

Ralf Doerner
Wolfgang Broll
Paul Grimm
Bernhard Jung *Eds.*



Virtual and Augmented Reality (VR/AR)

Foundations and Methods of
Extended Realities (XR)

 Springer

Virtual and Augmented Reality (VR/AR)

Ralf Doerner • Wolfgang Broll
Paul Grimm • Bernhard Jung
Editors

Virtual and Augmented Reality (VR/AR)

Foundations and Methods of Extended
Realities (XR)

 Springer

Editors

Ralf Doerner
Department of Design, Computer
Science, Media
RheinMain University of Applied Sciences
Wiesbaden, Germany

Paul Grimm
Department of Media
Darmstadt University of Applied Sciences
Darmstadt, Germany

Wolfgang Broll
Department of Computer Science
and Automation / Department for
Economic Science and Media
Ilmenau University of Technology
Ilmenau, Germany

Bernhard Jung
Institute for Informatics
TU Bergakademie Freiberg
Freiberg, Germany

ISBN 978-3-030-79061-5 ISBN 978-3-030-79062-2 (eBook)
<https://doi.org/10.1007/978-3-030-79062-2>

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2022

Translated from the German language edition: *Virtual und Augmented Reality (VR/AR)* by Dörner, © Springer-Verlag GmbH Deutschland, ein Teil von Springer Nature 2013, 2019. All Rights Reserved.

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Foreword

You hold in your hands – or are viewing on a screen – an excellent book on virtual and augmented reality. It will introduce you to a plethora of topics from a variety of viewpoints by multiple experts.

Virtual and augmented reality has indeed come a long way since Ivan Sutherland’s 1968 head-mounted 3D display. The paper starts:

The fundamental idea behind the three-dimensional display is to present the user with a perspective image which changes as he moves. ... if we can place suitable two-dimensional images on the observer’s retinas, we can create the illusion that he is seeing a three-dimensional object. ... The image presented by the three-dimensional display must change in exactly the way that the image of a real object would change for similar motions of the user’s head. (Sutherland 1968, first paragraph)

Every aspect of Sutherland’s above description – and much, much more – is addressed extensively in this book: (1) the basic elements of VR/AR systems, (2) the perceptual aspects of VR, (3) modeling of virtual worlds, (4) input devices and tracking, (5) output devices, (6) interaction, (7) real-time aspects of VR, (8) AR overview, (9) a group of case studies, (10) a tutorial on creating VR/AR applications with current hardware and software, and (11) mathematical foundations of VR/AR. This list is but a hint of the many topics in this volume.

This text will be useful and enjoyable for both the beginner as well as the experienced practitioner. For “old folks” like yours truly, many pages convey new information (trying to use VR to teach literacy to prisoners) and many trigger memories of old VR adventures (building 1980s VR systems with miniature TV displays and analog TV camera-based trackers).

In this volume you will learn not only about VR/AR topics but also about many other topics whose utility extends far beyond VR/AR boundaries: physically based modeling, user interfaces, real-time rendering, and haptic devices, to name just a few.

The authors present each chapter with a gentle introduction, clear organization, useful illustrations, clear exposition, and end each chapter with a summary, a set of self-study questions, a short list of recommended further readings, and the list of references for that chapter. The list of recommended further readings is particularly helpful for many of us who, when we delve into a topic of interest, often yearn for

more. This recommended reading list is a short annotated list, in contrast to the long, complete list of references for that chapter.

As a reader, I really appreciate this chapter-oriented organization; it is much easier to look up references and scan for other relevant readings at the end of a particular chapter than to consult a much longer list of all the combined references at the end of an entire book. When there are overlaps with other chapters, the authors make clear references to the other locations, to the related topics, and to the relevant illustrations.

This is indeed an excellent volume. It is sure to be useful both for adoption as a textbook in a VR/AR course and also for self-study. I highly recommend it.

The University of North Carolina at Chapel Hill,
Chapel Hill, North Carolina, USA
June 2021

Henry Fuchs

Preface

Professional associations and societies in the field of sciences, such as the Association for Computing Machinery (ACM) or the IEEE Computer Society, play an important role in advancing scientific discourse and education. Their activities lead to tangible outcomes, such as journals or conference proceedings. This book is the result of an initiative that started in the special interest group for virtual reality and augmented reality (VR/AR) of the German Society for Computer Science (GI). The members of this special interest group, who come not only from academia (e.g., universities, research institutes) but also from companies and organizations, started a project in 2010 that aimed to provide a scientifically based introductory book to VR/AR that can serve as a textbook for students of various disciplines and also cater to professionals and the interested public. It soon became clear that a multi-faceted and broad topic such as VR/AR needed the expertise of many authors from the group. However, just slapping chapters from different authors together does not necessarily lead to a good introductory text – especially for beginners, consistency and coherency are key. Therefore, we four editors saw our task as being to invest significant time and to heavily edit the initial texts from the authors to obtain a coherent and consistent book. In this context, we are especially grateful to Rolf Kruse, Professor of Digital Media and Digital Design, who ensured consistency and quality for all of the figures.

Since the German version of this book appeared for the first time in 2013, it has become widely popular, especially as basic literature for courses in VR and AR. The most recent German edition is from 2019. In 2020, we decided to publish this international edition. This did not just mean that we translated the book from German to English. For instance, the case studies that initially came only from Germany were replaced by case studies from all over the world.

This preface is a good opportunity to thank once again all those involved in this book project. These include not only the authors and those involved with Springer Nature but also all our readers of the German editions, especially our students and the members of the VR/AR special interest group of the German Society for Computer Science, who have given us incredibly valuable feedback that has been incorporated into this current edition. Among other things, we have complied with

the request to make the illustrations contained in the book available electronically for non-commercial use, for example, in lecture slides or student works. A corresponding package, which also contains code examples from Chap. 10, is available for free download at vr-ar-book.org.

Now, it is a pleasure for us to serve as your guide for your journey into the fascinating world of virtual and augmented reality.

Wiesbaden, Germany
Ilmenau, Germany
Darmstadt, Germany
Freiberg, Germany

Ralf Doerner
Wolfgang Broll
Paul Grimm
Bernhard Jung

Contents

1	Introduction to Virtual and Augmented Reality	1
	Ralf Doerner, Wolfgang Broll, Bernhard Jung, Paul Grimm, Martin Göbel, and Rolf Kruse	
2	Perceptual Aspects of VR	39
	Ralf Doerner and Frank Steinicke	
3	Virtual Worlds	71
	Bernhard Jung and Arnd Vitzthum	
4	VR/AR Input Devices and Tracking	107
	Paul Grimm, Wolfgang Broll, Rigo Herold, Johannes Hummel, and Rolf Kruse	
5	VR/AR Output Devices	149
	Wolfgang Broll, Paul Grimm, Rigo Herold, Dirk Reiners, and Carolina Cruz-Neira	
6	Interaction in Virtual Worlds	201
	Ralf Doerner, Christian Geiger, Leif Oppermann, Volker Paelke, and Steffi Beckhaus	
7	Real-Time Aspects of VR Systems	245
	Mathias Buhr, Thies Pfeiffer, Dirk Reiners, Carolina Cruz-Neira, and Bernhard Jung	
8	Augmented Reality	291
	Wolfgang Broll	

9 VR/AR Case Studies 331
Ralf Doerner, Alexander Tesch, Axel Hildebrand,
Stephan Leenders, Tobias Tropper, Wilhelm Wilke,
Christian Winkler, Julian Hillig, Alec Pestov, James A. Walsh,
Bruce H. Thomas, Gerhard Kimenkowski, Stephen Walton,
Torsten W. Kuhlen, Geert Matthys, Holger Regenbrecht,
Chris Heinrich, Xiumin Shang, Marcelo Kallmann, Benjamin Lok,
Francisco A. Jimenez, Cheryl Wilson, Marc Erich Latoschik,
Carolin Wienrich, Silke Grafe, Mario Botsch, and Jonny Collins

10 Authoring of VR/AR Applications 371
Wolfgang Broll, Florian Weidner, Tobias Schwandt, Kai Weber,
and Ralf Doerner

11 Mathematical Foundations of VR/AR 401
Ralf Doerner

About the Authors 413

Index 421

Chapter 1

Introduction to Virtual and Augmented Reality



Ralf Doerner, Wolfgang Broll, Bernhard Jung, Paul Grimm, Martin Göbel, and Rolf Kruse

Abstract What is Virtual Reality (VR)? What is Augmented Reality (AR)? What is the purpose of VR/AR? What are the basic concepts? What are the hard- and software components of VR/AR systems? How has VR/AR developed historically? The first chapter examines these questions and provides an introduction to this textbook. This chapter is fundamental for the whole book. All subsequent chapters build on it and do not depend directly on one another. Therefore, these chapters can be worked through selectively and in a sequence that suits the individual interests and needs of the readers. Corresponding tips on how this book can be used efficiently by different target groups (students, teachers, users, technology enthusiasts) are provided at the end of the chapter, as well as a summary, questions for reviewing what has been learned, recommendations for further reading, and the references used in the chapter.

1.1 What Is VR/AR About?

Let us first look at the ideal conception of a *Virtual Reality (VR)*: What is a perfect VR? In this extreme case the underlying ideas of VR become particularly clear. Then we will look at why perfect VR cannot be achieved today (and why one would not want to achieve it, e.g., for ethical reasons) and show how a *virtual environment* can still be created. We introduce the concept of *Augmented Reality (AR)*. Finally, we motivate what VR and AR can be used for today and why these topics are being dealt with intensively.

Dedicated website for additional material: vr-ar-book.org

R. Doerner (✉)
Department of Design, Computer Science, Media, RheinMain University of Applied Sciences, Wiesbaden, Germany
e-mail: ralf.doerner@hs-rm.de

1.1.1 *The Perfect Virtual Reality*

Humans perceive the world through sensory impressions. If, for example, light is reflected by a real object, such as a tiger, and enters a person's eye, photochemical processes are triggered in special sensory cells located in the retina. The light acts as a stimulus for these sensory cells. The light stimuli set off nerve impulses, which are modified via nerve cells that are connected in a complex way. These signals are then transmitted throughout the brain and processed further. Various areas of the brain that contribute to visual perception have already been identified. The perceived image is not created in the eyes, but rather in brain regions, mainly in the back of the head. The processes in the brain can be divided into several stages. At first, fast parallel processing of the visual sensory impressions takes place during which, for example, the yellow and black areas and also the pattern on the fur of the tiger are identified. Based on this, slower sequential processing follows, e.g., the composition of the colored surfaces to objects (as for example a paw or the teeth of the tiger) with the support of the person's memory. If the human being has already seen a tiger before, this can lead to recognition. We call the whole apparatus, from the sensory cells, via the visual nerves to the visual centers in the brain, the *visual system* of the human being. So, in our example, the human being sees the tiger thanks to the visual system and can draw conclusions about reality from this, e.g., that a real predatory cat is standing nearby and it would be a perfectly suitable time to start running away.

The connection between reality and what people perceive about it through their visual system is anything but simple. The same reality can cause different perceptions in different people. A wall that reflects light with a wavelength of 630 nm triggers the color perception "red" in many people – but some people have a different perception. Because they are in the minority, these people are called colorblind – after all, about 9% of men and 1% of women perceive colors differently than the rest of the population. Color, a term people use to describe visual perception, is therefore not a term that objectively describes reality. Color is not a physical property of the real wall but rather stands for a subjective perception that is indirectly triggered in people by the wall through reflected light.

Even in a single individual there is no simple connection between reality and visual perception of reality. If you look at Fig. 1.1, you can see black squares arranged on a grid. At the intersections of the grid, one can see alternating, partly flickering dark and bright points. But this does not correspond to the properties of the grid points in reality. All grid points are identical and always reflect the light in the same way (if this text is being read with an e-book reader, be assured that there is no trickery here). A number of such phenomena have been described in perceptual psychology, showing how the visual system combines, amplifies, filters out or recombines responses to external stimuli originating from the sensory cells during the complex process of perception. The same stimuli can lead to different perceptions in the same individual at different times, for example depending on whether the individual is concentrating on something or not – or whether the individual has

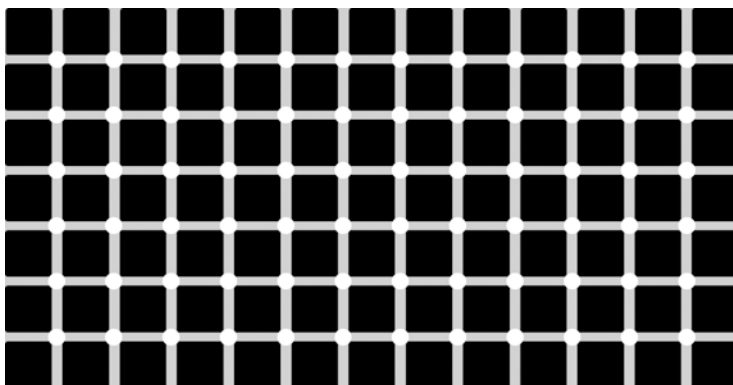


Fig. 1.1 A Hermann grid. Although in reality all grid intersections always reflect light to the same extent, a person sometimes perceives dark spots there. The dark spots disappear as soon as you try to look at them directly

just had a glass of vodka or not. A remarkable characteristic of the visual system is that it can also change its mode of operation over time, adapting itself. The psychologist George M. Stratton made this clear in an impressive self-experiment at the end of the nineteenth century. Stratton wore reversing glasses for several days, which literally turned the world upside down for him. In the beginning this caused him great difficulties: Just putting food in his mouth with a fork was a challenge for him. With time, however, his visual system adapted to the new stimuli from reality and he was able to act normally in his environment again, even seeing it upright when he concentrated. As he took off his reversing glasses, he was again confronted with problems: He used the wrong hand when he wanted to reach for something, for example. Fortunately for Mr. Stratton, an adaptation of perception is reversible, and he did not have to wear reversing glasses for the rest of his life. For him, everything returned to normal after one day.

We can conclude that there is no fixed, unambiguous and objective connection between (1) reality with the light stimuli it exerts on a human being and (2) the visual perception by the human being of this reality. This creates some leeway for manipulating the human visual perception of reality. A simple way is to replace a stimulus emanating from a real object with a similar, artificial stimulus. If the human visual system, stimulated by this artificial stimulus, comes to a similar perception as it would have done with a real object, the human being may even be under the mistaken impression that this object actually exists in reality. Images are a typical example of this approach. If one wishes to cause the visual perception “tiger” in a human being, then one does not need to inconvenience a real predatory cat. One can show the person a photograph of a tiger. Of course, this photograph of a tiger – a sheet of paper printed with pigments reflecting light in a certain way – is a fundamentally different object than a flesh and blood tiger. But both have something in common: They reflect light in a similar way, stimulate the visual system in a similar way and evoke similar visual perceptions in the human being.

Typically, a person will not be deceived so simply. People are usually able to distinguish a real tiger from a photo of a tiger. Therefore, let us assume that we could bring the light stimuli that emanate from a real tiger perfectly into the visual system of a human being, e.g., by playing in the impulses of sensory cells resulting from outside stimuli via a “socket” implanted into the brain. Let us go a step further in our thoughts and not limit ourselves to visual perception alone. Visual perception is the most important source of information about a person’s environment – more than 130 million sensory cells (about 70% of all human sensory cells) and more than four billion neurons, i.e., more than about 40% of the cerebral cortex, are involved in seeing. “Man is an eye animal” as Leonardo da Vinci put it. However, the human perception of reality is also based on other sensory impressions. For example, in addition to the cone cells in the retina that react to light, there are special sensory cells, such as Merkel cells, which respond to pressure, or the Pacinian corpuscles, which are stimulated by acceleration. Therefore, let us further assume that we could also transfer the reaction of all these other sensory cells directly to the brain via the imaginary “socket”. Besides seeing (visual perception) we would thus also manipulate

- hearing (auditory perception),
- smelling (olfactory perception),
- tasting (gustatory perception),
- feeling (haptic perception),
- and, as part of feeling, touch (tactile perception),
- sense of balance (vestibular perception),
- body sensation (proprioception),
- the sensation of temperature (thermoception),
- and the sensation of pain (nociception).

Would we then be in a position to have the stimuli emanating from a tiger calculated by a computer and played into the brain of a person in such a way that this person would be convinced that there was a real tiger nearby? Would we be able to put a human being into an apparent reality, a virtual reality, that the human being could no longer distinguish from the “real” reality? Can we create a perfect illusion of reality?

These are fascinating questions that the Wachowskis, for example, have vividly dealt with in their film *The Matrix* and its sequels. Other films, such as *Vanilla Sky* and science fiction novels by Stanislaw Lem, for example, also address this question. It also touches on philosophical questions such as those raised by Plato over 2400 years ago with his allegory of the cave. Plato wondered how people would react who had been trapped in a cave since childhood with their heads fixed in such a way that they never see objects behind them but only perceive the objects’ shadows cast on the cave wall visible to them. According to Plato’s Theory of Ideas, we do not directly recognize reality – the true being – but are only able to perceive indirectly “shadows”, images of reality in our “cave”, our world limited by the

realm of sensual experiences. Similar ideas can also be found, for example, in Indian mythology. Here, Maya, the goddess of illusion, prevents people from directly recognizing reality. Instead, Maya makes us experience only a projection of the world created by ourselves and our perception.

The French philosopher René Descartes went a step further. He stated that our perception of reality might not only be an imperfect image but a complete illusion and that all knowledge about reality is to be doubted. Descartes introduces the figure of the Genius Malignus, an evil spirit, who makes people believe in a reality that does not exist. So, you are not reading a book, but an evil spirit makes you believe that you have eyes and can read a book that does not exist in reality. In fact, the spirit is even so evil that it is a textbook about Virtual Reality.

The philosophical direction of skepticism doubts that there is such a thing as reality or such a thing as fundamental truths at all. With the “Brain in a Vat” experiment, a thought experiment similar to our considerations, the followers of skepticism justify their position. In this experiment, it is assumed that a brain extracted from a human being floating in a vat of nutrient solution is supplied by a computer with impulses that simulate an apparent reality. They answer our question of whether the consciousness in this brain can distinguish the faked reality from real reality, namely the disembodied brain floating in a tub, with a firm “No”. Therefore, the argument goes, we can never be sure whether we are in a Virtual Reality – just as most people in the feature film *The Matrix* never realize what their actual reality looks like.

1.1.2 The Simulation of the World

In order to realize a perfect Virtual Reality, at least to some extent, sensory stimuli must be generated that make a person perceive this alternative world. In the first flight simulators, a video camera was attached to a linkage and moved over a physical landscape model similar to a model railway. The images captured by the camera were displayed to the pilot in the flight simulator, who could thus perceive an image of the world when looking out of the cockpit. A more modern approach would be to use computer graphics to generate images or light stimuli for Virtual Reality.

But the generation of the stimuli is only one task on the way to the perfect Virtual Reality. People not only want to see and feel the world but also to act in it. For example, if a person perceives a ball in Virtual Reality, he or she might want to be able to kick the ball and run after it. This requires that the virtual world is simulated, that the actions of the person are known to the simulation, and that these actions can influence the simulation. The results of the simulation in turn have an effect on the generation of the stimuli – if a person moves in Virtual Reality, the generation of stimuli must also take the new position into account. The task of simulation can be performed by a computer system that must have a simulation model of the world at

its disposal. The simulation model of the world determines the behavior of the Virtual Reality. Consequently, the reactions of the virtual world in response to the actions of users must be simulated, as well as changes in the virtual world that do not depend on human actions. For example, a day-night cycle in the virtual world could be simulated that cannot be influenced by people.

One can strive to build the simulation model of the world in such a way that the behavior of the virtual world corresponds as closely as possible to that of reality. If a person kicks a virtual ball, the world simulation would move the ball according to the well-known laws of physics – the ball would have a virtual mass and a virtual frictional resistance, and would continue to roll on sloping virtual terrain until it reached a rest position. In Virtual Reality, however, one is not bound by the laws of reality. A kick against a virtual ball, for example, could also cause the ball to move along a serpentine path – or to turn it into a chicken. In this way you can create fantastic virtual worlds, virtual worlds that play in an imaginary future, or virtual worlds that recreate past times.

Being tasked with the recognition of human actions, the simulation of the virtual world, and the generation of stimuli for humans, the VR system can become highly complex. The simulation of a single virtual human being – which includes the generation of realistic images of skin and clothing, speech synthesis, and the simulation of human behavior, emotions, irony and willpower – is a major challenge today. The challenge is further increased by the requirement that this computer system must operate in real time, i.e., it has to keep pace with human beings. This implies that calculations must not take up arbitrary time but must adhere to strict time constraints. For example, a large number of images for Virtual Reality must be generated per second so that the human observer perceives movements in the virtual world as continuous and natural. The required number of images per second depends on the viewers and their current situation – typically 60 images per second are needed to meet the demand for real time (if the viewers have large amounts of alcohol in their blood, however, four images per second may be sufficient). This means that the computer system may not take more than 16 ms to generate images. Real-time requirements are even more demanding for haptic feedback. Typically, the VR system must generate haptic stimuli 1000 times per second in order to create a convincing sensation of touch.

We call a *VR system* a computer system consisting of suitable hardware and software to implement the concept of Virtual Reality. We call the content represented by the VR system a *virtual world*. The virtual world includes, for example, models of objects, their behavioral description for the simulation model and their arrangement in space. If a virtual world is presented with a VR system, we speak of a *virtual environment* for one or more users.

1.1.3 *Suspension of Disbelief*

The Matrix in the feature film of the same name and the Holodeck in the television series *Star Trek* both transport a person into Virtual Reality. There is one crucial difference: In the Matrix, people do not know that they are in Virtual Reality at all. On the contrary, people enter the Holodeck on the starship *Enterprise* consciously. They go through a door into the virtual environment and know that it is a simulation, but in reality, they are still in a large hall. Nevertheless, people seem to perceive the Holodeck as very real. Does it not bother you to know that you are in Virtual Reality? Can the illusion of a virtual world be achieved at all if you are aware of being in Virtual Reality?

Let us consider the following experiment. We put a helmet on a person, in which two small monitors, one for each eye, are attached. The person can no longer perceive the environment visually, but only the images in the monitors, which are fed in from outside. A sensor is built into the helmet which can determine how the person is turning their head and where the person is located. This information is used to adjust the generated images to the current head position: If the person looks up, images from the sky are shown; if the person tilts their head downwards, then he or she sees the ground; and if the person takes a step forward, then images from this new position are shown. We use a computer to create images of the roof of a virtual skyscraper and want to give the impression that the person is standing at a dizzy height on the edge of a huge building. If you observe people in this situation, you often see that they move forward very slowly and carefully. The closer they get to the edge of the building, the faster their pulse and breathing become. Their hands get wet. These are typical fear reactions that are caused by a danger such as an abyss in reality. The people are always aware that the building is only virtual, that in reality there is no abyss at all, and that they are standing safely in a room. Nevertheless, they succumb to the illusion of Virtual Reality and react to it as if it were the real world.

In certain situations, people possess the ability to blank out the obvious contradiction between a fictitious world and reality. Besides, people want to do this. The philosopher Samuel T. Coleridge coined the expression “*willing suspension of disbelief*”. For entertainment purposes, people are prepared to accept the figure of Scrooge McDuck and his virtual world Duckburg as existing, even if it is known that this character consists only of hand-drawn lines and that in reality older drakes do not bathe in money. In dubbed films, one fades out the fact that James Bond as an English agent obviously does not always speak perfect Japanese or German. However, this “suspension of disbelief” is not easy to describe and is sometimes selective. Cartoonist Gary Larson describes the indignation of his readers about the fact that in one of his cartoons a polar bear is surrounded by penguins. Readers criticized that this is impossible since polar bears live at the North Pole, but penguins live at the South Pole. However, at the same time readers are not in the least bothered by the fact that the penguins in the cartoon talk to each other and the polar bear has disguised himself as a penguin.

For the creation of Virtual Reality, this human characteristic of blanking out disbelief means that one does not have to resort to drastic measures. Fortunately, there is no need to drill holes in the top of someone's skull and directly manipulate the brain in order to put people into a virtual environment in which they feel present. In this way, Virtual Realities can be created at different stages of technological advancement, where the ultimate stage would allow the creation of the perfect Virtual Reality discussed above. In fact, highly believable virtual environments can already be created today with relatively little effort.

1.1.4 Motivation

What is the point of all this? Why would you want to build a virtual environment at all and put people into it? What are the advantages of dealing with Virtual Reality? There are many answers to these questions. We will consider some of them in the following.

If the world simulation is performed by a computer, then Virtual Reality is the interface between the computer system and the human being. Under this perspective, every Virtual Reality implements a human-machine interface. This interface can be characterized as being particularly natural and intuitive. For example, instead of using a mouse and keyboard, the use of a steering wheel and foot pedals for a car racing game is a step towards Virtual Reality that makes the operation of the virtual car and its navigation through the virtual world more natural. A perfect Virtual Reality can then be understood as a perfect user interface for software. Users can simply act as they are used to doing in the world. Ideally, they are completely unaware of the fact that they are interacting with a computer program. In this respect, the engagement with Virtual Reality can be understood as a methodical approach to finding new forms of human-computer interaction by working towards a vision of a perfect Virtual Reality. Even though this vision may never be achieved (or one may not even want to achieve this, because extensive manipulation of humans is ethically questionable), valuable new ideas can emerge along the way and innovative user interfaces can be designed to make it easier for humans to handle computer systems.

By exploiting its sophisticated visualization capabilities, Virtual Reality can also make it easier for people to absorb and understand data. For example, through years of study and experience, architects have acquired the ability to imagine a building in their minds by looking at 2D construction plans – many real-estate investors do not have this ability. Virtual Reality can also visualize the data in the construction plans for clients in such a way that they can get a good impression of the building and make more informed decisions regarding alternative design choices. Complex results of computer simulations, e.g., the calculation of how air would flow around a newly planned vehicle, can be visualized directly on a virtual vehicle. Engineers and designers can work together in the virtual world to develop aesthetically

pleasing body shapes that avoid air turbulence and reduce the vehicle's air resistance. Even completely abstract data can be displayed in Virtual Reality. In this way, an analyst can be transported to a virtual world of financial data.

Virtual realities offer researchers tools to find out more about human perception. For example, experiments can be conducted in Virtual Reality that help to gain insight into how people orient themselves in three-dimensional space. In addition to gaining knowledge in science, Virtual Reality can also offer a very practical use with tangible financial benefits, as case studies show, e.g., on the use of VR in construction (see Chap. 9).

Hardly any car is built today without using methods from Virtual Reality. For example, designs can be visualized more realistically, and prototypes can be created more cost-efficiently than in traditional model-making (see Chap. 9). How the robots in production lines of automobiles are adjusted to a new car model can be simulated in a virtual world and presented in Virtual Reality before the start of production. The analysis of the planning and the elimination of planning errors in a virtual plant or a virtual factory is much easier and more cost-efficient than performing it in the real world.

Pilots take advantage of Virtual Reality during their training in flight simulators. By not using a real aircraft, the airline saves money. But training in Virtual Reality does not only have financial advantages. As no kerosene is burned, less CO₂ is emitted, which benefits the environment. In comparison to a real aircraft, the pilots can rehearse extreme situations without danger. In addition to flight simulators, simulators of ships, trams, trains, and trucks are also commonly used. German air traffic control even operates a virtual airport where air traffic controllers can train. Another example is the training of personnel for complex systems, such as operating the control center of a coal-fired power plant or maintaining aircraft. Virtual Reality allows training to take place even before the real object is completed, so that well-trained personnel are already available at the time of commissioning. In addition to training in the civilian sector, Virtual Reality also has application potential in the military. For example, crews of fighter jets or tanks are trained in virtual environments.

Interested people can buy tickets to an attraction that allow them to drive a high-speed train through a virtual landscape sitting in a highly realistic mock-up of a locomotive. This is an example of how Virtual Reality is used for entertainment purposes in simulation games. Other game genres also benefit from the use of Virtual Reality, so players can experience adventures in fantastic worlds in adventure games. Very close to reality, tourists can experience historical cities such as ancient Rome by visiting them in Virtual Reality. Museums can offer engaging sensual experiences in virtual environments. Artists use Virtual Reality for installations. Virtual Reality arouses interest and can serve as an eye-catcher – accordingly, it offers potential for marketing, for example at trade show booths.

In medicine, there are possible applications in the field of training. Doctors can practice and plan operations in Virtual Reality without any risk for their patients. Nursing staff can train in the handling of patients. Virtual Reality can even be used

for treatment. As already described, people can be positioned at a virtual abyss. In this way, people with a fear of heights can be confronted with critical situations and their phobia can be treated. In Virtual Reality, the factors that cause fear can be safely controlled and dosed during treatment.

The range of possible applications of Virtual Reality can be significantly expanded by trying not to completely cut people off from reality when placing them in a virtual environment. Instead, one can try to integrate parts of a virtual world into reality. Let us look again at the example already described where we have placed a person on the edge of a virtual abyss. Would it not be more effective if we did not have to put a helmet on the person and instead could place him or her on a large glass plate? An image from the virtual world would be projected onto this glass plate from below instead of showing it on the small monitors in the helmet. If the person looks down, he or she can see not only the virtual edge of the building but also their own real feet. So the person still perceives reality, but additionally, at some points, integrated parts from a virtual world that fit into reality. The idea of augmenting images from reality in real time by exactly fitting virtual partial images opens up a whole field of new application possibilities for VR technologies. Another example is the use of special binoculars, similar to the well-known coin-operated binoculars, that are permanently installed at viewing points. When looking through the binoculars, the user sees not only reality but also parts of a virtual world that are displayed according to the area of reality currently being viewed. For example, if the observer is looking at the tower remains of an old castle ruin, the binoculars can display a virtual tower at exactly this point, just as it might have appeared several centuries ago. In this case, one no longer speaks of Virtual Reality (VR) but of *Augmented Reality (AR)*. The virtual and real portions of an image can vary. In fact, there is a smooth transition. One speaks of AR when the real parts predominate. In Sect. 1.2, we look at VR in more detail, while AR is the subject of Sect. 1.3.

So, there are many reasons to learn more about the theoretical foundations of VR and AR as well as the practice of creating convincing virtual and augmented worlds. If one embarks on this endeavor, one is confronted with many questions. What do you have to consider if you want to put people into a virtual world? What makes it believable? What is conducive to achieving suspension of disbelief – and what can destroy it? What effort must be made in a particular application area to achieve this? How is the transmission of different stimuli from a VR technically realized? Which devices are there to make it easier for a person to immerse him or herself in Virtual Reality? How is a computer system structured that generates the corresponding stimuli, e.g., generates images from a VR close to reality? What is the system architecture of a VR system? Which interfaces are there, which norms, and which standards? How do you build simulation models for the world simulation of VR? How does the simulation get information about the actions of people? How can people move in a virtual world? Which algorithms are used in VR? What is the runtime of these algorithms? How can the VR system meet real-time requirements? When looking at AR in comparison to VR, additional questions arise: Which technology is used to include parts of a virtual world into reality? What is the relationship between

virtual and real objects? Can they occlude each other? How is a virtual object illuminated with a real light source? How does a virtual object cast a shadow on a real object? How can a virtual object be placed on top of a real object?

In science, but also in practical implementation, many people have already dealt with such questions and contributed to finding answers. In this textbook, basic scientific knowledge in the field of VR and AR is compiled and its practical application is illustrated with case studies. The knowledge conveyed in the book is a solid foundation for all those who want to use VR and AR practically, but also for those who want to actively contribute to the vision of a perfect Virtual Reality through research and development in the field.

1.2 What Is VR?

As should be clear from the introductory remarks above, one can approach the field of VR in very different ways. At the visionary end of the spectrum, e.g., in science fiction movies and popular culture, “perfect VR” is presented as a comprehensive simulation which is no longer distinguishable from human reality. At the practical end of the spectrum, VR has long been established as a tool for product development in many industrial sectors. In this section, we examine how the scientific and technological field of VR is characterized by the members of the research community.

VR is a relatively young field of science and its development is strongly driven by rapid advances in the underlying hardware. In view of this, it may come as no surprise that the scientific discipline of VR has so far not produced a uniform definition of “Virtual Reality”. Nevertheless, there is very broad agreement on the essential or desirable features of VR. The following characterizations of VR take different perspectives to differentiate VR systems from traditional human–computer interfaces: the focus on technological aspects, the classification of VR as a new form of human–computer interaction, and the emphasis on the mental experience of VR.

1.2.1 *Technology-Centered Characterizations of VR*

“The ultimate display would, of course, be a room within which the computer can control the existence of matter. A chair displayed in such a room would be good enough to sit in. Handcuffs displayed in such a room would be confining, and a bullet displayed in such a room would be fatal. With appropriate programming such a display could literally be the Wonderland into which Alice walked.” (Sutherland 1965)

An iconic feature of VR in many photos or other visual depictions is the special input and output devices worn by the users such as head-mounted displays (HMDs), stereo glasses, spatial tracking devices or data gloves. Accordingly, one way to characterize VR is by highlighting its technological components. However, there is a certain danger with technology-centered approaches that such definitions of VR may refer too much to specific input and output devices (e.g., “wired data suits”), which become quickly outdated by technological progress. “Future-proof” definitions of VR should also be compatible with visionary ideas like Sutherland’s Ultimate Display or the Holodeck from *Star Trek*. The following technology-oriented characterizations from the early years of VR still apply to today’s VR systems:

“Virtual Reality (VR) refers to the use of three-dimensional displays and interaction devices to explore real-time computer-generated environments.” (Steve Bryson, Call for Participation 1993 IEEE Symposium on Research Frontiers in Virtual Reality)

“Virtual Reality refers to immersive, interactive, multi-sensory, viewer-centered, three-dimensional computer-generated environments and the combination of technologies required to build these environments.” (Carolina Cruz-Neira, SIGGRAPH ’93 Course Notes “Virtual Reality Overview”)

These characterizations of VR can perhaps best be understood in contrast to “traditional” computer graphics, as the science and technology field from which VR evolved. VR builds on 3D content from computer graphics but focuses in particular on *real-time* computer graphics. Matching the 3D content, *three-dimensional displays* are used for its presentation. In the case of the sense of vision, this is achieved using stereoscopic displays. Moreover, 3D content is often presented in a multi-sensory manner by addressing further senses such as hearing or touch, for which spatial audio and haptic feedback devices are employed. Besides 3D presentation, VR systems also facilitate *3D interaction*. 3D interaction devices are input devices whose position and orientation can be tracked in 3D space. Whereas in desktop systems the classic mouse and other “pointing” devices such as trackpads only provide 2D positional information, VR systems make use of 3D tracking to realize, for example, natural pointing gestures. By tracking body and finger movements, grasping of virtual objects can be simulated. *Interactivity* includes users receiving sensory feedback on their inputs, e.g. by mapping hand movements directly onto a virtual hand model. The tracking of the user’s position and orientation, particularly *head-tracking*, is the basis for another characteristic of VR systems: Viewer-dependent image generation. If a VR user moves in real space, the 3D environment is automatically displayed from her new perspective. Steve Bryson (2013) succinctly summed up this quintessential property of VR: “*If I turn my head and nothing happens, it ain’t VR!*”

Immersion is often considered as an essential feature to distinguish VR from other kinds of human–computer interfaces. Unfortunately, the term immersion is used in non-uniform ways in the literature. Following Slater and Wilbur (1997), we take a technology-centered view of immersion. According to Slater and Wilbur (1997), immersion is based on four technical properties of display systems: *Inclusive* (I) indicates the extent to which the user’s sensory impressions are generated by the computer, i.e., the user should be largely isolated from the real environment. *Extensive* (E) refers to the range of sensory modalities accommodated. *Surrounding* (S) indicates the extent to which the presentation is panoramic rather than limited to a narrow area. *Vivid* (V) indicates the resolution, fidelity and dynamic range of stimuli within a particular sensory modality. Immersion is therefore a gradual characteristic that is achieved to different degrees by different displays. For example, HMDs are usually considered highly immersive displays, since the visual sensations of the user are exclusively computer-generated. However, an HMD with a small field of view is less immersive than an HMD with a wider field of view. Similarly, multi-wall projections like CAVEs (see Sect. 9.2) are more immersive than single-screen projections.

The goal of total immersion is achieved by today’s VR displays to varying degrees. The terms *immersive VR* or *fully immersive VR* usually refer to VR systems based on HMDs or CAVEs. Desktop systems that provide stereoscopic displays and head-tracking are sometimes referred to as *non-immersive* and large single-screen or table-top displays as *semi-immersive VR*.

Besides the use of the term immersion as a technical property of VR displays, some authors also use the term to describe a mental quality of the VR experience (e.g., Witmer and Singer 1998). To differentiate between the two uses, one also speaks of *physical immersion* and *mental immersion* (Sherman and Craig 2003) and sometimes also of *physiological* and *psychological immersion* (Sadowski and Stanney 2002).

Table 1.1 summarizes the distinguishing features of VR as compared to conventional computer graphics.

Table 1.1 Features of VR as compared to conventional computer graphics

3D Computer Graphics	Virtual Reality
Visual presentation only	Multimodal presentation (i.e., addressing several senses, e.g., visual, acoustic and haptic)
Presentation planning/rendering not necessarily in real-time	Real-time presentation planning and rendering
Viewer-independent image generation (exocentric perspective)	Viewer-dependent image generation(egocentric perspective)
Static scene or precomputed animation	Real-time interaction and simulation
2D interaction (mouse, keyboard)	3D interaction (body, hand and head movements and gestures) + speech input
Non-immersive presentation	Immersive presentation

1.2.2 VR as an Innovative Kind of Human–Computer Interaction

“The promise of immersive virtual environments is one of a three-dimensional environment in which a user can directly perceive and interact with three-dimensional virtual objects. The underlying belief motivating most virtual reality (VR) research is that this will lead to more natural and effective human–computer interfaces.” (Mine et al. 1997)

Another way to characterize VR is to emphasize the goal of creating human–computer interfaces, which, in comparison to traditional user interfaces, enable much more natural or intuitive interaction with the three-dimensional simulated environment (see Fig. 1.2).

Graphical user interfaces (GUIs) and the associated WIMP (Windows, Icons, Menus, Pointing) interaction style represent a paradigm of human–computer interaction that has been dominant for several decades. The WIMP paradigm, which was originally developed for document-processing tasks, however, turns out to be rather inefficient when manipulating 3D content. For example, the task of repositioning an object in 3D space can be naturally achieved in VR by grasping and moving the object. In 2D GUIs, however, this task usually has to be broken down into several subtasks, e.g., first move the object in the xy -plane, then move in the z -direction. Besides the additional motor effort (e.g., two 2D mouse movements instead of one hand movement in 3D space), this also requires an additional cognitive effort for remembering when and how to change the system control state (e.g., how do you tell the computer that the next 2D mouse movements should be interpreted as translation in the z -dimension of 3D space?). As a prerequisite for successfully completing the task, the user must first learn how the 3D task can be broken down into a sequence of 2D subtasks, i.e., there is also a greater learning effort.



Fig. 1.2 Example of natural interaction: A virtual switch is turned like a physical switch using one’s hand

Virtual and Augmented Reality, along with other innovative forms of human-computer interaction, are examples of so-called *post-WIMP interfaces*. Post-WIMP interfaces build on interaction techniques that exploit prior knowledge and skills that the human user has already learned from everyday interactions with physical objects. For example, a person knows from everyday experience how one can use one's body to manipulate objects and has expectations of how these objects will typically behave as a consequence of this interaction. By using this knowledge, learning and further mental effort in natural interaction techniques may be greatly reduced when compared with WIMP techniques.

The following quote from Robert Stone explains the goal of *intuitive user interfaces* in the context of VR systems:

“An intuitive interface between man and machine is one which requires little training ... and proffers a working style most like that used by the human being to interact with environments and objects in his day-to-day life. In other words, the human interacts with elements of his task by looking, holding, manipulating, speaking, listening, and moving, using as many of his natural skills as are appropriate, or can reasonably be expected to be applied to a task.” (Stone 1993)

Compared to other innovative forms of human-computer interaction, VR offers great potential for an especially thorough realization of intuitive human-machine interfaces in the sense of Robert Stone. However, the goal of completely natural forms of interaction has arguably not yet been achieved nor is it always aimed for in today's VR systems. Nevertheless, through the use of 3D input and output devices, interactions in existing VR systems are typically much more natural than is the case with conventional 2D interfaces.

“The primary defining characteristic of VR is inclusion; being surrounded by an environment. VR places the participant inside information.” (Bricken 1990)

Metaphors represent another important aspect in the design of human-computer interfaces. Metaphors aim to explain the user interface through analogies with everyday life experiences. Within the WIMP paradigm, well-known examples of metaphors are the desktop, folders with documents in them, or cutting and pasting for transferring parts of one document to another. In the case of VR, the term Virtual Reality itself is a metaphor that makes the analogy to reality as such. The VR metaphor conveys to the user that the objects in the simulated world behave realistically and that natural forms of interaction are supported. Another aspect of the VR metaphor is that the user is situated within the simulated world and experiences it “from the inside” instead of looking at the simulated world “from the outside” through a

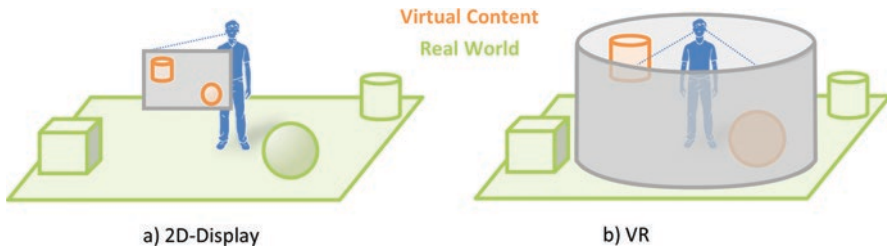


Fig. 1.3 Interaction models for desktop computers and VR: **(a)** When looking at the 2D display of a desktop computer, the user perceives both the real world and the computer-generated environment. **(b)** According to the VR metaphor the user is completely situated within the computer-simulated virtual world and fully isolated from the physical world. (c.f. Rekimoto and Nagao 1995)

window as with conventional desktop computers. According to the VR metaphor – which could be implemented using future perfectly immersive systems – the user is totally isolated from physical reality so that all sensory impressions are computer-generated. Fig. 1.3 contrasts the interaction models of conventional desktop computers with 2D displays and VR.

1.2.3 *Mental Aspects of the VR Experience*

“At the heart of VR is an *experience* – the experience of being in a virtual world or a remote location” (Rheingold 1991)

In perfect VR, all of the sensory impressions of the user would be generated by the computer, in the same quantity and quality as people are used to in the real world. Human actions in VR would have the same effects and virtual objects would affect people as they do in the real world. Today’s VR systems are by no means perfect, but the development of VR technology is aimed at ever more realistic experiences. But if the computer-generated sensory level is no longer (or hardly) distinguishable from physical reality, what effects does this have on higher-level processes of human perception? Does the user perceive the pixels of the visual displays as pictures or does the feeling of being at another place emerge? What other properties characterize the mental experience of VR? How can you measure or otherwise quantify these properties? How does this inform the design of virtual worlds and the setup of VR systems?

In VR research, these and similar questions regarding the mental experience of VR have played an important role right from the start. The fact that these questions are still the subject of research shows on the one hand their relevance for the research area of VR, but on the other hand that no generally accepted answers have yet become established. Unfortunately, the relevant terms in the literature such as

“immersion” and “presence” are sometimes used with different meanings. As noted above, we reserve the term “immersion” in this book, consistent with much of the research community, to exclusively describe the technical properties of VR systems. In contrast, some authors also use the term to describe the mental sensations of VR experiences. When reading different texts on VR, it is necessary to pay close attention to how key terms such as immersion are used. The following presentation of the most important concepts for the analysis of the mental experience of VR essentially follows the terminology of Slater (2003, 2009).

Presence is the central concept for describing the mental aspects of the VR experience. In a broad sense, it refers to the feeling of being within the virtual environment that is displayed by an immersive VR system (“being there”). The concept of presence was originally developed in the context of telerobotics. The aim was to provide the operator with the most realistic impression possible of the robots’ environment during remote control of robots, in particular using immersive VR technologies such as HMDs and data gloves. In the early 1990s, the concept of presence was transferred to VR (Held and Durlach 1992; Sheridan 1992). Evidence for (the feeling of) presence is, for example, when VR users react to the virtual environment as if it were a real environment. The concept of presence can be further decomposed to involve three different components:

First, the *place illusion* refers to the feeling of being in the location presented by the VR system (Slater 2009). The place illusion is the human response to a given level of immersion. It tends to arise naturally in highly immersive systems, but is more difficult to achieve with desktop systems (Slater 2009). Particularly important is the ability of the immersive VR system to display the scene from the perspective of the viewer. If the user turns their head, then the virtual environment should still be visible, just from a different perspective. If this is not the case, e.g., due to a single-screen setup, a *break in presence* may occur.

Second, the *plausibility illusion* arises when the events of the simulated environment are perceived as if they are really happening (Slater 2009). While the place illusion is largely induced by how the virtual world is presented, the plausibility illusion has to do with the content of the simulated world. The plausibility illusion relates in particular to events that affect the user but were not initiated by the user him or herself (e.g., a projectile suddenly flying towards the user or a virtual person who appeals to the user). The *believability* of the virtual environment seems to be more important than sensory realism for the emergence of the plausibility illusion. For example, a visually perfectly represented virtual person who communicates only in simple phrases would lead to a break of the plausibility illusion.

Third, *involvement* refers to the level of user attention or interest in the simulated world (Witmer and Singer 1998). Involvement, like the plausibility illusion, is mainly related to the content of the virtual environment. For example, in an immersive VR system, users might feel strongly that they are part of the simulated world (convincing place illusion), but may still get bored (low involvement).

To test whether and to what degree the feeling of presence arises with users, experimental studies with test persons are necessary. Different users may experience different levels of presence in one and the same VR application. One way to

record presence is to use special questionnaires (e.g., Witmer and Singer 1998). Furthermore, the behavior of the experiment's participants can be observed, for example movements (e.g., a user ducks away when an object comes flying towards them at high speed) and emotional expression such as fright. Other studies measure physiological parameters such as heart rate or skin resistance, which are often interpreted as signs of stress. In Slater et al. (2010) a "VR in VR" scenario is proposed as a further possibility for quantifying presence, in which the user can configure a VR system in the simulated world that generates a maximal level of presence.

Finally, the feeling of presence is not limited to VR, but may also arise with other media, such as books, movies or arcade machines, though perhaps not equally intensely. A further discussion on this can be found in Sherman and Craig (2003), for example.

1.3 What Is AR?

In the literature, a large number of different and sometimes contradictory definitions of AR exist. While Ivan Sutherland was the first to create an AR system in the late 1960s (Sutherland 1968), the definition according to Azuma from 1997 is widely used in science.

"Augmented Reality (AR) is a variation of Virtual Environments (VE), or Virtual Reality as it is more commonly called. VE technologies completely immerse a user inside a synthetic environment. While immersed, the user cannot see the real world around him. In contrast, AR allows the user to see the real world, with virtual objects superimposed upon or composited with the real world. Therefore, AR supplements reality, rather than completely replacing it." (Azuma 1997)

According to Azuma (1997), an AR system (see also Sect. 1.6) has the following three characteristic features: (1) It combines reality and virtuality. (2) It is interactive in real time. (3) The virtual contents are registered in 3D. While the second feature is also found in VR, the other two aspects differ significantly from VR. The combination of reality and virtuality is typically achieved by overlaying reality with (artificial) virtual content. That is, an observer (the AR user) simultaneously perceives the real environment and the virtual objects within it as a coherent whole. The virtual content allows for real-time interaction. Furthermore, the virtual content is registered in 3D (i.e., geometrically). This means that in an AR environment, a virtual object has a fixed place in reality and, as long as it is not changed by user interaction or changes itself, e.g., by animation, it remains there. In other words, from the user's perspective, it behaves exactly like a real object that would be in that location. As registration in 3D space and visual superimposition occur in real-time,

this does not change even if the user changes their perspective and therefore perceives a different part of the environment.

In the domain of popular science, the term AR is often used to refer to examples limited to the first of the features described by Azuma (i.e., the augmentation of reality by virtual content), while interactivity, real-time capability and especially 3D registration are frequently ignored.

More generally, AR may be defined as follows:

Augmented Reality (AR) refers to the immediate and seamless perception of the real environment enriched by virtual content in real-time, the latter resembling reality to the largest extent possible regarding its characteristics, appearance, and behavior, so that (if desired) sensory impressions from reality and virtuality may become indistinguishable (for any senses).

Implicitly, this definition also includes the aspects of interactivity and real-time capability, though it considers AR from the perceptual perspective. While AR today (as in much of this book) is mostly limited to the augmentation of visual perception, it can, just like VR, extend to any other form of sensory experience, including auditory, olfactory, gustatory, haptic (including tactile), vestibular, proprioceptive, thermoceptive and nociceptive perception. In contrast to VR, it is not intended to replace the sensory impressions completely by virtual ones. Rather, real and virtual sensory impressions are mutually superimposed.

In addition to AR, the term *Mixed Reality* (MR) is often used, indicating that real and virtual content are mixed together. Although MR and AR are often used interchangeably, MR, unlike AR, represents a continuum. The MR taxonomy of the reality–virtuality continuum introduced by Paul Milgram et al. (1995) is widely accepted in the research community (see Fig. 1.4).

Reality–Virtuality Continuum (according to Milgram): Mixed Reality (MR) is a continuum that extends between reality and virtuality (Virtual Reality), whereby the share of reality continuously decreases while that of virtuality increases. As far as the share of virtuality is prevailing here, without the environment being completely virtual (Virtual Reality), one speaks of Augmented Virtuality. If on the other hand the share of reality is larger, then we are talking about AR.

While Azuma sees AR as a special case of VR, Milgram et al. define AR as one representation of MR, whereas MR and VR are disjunct. Thus, while using the AR definition from Azuma, we will apply the taxonomy from Milgram throughout the remainder of this book. Furthermore, although the term XR as an abbreviation for *eXtended Reality* goes back to a patent application by the photographer Charles

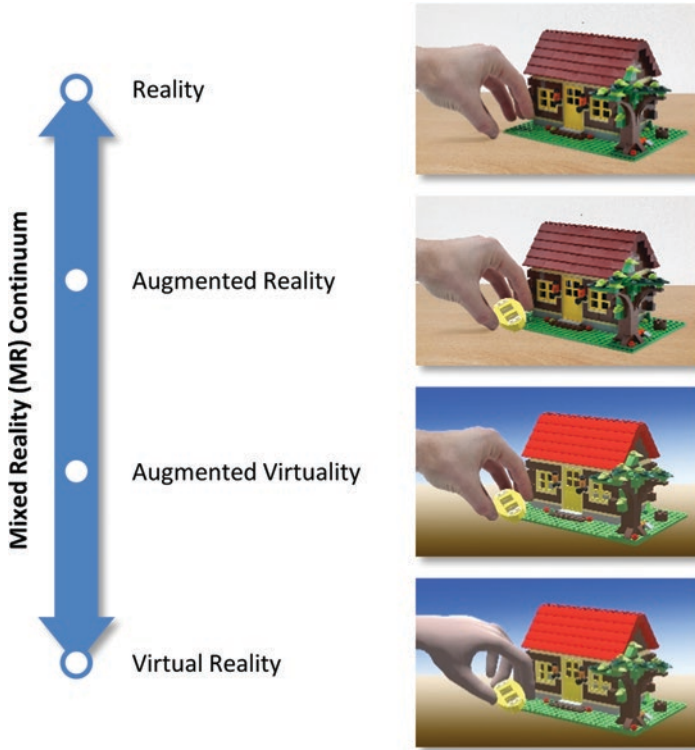


Fig. 1.4 Reality–Virtuality Continuum. (According to Milgram et al. 1995)

Wyckhoff in 1961, it has also been used since then by Sony, for example, to describe their X-Reality technology, or by Paradiso and Landay (2009) and others to describe types of *Cross-Reality*, i.e., a crossbreed between a Virtual Reality and ubiquitous sensor/actuator networks placed in reality. In this book, however, we will use the currently most common variant, namely XR as a generic term for VR and MR (and by that also AR). In this sense, the “X” may also be considered as a placeholder for “V”, “A” or “M”. As the “X” resembles a cross, XR is also sometimes referred to as Cross Reality (not to be confused with the concept of Cross-Reality mentioned above).

“Virtual Reality (VR) replaces the user’s perception of the real environment by a virtual world. In contrast, Augmented Reality (AR) augments or enhances the perception of reality by virtual content – Diminished Reality (DR) removes parts from the real environment. Augmenting, enhancing, deliberately diminishing, or otherwise altering the perception of the real environment in real time is referred to as Mediated Reality” (Mann 1999)

VR replaces the perception of the user’s real environment by that of a virtual world. AR enriches the user’s perception of the real environment by virtual content (see Fig. 1.5). In *Mediated Reality* the perception of the real environment is augmented, enriched, consciously diminished or otherwise changed in real time (Mann 1999). If the perception of reality is consciously reduced, i.e., real contents of the environment are deliberately removed from the perception of the user in real time, this is called *Diminished Reality* (DR). While not necessarily following the extended taxonomy of Mann et al. (2018), we will use their definitions of Mediated Reality and Diminished Reality in this book. Further, we will consider eXtended Reality (XR) to be a subset of Mediated Reality. For clarification of the taxonomy as used in this book, refer to Fig. 1.6.

Comparing AR with VR (see Table 1.2), it becomes obvious that many basic characteristics are very similar, if not identical. Both use a multimodal presentation, in that both interaction and simulation take place in real time, both visualize virtual 3D objects, and both use the egocentric perspective, i.e., the visualization is (at least conceptually) correct in terms of perspective for the respective viewer (although this

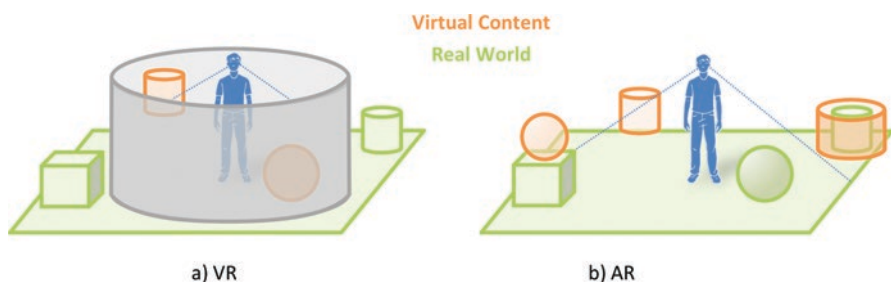


Fig. 1.5 AR compared to VR. In contrast to VR, the user interacts with the virtual content as well as with the real environment. Furthermore, an interaction between the real environment and the virtual content can take place. Virtual content and the real environment are not strictly separated from each other, but can overlap, be superimposed or penetrate each other

Fig. 1.6 Euler diagram showing the relationships between Augmented Reality (AR), Augmented Virtuality (AV), Mixed Reality (MR), Virtual Reality (VR), eXtended Reality (XR), Diminished Reality (DR) and Mediated Reality

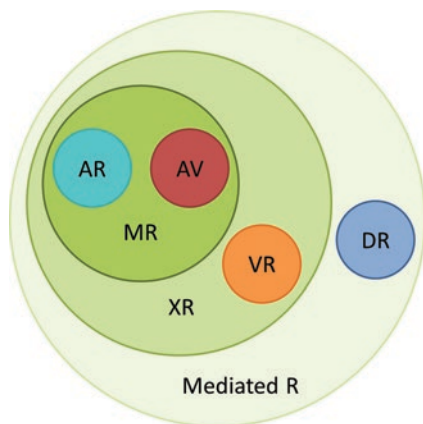


Table 1.2 Features of AR as compared to VR

Virtual Reality	Augmented Reality
Multimodal presentation	Multimodal presentation
Real-time presentation planning and rendering	Real-time presentation planning and rendering
Viewer-dependent image generation(egocentric perspective)	Viewer-dependent image generation(egocentric perspective)
Real-time interaction and simulation	Real-time interaction and simulation
Virtual 3D objects	Virtual 3D objects
All content purely virtual	Combination of reality and virtual content
Immersive presentation (central aspect)	Immersive presentation (open issue)
Tracking	Tracking and geometric (3D) registration
Implicit (restricted) and explicit navigation	Implicit (unrestricted) navigation
Stationary	Stationary or mobile
Indoor	Indoor and outdoor
Virtual illumination	Mutual influence of real and virtual illumination
Arbitrary scaling of the user perspective	User perspective always unscaled (virtual models may have limited scalability)

is not always the case with actual VR and AR systems). However, there are also a number of differences: The most obvious difference is that in VR all content is purely virtual, whereas in AR the virtual content is embedded in the real world. Accordingly, there is no real immersion in AR like there is in VR. For its application to AR, the concept of immersion would have to be significantly expanded. In AR, the focus is rather on the correct superimposition or fusion of reality and virtuality. This is achieved by registration. VR and AR also differ with respect to navigation. While in VR implicit navigation (the user moves in the virtual world analogous to movement in reality) is limited due to the inherent limitation of the dimensions of the room, the tracking area, the cable length of the HMD or the dimensions of the CAVE (see Sect. 1.4), navigation in AR is often unrestricted. For this purpose, VR additionally enables explicit navigation, in which the user changes their point of view by changing the camera position using specific interaction techniques. This allows, for instance, the user to fly through a virtual world, which is obviously not possible in AR at all. VR takes place primarily in closed rooms and these are usually stationary (location-bound) systems. Although there are many AR applications for indoor use, AR is generally not limited to these. Many AR applications are mobile and used outdoors. Also, the lighting and scaling of the virtual contents are fundamentally different. While in VR only the virtual lighting is of importance, in AR there is a mutual influence of the real and virtual lighting situation, although this is only rudimentarily or not at all considered by many applications. In VR, content can be scaled as desired. A user can, therefore, move between molecules or microbes as well as holding the entire Milky Way in their hands. With AR, in contrast, the real environment always provides a frame of reference, so that virtual objects usually have to be on a scale of 1:1. Of course one could also superimpose the Milky Way

in AR in such a way that the user is holding it in their hands. However, the perception would be fundamentally different. While in VR the users get the illusion that they have shrunk to the size of a microbe or grown to the size of a galaxy, in AR the users have the impression of holding a model of the Milky Way, since their own size remains unchanged in relation to the real environment.

Sometimes you may hear the question: Which one is better: VR or AR? This question cannot be answered because VR and AR are aimed at different application scenarios. There will rarely be a situation where you have a choice between VR and AR when it comes to implementing them. Rather, the application scenario usually determines the type of system to be used. This, however, does not mean that VR and AR cannot complement each other – in fact, quite the opposite! Thus, for example, in a purely virtual environment (VR), the details of a complex machine can be explained to trainees, problem and danger scenarios can be simulated and options can be tested that do not exist in reality (at least not on site). By using AR, the acquired knowledge can then be tested and further consolidated on the real machine with virtual support. For instance, it is possible to look into a component using virtual X-ray vision, etc. Basically VR, in contrast to AR, has no limitations: neither in content nor in physics (in a VR you can define your own physics!). On the other hand, the continuous use of VR is – at least currently – limited to rather short periods of time (minutes rather than hours). Since you always have to leave the real world for VR, this will not change fundamentally (unless we will live in the matrix one day). AR, on the other hand, has the potential to be used always and everywhere (24/7), although this potential currently cannot be fully exploited due to limitations in software and hardware.

1.4 Historical Development of VR and AR

The history of VR as a field of science and technology began in the 1960s. As part of his research on immersive technologies, Ivan Sutherland (1965) wrote “The Ultimate Display”, in which he described the vision of a room “within which the computer can control the existence of matter”. In his pioneering work, Sutherland took the first step towards connecting the computer with the design, construction, navigation, and experience of virtual worlds, even before the personal computer (PC) was invented (1970). In 1968, Sutherland created a *Head-Mounted Display System* consisting of a data helmet and a mechanical and alternatively ultrasound-based tracking system (see Fig. 1.7a). This system (Sutherland 1968) is often erroneously called the “Sword of Damocles” in the literature, although this was only the name of the mechanical tracking component of it. It enabled the viewer to view a simulated, albeit simple, 3D environment in the correct perspective. This system can also be regarded as the first AR system due to its see-through property.

The VIEW project (Virtual Environment Interface Workstations) of the NASA Ames Research Center in the mid-1980s had the goal of developing a multi-sensory workstation for the simulation of virtual space stations.



Fig. 1.7 Pioneering work in the field of VR/AR. (Left) Sutherland’s data glasses with 6-DOF ultrasound tracking; image courtesy of © Ivan Sutherland, all rights reserved. (Right) Replica of the MARS system of 1997 (Bell et al. 2002). (Image courtesy of © Steve Feiner, all rights reserved)

Around 1987 Thomas Zimmermann described the “DataGlove”, a glove that was equipped with glass fibers on the top of the hand to capture finger flexion. He and Jaron Lanier jointly founded the company VPL. Lanier is often credited as the first scientist to use the term “virtual reality”. Besides selling the “DataGlove”, VPL also developed the “EyePhone” data helmet, a continuation of Sutherland’s Head-Mounted Display from the 1960s. The LX version of the EyePhone offered a resolution of 442×238 pixels, while the HRX version offered 720×480 pixels.

Another milestone was the commercialization of electromagnetic trackers by Polhemus 3Space in 1989. This made it possible to control or determine a target at a certain distance from a computer.

Around the same time, the “BOOM” (Binocular Omni-Orientation Monitor) was developed by Fake Spaces Labs, a 3D visualization device with two monochrome cathode ray tubes, which received NTSC signals generated by a Silicon Graphics Workstation VGX380 (8 RISC processors, 33 MHz per processor, 1280×1024 pixels at the graphics output). This workstation was able to generate 800,000 small, transformed and shaded triangles per second that were also clipped at the border of the drawing area. One of the first applications to take advantage of this feature was the “Virtual Wind Tunnel” in the aerospace field by Steve Bryson in 1991.

Around 1988, various high-quality workstations for graphics were introduced to the market. These included Ardent, Stellar, Silicon Graphics (SGI) and HP, of which the SGI Reality Engine from Silicon Graphics prevailed on the worldwide market for high-end graphics systems around 1995. Commercial VR software systems were also introduced to the market. These were “RB2 – Reality built for two” by VPL, “dVS” by the English company Division and “WorldToolKit” by Sense8 (1990–1995).

The term “Augmented Reality” was coined in the early 1990s by a pioneering project at Boeing, which used information superimposed on the visual field to make it easier for workers to lay aircraft cables (Caudell and Mizell 1992).

In 1993, a student of the Massachusetts Institute of Technology (MIT) founded SensAble Technologies Inc., a company that developed and commercially distributed haptic devices. The “PHANTom” facilitated the experience of force feedback – a great innovation at that time.

At the beginning of the 1990s, groundbreaking research was undertaken in the field of Virtual Reality. For the first time, these made projection-based representations possible. The main representatives of these are the “Powerwall”, which consisted of a stereo screen, the “CAVE” (Cave Automatic Virtual Environment), which had four screens (developed at the University of Illinois in 1992), the “Responsive Workbench”, which arranged a screen horizontally analogous to a table surface (developed by GMD in 1993), and “iCONE”, which used semicircular screens.

With “MARS” (see Fig. 1.7b), the first mobile AR system was presented at Columbia University in 1997 (Feiner et al. 1997). The publication of ARToolkit in 1998 (Kato and Billinghurst 1999) made computer vision-based tracking for AR available and triggered a huge wave of research around the world.

After the development of electromagnetic tracking systems, ultrasonic tracking systems came on the market, which in turn were replaced by optical tracking systems based on infrared light around the year 2000. PC clusters also replaced the SGI Reality Engine II, reducing the price for the user to about one fifth. This made more extensive research possible. Founded in 1993, the company Nvidia released their GeForce graphics chips as a successor to the RIVA chip family in 1999. Introducing advanced features to consumer-level 3D hardware, the GeForce is a milestone in graphics hardware.

On the software side, Silicon Graphics developed a toolkit named OpenInventor (originally IRIS Inventor) to support application development that also benefitted VR applications in 1988. It was based on the ANSI standard PHIGS that introduced the concept of a scene graph. The Open Graphics Library (OpenGL) debuted in 1992. With the success of the World Wide Web, VRML, a dedicated markup language for VR, was developed and became an ISO standard in 1997. It would later evolve into X3D. This was also the time when dedicated VR software companies emerged and basic application areas were explored. For example, Henry Fuchs investigated telepresence applications as well as medical applications with VR/AR (Fuchs et al. 1998).

There is a regular exchange of information on the subject of VR throughout the world. In the USA there have been VRAIS Symposia since 1991 and in Europe EuroGraphics VE Workshops since 1993. In Japan the ICAT workshops have also taken place since the beginning of the 1990s. In 1999 the IEEE VR Conference was established as the successor to the VRAIS, which attracts about 500 participants from all over the world every year. Similarly, dedicated conferences on the topic of AR were introduced, e.g., ISMAR, the IEEE International Symposium on Mixed and Augmented Reality, which started in 2002 as a merger of the International Symposium on Augmented Reality (ISAR) and the International Symposium on

Mixed Reality (ISMR). Moreover, VR and AR have been featured in trade shows such as the consumer electronics show (CES).

For several decades, access to VR and AR technology was limited to research institutions, large industrial companies and government agencies, not least because of the sometimes astronomical prices for the necessary hardware. This changed abruptly with the introduction of the first high-end low-cost data glasses, Oculus Rift, in 2013. Since the delivery of the consumer version in 2016 and the market entry of numerous comparable displays (HTC Vive, Playstation VR, Microsoft's "Mixed Reality" displays, etc.) VR has experienced a boom. Approaches to AR glasses have not yet been able to achieve this success. For example, Google Glass has not yet prevailed in the market and Microsoft's HoloLens is considered a technical masterpiece but has not achieved widespread use quickly. A new phase in the evolution of AR applications started in 2017 with the introduction of several major software platforms for mobile AR on smartphones and tablet computers. Apple presented ARKit and Google presented ARCore, two modern frameworks that have started to strongly influence the commercial development of AR applications.

1.5 VR Systems

If we summarize the previous requirements for a VR system, we get the following situation: We need a computer system that recognizes the actions of users, simulates the world under this influence, and lets users perceive a virtual world via appropriate stimuli. Technically, three parts can be distinguished: input devices, output devices, and the world simulation. As simply as the tasks of a VR system can be broken down into these three parts, each subsystem can become rather complex: Which sensors can detect a user's actions? What coverage and resolution do these sensors have in terms of space and time? What range of actions do these sensors allow? Do the sensors restrict or limit the user? How can sensor data be passed on to the simulation of the world? How can knowledge about the world be made available to the simulation? How can stimuli be generated in a suitable way for all perception channels of the user? What is the quality of these stimuli? In what radius of action can the user sensibly perceive these stimuli? How can it be ensured that the response time of the overall system keeps pace with the actions of the user?

To demonstrate the importance of the individual subsystems of a VR system, let us revisit a prior experiment and examine it in more depth. In that experiment, we had placed a user in VR on the edge of a virtual abyss to observe the user's reactions to images of the user's surroundings. The user's position and viewing direction must be tracked by the input devices all the time to be able to calculate the correct perspective for the user in the virtual environment. In the first variant of the experiment, it was assumed that a sensor was built into the user's helmet to provide this position and orientation data. What does such a sensor look like? Is only the orientation of the head recognized or also the direction of the eyes? What distances of movement does such a sensor allow? In addition to tracking the head's orientation, is it possible to also track the position of the head so that bending forward is possible in the

virtual environment? Can you approach the virtual abyss by taking a step or two? Is it possible to walk on the entire roof of the virtual skyscraper? In addition, is it possible to track the whole body with all limbs to visualize the user's body as an avatar to support self-perception? Would this body tracking recognize only roughly the limbs or also individual finger movements, e.g., is it possible to press the elevator button with one finger to leave the roof of the virtual skyscraper by elevator?

In the early days of Virtual Reality, it was common to attach many of the sensors required here, and the input devices were mostly connected by cable (called *wired clothing*). Examples of this are helmets with monitors or data gloves to recognize finger movements. Electromagnetic and ultrasound-based devices have also been developed over time. Such systems usually consist of transmitter(s) and receiver(s), so that users had always something attached to their bodies. These days the trend is towards optical processes based on one or more cameras, whereby a distinction is to be made regarding the use of so-called markers or markerless systems. Markers are patterns known to the VR system that can be detected automatically with high reliability. Markers can be used to enable or stabilize the camera-based detection, as they are designed in a way that they are easy to detect in camera images and less prone to detection errors due to factors such as occlusions or unfavorable lighting situations. In addition to RGB cameras, markerless systems often use so-called depth cameras, which support the extraction of foreground objects and background. By using multiple cameras, accuracy can be improved and situations can be avoided, where tracking fails in single-camera setups due to occlusion.

Multiple, possibly redundant, input devices are often used at the same time to ensure the best possible recognition of user actions. An example of this is the combination of precise position tracking within a large action space in combination with hand/finger tracking and voice input. Here, the sensor data must be aggregated in a suitable form (sensor fusion) in such a way that overall plausible and non-contradictory data are provided reliably by combining sensor data of different types, even if single sensors fail due to occlusions.

When designing or configuring a VR system, one should always focus on the actual task and analyze which input devices are necessary. It is not always advantageous to include as many sensors as possible in a setup if this results in restrictions for the user. In our abyss experiment, it could be possible to measure the pressure distribution of the sole to infer whether the user is leaning forward or backward. This could be done using pressure-sensitive mats, which would require that the user may only stand on the mat, and thus the user's location would be fixed. This would be counterproductive in relation to other objectives, e.g., that the user should be able to move freely.

The output devices are the counterpart to the input devices. They serve to present the virtual world to the user in multiple modalities. This conversion of the virtual world model to sensory stimuli for the user is called rendering. According to the different sensory modalities through which humans perceive the real world, it is helpful to address as many of them as possible in Virtual Reality. Regarding our experiment, the visual output is of course highly important. Should the user be able to look around freely, as would be possible with a tracked helmet? Is it enough for the user to look down only, as in the second variant of the experiment, in which the image is projected onto the floor? Is it important for this use case that the user can turn and look around?

Which action space should be provided where the user can perceive the virtual environment? In which visual quality should the virtual world be presented? Is it important to recognize moving cars or pedestrians from the skyscraper? In addition to visual stimuli, other sensory modalities of the user can also be addressed. Should the noise of road traffic be perceived louder when you get closer to the edge of the building of the skyscraper? Should the user be able to perceive wind, and should it also change at the edge of the building? As already discussed, the time requirements for the stimulus calculation and rendering for the individual sensory modalities also differ. For the visual system, many new images must be calculated every second. In contrast, it is enough to determine the strength of the wind from the example once or twice per second. It is advisable to analyze exactly what is important for the actual application, instead of implementing everything that is technologically possible.

The task of world simulation is performed by a computer system that relies on an appropriate world model. This model determines the behavior. Depending on the application, physically based simulation models (e.g., for simulating flow behavior) or models based on artificial intelligence (AI) may be appropriate. The world simulation responds to data from the input devices. In addition to the question of the granularity in which the world is or can be modeled, which was dealt with in Sect. 1.1.2, there are questions relating to technical issues: Which delays occur from recognition by an input device to rendering in all output devices? To reduce this response time (which is called end-to-end latency), it may be helpful or even necessary to use pre-calculated simulation data instead of calculating everything in real time. For our experiment, the movements of road traffic can be calculated as well as the flow simulation for the winds between the skyscrapers. It may even be necessary

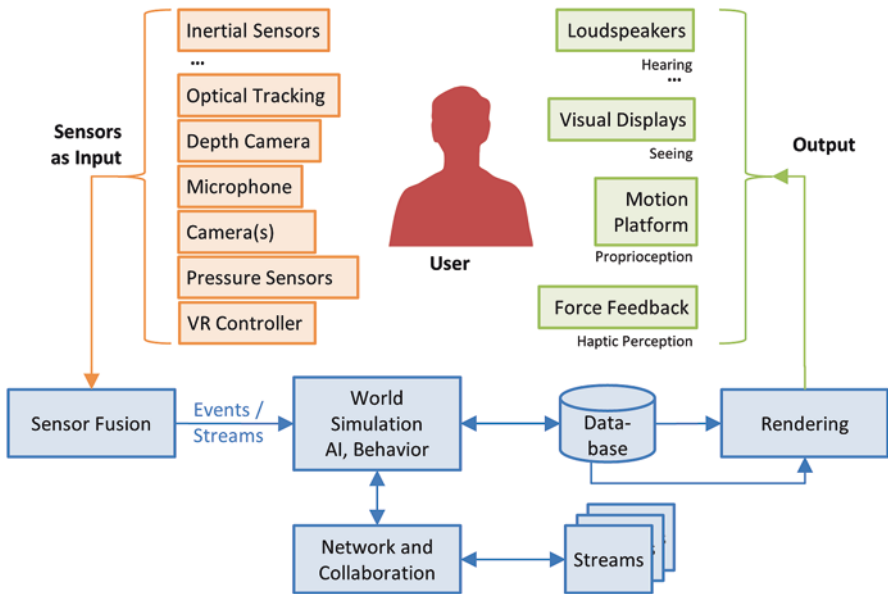


Fig. 1.8 Overview of the subsystems of a VR system

to make major simplifications to keep the delays to an acceptable level. It may also be necessary to distribute the calculation of the world simulation or the rendering to several computers. Does the world simulation rely on locally available data only or does it depend on remote data (e.g., current flight data for a simulator for air traffic controllers or data from VR systems that enable collaboration in virtual space)? Such data can be made available to the world simulation via network connections.

An overview of a VR system is shown in Fig. 1.8, with sensors, which can serve as input devices (in orange), output devices that address the various sensory modalities (in green), and all remaining subsystems of the VR system (in blue).

1.6 AR Systems

We define the term *AR system* by analogy with the already introduced term *VR system*.

We call an *AR system* a computer system that consists of suitable hardware and software to enrich the perception of the real world with virtual content as seamlessly and indistinguishably as possible for the user.

Even though an AR system typically looks different, its basic composition comprising subsystems is very similar to that of a VR system. Consider the requirements for an AR system: Again, we first need a computer system that performs a simulation depending on user activities. However, this simulation only affects certain parts of the world. One might be inclined here to limit the simulation of an AR system to the virtual part of the world perceived by the user. However, this is by no means sufficient for AR. Since the real and virtual contents are closely intertwined, i.e., there is an interdependency between the two, the parts of the real world that are influenced by the virtual content or, respectively, influence the virtual content, must also be simulated. In AR, the stimulus is generated in such a way that the real and virtual contents complement each other. Many aspects relating to sensors and stimuli apply in a similar way to AR systems. However, in contrast to VR systems, AR systems are usually not restricted to a specific location. This means that factors such as the operating range are omitted, but questions regarding the usability in certain environments have to be added. Can I use my AR System indoors or only outdoors? Will it still work in the subway? What if I am in a room with smooth white walls? Will the display work in sunlight? So, does an AR system have higher or lower technical requirements than a VR system? There is no general answer to these questions, but in a non-stationary system the amount of hardware is naturally limited. Thus, AR systems use on average fewer devices (sensors, output devices, computers, etc.) than VR systems. Nevertheless, the baseline requirements are rather high. While in the above example in the VR system we had a variety of configurations with more or

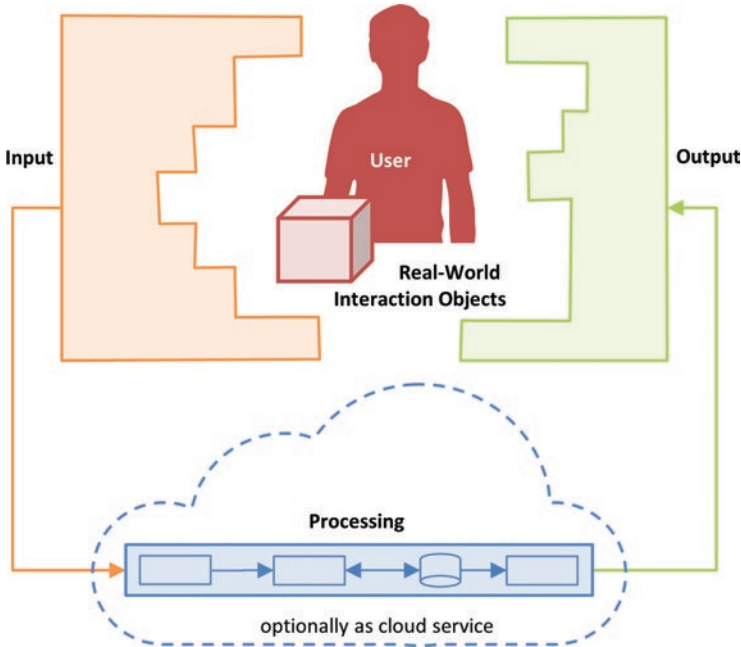


Fig. 1.9 Overview of the subsystems of an AR system. (See also Fig. 1.8)

less sophisticated sensor technology, an AR system must always guarantee the correct superposition of the real and virtual worlds with the proper perspective. On the other hand, many components of VR systems are not required. Through the awareness of reality, self-perception is always guaranteed. Also, navigation in the virtual world is not necessary, because users change their point of view by moving in their natural environment, the real world. While in VR systems the sensors, world simulation and stimulus generation are often distributed over a number of computer systems to ensure the required performance of the overall system, most AR systems are confined to a single computer system. This can be a mobile device such as a smartphone or tablet or it is sometimes completely integrated into AR data glasses (such as the Microsoft HoloLens). However, there are also approaches where optical tracking or rendering are outsourced to external systems to improve quality.

The overall view of an AR system is shown in Fig. 1.9: By analogy with Fig. 1.8, the sensors for input are shown in orange, output devices in green and the other subsystems of the AR system in blue.

1.7 Using the Book

In the following, you will find information on how the book is structured and suggestions on how the book can be used by different target groups for different purposes. Recommendations for use in academic courses are also given.

1.7.1 Structure of the Book

Following this introduction, the next chapter (Chap. 2) describes the basics of spatial perception. Starting from the human visual system, the theory of “depth cues” is presented, which describes the basic theory of spatial perception. The physiological aspects of stereoscopy are considered as well as supporting recommendations to enhance spatial perception. In addition to visual perception, the importance of other perceptual channels is discussed. The chapter on virtual worlds (Chap. 3) describes typical concepts employed to build them. Starting from data structures like the scene graph, advanced modeling concepts for virtual worlds are presented: Examples are animation methods, behavior descriptions and event models. In the chapters about VR input devices (Chap. 4) and VR output devices (Chap. 5), the characteristics of sensors and displays are described. After the introduction of underlying properties, methods for the tracking of user actions are shown as well as realization alternatives addressing the different sensory modalities of the user. Based on individual technologies, typical setups with VR hardware are also presented. Concepts and techniques for interactions in virtual worlds are presented in Chap. 6. Basic techniques for navigation and selection are described as well as the iterative approach to creating user interfaces based on user testing. Chapter 7 describes the requirements for the real-time capability of VR systems and presents solutions to meet these requirements. Based on fundamentals such as the importance of latency and efficient representations of large scenes, procedures for typical problems like synchronization and collision detection are discussed. Chapter 8 is dedicated to the topic of Augmented Reality. In addition to special input/output devices, the focus is on geometric and photometric registration as well as on the question of how authenticity or believability can be increased. Chapter 9 contains a series of small case studies that provide insights into the practice of VR/AR and illuminate the many facets of the topic. Software and tools for the practice of VR/AR development are the subject of Chap. 10, while Chap. 11 contains an introduction to the basic mathematics relevant to VR and AR.

1.7.2 Usage Instructions

Each further chapter of this book presumes having read in this chapter. For example, to work through Chap. 6, it is not necessary to read Chaps. 2, 3, 4, and 5 but only the first chapter. This means that the book can be used modularly and selectively – it does not have to be worked through from front to back. All the necessary basic knowledge has already been addressed in this chapter. Although the individual chapters of the present book differ considerably in the complexity of the material dealt with and thus in their scope, all chapters are structured according to a similar basic pattern. This enables the readers to find their way around the individual chapters quickly and to work through them similarly.

Chapters always start with an abstract that summarizes the most important content in a concise form. This enables readers who already have prior knowledge in

individual areas or are only interested in certain topics, i.e., who do not want to work through the book sequentially, to quickly identify and select the chapters relevant to them. The most important topics are then dealt with in the respective sub-chapters. The individual chapters are concluded with a list of questions on the topics covered and a list of recommendations for more in-depth or supplementary literature.

1.7.3 Target Groups

This book is primarily an academic textbook, i.e., it is intended to offer teachers and students a comprehensive and structured treatment of the topic of VR/AR. Therefore, fundamental aspects of VR and AR are covered. Prior knowledge in this field is therefore not necessary, but mathematical basics and basic knowledge of computer graphics are useful. Chapter 11 contains a summary of the most important mathematical elements of VR. A comprehensive and in-depth treatment of all topics relevant to VR/AR would go far beyond the scope of a single book – this book can serve here as an introduction and preparation for the study of specialist literature.

The book has a modular structure – each chapter only requires the reading of in this chapter. This allows students and teachers to adapt the order in which they work through the subject matter to the requirements of their course. It is also possible to select individual chapters and to omit other chapters (except in this chapter) without any problems, as it is not a prerequisite for understanding that all previous chapters have been read.

The creation of interactive virtual worlds is also one of the foundations of modern 3D computer games. Although the present book deals with these topics and there is considerable overlap with the realization of computer games, the book is not primarily aimed at developers of computer games, as game-specific aspects are not considered.

Lecturers in the Field of VR/AR

The book can be used directly as a basis for lectures and seminars in the field of VR/AR. Due to the modular structure of the book it is easy to vary the order of the different topics and thus to adapt to the individual requirements of the respective teaching unit. The individual chapters conclude with a collection of comprehension and transfer questions. These can be used directly as a basis for corresponding examinations or the preparation for them.

In the following, some typical combinations for individual courses are shown as examples. However, this can and should only serve to illustrate and in no way replaces the individual selection based on the respective curriculum and scope.

Introduction to VR/AR

Chapter 1

Sections 2.1, 2.2, 2.3, 2.4

Sections 3.1–3.3, optional 3.5

Sections 4.1, 4.2, 4.3, 4.6

Sections 5.1, 5.2, 5.3, 5.4

Sections 6.1, 6.2, 6.3, 6.4, 6.5

Sections 7.1, 7.2, 7.3

Section 8.1, 8.3, 8.4

3D User Interfaces

Chapter 1

Sections 2.1, 2.2, 2.3, 2.4, 2.5.2

Sections 4.1, 4.2, 4.3, 4.6

Chapter 6: all subchapters

Section 7.1

Section 8.5

Applications of Virtual Reality

Chapter 1

Sections 2.4, 2.5

Chapter 3: all subchapters

Sections 5.1, 5.2, 5.3

Chapter 6: all subchapters

Section 7.2

Section 8.6

Chapter 9 (VR examples)

Section 10.1, 10.2/10.3

Graphically Interactive Systems

Chapter 1

Chapter 2: all subchapters

Chapter 4: all subchapters

Chapter 5: 5.1

Chapter 6: all subchapters

Chapter 9: all subchapters

Chapter 10: all subchapters

Augmented Reality

Chapter 1

Chapter 3

Sections 4.1–4.4

Sections 5.1, 5.2, 5.3

Chapter 6

Chapter 8

Chapter 9 (AR examples)

Chapter 10

Students

The book offers students a universal companion and reference reading for courses on VR, AR and XR. In addition, it enables the self-study of the subject matter. The book is suitable for students of courses of study who might want to develop or extend VR/AR systems themselves, implement VR/AR applications or just use VR/AR applications. While the first aspect particularly appeals to students of Computer Science, Media Computing, Computational Imaging and Media Technology, the other aspects cover a wide range of natural and engineering sciences, humanities and social sciences, as well as creative and artistic fields.

Users and Those Who Want to Become Users

Potential users of new technologies such as VR and AR often have only a vague idea of the potentials and limitations as well as the resources required for their use. This leads to the fact that such technologies are often not used at all or are used too late. Or even worse, many introductions fail in the end. One of the main problems is that often extensive investments are made in hardware before it is clear whether and how it will be used afterward. Who are the users? Who benefits? How are the users trained? How is the infrastructure maintained and developed? Which applications should be created or used? How are they integrated into a production process or adapted to it? This book should help potential users of VR and AR to better assess these issues in advance and thus prevent or at least reduce misplanning. For users from both research and industry, the book enables them to deal with the topic in detail and thus to assess whether and to what extent the use of VR and AR appears to be sensible and what resources are required for this.

The Technology-Savvy

Ultimately, the book reflects the current status quo in the field of VR/AR and thus gives the technologically interested reader an insight into this fascinating world. New techniques and technologies that are currently still primarily used in research or research-related prototype and application development are presented, as well as those that are already an integral part of the production chain today, for example in the automotive industry.

1.8 Summary and Questions

There is no single generally accepted definition of VR today. One can approach the term from a technology-centered perspective and understand it to mean computer systems that build immersive and interactive environments using

appropriate hardware, such as stereo displays. But VR can also be described as a methodology to give users the experience of inclusion in an alternative reality. The goal is not necessarily to achieve a perfect Virtual Reality that can no longer be distinguished from reality. Peculiarities of human perception and cognition such as the suspension of disbelief can be exploited to successfully create virtual environments for people and give them the feeling of presence in a VR. This can serve different purposes: research (e.g., about human perception), education, entertainment, communication support, visualization of simulation results or economic goals (e.g., prototyping to increase efficiency or save costs). The basic purpose of VR is to create an innovative interface between humans and computers. The idea of leaving users present in reality, but extending it with parts from a virtual world, leads to Augmented Reality. For the realization of virtual or augmented environments a virtual world and a VR/AR system are required. The virtual world provides the content to be shown in the environment (e.g., description of the geometry, appearance, and behavior of the virtual objects occurring in it). With regard to the VR/AR system, a computer system needs to be implemented that comprises the essential components for the collection of information about the users and their interactions (e.g., by tracking), the generation of stimuli for the user (e.g., images and sounds) as well as the simulation of the virtual world. Despite its more than 50 years of existence, VR/AR is still a young science. Four generations can be distinguished in its development, which can be characterized by the hardware used: (1) HMD and data glove, stereo projection and optical tracking, (2) high-resolution displays and low-cost tracking without the use of artificial markers, (3) consumer HMD including tracking and controllers, and (4) AR on smartphones and tablets.

Check your understanding of the chapter by answering the following questions:

- What would your definitions of the terms “virtual reality”, “virtual world”, “virtual environment”, “augmented reality”, “mixed reality”, “immersion”, “presence”, “simulation”, “tracking”, “user”, “human-machine interaction” and “suspension of disbelief” be?
- The text describes a scenario in which a user stands on a glass plate that is used as a projection screen. This gave the user the impression of standing on a virtual high-rise building where the user could see their real feet. Is this scenario an example of VR or AR?
- Suppose you want to create a jogging app where you run against other runners (or even yourself the day before). Would you implement this with VR or AR? What might this depend on? What would your environment look like? Which hardware would you use for this?
- What can VR and AR be used for? Which application examples do you know, or can you imagine? Why are you interested in VR/AR?

Recommended Reading

Angel E, Shreiner D (2015) *Interactive computer graphics: a top-down approach with WebGL*. Pearson Education, Harlow – *Textbook covering the basics of computer graphics, e.g., discussing the generation of images with the computer. It also introduces OpenGL and WebGL, a programming library for computer graphics, and discusses the possibilities of using graphics processors (GPUs) in the form of so-called shaders.*

Rabin S (2009) *Introduction to game development*, 2nd edition. Charles River Media, Boston – *a standard work on computer games. Due to the manifold points of contact between VR and computer games, literature from the field of computer games is also relevant.*

Original scientific literature can be found in specialist journals and conference proceedings which can be researched and accessed in digital libraries (e.g., dl.acm.org, ieeexplore.org, link.springer.com) or via search engines (e.g. scholar.google.com). In the field of VR the IEEE VR Conference (ieeevr.org) takes place annually. Moreover, there is the Eurographics Symposium on Virtual Environments (EGVE) as well as the VR Conferences of euroVR, which are partly jointly organized as Joint Virtual Reality Conference (JVRC). With the focus on AR, ISMAR, the IEEE Symposium for Mixed and Augmented Reality, is held annually. In addition, there are special events that focus on aspects of user interfaces of VR and AR, such as the ACM VRST conference or the 3DUI, the IEEE Symposium for 3D User Interfaces. There are also further events dealing with special applications of VR, for instance in the industrial sector (e.g., VRCAI – ACM International Conference on Virtual Reality Continuum and Its Applications in Industry). Some scientific journals also focus on VR and AR, e.g., *Presence – Teleoperators and Virtual Environments* by MIT Press, *Virtual Reality* by Springer Verlag or the *Journal of Virtual Reality and Broadcasting (jVRb)* as an open access e-journal.

In addition to conference proceedings and professional journals that deal primarily with VR and AR, literature is also recommended that deals with essential aspects of VR and AR, such as *Computer Graphics* (e.g., ACM SIGGRAPH and the ACM Transactions on Graphics), *Computer Vision* (e.g., IEEE ICCV) or *Human–Machine Interaction* (e.g. ACM SIGCHI).

References

- Azuma R (1997) A survey of augmented reality. *Presence Teleop Virt* 6(4):355–385
- Bell B, Feiner S, Hoellerer T (2002) Information at a glance. *IEEE Comp Gr Appl* 22(4)., July/August, 6–9
- Bricken W (1990) Virtual reality: directions of growth. Notes SIGGRAPH '90 Panel (HITL Technical Report R-90-1), University of Washington, Seattle
- Bryson S (2013). Virtual Reality: a definition history – a personal essay. *ArXiv*, abs/1312.4322

- Caudell TP, Mizell DW (1992) Augmented reality: an application of heads-up display technology to manual manufacturing processes. In: Proceedings of 25th Hawaii International conference on system sciences, Vol. 2, 659–669
- Feiner S, MacIntyre B, Höllerer T (1997) A touring machine: prototyping 3D mobile augmented reality systems for exploring the urban environment, digest of papers. In: First International Symposium on Wearable Computers, pp 74–81
- Fuchs H, Livingston MA, Raskar R, Colucci D, Keller K, State A, Crawford JR, Rademacher P, Drake SH, Meyer AA (1998) Augmented reality visualization for laparoscopic surgery. In: Wells WM, Colchester A, Delp S (eds) Medical image computing and computer-assisted intervention — MICCAI'98, LNCS, vol 1496. Springer, Berlin/Heidelberg
- Held RH, Durlach NI (1992) Telepresence. *Presence Teleop Virt* 1(1):109–112
- Kato H, Billinghurst M (1999) Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In: 2nd IEEE and ACM international workshop on augmented reality (IWAR), pp. 85–94, IEEE
- Mann S (1999) Mediated Reality. *Linux Journal*, Article No 5, Issue 59
- Mann S, Furness T, Yuan Y, Iorio J, Wang Z (2018) All Reality: Virtual, Augmented, Mixed (X), Mediated (X,Y), and Multimediased Reality. <https://arxiv.org/abs/1804.08386>
- Milgram P, Takemura H, Utsumi A, Kishino F (1995) Augmented reality: a class of displays on the reality-virtuality continuum. *Proc SPIE* 2351:282–292
- Mine MR, Brooks Jr FP, Sequin CH (1997) Moving objects in space: Exploiting proprioception in virtual-environment interaction. In: Proceedings of SIGGRAPH 1997, pp 19–26
- Paradiso JA, Landay JA (2009) Guest editors' introduction: cross-reality environments. *IEEE Perv Computing* 8(3):14–15
- Rekimoto J, Nagao K (1995) The world through the computer: computer augmented interaction with real world environments. In: Proceedings of UIST '95, pp 29–36
- Rheingold H (1991) *Virtual reality*. Summit Books, New York
- Sadowski W, Stanney KM (2002) Presence in virtual environments. In: Stanney KM (ed) *Handbook of virtual environments: design, implementation, and applications*. Lawrence Erlbaum Associates Inc, Mahwah
- Sheridan TB (1992) Musings on telepresence and virtual presence. *Presence Teleop Virt* 1(1):120–125
- Sherman W, Craig A (2003) *Understanding virtual reality*. Morgan Kaufmann, San Mateo
- Slater M (2003) A note on presence terminology. *Presence Connect* 3:3
- Slater M (2009) Place illusion and plausibility can lead to realistic behaviour in immersive virtual environments. *Phil Trans R Soc B* 364(1535):3549–3557
- Slater M, Wilbur S (1997) A framework for immersive virtual environments (FIVE): speculations on the role of presence in virtual environments. *Presence Teleop Virt* 6(6):603–616
- Slater M, Spanlang B, Corominas D (2010) Simulating virtual environments within virtual environments as the basis for a psychophysics of presence. *ACM Trans Graph* 29(4):92
- Stone RJ (1993) In: Earnshaw RA, Gigante MA, Jones H (eds) *Virtual reality systems*. London, Academic
- Sutherland IE (1965) The ultimate display. Proceedings of the IFIP congress, 506–508
- Sutherland I (1968) A head mounted three dimensional display. In: Proceedings of the AFIPS fall joint computer conference. Thompson Books, Washington, DC, pp 757–764
- Witmer BG, Singer MJ (1998) Measuring presence in virtual environments: a presence questionnaire. *Presence Teleop Virt Environ* 7(3):225–240

Chapter 2

Perceptual Aspects of VR



Ralf Doerner and Frank Steinicke

Abstract Virtual Reality (VR) has the special ability to provide the user with the illusion of presence in a virtual world. This is one aspect of the valuable potential that VR possesses concerning the design and realization of human–machine interfaces. Whether and how successfully this potential is exploited is not only a technical problem. It is also based on processes of human perception to interpret the sensory stimuli presented by the virtual environment. This chapter deals with basic knowledge from the field of human information processing for a better understanding of the associated perceptual issues. Of particular interest in VR are the perception of space and the perception of movement, which will be dealt with specifically. Based on these fundamentals, typical VR phenomena and problems are discussed, such as double vision and cybersickness. Knowledge of human perception processes can be used to explain these phenomena and to derive solution strategies. Finally, this chapter shows how different limitations of human perception can be utilized to improve the quality and user experience during a VR session.

2.1 Human Information Processing

The way that people perceive and process information is essential for the design of virtual environments and the interaction within them. Ultimately, every virtual environment is used by humans. For this reason, it is useful to study the basic functions of human information processing to better understand the various effects and phenomena of VR and to be able to take advantage of possible limitations.

Dedicated website for additional material: vr-ar-book.org

R. Doerner (✉)
Department of Design, Computer Science, Media, RheinMain University of Applied Sciences, Wiesbaden, Germany
e-mail: ralf.doerner@hs-rm.de

Humans perceive their environment through different senses. In the context of today's VR technologies, the most important senses are:

- the visual sense,
- the acoustic sense, and
- the haptic sense.

In most of today's VR systems, other senses, such as the olfactory (smelling) or gustatory (tasting) senses, are not stimulated. Thus, most information presented in the virtual environment is perceived through the eyes, ears, or skin. At first glance, perception in a virtual environment does not differ from perception in a typical desktop environment and the associated senses and sensory impressions. The virtual worlds presented on the screen or from the loudspeakers act as visual or acoustic stimuli; haptic impressions are conveyed via mouse and keyboard. An important aspect of the VR experience is the possibility to explore the virtual world in an immersive way. In contrast to desktop-based environments, in VR this is not only done by mouse and keyboard but by 3D input devices or by movements of the user in real space, which are mapped to corresponding movements in the virtual world. In addition to these inputs into the VR system, there are other forms of input, such as speech, gestures, and other forms of human expression (Preim and Dachsel 2015).

To better understand the complexities of human perception and cognition, it is helpful to imagine humans as an information processing system (see Fig. 2.1). In this metaphor from the field of computer science, all physical characteristics of humans are assigned to hardware and all psychological characteristics to software. The chain of information processing starts with an *input*, which is *processed* in the computer and finally presented as *output* on the output media. In human information processing, stimuli from the external world are thus first transferred to the perceptual system as input and perceived there (Card et al. 1986a). This *perceptual processor* has access to memory (e.g., visual memory) and processor (e.g., for pre-filtering) similar to the input to the computer. The processing of the resulting perceived stimuli then takes place in the *cognitive processor*. Here, further memories, i.e., the working and long-term memories, can be accessed to interpret the stimuli and plan appropriate action. The actual action then takes place in the *motor processor*, which initiates corresponding movements.

These partly substantial simplifications only approximate the much more complex biological processes, but they allow us to make predictions about human information processing. For example, Card et al. (1986a) were able to predict the time required for a whole series of human interaction tasks. This model makes it clear, among other things, why tasks that require the cognitive processor to be run through several times (e.g., comparisons) require more time than those tasks in which the cognitive processor is only run through once (e.g., simple response to stimulus).

In this context, a whole range of other models, such as *GOMS* or the *Keystroke-level Model* (KLM), can be mentioned, which are used in the field of human-computer interaction (Card et al. 1986b; Sharp et al. 2019; Shneiderman et al. 2018). In the following, we want to give a more detailed insight into the individual components of human information processing.

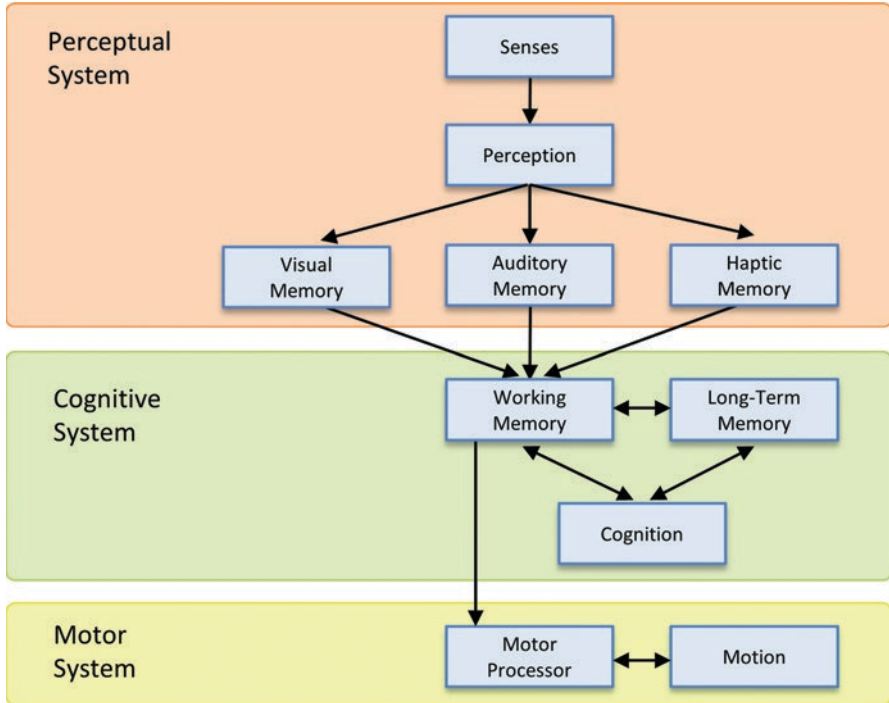


Fig. 2.1 Model of human information processing. (According to Card et al. 1986a)

2.2 Visual Perception

The visual system is the part of the nervous system responsible for processing visual information. The structure of the human eye allows light to be projected through the lens onto the inner *retina*. There are about 120 million photoreceptor cells. These are divided into the *rods*, which only perceive brightness, and the approximately 7 million *cones*, which are responsible for color vision. The cones, in turn, can be divided into three types, each of which reacts to blue, green or red hues. The optical apparatus of the eye produces an upside-down and reversed image on the retina. For the perceived image to arrive sharply on the retina, the lens must be correctly adjusted by muscles depending on the distance of the object being viewed. This process is called *accommodation*. The *fovea* is the retina area with the highest image sharpness and the highest density of photoreceptor cells. Although the eye has an aperture angle of approximately 150° (60° inside, 90° outside, 60° above, and 75° below), only 2° to 3° of the field of vision is projected onto the fovea. Under ideal conditions, the resolving power is about 0.5–1 min of angle. This means that a 1 mm spot can be perceived from a distance of about 3–6 m. The eye only remains at such a fixation point for a period of about 250 ms to 1 s before rapid, jerky eye movements (known as *saccades*) occur. These saccades serve to complement peripheral

perception, in which the resolution is only about one-fortieth of the foveal resolution, and thus enable us to perceive a complete high-resolution image.

In particular, visual perception enables us to identify objects. For this purpose, the projected image of the scene is already analyzed in the retina (e.g., brightness, contrast, color and motion) and processed (e.g., brightness compensation and contrast enhancement). During transmission via the optic nerve, the spatial relationships of the photoreceptors are retained in the nerve tracts' positional relationships and synapses. This positional relationship can be detected in the visual cortex as a neural map and supports, for example, the identification and differentiation of objects (Marr 1982). The recognition of individual elements and their meaning is probably done by comparison with already stored experiences (scenes linked to body sensation, emotions, smell, sounds, and much more).

2.2.1 Stereo Vision

As an example of how human perception works and how it can be manipulated by a VR system to create presence in the virtual environment, we consider a phenomenon important for VR: *stereopsis*, also called stereo vision. Humans have two eyes but do not perceive two separate images of reality. In addition, the visual system succeeds in obtaining a three-dimensional impression of the environment from the light stimuli impinging on the two-dimensional retina of the eyes.

Let us consider point A in Fig. 2.2a. If we assume the eyes have fixated on point A , then they have been adjusted so that light from point A enters both the fovea of the left eye (and impinges on the retina at point A_L) and the fovea of the right eye

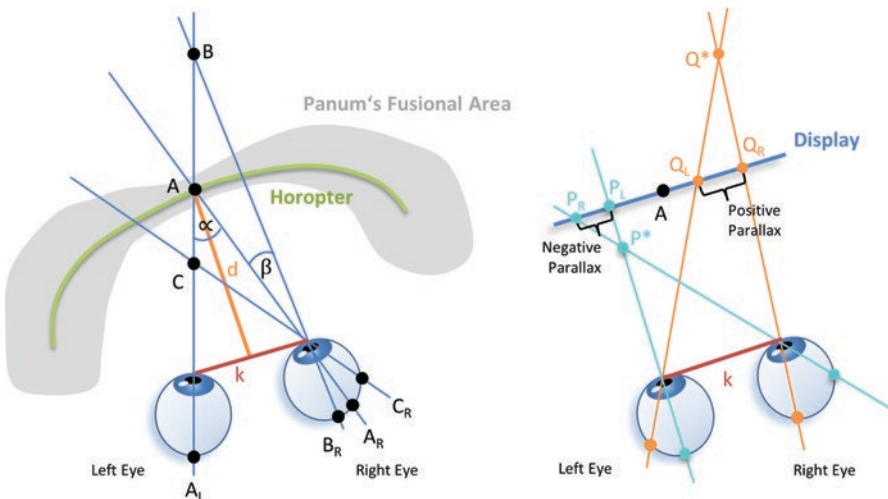


Fig. 2.2 (a) Stereopsis. (b) Manipulation of the stereopsis with a stereo display

(there at point A_R). Adjusting means that the eye muscles are moved accordingly. The closer the point A between both eyes is to the observer, the more the eyes must be turned inwards towards the nose to fixate on A . This movement of both eyes is called *convergence*. As the visual system has information on how big the convergence is, the angle α can be estimated in the triangle A , A_L and A_R , because the bigger the convergence, the bigger α is. With the knowledge of α and the distance k of both eyes, which is constant for a person, the distance d of point A from the observer can be concluded. By simple trigonometry, the following relationship between d and α can be established: $d = k / (2 \cdot \tan \alpha)$. With this triangulation of A , which is only possible with two eyes, the visual system can thus perceive the distance of A .

The points A_L and A_R are called *corresponding points* of the retina. They would be in the same place if the two eyes were thought to be superimposed. The visual system is able to determine this correspondence. All points in reality that are mapped onto corresponding points on the retina form the *horopter*. It has the shape of a surface curved around the head, which contains the fixation point. Let us now look at point B in Fig. 2.2, which is not on the horopter. In the left eye, light from B still strikes at point A_L , while in the right eye, it strikes at point B_R . The points A_L and B_R are not corresponding points. The difference between B_R and the point A_R corresponding to A_L is called the *disparity* created by B . Disparities are often given as angles; in our example in Fig. 2.2 this would be the angle β . The larger β is, the more the point B is away from the horopter. The disparity generated by B thus provides a point of reference for determining the distances of points like B , which, unlike A , are not fixated on and whose distance cannot be determined directly based on eye convergence alone.

Two Small Experiments on Convergence and Disparity

1. Hold a pen at a distance of about 1 m in front of a person's face. Ask the person to fixate on the tip of the pen and leave it fixed. Now move the pen towards the person's nose so that you can easily observe the convergence: the eyes are directed inwards towards the nose.
2. Sit in front of a rectangular object (e.g., a monitor), close your right eye and hold your index finger so that the left index finger points to the left edge of the object and the right index finger to the right edge. Now open the right eye and close the left one. The object seems to jump relative to the fingers – the right and left eyes perceive a slightly different image; there are disparities.

Retinal disparities also allow us to obtain information about the distance of points that are in front of the horopter from the observer. Point C in Fig. 2.2 is such a point, and while light from C in the left eye also arrives at point A_L , this happens in the right eye at point C_R . The disparity now exists between A_R (the point corresponding to A_L) and C_R . The point C_R lies to the right of A_R , while B_R lies to the left of A_R . B creates an *uncrossed disparity* and C a *crossed disparity*. Whether a point lies behind or in front of the horopter can be distinguished by the fact that in the first

case uncrossed disparities are generated and in the second case crossed disparities are generated.

If the disparity becomes too large, i.e., the point generating the disparity is too far away from the horopter, the visual system is no longer able to fuse the image information of both eyes into one image. As a result, one no longer sees one point but two points. All points in the world that create disparities small enough to allow a fusion of the image information from the left and right eye form *Panum's fusional area*. This area has the smallest extension around the point the eyes fixate on.

In a virtual environment, stereopsis can be manipulated with the aim of creating a three-dimensional impression, even though only a two-dimensional display surface is used. Figure 2.2b shows that a *display surface* is viewed by an observer. Viewing means that the observer fixates on a point A on the display surface with the eyes. We now illuminate two points P_L and P_R on the display surface. By taking the technical precautions described in detail in Chap. 4, we ensure that light from P_L only hits the left eye and light from P_R only the right eye. The distance between P_L and P_R on the display surface is called *parallax*. The visual system can react to this situation in two ways. First, two different points are perceived. In reality, it happens all the time that light from points in the world only enters one of the eyes. The visual system can also spatially arrange such points in relation to points from which light falls into both eyes and whose location could already be deduced (*DaVinci-stereopsis*). Secondly, the visual system explains the light stimuli at points P_L and P_R by the fact that the light comes from a single point P^* located in front of the display surface. P^* is the *fusion* of P_L and P_R . Which of the two cases actually occurs depends on a number of factors, such as how far the apparent point P^* is located from the display surface. If the visual system merges P_L and P_R , then a point outside the display surface is successfully displayed. It is also possible to create points behind the display surface by reversing the order of the points for the left and right eyes on the display surface. This is shown in Fig. 2.2 at point Q_L and Q_R , where the two points shown on the display could be fused to form a point Q^* behind the display. When P_L and P_R are displayed, this is called *negative parallax*, while in the case of Q_L and Q_R one speaks of *positive parallax*.

In VR, it is, therefore, possible to create a *stereo display* by exploiting the peculiarities of human perception. The visual system creates not only a two-dimensional but also a plastic three-dimensional image impression, in which objects appear in front of or behind the screen based on an appropriate selection of the parallax. This must be distinguished from true three-dimensional displays (*volumetric displays*), in which, for example, a display surface is moved in space.

2.2.2 Perception of Space

Not only disparities are used by the visual system to perceive spatiality and the arrangement of objects in space. This can be seen by the fact that there are people who are unable to evaluate information from disparities ('*stereo blindness*') but

nevertheless develop a three-dimensional idea of the world. There are no exact figures, but it is estimated that about 20% of the population is stereo blind. A test can be used to determine stereo blindness in the same way as a test for color vision defects. It is recommended to perform such a test, especially for people who are active in the field of VR. Many people are not aware that they are stereo blind.

Today we know a whole series of clues, called *depth cues*, which are used by the brain for the perception of space. Disparity is an example of a depth cue. If a car covers a tree, the visual system can derive the information that the car is closer to the observer than the tree. This information does not require the interaction of both eyes. Thus, this clue is called a *monocular depth cue*. As it is still possible to obtain depth cues even from 2D images, this is also referred to as a *pictorial depth cue*. Disparity, on the other hand, is a *binocular depth cue*. With depth cues, one can distinguish whether they help to estimate the spatial position of an object absolutely or only relative to another object. Convergence, for example, allows an absolute position determination, whereas occlusion only permits a determination relative to the occluded object.

The informative value and reliability of the various depth cues depend in particular on the observer's distance to the respective object. While occlusion provides reliable information in the entire visible range, this is not the case for disparity. The further away a point is from the observer, the lower the disparity it generates. A point at a distance of 2–3 m produces a very small disparity. From a distance of 10 m, the disparity is de facto no longer perceptible. For VR, this means that for virtual worlds where significant objects are within arm's reach, the effort to use stereo displays should be invested. Disparity is essential in this area. For virtual worlds, however, where objects are more than 3 m away from the viewer, the use of a stereo display does not contribute much to the perception of space and may be superfluous.

Table 2.1 lists various depth cues and gives details of the area of action and the information content (indications of relative arrangement or absolute distance determination), as well as the category (monocular depth cue, binocular depth cue or *dynamic depth cue*, the latter being understood as depth cues that the observer receives through movement). The depth clues mentioned in the list are all of a visual nature, but the brain can also obtain cues from other senses, e.g., by interpreting information from touch or by analyzing the pitch of a moving object's sound. As it is essential for a good perception of a virtual world to give as many depth clues as possible in VR, we go through the list below. *Occlusion*, *disparity* and *convergence* have already been discussed. Similar to convergence, where muscle tension is taken into account to align the eyes, the brain also uses the muscle tension necessary for *accommodation*, the adjustment of the refractive power of the eye lens, as a depth cue. To see nearby objects clearly, the eye lens must be pressed together with more muscle power than is the case with distant objects. If a person fixates on an object at a certain distance, other objects appear sharp only in the vicinity of this object (e.g., in the distance range 75 cm to 1.5 m if the fixed object is 1 m away from the observer). Objects that are too far away or too close to the observer appear blurred. From the *image blur*, it is, therefore, possible to draw conclusions about the distance

Table 2.1 List of depth cues (with range of action and classification)

Depth cue	Range of action	Classification	Positioning
Occlusion	Complete range	Monocular	Relative
Disparity	Up to 10 m	Binocular	Relative
Convergence	Up to 2 m	Binocular	Absolute
Accommodation	Up to 2 m	Monocular	Absolute
Image blur	Complete range	Monocular	Relative
Linear perspective	Complete range	Monocular	Absolute
Texture gradient	Complete range	Monocular	Relative
Relative size	Complete range	Monocular	Absolute
Known quantity	Complete range	Monocular	Absolute
Height in the field of view	Over 30 m	Monocular	Relative
Atmospheric perspective	Over 30 m	Monocular	Relative
Shape from shading	Complete range	Monocular	Relative
Shadows	Complete range	Monocular	Relative
Motion parallax	Over 20 m	Dynamic	Relative
Accretion	Complete range	Dynamic	Relative

of objects. *Linear perspective* is a depth indication based on perspective distortion. Objects further away appear smaller; in reality, parallel lines seem to converge at a vanishing point (see, for example, the street in Fig. 2.3a).

Also, with textures, the texture elements become smaller with increasing distance. Thus, the *texture gradient* can serve as a depth cue. For similar objects, such as the three squares in Fig. 2.3a, which have different sizes in the image, the visual system assumes that the differences in size can be explained by different distances (and not by the fact that the objects themselves are of different size: assumption of *size constancy*). This depth cue is called *relative size*. However, the *known size* also contributes to distance estimation. We get a good impression of the size and orientation of the triangle in Fig. 2.3a because a person is standing next to it – and thus an object of which we know the size and the usual orientation in space. Moreover, the *height in the field of view* is an indication of depth. In Fig. 2.3a, square C is arranged higher in the image than square A and thus closer to the horizon line. This indicates that square C is further away. Connected to this is also the direction of view. If one has to look straight ahead or raise the head, the object is assumed to be further away (Ooi et al. 2001). Very distant objects do not appear so rich in contrast and have a slightly bluish coloration (cf. Fig. 2.3b), because more air and the particles it contains lie between the observer and the object (*atmospheric perspective*). The illumination of objects gives clues about their arrangement in space. On the one hand, shaded objects appear more spatial (*shape from shading*, cf. left pyramid with shading, right pyramid without in Fig. 2.3c); on the other hand, the *shadows* cast give cues about the spatial arrangement of objects (cf. shadows of spheres in Fig. 2.3d). It is especially effective when shadows are cast from above on a base surface because the visual system is used to a light source from above: the Sun. If the object is in motion, the shadow of this object is particularly useful for depth perception.

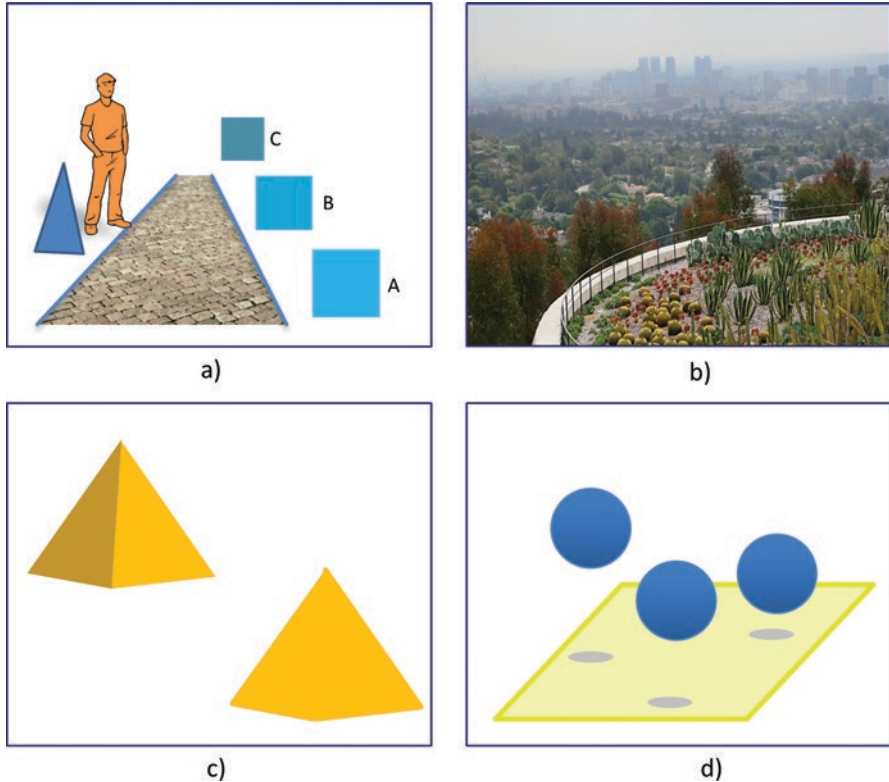


Fig. 2.3 Examples of depth cues

Finally, certain depth cues are based on movement: movement of objects or movement of the observers themselves. This includes *motion parallax*: the light stimuli from near objects move faster across the retina than those from farther away. If we drive through an avenue in a car, the nearby trees pass us quickly while the mountains in the background move only slowly. Through movement, objects suddenly become occluded or reappear behind the objects that are obscuring them. This change, called *accretion*, also gives cues to the spatial arrangement of the objects.

Depth cues are not to be considered independently of each other. For example, accommodation and convergence depend on each other (Howard 2002). Also, depth cues are of varying strength. For example, while accommodation is a weak depth cue, occlusion is a strong depth cue. All depth cues are considered for spatial perception in the form of a weighted sum. How much weight is given to a depth cue is flexible and depends on the distance of the object to be assessed. One theory (Wanger et al. 1992) assumes that the weights also depend on the current task the observer is engaged in. If the task is to estimate the spatial arrangement of distant objects, then motion parallax, linear perspective, texture gradient and shadows have a high weight. If the task is to grasp an object, disparity, convergence and

accommodation are important. According to this, the depth cues in the brain are not used to form a single model of the 3D world, which is then used for different tasks, but rather task-dependent models are formed. Therefore, if not all depth cues can be generated in a VR, then a prioritization should be made depending on the task the user has to perform.

2.3 Multisensory Perception

Even though the visual sense is undoubtedly the most important source of information in the perception of virtual worlds, the auditory and haptic senses also play an increasingly important role (Malaka et al. 2009). In this respect, these two senses will also be examined more closely in the context of this chapter. Other senses, such as smell and taste, play more of an exotic role and are currently mostly used as prototypes in research laboratories. At this point, it should be noted that perceptions via the individual sensory organs are by no means processed separately, but rather an integration of the different impressions is created. For further literature, please refer to Ernst (2008).

2.3.1 Auditory Perception

The ears enable humans to perceive air movements. Such air and pressure fluctuations generate mechanical waves that hit the ear, which is made up of the outer, middle and inner ear. The auricle (outer ear) collects sound waves and transmits them to the middle ear. In the middle ear, sound waves are converted into vibrations of the eardrum. The eardrum vibrations are transmitted to the cochlea via the ossicles (anvil, malleus and stapes). The sensory cells in the cochlea then convert the mechanical energy into electrical signals. Finally, these electrical nerve impulses are transmitted to the brain via the auditory nerve. The different frequencies can be perceived by hair cells in the inner ear. The waves perceived by humans have lengths of about 0.02–20 m, which correspond to audible frequencies in the range of about 18–0.016 kHz (Malaka et al. 2009). In contrast to the visual sense, the spatial resolution is much lower. The *Head-Related Transfer Function* (HRTF) or outer ear transfer function describes the complex filter effects of the head, outer ear, and trunk. The evaluation and comparison of the amplitudes are, along with the transit time differences between the ears, an essential basis of our acoustic positioning system. However, the absolute distinguishability of intensity and frequency has clear limits, so that two noise sources are only distinguished if they are several degrees apart. In contrast, the temporal resolution is much better and acoustic stimuli can be distinguished already at 2–3 ms temporal discrepancy. The principle of localizing noise sources at different receiver positions is also used in acoustic tracking systems (see Chap. 4).

2.3.2 *Haptic Perception*

Haptics, or haptic perception, describes the sensory and/or motor activity that enables the perception of object properties such as size, contours, surface texture and weight by integrating the sensory impressions felt in the skin, muscles, joints and tendons (Hayward et al. 2004). The senses that contribute to haptic perception are divided into:

- tactile perception (element of surface sensitivity),
- kinesthetic perception/proprioception (depth sensitivity) and
- temperature and pain perception.

These senses thus enable the perception of touch, warmth and pain. Such perception phenomena are based on receptors in the skin. The more such receptors are available, the more sensitive the respective body part (e.g., hand, lips or tongue) is. The most important receptors are the mechanoreceptors (e.g., pressure, touch or vibration), the thermoreceptors (heat, cold) and the nociceptors (e.g., pain or itching). The mechanoreceptors, for example, convert mechanical forces into nerve excitation, which are transmitted as electrical impulses to the sensory cortex, where they are processed. As a result, shapes (roundness, sharpness of edges), surfaces (smoothness and roughness), and different profiles (height differences) can be perceived.

Haptic output devices stimulate the corresponding receptors, for example, by vibration (see Chap. 5).

A small experiment on the spatial resolution of haptic perception: take a compass or two sharp pencils and test with somebody else or yourself where in your upper extremities you can best distinguish between two points of contact and where you can distinguish least.

2.3.3 *Proprioception and Kinaesthesia*

In contrast to surface sensitivity, depth perception describes the perception of stimuli coming from inside the body. Depth perception is essentially made possible by proprioception and kinaesthesia. Both terms are often used synonymously. However, we will use the term *proprioception* to describe all sensations related to body position – both at rest and in motion – whereas *kinaesthesia* describes only those sensations that occur when active muscle contractions are involved. Proprioception thus provides us with information about the position of the body in space and the position of the joints and head (sense of position) as well as information about the state of tension of muscles and tendons (sense of strength). Proprioception enables us to know at any time what position each part of our body is in and to make the

appropriate adjustments. Kinaesthesia (sense of movement) enables us to feel movement in general and to recognize the direction of movement in particular.

These two senses are essential, considering that interaction in a virtual environment is largely carried out by active movements of the limbs. In VR, various devices are available to stimulate these senses, such as haptic joysticks, complete exoskeletons or motion platforms (see Chaps. 4 and 5).

2.3.4 Perception of Movement

Movement is a fundamental process in real and computer-generated environments. We move through the real world, for example, by simply walking, running, or driving a car or bicycle. In addition to the user's actual movements, most virtual worlds contain a multitude of movements of other objects. From a purely physical point of view, motion is defined as a change of location over time. In visual perception, the movement of a stimulus leads to a shift in the corresponding retinal image. Provided it has the same speed, the further away the stimulus is, the smaller is the retinal shift. Still, we mostly perceive the physical and not the retinal speed. This ability is called *speed constancy* (analogous to size constancy; see Sect. 2.4.5). The human body has elementary motion detectors available for the visual perception of movement, which detect local movements in a certain direction at a certain speed. More complex, global movements are composed of local movement stimuli.

Another essential sense in the perception of movement is the *vestibular sense*. Hair cells in the inner ear detect fluid movements in the archways of the organ of equilibrium. This then makes it possible to perceive linear and rotational accelerations. To stimulate the vestibular sense, motion simulators (platforms) are used in some VR systems. It is also possible, however, to create the illusion of an own movement by visual stimuli only. This illusion is called *vection* and is created, for example, in a standing train when looking at another train that starts moving next to it. This illusion is mainly based on the perception of the *optical flow*. The optical flow can be modeled as a vector field, i.e., each point P on an image is assigned a vector – whereby the image is not isolated but is part of a sequence of images in which pixels corresponding to P can be found. The direction of this vector indicates the direction of movement of the pixel P in the sequence of images. The speed of the movement can be determined from the length of the vector. In this respect, the optical flow is a projection of the 3D velocity vectors of visible objects onto the image plane. Accordingly, when we humans move, we receive a whole series of different movement cues, which are all integrated to derive a final perception of movement (Ernst 2008).

2.3.5 *Presence and Immersion*

As described at the beginning of this chapter, an essential potential of VR lies in the possibility to create in the user the illusion of *presence* in a virtual world. For example, the user should get the feeling of complete immersion in the virtual world. The term presence (cf. Chap. 1) describes the associated subjective feeling that one is oneself in the virtual environment and that this environment becomes real. Stimulus from the real environment is thereby faded out. On the other hand, immersion describes the degree of inclusion in a virtual world caused by objective, quantifiable stimuli, i.e., multimodal stimulations of human perception. Various studies have shown that presence occurs, particularly when a high degree of immersion is achieved. Presence is achieved when the user feels located in VR and behaves as in the real world. Various studies have shown that various virtual environment parameters have the potential to increase the presence of test subjects, such as a large field of vision, activated head-tracking and real walking (Hendrix and Barfield 1996). There are several questionnaires to measure the subjective feeling of presence (Witmer and Singer 1998; Slater et al. 1994). However, it is also possible to determine the degree of presence based on physiological data or human behavior. For example, a user with a high degree of presence in an apparently hazardous situation occurring in VR will respond physiologically, e.g., with increased skin conductance or heart rate (Slater et al. 1994).

2.4 Phenomena, Problems, Solutions

When using VR, one can observe surprising phenomena. From 1 s to the next, the presentation of a virtual world in a stereo display no longer succeeds. The viewer no longer sees the world plastically but sees everything twice. Users of VR start to complain about headaches or even vomit. Although the car's interior appeared spacious when first viewed in VR, the space in the real car is then perceived as disappointingly tight, even though the virtual car and the real car are identical in terms of proportions. With knowledge of human perception, one can try to explain these phenomena and develop solution strategies to avoid or at least mitigate the resulting problems. With today's VR, we are not able to reproduce reality 1:1; there are always deviations. For example, the two images required for stereopsis for the right and left eye may have been generated at a distance between the two virtual cameras that does not correspond to the actual eye distance of an individual observer. Is that bad? Knowledge of human perception helps us to assess the magnitude of the problem associated with these deviations. The following eight subsections deal with VR-typical phenomena and problems. In each subsection, the currently known attempts at explanation are also presented as well as approaches to solutions that can be derived from them.

2.4.1 *Deviating Observation Parameters*

Let us assume that we recreate the Eiffel Tower and its surroundings in a virtual environment. With a virtual camera, we create an image and show it to a human observer. Light stimuli from this image are projected onto the retina in the eyes of the observer and create a visual sensation. Ideally, the image of the virtual Eiffel Tower creates the same impression that viewers would have if they were standing in front of the real Eiffel Tower. However, aberrations usually occur, which can be explained by deviations in the viewing parameters. The virtual camera generates images on a plane, while human retinas are curved. The angle of view of the virtual camera can deviate from the *field of view* of the observer. The observer does not necessarily look at the image from the same place where the virtual camera was standing – the observer might be closer or further away, perhaps not looking perpendicularly at the image but from the side. As a result, enlargements or reductions, as well as distortions of image impressions, occur. This affects the estimation of distance or the perception of the inclination of objects (Kuhl et al. 2006).

However, the distortions caused by looking at the image of the virtual world from a different perspective are surprisingly not experienced as bothersome. One speaks of the *robustness of linear perspective* in human perception (Kubovy 1986). This phenomenon can also be observed in a cinema – if the viewer sits in the first row on the very outside, he or she is very likely to have a completely different perspective than the camera that shot the film. There is, if at all, only one place in the whole cinema where the perspective of the film camera is maintained. Although this means that almost all viewers see the film in a distorted way, they do not mind. One explanation for this phenomenon is that the viewer's visual system actively corrects the distorted image impression. This correction is based, among other things, on the deviation of the viewing direction from the normal of the image plane (Vishwanath et al. 2005). Conversely, this active correction could be responsible for the fact that images taken with a wide opening angle of the virtual camera ('wide-angle perspective') may appear distorted even when viewed from the correct position.

Although deviating viewing parameters are not experienced as particularly irritating, it is advisable to strive to minimize the deviation. This is especially true for applications where the correct estimation of distances or orientation of objects in space is of high importance. It is particularly relevant if the virtual world is not only viewed passively, but active actions (grasping objects, movement) are performed. Moreover, the virtual world and one's own body should not be perceived simultaneously from different viewing positions. An approach to minimization of such deviations frequently pursued in VR consists of determining the current viewing parameters (e.g., by head-tracking, see Chap. 5), such as position and direction of gaze. If these are known, they can be transferred to the virtual camera. Another approach is to simulate long focal lengths in the virtual camera, i.e., to realize almost a parallel projection. This reduces the distortions caused by a deviating viewer position (Hagen and Elliot 1976).

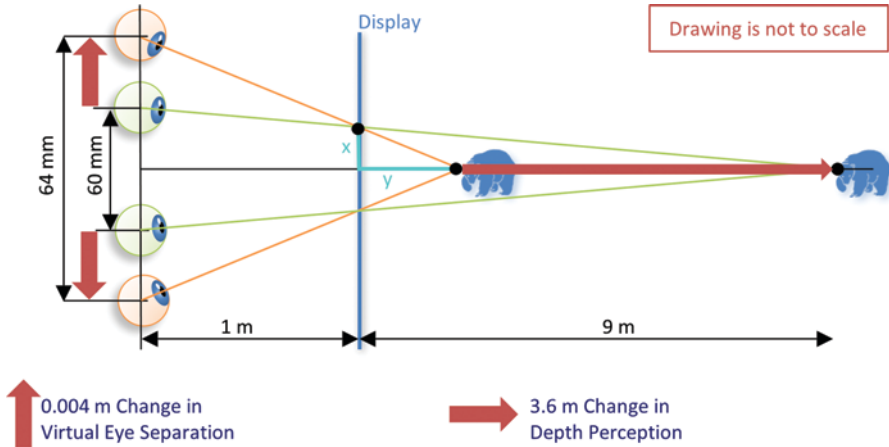


Fig. 2.4 Geometric effect of changing the virtual eye separation (drawing is not to scale). The geometric effects also influence perception (Bruder et al. 2012a)

Stereo displays can cause additional deviation because the two virtual cameras that generate the image for the left and right eyes have a distance (called *virtual eye separation*) that may differ from the distance of the viewer's pupils. On average, the *pupil distance* is 64 mm, but the individual range is large and lies approximately in the interval from 45 mm to 75 mm. Figure 2.4 shows an example that small changes in pupil distance can result in large changes in depth perception. In this example, the pupil distance is initially 64 mm and the object shown on the projection surface appears to be 9 m behind the projection surface. If the distance between the eye points is reduced by 4 mm, it follows from the set of beams that the virtual object moves forward by 3.6 m. But as with deviations in the viewing position, deviations between virtual eye separation and pupil distance are compensated by adaptation in such a way that they do not irritate the viewer. In fact, the distance between the virtual cameras can be changed several times in 1 s without the viewer even realizing it. In VR, it is therefore not absolutely necessary to first measure the distance between the two eyes of the viewer and then adapt the distance between the two virtual cameras accordingly. However, side effects such as nausea (see Sect. 2.4.7) can occur, even if the user does not consciously notice the difference.

2.4.2 Double Vision

If the viewer of a stereo display is not able to fuse the two different images shown to the left and right eyes, *diplopia* occurs. This is a severe problem in VR, as it is perceived as extremely irritating and has a negative effect on the feeling of presence in VR. Thus, diplopia should be avoided at all costs.

The reason for diplopia has already been explained in Sect. 2.2.1: the point to be merged lies outside Panum's fusional area. Since accommodation always occurs to the display plane, the visual system tends to move Panum's fusional area near the display surface of the stereo display as well (see vergence-focus conflict, Sect. 2.4.4). This means that a stereo display cannot make objects appear arbitrarily far in front of or behind the display surface. So, if one wants to display a virtual world with the help of a stereo display, there is only a limited area available in which the virtual objects can be placed in front of or behind the display (*parallax budget*) without diplopia. Williams and Parrish (1990) state that -25% to $+60\%$ of the distance from the viewer to the display surface are the limits for the usable stereo range (in the case of an HMD, the virtual distance of the display is to be used). Here, Panum's fusional area has its thinnest extent in the area of the point that the eyes fixate on. In the worst case, it has only a width of $1/10$ degree viewing angle. At a distance of 6° from the fixated point, Panum's fusional area increases in width. Then, it has a visual angle of about $1/3$ degree. If a display is at typical monitor distance and has 30 pixels per cm, then points can only be arranged in a depth range of 3 pixels before diplopia occurs (Ware 2000). The situation is aggravated by the fact that the entire Panum's fusional area should not be used, since only in a partial area can fusion be achieved without effort even over longer periods of time. This partial area is called *Percival's zone of comfort* and it covers about one-third of Panum's fusional area (Hoffmann et al. 2008).

One strategy to avoid diplopia is to enlarge Panum's fusional area. The size of this area depends, among other things, on the size and richness of detail of the objects being viewed and on the speed of moving objects. By blurring the images to be fused, the amount of detail is reduced. This way, Panum's fusional area can be enlarged. Another strategy is to bring virtual objects closer to the display area and thus into Panum's fusional area. With virtual eye separation, we have already learned a technique for this. If one reduces the distance between the virtual cameras, objects meant to appear behind the display can be brought closer to the display surface. Since human perception is robust against this manipulation, changing the virtual eye separation is useful to avoid diplopia. Ware et al. (1998) propose the following formula: virtual eye separation $v = 2.5 \text{ cm} + 5 \text{ cm} \cdot (a / b)^2$, where a is the distance of the point in the scene closest to the viewer and b is the distance of the point furthest away. Another technique to bring the virtual world into Panum's fusional area is the *cyclopean scale* (Ware et al. 1998). Here, the whole scene is scaled by one point between the two virtual cameras (cf. Fig. 2.5). The cyclopean scale can be combined with the manipulation of virtual eye separation, where scaling should be performed first. Such scaling is not only useful to bring a virtual world that is too spatially extended into Panum's fusional area, but also in the opposite case: a virtual world that does not use the limited area around the stereo display can be made to appear more three-dimensional by extending it. In VR, it is useful to be clear about the available parallax budget and its use. In a stereo display, the parallax that can be displayed cannot be arbitrarily small. The lower limit is the width of one pixel.

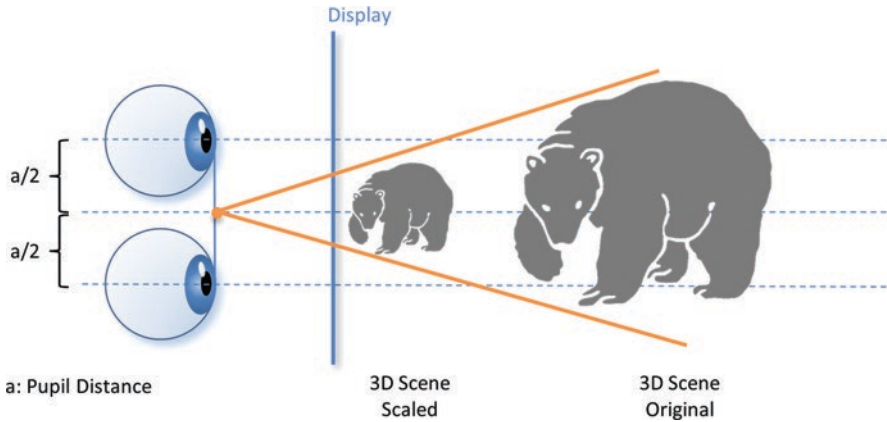


Fig. 2.5 Cyclopean scale

2.4.3 Frame Cancellation

The displays used for the presentation of virtual worlds usually have several imperfections, e.g., they cannot display the brightness levels found in reality, such as in sunlight. Also, the surface of the display is usually recognized as such and can be distracting. In particular, the edge of a display surface can be perceived as irritating. Let us assume we use a stereo display to make an object appear in front of the display plane. In case this object approaches the edge of the display and finally touches it. The following phenomenon can be observed. The illusion that the object is in front of the display is suddenly lost and the object snaps back to the level of the display. Moreover, diplopia can also be observed. This phenomenon is called *frame cancellation*, *paradoxical window* or *stereoscopic window violation* (Mendiburu 2009).

This phenomenon can be explained by the fact that the object has conflicting depth cues. According to the disparities, the object is in front of the display. However, the edge of the display seems to occlude the object, which suggests that it is behind the display. Occlusion is a stronger depth cue than disparity, which is why the object is perceived to be behind the display. Other explanation attempts point out that the object can only be seen by one eye when it is at the edge.

Keeping objects with negative parallax away from the edge or moving them quickly at the edge so that they are either completely visible or completely invisible on the image are simple strategies to avoid frame cancellation. Another strategy is to darken objects at the edge of the display and color the edge itself black so that the contrast between the edge and the object is small. Finally, black virtual stripes can be inserted in the depth of the object in the scene, thus seemingly bringing the display edge forward. The virtual stripes cover the virtual object when it approaches the display edge.

2.4.4 *Vergence-Focus Conflict*

In contrast to reality, some depth cues may be completely missing in VR, e.g., because the VR system's performance is not sufficient to calculate shadows in real time. Depth cues can also be wrong, e.g., the image blur might not be displayed correctly because it is difficult to determine the exact point the observer fixates on. While in reality the depth cues are consistent, they can be contradictory in VR, as the frame cancellation example shows. Contradictory depth cues not only have consequences such as a misjudgment of the spatial arrangement of objects in space or the loss of presence because the virtual world appears unnatural; other negative consequences can include eye stress, exhaustion and headaches. An example of this is the *vergence-focus conflict* (Mon-Williams and Wann 1998), also called *accommodation-convergence discrepancy* or *vergence-accommodation conflict*.

No matter whether a virtual world is viewed on a computer monitor, a projection or a head-mounted display (see Chap. 5), the viewers must adjust their eyes so that the display surface is seen sharply to easily perceive what is shown there. If a stereo display is used and an object appears in front of or behind the display surface due to disparity, the convergence is not set to the distance of the display surface but the apparent distance of the virtual object. Therefore, if the viewer wants to focus on a virtual object that appears to be in front of the display surface, the viewer must increase the convergence. As a result, however, the object suddenly appears unexpectedly blurred, as the eyes no longer focus on the display surface. This can also cause a contradiction between convergence and image blur. Convergence and focus information are therefore in conflict. As a result, headaches can occur. The risk of this increases with the duration of viewing of the virtual world (Hoffman et al. 2008).

The contradiction between the above depth cues can be reduced by bringing the virtual objects as close as possible to the display surface. For this purpose, the already discussed techniques, such as the cyclopean scale or the change of virtual eye separation can be used. These techniques can have side effects, such as falsification of depth perception. These side effects must be weighed against phenomena like fatigue or headache. There is no way to avoid the viewer's eyes converging on the display surface, as this is the only way to ensure that the image shown on the screen can be perceived sharply. The approach of subsequently introducing *depth of field* into the image (computer calculations of images allow the creation of images that are sharp everywhere – in contrast to real imaging systems such as a camera or the human eye) by blurring parts of the image and thus adapting the focus information to the convergence has not proven to be successful (Barsky and Kosloff 2008).

2.4.5 *Discrepancies in the Perception of Space*

In applications from the fields of architecture, CAD, urban visualization, training, simulation and medicine, three-dimensional spaces are presented. In these applications, it is essential that the users correctly perceive the virtually presented space, so

that they can draw conclusions about their actions and decisions in the real world. Discrepancies between the perception of size and distance in the virtual and real worlds are particularly critical in this application context. For example, a physician simulating an operation in the virtual world should not train wrong movements due to misjudgments of space. The correct perception of sizes and distances is essential for many applications in the field of VR.

Unfortunately, many studies show that there can always be discrepancies in the perception of virtual space. For example, it has often been shown that users tend to underestimate distances in the virtual world by up to 50% (Interrante et al. 2006; Steinicke et al. 2010a). A common approach to measuring distance estimation is, for example, blind or imaginary walking. Here the user is shown a mark at a certain distance (e.g., 4 m, 6 m or 8 m) on the floor, and the user must then walk to this mark with eyes closed. In the real world, this task is easy to accomplish, and we walk almost exactly up to the mark. A user in the virtual world who sees the same scene (geometrically correct) on a head-mounted display, for example, will most likely walk much too short a distance; in some cases by up to 50%. This effect can be observed with many techniques for evaluating spatial perception (e.g., triangular completion, blind throwing, imaginary walking or verbal assessment). Many studies have shown the influence of some factors (such as stereoscopic imaging, limited field of view, realistic lighting or shading) on this distance underestimation, but up till now, there is no complete explanation for this phenomenon.

According to *Emmert's law*, there is a clear connection between sizes and distances. In this respect, the phenomenon of underestimating distances can also be observed as a phenomenon of overestimating sizes. The law states that the perceived size is proportional to the product of perceived distance with retinal size, i.e., the size of the image on the retina. The resulting law of *size constancy* is used by humans already in infancy. If, for example, a mother distances herself from her child, the projection of the mother on the retina of the child becomes smaller, but the child is aware that the mother is not shrinking, but merely moving further away. It is also the case that the more of the above-mentioned depth cues are missing, the more the angle of vision is used for size estimation. Misjudgments in the real world can also occur. These can be exploited in perspective illusions, for example. However, such misjudgments result not only from perceptual errors but also from cognitive processes. Distances are considered to be greater, for example, when subjects carry a heavy backpack (Proffitt et al. 2003) or are asked to throw a heavier ball (Witt et al. 2004). Thus, not only optical stimuli and their processing play a role in depth perception but also the intended actions and the associated effort. Furthermore, studies have shown that presence influences the perception of distances. The more present we feel in the virtual world, the better our assessments of distances become (Interrante et al. 2006). This illustrates that the correct assessment of space can be a complex task even in the real world, depending on perceptual, cognitive and motor processes.

Various approaches exist to improve the estimation of distances or sizes in the virtual world or to make the space presented or the objects displayed in it appear larger or smaller. For example, one could simply scale the entire geometry. Now the

test persons would perceive the space as they would in the real world, but this does not solve the problem. Similar effects can be achieved by enlarging the *geometric field of view*. The geometric field of view refers to the area presented by the virtual scene, which is defined by the horizontal and vertical opening angle of the virtual camera. If this is enlarged, the viewer sees a larger area of the virtual world. However, since the same physical display is still used, this larger area must be mapped to the same area of the screen. Thus, the scene is minified, and objects appear further away (Kuhl et al. 2006). This is illustrated in Fig. 2.6. Similar effects can be achieved by changing the pupil distance. However, these approaches have the disadvantage that they actually present a different space utilizing, for example, perspective distortion. Subjects now continue to walk further, but they do so in another room that is projected with different geometric properties (see Fig. 2.6).

Alternative approaches are based on the idea of exaggerating the given depth cues to give the users clearer indications for the assessment of distances. For example, artificial shadows created by drawing lines to the base surface can give just as effective depth indications as stereoscopy. By using fog to desaturate the colors of distant objects, atmospheric depth can be imitated. This helps the user to better estimate distances, for example in virtual city models.

As already indicated above, cognitive factors also influence the assessment of space. It has been shown that the estimation of distances is significantly better in virtual space that is an exact representation of real space (Interrante et al. 2006). Follow-up studies have shown that this is not only due to the knowledge of real space, but especially to the higher sense of presence in such virtual worlds. This improved ability to assess distance can even be transferred to other virtual worlds.

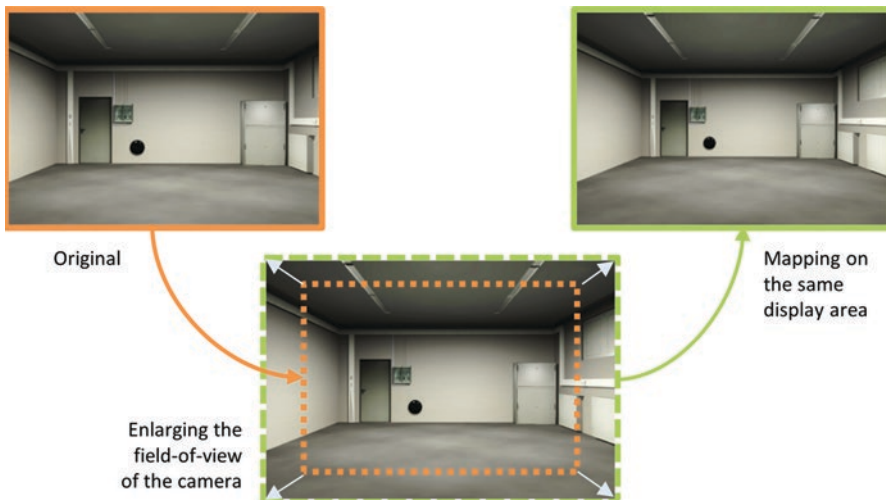


Fig. 2.6 Presentation of the same virtual space with (left) small and (right) large geometric field of view. (According to Steinicke et al. 2009)



Fig. 2.7 Representation of a virtual portal through which users can travel to different virtual worlds. (According to Steinicke et al. 2010b)

For instance, a transfer can succeed if one is teleported from a virtual space exactly simulating real space to these other virtual worlds through a portal (see Fig. 2.7).

2.4.6 *Discrepancies in the Perception of Movement*

A similar effect as with distance underestimation can also be observed in the perception of movement, such that speeds of movement or distances covered are over- or underestimated. For example, many studies have shown that forward movements along the line of sight are underestimated (Lappe et al. 2007; Loomis and Knapp 2003). This is particularly true if the movement is only visually presented, and the user essentially perceives only the optical flow. Even if the user moves simultaneously and the movements are mapped 1:1 onto the virtual camera, this underestimation of forward movements along the line of vision occurs. In contrast to virtual straight-line movements, virtual rotations often lead to an overestimation (Steinicke et al. 2010a).

In principle, these discrepancies in the perception of movement can be resolved relatively easily by applying *gains* to the tracked movements. For example (t_x, t_y, t_z) is a measured vector that describes the head movement of a user from one frame to the next. By means of a gain g_T , this movement can now be scaled simply by $(g_T \cdot t_x, g_T \cdot t_y, g_T \cdot t_z)$. If $g_T = 1$ no scaling occurs; for $g_T > 1$ the motion becomes faster; and for $g_T < 1$ the motion becomes slower. Psychophysical studies have shown that, for example, forward movements must be slightly accelerated (approx. 5–15%) to be

considered correct by users. In contrast, rotational speeds should be reduced slightly (by approximately 5–10%).

These manipulations now lead to the fact that the virtually represented movements are correctly perceived, i.e., the visually perceived movements match the vestibular-proprioceptive as well as the kinesthetic feedback. However, the users now actually perform different movements in the virtual and real environments, with the effect that, for example, certain distance estimation methods, such as counting steps, no longer work. More recent approaches by Bruder et al. (2012b) prevent such discrepancies between real and virtual movements by manipulating the optical flow. Such optical illusions only manipulate the perception of the movement but not the movement itself.

2.4.7 Cybersickness

Users of a VR/AR application may experience undesirable side effects: headaches, cold sweat, paleness, increased salivation, nausea and even vomiting, *ataxia* (disturbance of movement coordination), drowsiness, dizziness, fatigue, apathy (listlessness) or disorientation.

It is generally known that the use of IT systems is not free of health side effects. Just working at a screen can lead to headaches, for example, because the eyes are overstrained by focusing on one plane for a long time, or the visual system is stressed by flickering at low refresh rates or blurred images. These visual disturbances, known as *asthenopia* (eye strain), can also occur in VR/AR applications because they also use monitors. The symptoms can be more severe, e.g., because the displays in an HMD may be closer to the eyes or fusion may still be necessary for stereo vision. An early study (Stone 1993) concluded that 10 min of use of an HMD is as stressful for the visual system as sitting in front of a computer monitor for 8 h. The situation is worse for individuals who suffer from vision disorders and, for example, have problems with eye muscle coordination.

Side effects can also be expected when users are moving or being moved within an application, e.g., by means of a motion platform, or by simply walking. The syndrome of symptoms known as *seasickness* (more generally: *motion sickness*) has been known for a long time and has also been the subject of research. It is possible to characterize movements that cause seasickness – for example, it is known that low-frequency vibrations (which may also occur in VR installations) lead to seasickness. In flight simulators, which move an entire replica of a cockpit, it was observed early on that a significant proportion of pilots complain of feeling unwell (*simulator sickness*).

It is noteworthy that in VR/AR applications, the physiological symptoms mentioned at the beginning, which sometimes also occur in motion sickness or simulator sickness, can be observed even when the users are not moving at all. Just seeing images seems to cause discomfort. Therefore, a separate term has been coined: *cybersickness* (sometimes also called *VR sickness*). Cybersickness can occur not

only during VR/AR use but also for some time afterward. Usually, the symptoms disappear by themselves. However, users may still be sensitized even after the symptoms have subsided, i.e., they may suffer from cybersickness more quickly if they repeatedly use VR/AR systems within a certain period.

The exact causes of cybersickness are not known today. Probably there is also no single cause, but it is a multifactorial syndrome. One theory often used to explain cybersickness and motion sickness is the *sensory conflict theory*: problems occur when sensory perceptions are inconsistent. If, for example, a passenger is below deck while heavy seas are moving the ship, the brain receives information via the vestibular sense that strong movements are present. In contrast, the visual sense suggests precisely the opposite when no movement is detected in the cabin. Treisman (1977) motivates the sensory conflict theory by means of evolution: in the past, such inconsistencies in sensory perception only occurred if one had eaten the wrong mushrooms – and it is a sensible protective mechanism to quickly get rid of the poisoned stomach contents. Although in motion sickness inconsistencies between the visual sense and the sense of balance are particularly important in explaining symptoms, in cybersickness inconsistencies within a sense (e.g., contradictory depth cues in the visual sense, as in the vergence-focus conflict) are also considered, or even inconsistencies between the expected sensory impressions of a user and what is actually perceived. However, the sensory conflict theory cannot explain all phenomena in the area of cybersickness, and in particular, the extent to which symptoms occur can only be predicted with difficulty. Other attempts at explanation are therefore being sought. For instance, the *postural instability theory* (Riccio and Stoffregen 1991) assumes that people cannot cope with unfamiliar situations (such as those that can occur in a virtual environment) and that there is a disruption in the control of body posture that causes further symptoms.

Even though cybersickness's exact causes cannot be explained, factors have been identified that promote cybersickness's occurrence. The first group of factors depends on the individual. Age, gender, ethnicity and also individual previous experiences with VR and AR can influence the occurrence of cybersickness. Remarkable are significant individual differences in the susceptibility to cybersickness. People who frequently suffer from motion sickness are also more susceptible to cybersickness. The second group of factors is related to the VR/AR system. Influencing factors include image contrast and associated flicker, refresh rate, tracking errors, quality of system calibration and use of stereo displays. The larger the field of view (and the more peripheral vision is involved), the more frequently the occurrence of cybersickness is observed. Other essential factors are latencies, e.g., the time offset between head movement, the new head position's detection, and the correct image display of this new head position. A rule of thumb says that latencies above 40 ms are too high and that latencies below 20 ms should be aimed for. Finally, there is a third group of factors that are related to the application. Does the user spend a long time in the application? Does the user have to move the head frequently? Does the user rotate, perhaps even more than one axis at a time? Is the head tilted off the axis around which the user is rotated (*Coriolis stimulation*)? Is the user standing instead of sitting or lying down? Do users look directly down at the area in front of their feet

and cannot see far in the scene in general? Is it difficult to orientate in the scene, e.g., because a static frame of reference is missing? Is there much visual flow? Do users move quickly and a lot in a virtual world? Are there frequent changes in speed, are movements oscillating rather than linear, and are there abrupt movements? Does the user jump often or climb stairs? Are there unusual movements? Are users anxious? The more questions are answered in the affirmative and the more emphatic the agreement, the more cybersickness can be expected. Another factor is the degree of control (combined with the anticipation of movement) that a user has when navigating through a virtual environment. This is consistent with the phenomenon that the driver of a car or the helmsman of a ship suffers less often from motion sickness. Finally, a further factor is whether the application favors *vection*, i.e., the illusion of moving even though no movement is taking place.

If one wants to reduce the risk of cybersickness, one can minimize the influence of the factors mentioned, such as reducing latencies by improving the technical realization, reducing movements of the user by increased use of teleportation, or by inserting artificial blurring during the rotation of the user. Individually, one can avoid the occurrence of cybersickness by slowly getting used to VR/AR applications (McCaughey and Sharkey 1992). Chewing gum and adequate fluid intake are recommended. In extreme cases, one can take medication against motion sickness. As a herbal remedy, ginger does not prevent cybersickness, but it does counteract nausea and vomiting. Ultimately, it must be accepted that the occurrence of cybersickness cannot be prevented with certainty. Consequently, users should be given an easy way to terminate a VR/AR application at any time. It is also important to inform users about the possible side effects and to obtain the explicit consent of users, especially in user tests.

Whether and to what extent cybersickness occurs is usually determined by observing or asking users. For this purpose, it makes sense to use standardized questionnaires. Although not intended for cybersickness, the *Simulator Sickness Questionnaire (SSQ)* and the *Motion Sickness Assessment Questionnaire (MSAQ)* are often used (Kennedy et al. 1993). Alternatively, users can be watched to detect symptoms – but this is sometimes difficult, e.g., headaches are difficult to detect, but vomiting is easy. Physiological body values (e.g., heart rate, skin conductivity) are sometimes measured. Here, especially, the interpretation of the measured values is difficult. Based on such measurements, studies such as Lawson (2015) conclude that 60–80% of users of a VR application show symptoms of cybersickness. Around 15% show symptoms so severe that they have to stop using the application. However, such figures should be applied with great caution to a specific VR/AR application – there are many possible influencing factors and, therefore, strong fluctuations in the values. Individual differences among users are also considerable; the same user can react very differently to a scenario repeated several times during each repetition. Nevertheless, these figures show that cybersickness is not a marginal problem, but a real barrier to the use of VR and AR. Consequently, cybersickness should be taken into account in the development of every VR/AR application.

2.4.8 Vertical Parallax Problem

One problem with the technical implementation of stereo vision is that the virtual projection plane used in rendering cannot be brought into alignment with the display's real plane if the two are not parallel to each other. This leads to *vertical parallax*, which the viewer perceives as a strain and can lead to errors in depth perception, blurring at specific image points or double images. Let us look at Fig. 2.8a. An observer fixates on point P, and thus the eyes are aligned accordingly – the directions of gaze are no longer parallel and convergence occurs. If we reproduce this when rendering the images, i.e., if we apply the *toe-in method*, the two projection planes intersect at point P and are not parallel to each other. Most of the time, it is technically not possible to realize that, for each of the two projection planes, there is a separate display available that can be aligned accordingly. Instead, a common real display is used for both projection planes. The point A has the distance v from the display. This is the unwanted vertical parallax. The further point A is from point P, the greater the vertical parallax, and the more blurred or distorted the image appears. As with horizontal parallax, you can distinguish between negative parallax (located before the display plane, such as point A) and positive parallax (located behind the display plane, such as point B).

Because of the problem of vertical parallax, the toe-in method is avoided, and the *off-axis method* is used instead. This is shown in Fig. 2.8b. Here, each eye has a fictitious point of view P' or P'', so that both projection planes lie on top of each other. This means that both projection planes can also be mapped exactly onto a single display plane. As a result, the viewing volumes are no longer symmetrical. Accordingly, an *asymmetrical viewing volume* must be set during rendering. This is

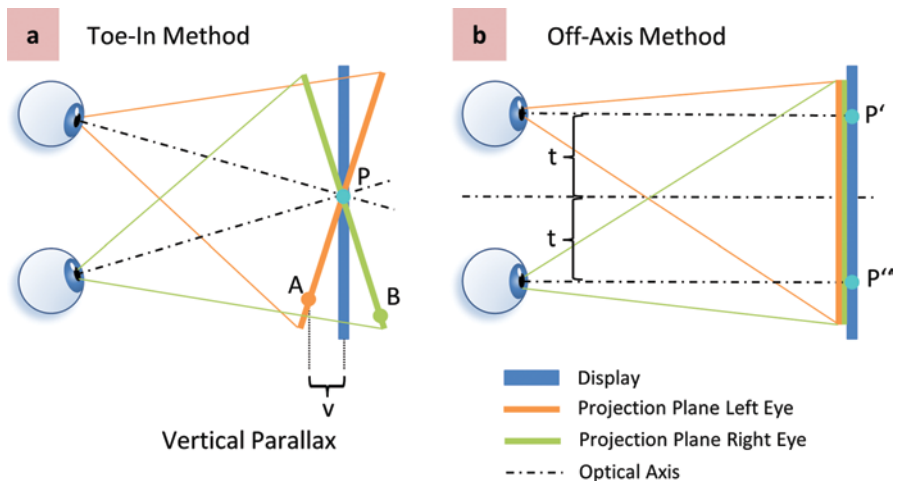


Fig. 2.8 (a) The toe-in method leads to the occurrence of vertical parallax. (b) The off-axis method solves this problem

shifted by the distance t from the center axis ('off-axis'). The exact size of the view volumes can be calculated through a set of rays if the distance between the projection plane and the eyepoint is known. This solves the problem of vertical parallax.

2.5 Use of Perceptual Aspects

With knowledge of human perception, we can not only explain problems occurring in VR. Knowledge about the operation of human perception can also be useful to improve a VR experience or to use available resources well. In Sect. 2.4.1 we have already seen an example of how the ability of the human visual system to adapt makes complex technical solutions superfluous: we do not have to measure the distance between the pupils of an observer to adjust the virtual cameras correctly. On the contrary, we can manipulate virtual eye separation to prevent diplopia because we know that human perception reacts robustly to changes in virtual eye separation. Besides adaptation, there are two other important perceptual aspects of VR that are exploited in VR: salience and user guidance.

2.5.1 Salience

Human perception does not have the capacity to process all environmental stimuli in equal detail. Priorities are set, and people can focus *attention* on certain aspects. In the human visual system, for example, differentiation is already inherently built-in through the uneven distribution of sensory cells on the retina of the eye – humans can align the fovea in such a way that light stimuli from environmental objects classified as particularly relevant hit this point in the retina, which possesses a high number of sensory cells.

VR makes use of this characteristic of human perception because VR systems often do not have the capacity to artificially generate all environmental stimuli equally well. If you know what the user of a VR system is focusing his or her attention on, you can adjust the quality of the rendering (e.g., simulation of surface materials, quality of the object models, effort invested in anti-aliasing), sound quality, quality of the animation or accuracy of the world simulation. Conversely, one does not need to invest any or only a few resources of a VR system in areas that are not the focus of attention. In extreme cases one can even observe *inattentional blindness*. In an experiment, Simons and Chabris (1999) showed nearly 200 students 75 s long videos in which basketball players throw a ball at each other. The viewers had the task of counting how many passes a team makes – attention was thus focused on the ball. The video showed an unusual event for 5 s, e.g., a person dressed as a gorilla walking across the field. About half of the viewers did not notice this at all. So why go to the trouble of creating images of a gorilla in a VR version of this scene if the viewer does not notice it?

There are two obstacles to exploiting these phenomena of human perception. On the one hand, while it is possible to make statements about probabilities, it is not possible to predict with certainty which environmental stimuli are considered important for an individual in a concrete situation. Hence, we could make mistakes. For example, we leave out the gorilla in our VR scene even though the viewer would have seen it in the concrete situation. Here it is essential to weigh up the likelihood of making a mistake and the consequences. Due to the limited performance of VR systems, one may have no choice but to set priorities to meet real-time requirements. Violating real-time conditions (e.g., the virtual environment reacts with a noticeable delay to a user's action; see Chap. 7) can have more serious consequences than choosing the wrong priorities.

On the other hand, there is the issue that the information is needed on which the viewer's attention is currently focused. There are different approaches to obtaining this information. Firstly, technical systems can be used to determine where the observer is currently looking (eye-tracking; see Chap. 4). Secondly, through knowledge about the application and the current goals and tasks of the user of VR, it can be estimated which objects of the virtual world are likely to attract a high level of attention (Cater et al. 2003). In the gorilla example, we could deduce from the task given to the viewers that the ball is the center of attention. Myszkowski (2002) creates *task maps* that assign each object a priority for rendering, with moving objects automatically getting a higher priority. A third approach (Treisman and Gelade 1980) is based on the *feature integration theory*. This approach is attractive for VR because it does not require any additional knowledge about the application or the viewing direction of the viewer but can work solely on the images of the 3D scene: the *saliency* (also called *saliency*) of objects is determined as a measure of their importance.

Saliency describes how strongly an object stands out from its surroundings (e.g., in color, orientation, movement, depth). If one shows a person a picture with 50 squares of equal size, 47 of which are grey and 3 are red, the 3 red squares stand out and are immediately noticed. The person can easily and quickly answer the question of how many red squares can be seen in the picture. Even if the number of gray squares is quintupled, the person can just as quickly recognize that there are 3 red squares present. The feature integration theory explains this observation by postulating that human perception works stepwise. In the first stage, all incoming image stimuli are processed in parallel and examined for specific features. This happens subconsciously. It is called *preattentive processing* (see Fig. 2.9). Anatomically, receptive fields have already been identified, i.e., groups of nerve cells in the brain that are responsible for these tasks of feature extraction. The result of preattentive processing then serves as the basis for the decision in the next stage as to which regions in the image are to receive attention.

If one wants to take advantage of this in VR, one must first calculate an *attention map* (*saliency map*) of an image in which every pixel of an image is assigned a saliency value. Today's algorithms for this purpose are based on the work of Itti et al. (1998). The procedure consists of first splitting the input image into feature images, e.g., extracting a luminance image that contains only brightness values.



Fig. 2.9 Example of preattentive processing: the time required to find the number of occurrences of the digit ‘7’ in a series of numbers can be reduced considerably if the number ‘7’ is displayed in a different color. This is processed in a preattentive stage. If the number series size increases, the time for the task completion increases if the number ‘7’ is not highlighted; otherwise it remains the same

These feature images are examined in parallel with image processing methods, whereby the operation of the receptive fields in the brain is modeled mathematically. Receptive fields that recognize orientation in a feature image can be described, for example, by *Gabor filters*. A Gabor filter is constructed from a Gaussian function modulated by a sinusoidal function and can thus map the sensitivity for different frequencies and orientations. The results of processing the individual feature images are normalized. The saliency values are determined from this by weighted summation. The weighting can also be chosen depending on the current task of the observer. It is often determined by machine learning, e.g., utilizing neural networks. In this processing step, another phenomenon of human perception can be mimicked: *inhibition*. Inhibition means that nerve cells can not only be stimulated but also inhibited by stimuli, which increases differences. Algorithmically, this can be realized, for example, with a *winner-takes-all approach*, i.e., the greatest value is used for saliency, while saliency in the vicinity of the greatest value is reduced to enhance its significance further. The saliency map finally obtained then serves as a basis for decisions on how to use resources of the VR system, e.g., for areas with high saliency 3D models with a high level of detail are used. Further data can also be obtained, e.g., *fixation maps* (Le Meur et al. 2006), which predict what an observer is likely to fixate on. Since saliency maps are two-dimensional, a relatively complex back-calculation into the 3D scene is necessary to assign a saliency value to virtual 3D objects. Therefore, approaches are also being considered that directly examine characteristics of 3D objects and derive a saliency value from them (Lee et al. 2005).

2.5.2 User Guidance

The area covered by the virtual environment’s hardware platform in which users can move around is usually much smaller than the virtual world represented in it. Clearly, without additional input devices, the users can only explore a very small part of the virtual world by their own movements. There is a variety of so-called locomotion devices that prevent the user from moving from one place to another in the real world

while walking. Examples are omnidirectional treadmills or the *Virtuix Omni* (see Chap. 4). Another approach is based on the idea of manipulating users in such a way that they walk on different paths in the real world than those perceived in the virtual world. If, for example, a small virtual rotation to one side is introduced during a user's forward movement, the user has to compensate for this rotation in the real world to be able to continue walking virtually straight ahead. This results in the user walking on a curved path in the opposite direction. Thus, users can be guided on a circular path in the VR setup while they think they are walking straight ahead in the virtual world. Investigations have shown whether and from when on test persons can detect such manipulations through *re-directed walking* (Steinicke et al. 2010a). For instance, test persons who walk straight ahead in the virtual world can be guided on a circle with a radius of about 20 m in the real world without noticing this.

2.6 Summary and Questions

In this chapter, you have acquired basic knowledge in the field of human information processing. In particular, we have dealt with some of the most important aspects in the field of spatial perception and the perception of movement. Based on this, you have learned about relevant phenomena and problems of VR. You have also seen examples of how different limitations of human perception can be exploited to improve the quality and user experience during a VR session. To design effective virtual worlds, it is essential to take findings from perceptual psychology on human information processing into account. Aspects related to perception have become increasingly important in recent years, which is reflected in the increased number of research projects in this field.

Check your understanding of the chapter by answering the following questions:

- Why is the response time for a subject longer when deciding whether a stimulus displayed on the screen matches a previously displayed stimulus than when the subject only has to respond when the stimulus appears?
- Compare a photo of a beach in the Caribbean and a photo of the streets of Manhattan. What pictorial depth cues are present in the photos?
- How does the object in Fig. 2.4 move if the virtual eye separation is not reduced from 64 mm to 60 mm, but instead increases to 70 mm?
- Why should a cyclopean scale be performed before virtual eye separation?
- Take a stereo display and conduct experiments to determine Panum's fusional area of the stereo display. Try using the techniques presented in Sect. 2.4 to fit a 3D scene that initially protrudes over the panorama area.
- Find more examples of conflicting depth cues in VR.
- You would like to build a light rail simulator with which a learner can drive a streetcar through a virtual city. Think about where perceptual aspects need to be considered. Which problems can potentially arise? Where can perceptual aspects be exploited in the technical realization of the simulator?

Recommended Reading¹

- Goldstein EB (2016) *Sensation and Perception* (10th edn). Cengage Learning, Belmont – *Standard work from the psychology of perception which is not limited to visual perception. Very informative and with many examples.*
- Thompson WB, Fleming WF, Creem-Regehr SH, Stefanucci JK (2011) *Visual Perception from a Computer Graphics Perspective*. CRC Press, Boca Raton – *Textbook which also explains essential aspects of perception for VR and always makes the connection to computer graphics.*

References

- Barsky BA, Kosloff TJ (2008) Algorithms for rendering depth of field effects in computer graphics. In: Proceedings of 12th WSEAS international conference on computers, pp 999–1010
- Bruder G, Pusch A, Steinicke F (2012a) Analyzing effects of geometric rendering parameters on size and distance estimation in on-axis stereographic. In: Proceedings of ACM Symposium on Applied Perception (SAP 12), pp 111–118
- Bruder G, Steinicke F, Wieland P, Lappe M (2012b) Tuning self-motion perception in virtual reality with visual illusions. *IEEE Trans Vis Comput Graph* 18(7):1068–1078
- Card SK, Moran TP, Newell A (1986a) The model human processor: an engineering model of human performance. In: Handbook of perception and human performance. Vol. 2: cognitive processes and performance, pp 1–35
- Card SK, Moran TP, Newell A (1986b) The psychology of human–computer interaction. CRC Press
- Cater K, Chalmers A, Ward G (2003) Detail to attention: exploiting visual tasks for visual rendering. In: Proceedings of Eurographics workshop on rendering, pp 270–280
- Ernst MO (2008) Multisensory integration: a late bloomer. *Curr Biol* 18(12):R519–R521
- Hagen MA, Elliott HB (1976) An investigation of the relationship between viewing conditions and preference for true and modified perspective with adults. *J Exp Psychol Hum Percept Perform* 5:479–490
- Hayward V, Astley OR, Cruz-Hernandez M, Grant D, La-Torre GR-D (2004) Haptic interfaces and devices. *Sens Rev* 24(1):16–29
- Hendrix C, Barfield W (1996) Presence within virtual environments as a function of visual display parameters. *Presence Teleop Virt* 5(3):274–289
- Hoffmann DM, Girshick AR, Akeley K, Banks MS (2008) Vergence-accommodation conflicts hinder visual performance and cause visual fatigue. *J Vis* 8(3):1–30
- Howard IP (2002) Seeing in depth: Vol. 1. Basic mechanisms. I Porteous, Toronto
- Interrante V, Anderson L, Ries B (2006) Distance perception in immersive virtual environments, revisited. In: Proceedings of IEEE virtual reality 2006, pp 3–10
- Itti L, Koch C, Niebur E (1998) A model of saliency-based visual attention for rapid scene analysis. *IEEE Trans Pattern Anal Mach Intell* 20:1254–1259
- Kennedy RS, Lane NE, Berbaum KS, Lilienthal GS (1993) Simulator sickness questionnaire: an enhanced method for quantifying simulator sickness. *International Journal of Aviation Psychology* 3(3):203–220

¹The ACM Symposium on Applied Perception (SAP) as well as the journal *Transaction on Applied Perception (TAP)* deal with multisensory perception in virtual worlds.

- Kubovy M (1986) *The psychology of linear perspective and renaissance art*. Cambridge University Press, Cambridge
- Kuhl SA, Thompson WB, Creem-Regehr SH (2006) Minification influences spatial judgement in immersive virtual environments. In: *Symposium on applied perception in graphics and visualization*, pp 15–19
- Lappe M, Jenkin M, Harris LR (2007) Travel distance estimation from visual motion by leaky path integration. *Exp Brain Res* 180:35–48
- Lawson B (2015) Motion sickness symptomatology and origins. In: Hale KS, Stanney KM (eds) *Handbook of virtual environments: design, implementation, and applications*. CRC Press, pp 532–587
- Le Meur O, Le Callet P, Barba D, Thoreau D (2006) A coherent computational approach to model the bottom-up visual attention. *IEEE Trans Pattern Anal Mach Intell* 28(5):802–817
- Lee CH, Varshney A, Jacobs DW (2005) Mesh saliency. In: *Proceedings of SIGGRAPH 2005*, pp 659–666
- Loomis JM, Knapp JM (2003) Visual perception of egocentric distance in real and virtual environments. In: Hettinger LJ, Haas MW (eds) *Virtual and adaptive environments*. Erlbaum, Mahwah
- Malaka R, Butz A, Hußmann H (2009) *Media informatics – an introduction*. Pearson, Munich
- Marr D (1982) *Vision: a computational investigation into the human representation and processing of visual information*. MIT Press, Cambridge
- McCauley ME, Sharkey TJ (1992) Cybersickness: perception of self-motion in virtual environments. *Presence Teleop Virt* 1(3):311–318
- Mendiburu B (2009) *3D movie making: stereoscopic digital cinema from script to screen*. Focal Press, New York
- Mon-Williams M, Wann JP (1998) Binocular virtual reality displays: when problems do and don't occur. *Hum Factors* 40(1):42–49
- Myszkowski K (2002) Perception-based global illumination, rendering and animation techniques. In: *Spring conference on computer graphics*, pp 13–24
- Ooi TL, Wu B, He ZJ (2001) Distance determination by the angular declination below the horizon. *Nature* 414:197–200
- Preim B, Dachselt R (2015) *Interaktive Systeme (Band 2)*. Springer Vieweg, Berlin, Heidelberg
- Proffitt DR, Stefanucci J, Banton T, Epstein W (2003) The role of effort in distance perception. *Psychol Sci* 14:106–112
- Riccio GE, Stoffregen TA (1991) An ecological theory of motion sickness and postural instability. *Ecol Psychol* 3(3):195–240
- Sharp H, Preece J, Rogers Y (2019) *Interaction design: beyond human–computer interaction*. Wiley, Indianapolis
- Shneiderman B, Plaisant C, Cohen M, Jacobs S, Elmqvist N, Diakopoulos N (2018) *Designing the user interface – strategies for effective human–computer interaction*. Pearson Education Ltd, Harlow
- Simons DJ, Chabris CF (1999) Gorillas in our midst: sustained inattention blindness for dynamic events. *Perception* 28(9):1059–1074
- Slater M, Usoh M, Steed A (1994) Depth of presence in virtual environments. *Presence Teleop Virt* 3:130–144
- Steinicke F, Bruder G, Kuhl S, Willemsen P, Lappe M, Hinrichs KH (2009) Judgment of natural perspective projections in head-mounted display environments. In: *Proceedings of VRST 2009*, pp 35–42
- Steinicke F, Bruder G, Jerald J, Frenz H, Lappe M (2010a) Estimation of detection thresholds for redirected walking techniques. *IEEE Trans Vis Comput Graph* 16(1):17–27
- Steinicke F, Bruder G, Hinrichs KH, Steed A (2010b) Gradual transitions and their effects on presence and distance estimation. *Comput Graph* 34(1):26–33
- Stone B (1993) Concerns raised about eye strain in VR systems. *Real-Time Graph* 2(4):1–13
- Treisman M (1977) Motion sickness: an evolutionary hypothesis. *Science* 197:493–495

- Treisman AM, Gelade G (1980) A feature integration theory of attention. *Cogn Psychol* 12(1):97–136
- Vishwanath D, Girshick AR, Banks MS (2005) Why pictures look right when viewed from the wrong place. *Nat Neurosci* 8(10):1401–1410
- Wanger LR, Ferwanger JA, Greenberg DA (1992) Perceiving spatial relationships in computer-generated images. *IEEE Comput Graph Appl* 12(3):44–58
- Ware C (2000) *Information visualization – perception for design*. Morgan Kaufmann, San Francisco
- Ware C, Gobrecht C, Paton M (1998) Dynamic adjustment of stereo display parameters. *IEEE Trans Syst Man Cybern* 28(1):56–65
- Williams SP, Parrish RV (1990) New computational control techniques and increased understanding for 3-D displays. In: *Proceedings of SPIE Stereoscopic Display Applications*, pp 73–82
- Witmer BG, Singer MJ (1998) Measuring presence in virtual environments: a presence questionnaire. *Presence: Teleoperators Virtual Environ* 7(3):225–240
- Witt JK, Proffitt DR, Epstein W (2004) Perceiving distance: a role of effort and intent. *Perception* 33:577–590

Chapter 3

Virtual Worlds



Bernhard Jung and Arnd Vitzthum

Abstract Virtual worlds, the contents of VR environments, consist of 3D objects with dynamic behavior that react in real time to user input. After a brief overview of the creation process of virtual worlds, this chapter introduces a central data structure of many VR/AR applications, the *scene graph*, which allows us to structure virtual worlds in a hierarchical manner. Afterwards, different ways to represent 3D objects are presented and discussed in the context of interactive virtual worlds. Special attention is given to methods for optimizing 3D objects with respect to the real-time requirements of virtual worlds. Subsequently, an overview of basic methods for generating the dynamic behavior of 3D objects is given, such as animations, physics-based simulations and the support of user interactions with 3D objects. A section on sound, lighting and backgrounds describes elements of virtual worlds that are supported by common scene graph systems. The concluding section on special-purpose systems deals with 3D objects that are usually modeled with the help of custom methods and tools, such as virtual humans, particle systems, terrains and vegetation.

3.1 Introduction

The term *virtual world* refers to the content of VR environments. Virtual worlds consist of 3D objects that exhibit dynamic behavior and can react to user input. Besides the actual 3D objects, virtual worlds also contain abstract, invisible objects that support the simulation and rendering of the virtual world. These include light and sound sources, virtual cameras and proxy objects for efficient collision checks or physics calculations. In the following, a simplified overview of the steps in modeling virtual worlds and their integration into VR systems is given.

Dedicated website for additional material: vr-ar-book.org

B. Jung (✉)

Institute for Informatics, Technical University Bergakademie Freiberg, Freiberg, Germany
e-mail: jung@informatik.tu-freiberg.de

3.1.1 *Requirements on 3D Object Representations for Virtual Worlds*

In contrast to other areas of 3D computer graphics that often emphasize photorealism and high visual detail of still images or animations, virtual worlds demand *real-time capability* and *interactivity*.

In simple terms, *real-time capability* means that the virtual world is updated and displayed immediately, i.e., without any noticeable delay. Ideally, the user would not perceive any difference from the real world in terms of the temporal behavior of the virtual world. For a more detailed description of the topics *real-time capability* and *latency* in the context of entire VR systems, refer to Sect. 7.1. For each time step or *frame*, e.g., 60 times per second, the subtasks of user tracking and input processing, virtual world simulation, rendering and output on the displays have to be performed by a VR/AR system (see Sect. 1.5). The way in which 3D objects are modeled directly influences the subtasks of world simulation and rendering. If the virtual world model becomes too complex, real-time capability may no longer be possible.

Interactivity means first of all that the system will respond to (any) activities of the user, such as moving around in the virtual world or influencing the behavior of the 3D objects contained therein. User interaction techniques for, e.g., navigation and object manipulation in VR, are the topic of Chap. 6. While the implementation of interactive behavior usually requires scripting or other forms of programming, certain measures can already be taken at the modeling stage of virtual worlds to make these interactions effective and efficient. For example, to accelerate user interactions as well as the dynamic behavior of 3D objects resulting from these interactions, 3D objects are often enriched with simpler *collision geometries* such as cuboids or spheres. This allows efficient collision checks not only of the 3D objects with each other, but also, during user interactions, with the virtual representation of the user or a virtual pointing ray emanating from an interaction device (see also Sects. 6.2 and 6.4 for selecting and manipulating 3D objects, and Sect. 7.2 for collision detection).

Concerning the *visual realism* of virtual worlds, a wide spectrum of requirements exists in different kinds of VR/AR applications. While virtual worlds for training purposes should strongly resemble the real world, the visual appearance of gaming applications may range from toon-like, through realistic to artistically fanciful. In scientific applications, often clearer form and color schemes are preferred over realistic appearance. Even in VR/AR applications with high demands on visual quality, however, the requirements regarding real-time and interactivity of the virtual world generally take precedence.

3.1.2 Creation of 3D Models

The first step in the creation process of virtual worlds is the creation of the individual 3D models. This can be done in different ways:

- ‘Manual’ modeling of 3D objects in *3D modeling tools*. Widely used examples are Autodesk’s 3ds Max and Maya, and the open-source tool Blender. 3D modeling tools typically also support the creation of animations, for example by integrating motion capture data to animate virtual humans. In the technical domain, CAD systems are used which often provide very precise geometric modeling. Before import into VR systems, it is typically necessary to simplify the often very complex CAD models (see below and Sect. 3.3.4).
- *Procedural modeling* techniques are used for the automatic generation of very large or very complex objects, whose modeling by hand would be too time-consuming. An example is the automatic generation of 3D models of buildings or entire cities, possibly based on real-world geodata. Another example is the generation of objects with fractal shapes, such as terrain or trees (see Sect. 3.5).
- Furthermore, 3D models can be acquired as *3D scans* of real objects or environments. For this purpose, e.g., laser scanners, which provide depth information, are used in combination with color cameras to obtain the object textures. By means of *photogrammetric* methods it is also possible to create 3D models solely on the basis of multiple camera images of the object (see Fig. 3.1). Raw 3D scans may require complex post-processing steps, such as filling gaps (in areas not captured by the camera due to occlusion), simplifying the geometry and removing shadows or viewpoint-dependent highlights from the object textures. A good overview of the algorithmic procedures for the 3D reconstruction of objects from 2D images can be found in the book by Hartley and Zisserman (2004). Among the more frequently used software tools are Agisoft Metashape, Autodesk ReCap, 3DF Zephyr and the open source VisualSFM.

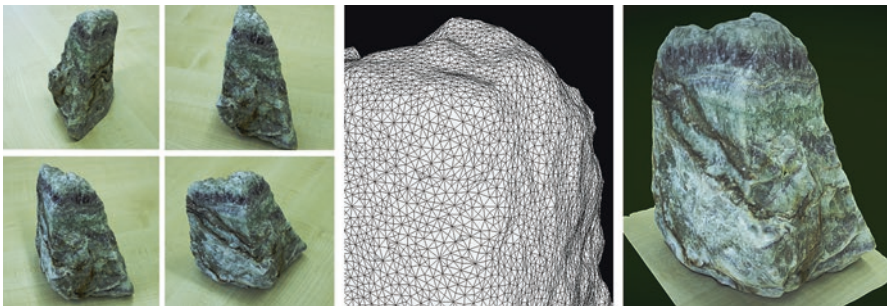


Fig. 3.1 Generation of 3D models using photogrammetry software. Left: Selection of photos of an object; typically several dozen photos would be used. Middle: Generated 3D model in wire-frame view. Right: Textured 3D model

3.1.3 Preparation of 3D Models for VR/AR

3D objects created or acquired by the above methods usually require post-processing so that they can be included in virtual worlds. This typically concerns simplification of the object geometry and the adaptation of visual detail. Further, objects must be converted into file formats suitable for the respective VR/AR system.

The simplification of the object geometry aims, among other things, at enabling an efficient rendering of the 3D objects. Essentially, the goal is to reduce the number of polygons of a 3D object. This can be done, for example, mostly automatically by special programs for *simplification of polygon meshes* (some manual postprocessing is typically required, however). Another option is to model an additional, low-resolution variant of the 3D object, which is textured with renderings of the original, high-resolution 3D object (*texture baking*). Furthermore, it can be useful to provide several variants of a 3D object in different resolutions, between which it is possible to switch at runtime depending on the distance to the viewer or the field-of-view covered (*level of detail*). These and other techniques are elaborated in Sect. 3.3.

The 3D objects must also be converted into a file format that is supported by the respective runtime environment of the virtual world. This step can be done using special conversion programs or export options of 3D modeling tools. For commercial game engines, the proprietary FBX format by Autodesk is primarily relevant. Popular file formats are also, for example, the somewhat older but still widely supported formats Wavefront (.obj) and Autodesk 3DS (.3ds). Open standards include COLLADA (.dae), glTF (.gltf) and X3D (.x3d).

X3D (Web 3D Consortium 2013) is an XML and scene graph-based description language for 3D content. The successor of *VRML* (Virtual Reality Markup Language), X3D was adopted by the W3C Consortium as a standard for the representation of virtual worlds in web applications. Many common 3D modeling tools offer an export option to the X3D format, which thus also plays an important role as an exchange format for 3D models and 3D scenes.

3.1.4 Integration of 3D Models into VR/AR Runtime Environments

Finally, the individual 3D models must be combined into complete virtual worlds. For this, the 3D objects are arranged in a *scene graph*. This could be done, for example, by creating a single X3D description of the entire virtual world. More common, however, is to load the individual objects into a *world editor* of a game engine and to create the scene graph there. Furthermore, to simplify collision detection and collision handling as part of the world simulation, it is often advisable to equip the 3D objects with simplified collision geometries at this point (see Sect. 3.4

and in-depth Sect. 7.2). In addition to the actual 3D objects, virtual worlds contain special objects such as virtual cameras, light sources, audio sources and backgrounds, which should now also be defined (see Sect. 3.5).

3.2 Scene Graphs

The elements of the virtual world, such as its 3D objects, sounds, cameras and light sources, as well as information on how these elements are spatially arranged and hierarchically structured are described by the so-called *scene*. At runtime, the scene is rendered from the user's point of view, i.e., converted into one, or in the case of stereo displays two, or in the case of multi-projector systems multiple 2D raster graphics (bitmap images). The rendered raster graphics are then displayed on suitable devices (e.g., monitor, head-mounted display, projection systems such as a CAVE, etc.; cf. Chap. 5). In addition, audio information contained in the scene is output via speakers or headphones. A scene can change dynamically at runtime. For example, the positions of 3D objects can vary over time. This is referred to as an *animated* scene. If 3D objects also react to user input, the scene is *interactive*. The ability of an object to react to events such as user input or interaction with other objects by changing its state is called *behavior*.

A *scene graph* describes the logical and often spatial structure of the scene elements in a hierarchical way. Common data structures for scene graphs are *trees* and, more general, *directed acyclic graphs (DAGs)*. Conceptually, a scene graph consists of nodes connected by directed edges. If an edge runs from node A to node B, A is called the parent node and B is called the child node. Scene graphs contain exactly one *root node*, that is, a node that does not have a parent node. Nodes without children are called *leaf nodes*. Unlike a tree, which is a special kind of DAG, child nodes are allowed to have multiple parent nodes in DAGs. The scene graph is traversed from the root to the leaves at runtime, collecting information for rendering, among other things (see Sect. 7.3).

Scene graphs allow a compact representation of hierarchically structured virtual worlds. Figure 3.2 shows an example of a scene comprising a vehicle, a road and a nail. The vehicle consists of several sub-objects, i.e., the body and four wheels. The hierarchical relationship is modeled by grouping them in a *transformation group*. By using a transformation group instead of a 'plain' group, the vehicle can be moved as a whole. The four wheels are also each represented by a transformation group that allows the wheels to rotate while the car is moving. Figure 3.2 also illustrates an advantage of scene graphs having a DAG structure rather than being trees, i.e., the ability to reuse 3D objects (or groups of them) very easily. In the vehicle example, only one geometry object of the wheel has to be kept in memory instead of keeping four separate copies.

The *leaf nodes* of the scene graph represent the actual (mostly geometric) 3D objects. All *internal nodes* have a grouping function. The *root node* represents the entire scene, as it encompasses all 3D objects. *Transformation groups* deserve

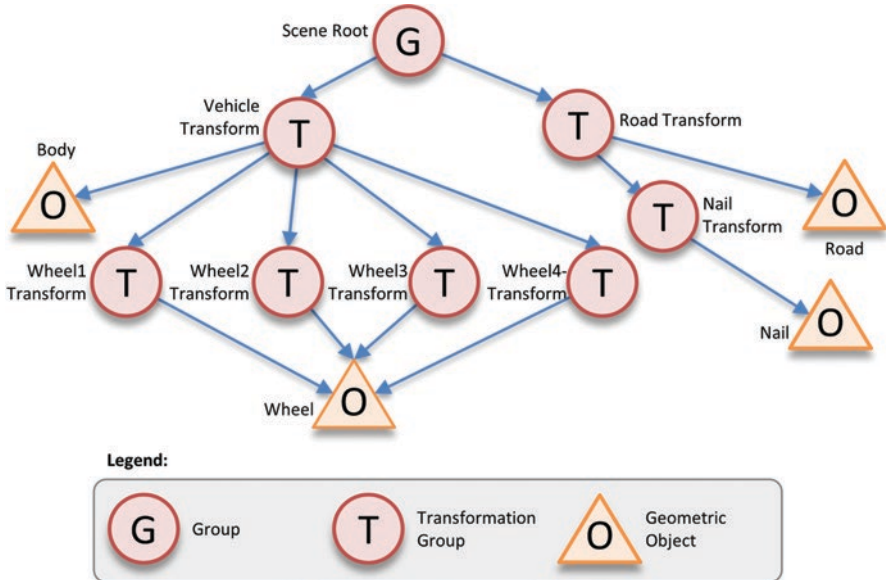


Fig. 3.2 Example of a *scene graph*. The scene consists of a vehicle with four wheels and a road with a nail on it. The 3D object for the wheel only has to be loaded into memory once, but is reused several times

special elaboration. They define a *local coordinate system* for their child nodes, usually by means of a *transformation matrix* contained as an attribute of the node. The transformation defined by such a node then describes the displacement, rotation and scaling of the local coordinate system with respect to the coordinate system of the parent node. To determine the global position, orientation and scaling of an object, the path from the root of the scene graph to the object must be traversed. For all transformation nodes occurring on the path, the corresponding transformation matrices must be chained together in the order of the path by right multiplication. The resulting matrix must now be multiplied by the vertex coordinates of the object. The mathematics of calculating with transformation matrices is explained in Chap. 11. Figure 3.3 illustrates the typical node types of scene graph architectures. The meaning and usage of these and other node types will be explained in more detail at the appropriate places within this chapter. In addition to the actual geometric 3D objects, the *scene graph* usually contains other elements, such as audio sources, light sources and one or more virtual cameras (or *viewpoints*). Lens parameters such as the horizontal and vertical view angle (or *field of view*) as well as the orientation and position of a virtual camera determine the visible section of the virtual world.

The hierarchical structure of scene graphs also offers the interesting possibility of representing an object in the coordinate system of another object (the reference object). For example, the vertex coordinates of a geometric object can be transformed into the coordinate system of the virtual camera. For this purpose, a path in the scene graph must be traversed from the node of the reference object to the

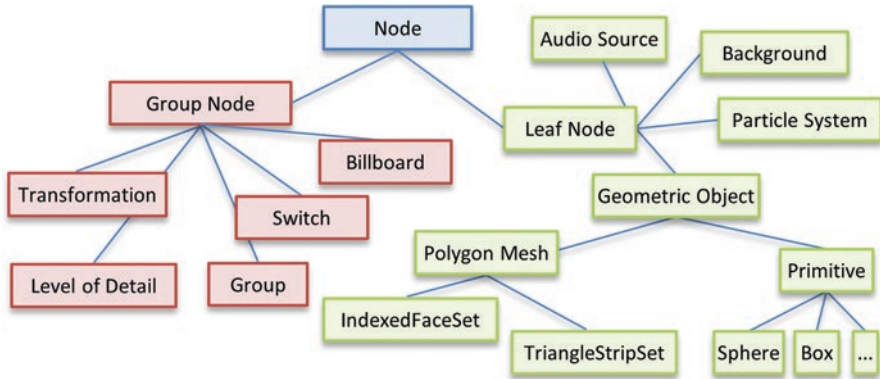


Fig. 3.3 Selection of typical node types in scene graph architectures. The leaf nodes (green) in the scene graph are usually displayed visually or audibly, group nodes (red) serve to structure the scene

respective object node. Edges can also be traversed in the reverse direction. As before, the transformation matrices occurring on the path must be multiplied. If the corresponding transformation group is reached via an edge in the reverse direction, multiplication with the *inverse matrix* must be performed.

As an example, the transformation matrix $M_{Nail \rightarrow Wheel1}$ is to be determined, which transforms the object coordinates of the first wheel of the vehicle into the coordinate system of the nail lying on the road (see Fig. 3.2). This yields the following matrix multiplication:

$$M_{Nail \rightarrow Wheel1} = M_{Nail}^{-1} \cdot M_{Street}^{-1} \cdot M_{Vehicle} \cdot M_{Wheel1}$$

A widely used, platform-independent scene graph library is the C++-based *OpenSceneGraph*, which is used, e.g., for the development of immersive VR systems. With the X3DOM framework, which is also open source, X3D-based virtual worlds can be displayed in web browsers. In game engines, scene graphs are also common. Popular examples are Unity, Unreal Engine and the open-source Godot engine. Scene graphs of game engines, however, usually have a tree structure, which is a special case of a DAG. To achieve memory-efficient reusability of 3D objects, other mechanisms such as instantiation are used here.

3.3 3D Objects

3D objects are the most important elements of virtual worlds. 3D models should define the object geometry both as precisely as possible and in a form that can be efficiently processed by a computer. Some common ways of representing objects

for VR/AR applications are presented below. A fundamental distinction exists between surface and solid models. *Surface models*, such as polygon meshes, describe surfaces that may, but are not guaranteed to, enclose a 3D volume. *Solid models*, e.g., b-reps, in contrast, always describe objects that enclose a volume.

3.3.1 Surface Models

In computer graphics, it is often sufficient to model what a 3D object looks like when seen from a certain distance, but unnecessary to model the invisible interior. *Surface models* thus capture only the outer appearance of objects but not their inside. While some surfaces are of simple, regular shape, the natural world also contains many complex, curved surfaces, such as human faces or hilly landscapes.

Polygonal Representations

Polygon-based surface representations are widely used in computer graphics as they both allow us to model arbitrary shapes and can be efficiently rendered. A disadvantage, however, is that the geometry of curved surfaces can only be reproduced approximately, since it is modeled by a mesh of planar polygons. To describe a curved surface with sufficient accuracy, a high number of polygons is therefore necessary, which in turn requires a larger amount of memory and makes rendering more complex.

On modern graphics hardware, so-called *tessellation shaders* are available which allow the creation of polygons directly on the GPU. With the help of the tessellation shaders, curved surfaces can be represented with low memory requirements and rendered efficiently. However, tessellation shaders are not yet supported by many modeling tools. Instead, when exporting 3D models with curved surfaces, polygon meshes with a high polygon count are typically generated. Thus, *memory efficiency* is an issue when choosing an appropriate data structure for polygonal representations.

Polygons

A *polygon* is a geometric shape that consists of *vertices* that are connected by *edges*. Only *planar* polygons are of interest here, i.e., polygons whose vertices lie in a plane. The simplest and necessarily planar polygon is the *triangle*. Slightly more complex is the *quadrilateral* (or *quad* in computer graphics speak). Also possible, but less common in computer graphics, are *n-gons*, i.e., polygons with n vertices. For the purpose of rendering, more complex polygons are typically split into triangles, as the graphics hardware can process triangles very efficiently. Polygons that are part of an object surface are also called *faces*. Figure 3.4 shows the conceptual relationship between objects, faces, triangles, edges, and vertices.

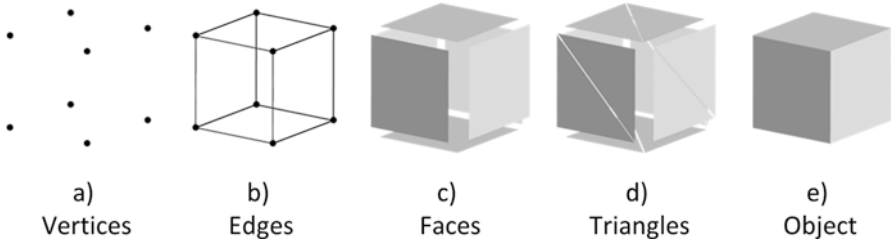


Fig. 3.4 Elements of polygonal object representations

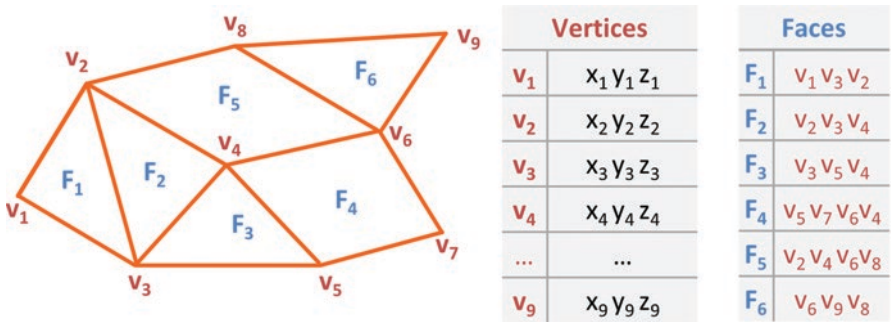


Fig. 3.5 Representation of a polygon mesh by separate lists for vertices and faces as an *indexed face set*. An indexed face set can contain different kinds of polygons, i.e., triangles, quadrilaterals or general n -gons, but each face must be planar

Polygon Meshes

A polygon mesh consists of a number of connected polygons that together describe a surface. As the vertices in a polygon mesh are shared by different faces, the *indexed face set* (or *indexed mesh*) is often a good choice as a data structure for storing the mesh. Two separate lists are defined for faces and vertices. A face is then defined by references (indices) to the vertex list (Fig. 3.5). Compared to an independent definition of the individual faces, the indexed set saves memory space. Furthermore, topology information (relationships between vertices, edges and surfaces) can be derived from the data structure.

Triangle Strips

An even more memory efficient representation of polygon meshes (or, more precisely, triangle meshes) is achieved by *triangle strips*. Here only the first triangle is defined by explicitly specifying all three vertices. Each further vertex then creates a new triangle by reusing two of the previously defined vertices (Fig. 3.6). Thus, for N triangles, only $N + 2$ vertices need to be defined instead of $3 \cdot N$ vertices. In addition to saving memory space, the fast processing of triangle strips is supported by

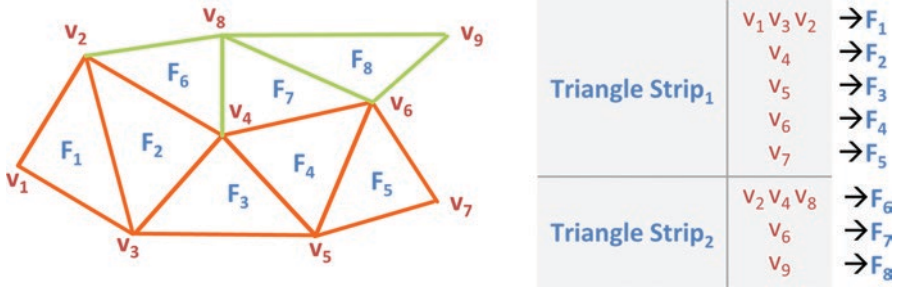


Fig. 3.6 Representation of a triangle mesh by triangle strips. The first triangle of each strip is specified by three vertices, and the following triangles by only one vertex. For example, the first triangle F_1 is specified by vertices v_1 , v_3 and v_2 . The following vertex v_4 specifies the triangle F_2 with vertices v_3 , v_2 , v_4 and the vertex v_5 specifies the triangle F_3 with vertices v_3 , v_5 , v_4 , etc.

the graphics hardware. Efficient algorithms exist for the automated conversion of other polygonal representations into triangle strips. Some scene graph architectures may provide special geometry nodes, so-called *TriangleStripSets*, which describe objects as a set of triangle strips. Also, many real-time oriented computer graphics environments, including those for VR/AR, may try to automatically optimize 3D models when loading them, e.g., by converting them to triangle strips.

For a more in-depth discussion of triangle strips and other polygonal representation, see Sect. 7.3.

3.3.2 Solid Models

A surface by itself does not have to enclose a volume, i.e., it does not necessarily have to describe a *solid*. While surface representations are often good enough for rendering purposes, other cases may require solids, e.g., in a physical simulation to calculate the volume or the center of mass of an object. Similarly, for collision detection, it can be advantageous to approximate objects by bounding volumes, i.e., simple solid bodies that fully enclose the actual objects (see also Sects. 3.4.2 and 7.2.1).

Boundary Representations (B-Reps)

A *boundary representation (b-rep)* defines a solid as a set of surfaces that define the border between the interior and the exterior of the object. A simple example is a polygon mesh that encloses a volume in a watertight manner. To execute certain algorithms efficiently, e.g., for checking the validity of the boundary representation (i.e., the ‘watertightness’ of the polygon mesh), data structures are required that provide information about the topology of the object surface (as relationships

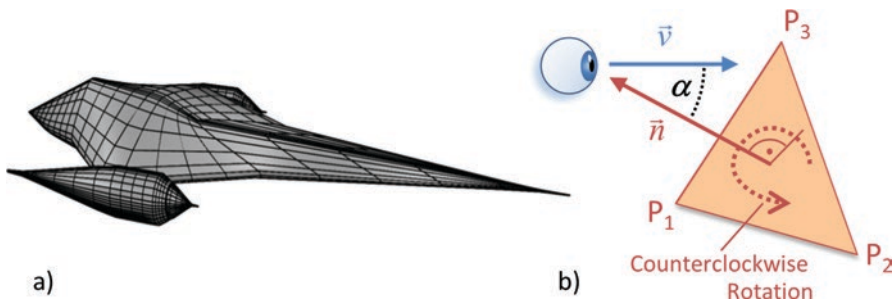


Fig. 3.7 (a) Example of a b-rep solid. (b) Determination of the front or back side of a polygon. If the polygon normal \vec{n} is approximately opposite to the viewing direction \vec{v} , or more precisely: if \vec{n} and \vec{v} form an angle between 90° und 270° , then the viewer is looking at the front of the polygon

between vertices, edges and faces). This is where data structures such as the indexed face sets discussed above come into play. In addition, it must be possible to distinguish the inside and outside (or back and front) of a boundary face. For this purpose, vertices or edges of the face can be defined in a certain order, e.g., counterclockwise. The order of the vertices determines the direction of the normal vector, which is perpendicular to the polygon front. Alternatively, the normal vector can be defined explicitly. An observer looks at the front side of a polygon when its normal vector points approximately in the direction of the observer (Fig. 3.7b). For b-reps (and solids in general) the drawing of the polygon back sides (*backfaces*) can be omitted, because they never become visible. In many scene graph libraries, the node classes for polygon meshes contain a binary attribute that indicates whether the polygon mesh models a solid.

Primitive Instancing

Primitive instancing is based, as the name already suggests, on the instantiation of so-called primitives. These are predefined solid objects, such as spheres, cylinders, capsules and tori, or sometimes more complex objects, such as gears. The properties of a primitive instance (e.g., the radius in the case of a sphere) can be set via parameters. Many scene graph libraries offer support at least for simple primitive objects like spheres, cuboids, cylinders and cones.

3.3.3 Appearance

While the surface or solid models discussed above describe the shape of 3D objects, their appearance is modeled by ‘materials’. Different types of *textures* play important roles for this.

Materials

The visual appearance of objects is mainly characterized by their *material* properties regarding reflection and transmission (transparency and translucency) of incident light. In computer graphics, a multitude of lighting models have been proposed which, with more or less computational effort, aim to approximate the underlying physical processes at least in effect. The lighting models differ, among other things, in their *material systems* used to model the appearance of the objects.

Two main approaches are currently relevant for real-time 3D applications such as VR and AR. Modern *game engines*, including Unity and the Unreal Engine discussed in Chap. 10, use *physically based rendering (PBR)* with associated material systems, which may, however, differ in detail between the various engines. PBR (e.g., Pharr et al. 2016). delivers comparatively photorealistic image quality but places higher demands on the available computing power. The older, ‘classical’ approach follows the illumination model by Phong (1975), which is also well suited for applications in web browsers and mobile devices due to the lower requirements regarding computing power. Older, but still common, file formats for 3D objects like Wavefront obj only support the well understood Phong model, so knowledge of it is still useful for application development with modern game engines.

glTF (GL Transmission Format) is a royalty-free standard for storage and network transmission of 3D models, which in particular offers support for physics-based rendering (Khronos Group 2017). Models using the metalness-roughness material system can be described in a purely declarative manner. For models that use other material systems, shader programs can be embedded.

According to Phong’s illumination model, the light reflected from a surface is composed of three components, which must be specified separately for each material: *ambient*, *diffuse* and *specular* reflection. *Ambient reflection* models the influence of directionless ambient light and provides the basic brightness of the object. *Diffuse reflection* occurs on matte surfaces and depends on the orientation of the object surface to the light source. The resulting shades contribute significantly to the spatial impression of the 3D object. *Specular reflection* creates shiny highlights on smooth surfaces. Material specifications according to the Phong model are usually supplemented by *emission* properties to model objects that themselves emit light.

Among the advantages of the Phong model are its conceptual simplicity and the low computing power requirements. An obvious disadvantage is that it cannot compete with modern PBR approaches in terms of visual realism. For example, the Phong model still produces reasonably attractive results for matte surfaces, but is less suitable for smooth surfaces, which often give the impression of plastic even when metals are to be displayed. A further disadvantage is that the specification of the different material properties requires a certain understanding of the different parameters of the model. For example, the ambient, diffuse and specular reflection

properties can be specified independently of each other, although there are physical dependencies between them. In practice, this may tempt the 3D designer to experiment with the parameter settings of the materials until a visually appealing result is achieved, but which violates basic physical laws. Such an object, with physically impossible reflective properties, may even look good under the lighting conditions of a given application, but is unlikely to be easily reusable in other applications.

PBR, which is used in modern game engines, is essentially a methodology with many variations, but not a standardized model. The various concrete forms share the common goal of achieving the most photorealistic renderings possible by implementing concepts that are comparatively close to physics. For example, PBR approaches ensure *energy conservation*, i.e., it is guaranteed that no more light is reflected than is incident on a surface. The calculation of light reflection often follows the Cook-Torrance model (Cook and Torrance 1981), which, among other things, makes a physically well-founded distinction between metals and non-metals ('dielectrics') with respect to material types. This takes into account, for example, that in the case of metals specular reflections occur in the object color, whereas in the case of non-metals specular highlights occur in the light color. Furthermore, PBR approaches conceptually regard surfaces as consisting of many micro-facets (Torrance and Sparrow 1967). The orientation of the micro-facets, in similar or varying directions across the surface, models smooth or rough surfaces. Corresponding to the multitude of concrete implementations of the PBR approach, modern game engines offer the 3D designer a number of different shader models to choose from. Most engines offer 'standard shader models', but these may differ between engines (or versions of the same engine). The shader models available in game engines typically try to provide the 3D developer with parameters that are as intuitive as possible, i.e., that 'hide' the complexity of the underlying physics of light.

A typical minimal PBR material system contains the following parameters: *albedo*, *metalness* and *roughness/smoothness*. *Albedo* is the basic color of the object. In contrast to the Phong model, no other color needs to be specified. Albedo corresponds approximately to the diffuse color of the Phong model. *Metalness* describes whether the material is a metal or not. Formally, the parameter usually allows values between 0 and 1. In practice, binary modeling with the exact values 0 or 1 or values that are close is often sufficient. The *roughness* parameter also allows values in the range between 0 and 1, although here the whole range of values can be used to define more or less smooth surfaces (Fig. 3.8). In *game engines*, these parameters can be specified for an entire object or, what is more common in practice, per pixel, using textures (see below). The material systems of game engines typically also provide options for specifying emission properties and textures such as bump, normal and ambient occlusion maps (see below).

With regard to the light transmission of objects, a rough distinction can be made between *transparency* and *translucency*. If the objects behind the considered object are still clearly visible, this is called transparency, e.g., clear glass, otherwise it is called translucence, e.g., frosted glass. Physically, the transition between transparency and translucency is continuous. In the simplest case, transparency is modeled

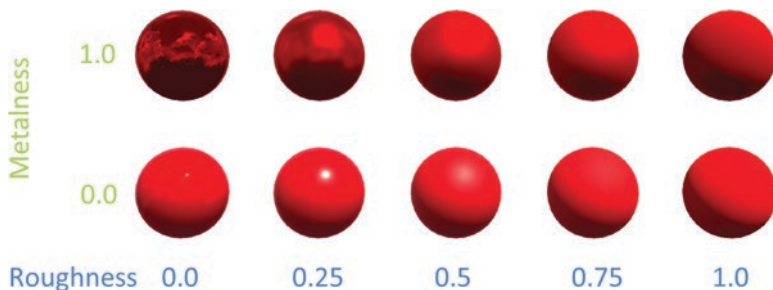


Fig. 3.8 Effects of the metalness and roughness parameters in a typical PBR material system. Mooth surfaces reflect the environment sharply, whereas on rougher surfaces the environment reflection is blurred or even imperceptible. Highlights on metals shine in the color of the surface, highlights of non-metals in the color of the light source

using an *opacity* value (opacity is the opposite of transparency). For example, the alpha value of *RGBA textures* is such an opacity value. More complex models also account for the *refraction* of light when it passes into other media, e.g., from air to water, for which physical parameters such as a refractive index or the *Fresnel reflection* ‘F0’ are required. To make it easier for 3D designers to use such models in practice, game engines typically provide specialized shader models for this purpose, as well as for related effects such as *subsurface scattering* or *clear coat* surfaces.

A more in-depth introduction to the concepts and methods of PBR is given, for example, in Pharr et al. (2016) and Akenine-Möller et al. (2018).

Textures

To represent fine-grained structures, e.g., of wood or marble, or to represent very fine details, a trick is used which can also be found in many old buildings, such as churches: the details are only painted on instead of modeling them geometrically. In computer graphics this is called *texturing*. *Textures* are raster images that are placed on the object surfaces. The exact mapping of pixels of the texture to points on the object surface is achieved by assigning normalized texture coordinates (i.e., raster image coordinates) to the vertices of the polygons representing a surface. During rendering, texture coordinates for pixels located between the vertices of a polygon are calculated by the graphics hardware by means of interpolation (Fig. 3.9a).

Even more realistic surface structures can be created using methods such as *bump mapping*, *normal mapping* or *displacement mapping*. In *bump mapping*, the pixel colors of the object surface are modified based on a grayscale image (the bump or height map). The bump map represents the ‘height profile’ of the object surface, with small (i.e., ‘dark’) values usually representing lowered areas and large (i.e., ‘light’) values representing raised areas of the object surface. A bump map is placed on the object’s surface like a conventional texture. However, the values of the bump map are not interpreted as colors, but modify the normals on the corresponding

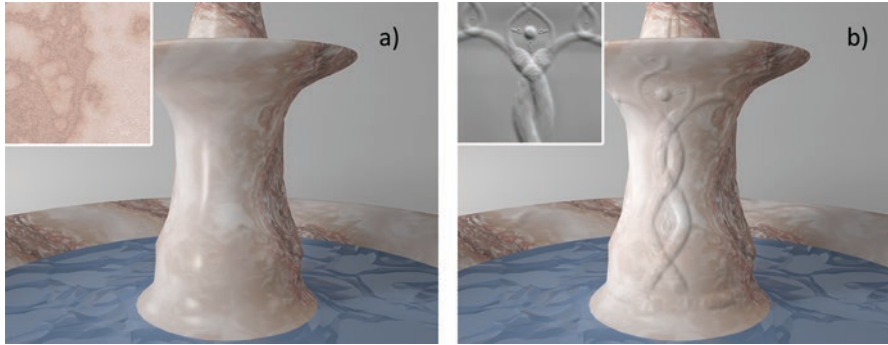


Fig. 3.9 (a) Object with image texture; section of the texture at the top left of the image, (b) object with image texture and bump map; section of the bump map at the top left of the image

points on the object surface. Thus, as normals play an important role in illumination calculations, the surface brightness can be varied pixel by pixel (Fig. 3.9b). *Normal mapping* is a variant of bump mapping with the difference that normal vectors are stored directly in a so-called normal map. Nevertheless, both bump-mapping and normal-mapping are ‘display tricks’ which create the visual effect of rough surfaces even on coarsely resolved polygon models without actually changing the object’s geometry. In contrast, *displacement mapping* indeed manipulates the geometry of the object’s surfaces. It may be necessary to refine the polygon mesh for this purpose.

Ambient occlusion maps are also quite common, modeling how much ambient light arrives at the different parts of a surface. For cracks, this value will tend to be low, but higher in exposed areas. Ambient occlusion maps are typically calculated from object geometry during the modeling stage using *texture baking* (see Sect. 3.3.4).

Shader

To enable an even more varied design of object surfaces, so-called *shaders* can be used. Shaders are small programs that are executed on the graphics hardware (*graphics processing unit, GPU*). Shaders are written in a special shader language like the OpenGL Shading Language (GLSL) or the High Level Shading Language (HLSL) by Microsoft. The most commonly used shader types are *vertex shaders*, which modify vertex information, and *fragment shaders* (often also referred to as *pixel shaders*), which allow manipulation of color values in the rasterized image of an object surface. For example, displacement mapping could be realized based on a vertex shader and bump mapping based on a fragment shader. The final color of a pixel on the screen may also result from color fragments of several objects, e.g., if a semi-transparent object is in front of a more distant object from the viewer’s point of view.

Modern GPUs contain thousands of processing units (also known as *hardware shaders*, *shader processors* or *stream processors*) that enable highly parallel execution of shader programs. GPUs are increasingly being used for tasks beyond computer graphics, including high-performance computing, crypto-mining and machine learning. Accordingly, newer GPUs also offer hardware support for such tasks, e.g., specialized deep learning processing units (e.g., Nvidia's 'tensor cores'). These new capabilities of the graphics hardware open up novel possibilities for using machine learning when rendering virtual worlds in the future. For example, an approach presented by Nvidia in 2018 uses deep neural networks to evaluate the visual quality of shadow renderings in real-time applications, so that they can be improved by means of *ray tracing* if necessary.

3.3.4 Optimization Techniques for 3D Objects

Rendering efficiency is a crucial factor for maintaining real-time performance and thus for compelling VR experiences. The rendering efficiency can be significantly improved by simplifying complex object geometries. In this section several useful optimization approaches are presented, namely simplification of polygon meshes, level of detail techniques and texture baking for replacing geometry with textures.

Simplification of Polygon Meshes

An important measure to obtain real-time 3D models is the reduction of the number of polygons. A common method for triangle meshes is the repeated application of 'edge collapse' operations (Hoppe 1996). For example, to remove vertex v_1 from the mesh, it is merged with an adjacent vertex v_2 into a single vertex v_2 . First, the two triangles that share the edge (v_1, v_2) under consideration are removed from the mesh. Then, in all triangles of the mesh that still contain v_1 , v_1 is placed by v_2 . Finally, the position of the unified vertex v_2 is adjusted, e.g., halfway between the old positions of v_1 and v_2 . This procedure effectively removes one vertex and two triangles from the mesh.

A question that arises, however, is according to which criteria the vertices to be deleted are selected by an automated procedure. Intuitively, the number of polygons in a mesh can be reduced at points where the surface is relatively 'flat'. For a triangle mesh, for example, the variance of the normals of the triangles sharing a vertex can be checked (Schroeder et al. 1992). If the variance is rather small, at least in the local neighborhood of the vertex, the surface is 'flat' and the vertex can be deleted. Depending on the choice of threshold value for the variance, the triangle reduction can be stronger or weaker.

Level-of-Detail Techniques

With increasing distance of a 3D object to the viewer, less and less detail is perceptible. This fact can be used to optimize rendering efficiency if a 3D object is stored in several variants of different *level of detail (LOD)*. Object variants with different levels of detail can be created, for example, by gradually simplifying a polygon mesh as described above, or by lowering the resolution of textures. Also, the two techniques addressed in the following, i.e., texture baking and billboarding, can be used to generate object variants at lower levels of detail.

At runtime, a suitable level of detail is selected by the VR system depending on the distance to the viewer. For example, if the object is further away, a 3D model is displayed that consists of relatively few polygons or uses smaller, less detailed textures and can therefore be rendered faster. In contrast, a more detailed model is rendered at shorter distances. The distance ranges for the detail levels are usually defined per object during the modeling stage. When defining these distance ranges, care should be taken that the VR user will not notice the transitions between the detail levels. In practice, three detail levels are often sufficient.

Some scene graph architectures support this mechanism directly through a dedicated *LOD node* type (e.g., in X3D). Alternatively, customized switch nodes could be used. A *switch node* is a group node where only one of the child nodes is displayed. The child node to be displayed can be selected at runtime. To mimic the behavior of an LOD node, one could select the child node to be displayed depending on the distance to the virtual camera (see also Sect. 3.4.3 on using switch nodes to display the state changes of dynamic objects).

Some modern game engines provide even more sophisticated LOD mechanisms. Besides the use of lower-detail models, it may also be possible to vary certain properties of the rendering process, depending on the detail level. For example, the more accurate per-pixel lighting might be replaced by faster per-vertex lighting, or computationally expensive indirect lighting methods could be turned off at lower detail levels.

Texture Baking

It is often necessary to reduce the number of polygons of a high-resolution 3D object to guarantee the real-time requirements mandated by VR/AR applications. To still get the impression of a detailed representation, the technique of texture baking is commonly used. Here, the color information of the illuminated surface of a high-resolution 3D model is stored in a texture. The texture ‘baked’ in this way is then applied to the low-resolution, polygon-reduced version of the 3D model. Instead of a color texture, this technique can be used in a similar way to create a bump map or normal map for the corresponding low-resolution model from a high-resolution model (Fig. 3.10).

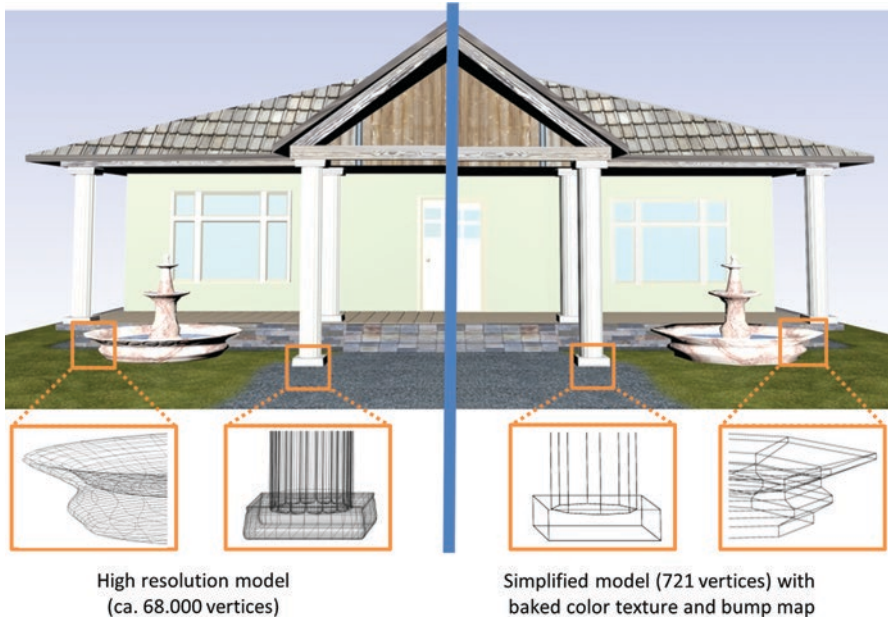


Fig. 3.10 Example of Texture Baking. Left: high resolution original scene. Right: scene with simplified geometry and baked textures for color and bump mapping

Billboards

Billboards are special transformation groups that are automatically aligned to face the observer. Billboards often contain very simple geometries, such as textured quadrilaterals. For example, it is much more efficient to render a billboard with the image of a distant tree as a texture than to render a detailed geometric tree model. Accordingly, billboards with textured quadrilaterals are often used in conjunction with LOD methods. Another important use case of billboards is the visual representation of individual particles in *particle systems* for fire, smoke, explosions etc. (see Fig. 3.15). Compared to a ‘true’ geometric model, the billboard has the disadvantage that the observer always sees the object from the same side. Therefore, it is generally recommended to use billboards only for more distant or very small objects whose details are less visible. An exception is the display of text, e.g., in textual labels or menu items, where the auto-aligning property of billboards can be exploited to ensure readability.

3.4 Animation and Object Behavior

If the properties of objects in the virtual world change over time, they are called *animated* objects. A wide variety of properties can be modified, such as position, orientation, size, color and geometry (vertex coordinates). In the following, two basic types of animation are briefly explained: keyframe and physics-based animation.

3.4.1 Keyframe Animation

A very common and simple method for animating 3D objects is *keyframe animation*. Here, the animator defines the values of a property to be animated, e.g., the position of an object, at selected time steps of an animation sequence – the so-called *keyframes*. Values at time steps between two keyframes are determined automatically by interpolation of the key values (Fig. 3.11). Different interpolation methods can be used, such as linear or cubic spline interpolation.

3.4.2 Physics-Based Animation of Rigid Bodies

It is often desirable to generate object movements in an at least approximately realistic manner. A common approach is to treat 3D objects as *rigid bodies* – which in contrast to *soft bodies* are not deformable – and to simulate their behavior based on physical laws. For this, several physical object properties must be modeled or computed. Important physical properties of an object include:

- its mass, to determine accelerations when forces or torques are applied to the object, e.g., after a collision,
- its linear velocity and (when rotating) angular velocity,

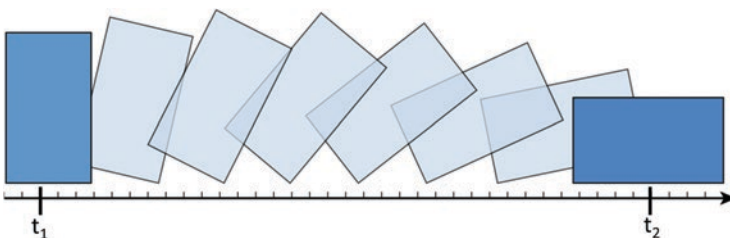


Fig. 3.11 Keyframes at time steps t_1 and t_2 , interpolated frames in between. In this example, the rotation angle of the object is animated

- material-related damping parameters to damp the movement of the object due to friction,
- elasticity values to simulate the reduction in speed due to the loss of kinetic energy after a collision.

Furthermore, the initial forces and torques acting on a body at the beginning of the simulation must be defined. Global influences, such as the gravity force permanently acting on all bodies, must also be taken into account by the simulation. For each time step, the behavior of a rigid body is calculated by the physics simulation. Its updated position and orientation are then applied to animate the 3D object.

Another important task in physics-based animation is *collision detection*. To facilitate efficient collision testing, the actual geometry of the body is usually approximated by a bounding volume (Fig. 3.12). The bounding volume is assigned to a proxy object (often called *collision proxy* in this context). The collision proxy is not rendered and thus remains invisible. Simple bounding volumes are spheres, cuboids or capsules. A more accurate approximation of an object's detail shape is its *convex hull* (the convex hull is a polygon mesh that is also a b-rep solid; see Sect. 3.3.2). Whether simpler or more accurate collision proxies are useful depends on the application. The augmentation of geometric objects with suitable collision proxies is therefore typically a task during the modeling stage of the virtual world. For a more detailed discussion of collision detection, see Sect. 7.2.

The rigid body simulation is usually performed within a *physics engine* that manages its own 'physics world' that is separate from and exists parallel to the actual scene of renderable objects – the 'geometry world'. Collision detection calculations are sometimes performed in a special *collision engine*, but when a physics engine is present the latter will usually both detect and handle collisions.

Not every geometric object of the visually displayed scene necessarily has to be represented by a corresponding physical rigid body. For example, it is not necessary to include distant background objects in the physics simulation if it is clear in advance that these objects will never collide with other objects. When augmenting the 'geometry world' with rigid bodies for the 'physics world', a suitable aggregation of single geometries is usually sensible. For example, a car may be composed of several individual geometric objects, e.g., body and four wheels (cf. Fig. 3.2), but for the special application case it may be sufficient to simulate the whole car as a single rigid body. At the end of each simulation step, the position and orientation values calculated by the physics engine are transferred to the corresponding

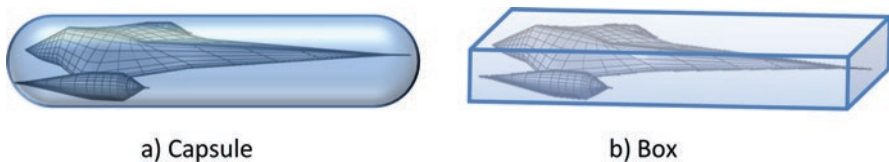


Fig. 3.12 The detail geometry of an object is approximated by two different bounding volumes, a capsule and a box. Bounding volumes are used instead of the actual geometry to efficiently detect collisions between objects

property fields of the geometric objects in the visual scene. When the scene is rendered next, the object movements become visible.

In some cases, the freedom of movement of bodies is restricted because they are connected by joints. Typical joint types are ball joints, sliders and hinges. For example, the elbow of a virtual human could be modeled in a somewhat simplified form as a hinge joint. Furthermore, the maximum opening angle of the joint in this example could be defined as approximately 180°. Such *motion constraints* must also be ensured by physics engines during the simulation.

While rigid-body dynamics only considers the motion of non-deformable objects, *soft-body dynamics* is concerned with the simulation of *deformable objects* such as clothes. Furthermore, *fluid animation* addresses fuzzy phenomena of unstable shape and undefined boundaries such as water and smoke. Soft-body and fluid simulations are supported by several modern game engines. However, they require special editing tools and effort at the modeling stage and induce relatively high computational costs at runtime.

3.4.3 Object Behavior

A high-level method of controlling animations of objects is the specification of their *behavior* when certain events are inflicted on them. For example, when a vehicle is involved in a severe collision, its state may change from ‘new’ to ‘demolished’ along with a corresponding change in its visual appearance. Similarly, the keyframe animation applied to a virtual human should change when transitioning from an idle to a walking state. In general, state changes can affect all kinds of properties of the 3D object, such as color, shape, position or orientation.

Different methods of specifying object behavior exist. A simple, yet powerful, way is the use of *state machines* (or *finite state machines*, *FSM*). State machines are formally well understood and supported by the major game engines. Further, special description languages have been proposed for behavior specification based on state machines, such as Behavior3D (Dachselt and Rukzio 2003) and SSIML/Behaviour (Vitzthum 2005). In scene graph architectures, state changes could be realized with the help of a switch node. In the vehicle example, a switch node could be defined with two child nodes: one geometry node for the vehicle before and another one for the vehicle after the collision. The task of a state machine is then to change the state of the switch node when the relevant event – here a car crash – occurs in the virtual world.

Besides instant changes of an object property, a state transition can also trigger the execution of an animation. This animation can also be repeated until the next state transition is triggered by another event. The example in Fig. 3.13 illustrates a state machine for the behavior of a door. Here, keyframe animations for opening and closing the door are executed in the corresponding states.

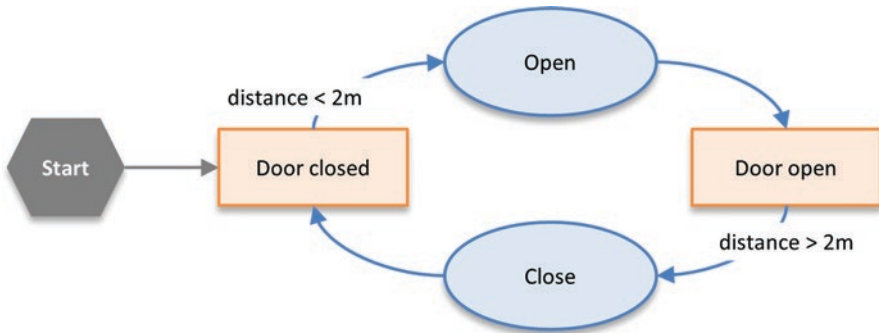


Fig. 3.13 State machine for defining the behavior of a door: if the distance between the VR user and the door is less than two meters, the door is opened, and closed again in the opposite case

Triggers for state transitions can be events of various types. In the simplest case, a state transition can occur after a defined period of time has elapsed (*timer event*). Another typical event would be the selection of an object by the user (*touch event*). Further, a *proximity event* may be triggered when the user approaches an object and the distance falls below a certain threshold. For example, a (virtual) door could be opened when the user moves close to it. A *visibility event* may be triggered when an object enters the user’s field of view, e.g., causing an animation of the object to start. Similarly, the animation of the object could be stopped when the user no longer sees it to save computational resources.

State machines are conceptually simple and widely supported in modern game engines. When it comes to the modeling of more complex behaviors, e.g., the ‘game AI’ of non-playing characters in games, extensions or alternative means of behavior specification are also commonly used. These include *hierarchical finite state machines*, *decision trees* and *behavior trees* (see Colledanchise and Ögren 2018).

3.4.4 Behavior and Animation in Scene Graphs

To implement animations and behavior, the scene graph must be dynamically updated in each frame before rendering. In addition to the obvious option of modifying the scene ‘from the outside’, e.g., by using an external physics engine (Sect. 3.4.2) or other procedures to simulate object behavior (Sect. 3.4.3), some scene graph architectures provide native support for keyframe animations (Sect. 3.4.1) by means of special node types. For example, X3D features nodes that generate certain events (e.g., *proximity sensors*, *touch sensors*) and keyframe animations in conjunction with timers and interpolation nodes. To update the scene graph before

displaying it, all new or not yet handled events must be evaluated, interpolation values must be calculated and animation-related actions (e.g., updating the position of an object or playing a sound) must be executed. This programming model of X3D and some other scene graph architectures allows elegant specifications of simple animations and behavior. On the other hand, the propagation of the relevant events through the scene graph – in the general case involving a multitude of nodes – induces relatively high runtime costs, which is why this is not done in performance-optimized scene graph architectures.

3.5 Light, Sound, Background

This section gives a brief overview of various further objects that are typically part of virtual worlds: light sources, sound sources and background objects. Due to the common use of these objects, scene graph architectures typically provide special nodes to integrate them into the scene.

3.5.1 Light Sources

Rendering of virtual worlds is based on calculations of how much incident light is reflected back from the surfaces of the 3D objects. Without *lighting*, all objects would appear pitch black. Virtual worlds thus should also include at least one but usually several *light sources*, in addition to the 3D objects. In computer graphics, typically, a distinction is made between directional light, point light and spot light sources. *Directional light* models a very far away or even infinitely distant light source (such as the Sun), whose rays arrive in the virtual world in parallel directions. Similar to a light bulb, a *point light* source emits light spherically in all directions (point light is sometimes called *omni light*, as it is omnidirectional). A *spot light* source produces a light cone, just like a flashlight. For all types of light sources, the light color and intensity can be defined. In the case of point and spot lights, the light intensity also decreases with increasing distance from the light source (*light attenuation*). For directional light, in contrast, distance-dependent attenuation does not make sense, as the distance to the light source is not defined (or infinitely large). In the real world, more distant light bulbs or flashlights cover a smaller area in the observer's field of view than closer ones do. This could be modeled as light attenuation that is proportional to the inverse square of the distance. In computer graphics practice, light attenuation is, however, often modeled with a less steep falloff so that a light source casts its light in a wider area, e.g., as proportional to the inverse of the distance to the light source. Generally, it is possible to define and tweak an attenuation function according to the application's needs, for example, to also account for dust or other particles in the air (*atmospheric attenuation*). In the case of spot lights, additionally, the light intensity decreases not only with increasing distance but also

towards the edge of the light cone. The strength of this radial falloff can be adjusted, as can the radius of the spot light cone. Moreover, often a radius of influence can be defined around a point or spot light source. Only objects that lie within the sphere defined by the radius of influence are illuminated by the light source.

In some VR/AR environments, *area lights* are offered as a further type of light source. Area lights emit light from a 2D rectangular area, in one direction only. Compared with point and spot lights, area lights produce much more realistic shadows, for example. However, the computational costs are also considerably higher for area lights.

In real-time applications such as VR and AR, for efficiency reasons often only *local illumination models* are implemented that only account for direct light paths between light sources and 3D objects. However, real-world lighting is much more complex. Let us take the example of a street canyon in a city center with many high-rise buildings. Even in the early afternoon, no direct sunlight arrives at street level, which is in the shadow of the tall buildings. Nevertheless, it is not completely dark there either, as the Sun's rays are reflected by the buildings' facades and – possibly after several reflections from facade to facade – arrive at the bottom of the street. In real-time applications, this *indirect lighting* is usually not simulated but instead accounted for by the simplified concept of a global *ambient light*. Ambient light is classically assumed to be directionless, equally strong throughout the whole scene and defined just once for the entire virtual scene. A variant makes use of a textured sky box (see Sect. 3.5.3) that acts as the source of ambient light that is now directional. Another extension is *ambient occlusion* techniques (see Sect. 3.3.3 on textures) that locally attenuate the ambient light intensity to approximate the effect of occlusions. While these variants improve the visual quality of rendered images, they still constitute a drastic simplification of real-world light propagation.

Global illumination models, such as raytracing, pathtracing or radiosity, in contrast, also account for light reflections over several surfaces (indirect illumination). However, global illumination models are not yet computable in real time for fairly complex virtual worlds. Some game engines offer *precomputed global illumination* that is used to improve the illumination of static objects. For this, a global illumination method, e.g., pathtracing, is applied at the end of the virtual world's modeling stage and the (direct and indirect) light arriving at a surface is stored in a special texture called a *lightmap*. This trick of *lightmap baking*, however, cannot be applied to dynamic, animated objects because, e.g., their positions during gameplay are not known at the time of the precomputation.

To apply indirect lighting to moving objects, several game engines offer the possibility to distribute *light probes* throughout the virtual world. Light probes capture and 'bake' the lighting conditions at modeling time at selected locations in the virtual world. In the example of the street canyon, light probes could be positioned in open space along the street, even at different heights. At runtime, indirect lighting can then be applied to moving objects by interpolation between nearby light probes. Of course, light probes can only provide a coarse approximation of true real-time global illumination, since they 'bake' indirect lighting at only one point in time and only a few points in space. Some game engines may even provide an option to

periodically update the light probes every few frames. Calculating global illumination at runtime is, however, computationally expensive and may slow down overall game play.

A special light source in many VR applications is the *headlight*, which moves along with the viewer, similar to a real headlight attached to a person's head. The headlight is typically realized as a directional light source whose direction is aligned with the viewing direction. Through this, the objects in the observer's field of view are well lit, even if they are insufficiently illuminated by other light sources. A possible disadvantage of using the headlight is that it changes the lighting conditions of a virtual world with carefully modeled light sources. In such cases the headlight should be explicitly switched off.

3.5.2 Sound

Besides light sources, *audio sources* can also be part of the virtual world. These can be integrated into the world just like other objects. In scene graph systems this integration is accomplished in the form of audio nodes. However, the extent and type of sound support differs from system to system. Typical types of audio sources in virtual worlds are presented below. For an overview of audio output devices, see Sect. 5.5.

When adding an audio source to a scene, one first has to specify the sound emanating from the source (based on an audio clip or an audio stream) and whether the sound is played only once or repeated in a loop. Probably the simplest audio source type is the *background sound* (e.g., birdsong) that is not bound to a defined spatial position and can be heard everywhere. In contrast, *spatial audio sources* have a defined position in the 3D world. These include point sources that can be heard within a certain radius, similar to point light sources that emit light in all directions. Similar to spot light sources, there can also be audio sources that emit sound waves within a conical volume. Since a purely conical emission hardly ever occurs in reality, it is recommended to combine a sound cone with a point source to model a more realistic sound propagation.

The volume of most audio sources decreases with increasing distance from the listener. This *acoustic attenuation* can be modeled approximately, for example, by a piecewise linear, monotonically decreasing function. Background sound is an exception, as the position of the audio source is undefined and therefore no distance to the listener can be calculated.

In the real world, *binaural hearing* enables a spatial perception of sound and the localization of sound sources. In VR, this can be an important navigation aid and generally improves the feeling of immersion. The ear that is closer to the sound source hears the sound signal a little bit earlier than the other ear. Moreover, the sound signal is slightly attenuated by the head, so that the sound level between the two ears also varies slightly. This situation can be reproduced by using two (stereo) or more output channels. The sound is played with slightly different delays for each

channel, possibly also with slight differences in the sound volume. As alternatives to multichannel sound processes that work with a fixed number of channels or loudspeakers, methods such as *Ambisonics* (Gerzon 1985) and *wave field synthesis* (Berkhout 1988) do not assume a fixed number of loudspeakers. For example, Ambisonics calculates the audio signals for the individual loudspeakers based on sound property values at the respective loudspeaker positions.

A physically exact real-time calculation of sound absorption, reflection and diffraction through arbitrary obstacles – similar to the reflection and refraction of light rays – requires very high computing performance and is therefore not supported by game engines and scene graph systems. However, both modern game engines and some scene graph systems – the latter using additional libraries that use low-level programming interfaces for real-time 3D audio such as FMOD or OpenAL – offer various advanced audio effects. These include reverb and echo, simulation of the change in the sound signal caused by obstacles between the sound source and the listener, and simulation of the *Doppler effect*. The Doppler effect increases the sound frequency (pitch) as the sound source, e.g., a fast-moving ambulance, moves towards the listener and decreases the pitch as the distance increases.

3.5.3 Backgrounds

In addition to the actual objects in the scene, the scene background, such as the sky, must also be displayed. In the simplest case, a static image can be used for this. Another option is to use a three-dimensional volume, such as a large sphere or box, whose inner surface is textured with the background graphics. This volume is usually modeled large enough so that it contains all (other) objects of the virtual world. The center of this volume is always at the current camera viewpoint. Thus, while different parts of the background volume might become visible by camera rotations, camera movements will not change the distance to the volume's surface. By rotating the *sky sphere* or *sky box*, effects like passing clouds can be simulated. In modern game engines, backgrounds typically also contribute to the illumination of the scene. Thus, for example, objects with smooth surfaces could show reflections of clouds.

3.6 Special Purpose Systems

Rounding off this chapter on virtual worlds, this section discusses special 3D objects that make virtual worlds more interesting, but whose modeling and animation pose distinct challenges, such as *virtual humans*, *particle systems*, *terrains* and *vegetation*, e.g., trees. These are often managed within special purpose systems of game engines, scene-graph systems and 3D modeling tools. The presentation of the

individual topics has an overview-like character while providing references to further literature.

3.6.1 *Virtual Humans*

Virtual worlds are often populated with *virtual humans* (or *virtual characters*). The function of these virtual humans can vary greatly depending on the application area of the respective virtual worlds. In game-oriented scenarios, virtual humans act as autonomous opponents or fellows (*non-player characters*, *NPCs*). In multi-player games and social virtual worlds, *avatars* serve as virtual representatives of the various participants. In virtual prototyping, virtual humans are used in ergonomics studies. Other application areas include training scenarios, architectural applications and the virtual reconstruction of historical environments. The following presentation focuses on the basic procedures for computer graphics modeling and animation of virtual humans in today's game engines and VR environments.

An *avatar* is a virtual figure that acts as representative or proxy of a VR user in a virtual world. Avatars often, but not necessarily, have a human-like appearance. Avatars are distinguished from *non-playing characters* (*NPCs*) or *bots* whose behavior is generated by control programs of the game engine or VR environment.

A simple method to model virtual humans is to represent the different body parts, such as the upper body, upper and lower arms, hands, head and legs, by separate, hierarchically structured 3D objects. Since this simple modeling often does not appear very realistic (e.g., unnatural gaps at the joints typically occur when animating such models), another method, *skeleton-based animation*, has become established, which distinguishes between the underlying skeleton structure (*'rig'*) and a deformable surface model (*'skin'*). During animation, the surface model is automatically deformed according to the respective skeleton pose. A prerequisite for this is that the vertices of the skin have been coupled to suitable bones of the skeleton in a prior modeling step called *'skinning'*. The even earlier process of setting up a suitable skeleton structure for a given surface model is called *'rigging'*. Figure 3.14 illustrates the principle of skeleton-based animation.

The skeleton structure defines the hierarchical structure of abstract bones of the virtual human model. The individual bones, e.g., thigh, lower leg and foot, are connected by joints. Thanks to the hierarchical skeleton structure, a rotation of the knee joint not only affects the lower leg, but also the position of the foot. The facial expression of virtual humans can be animated by defining suitable *'facial bones'*. Compared to the skeletons of natural humans, the skeletons of virtual humans are usually greatly simplified. There are different conventions concerning the number,

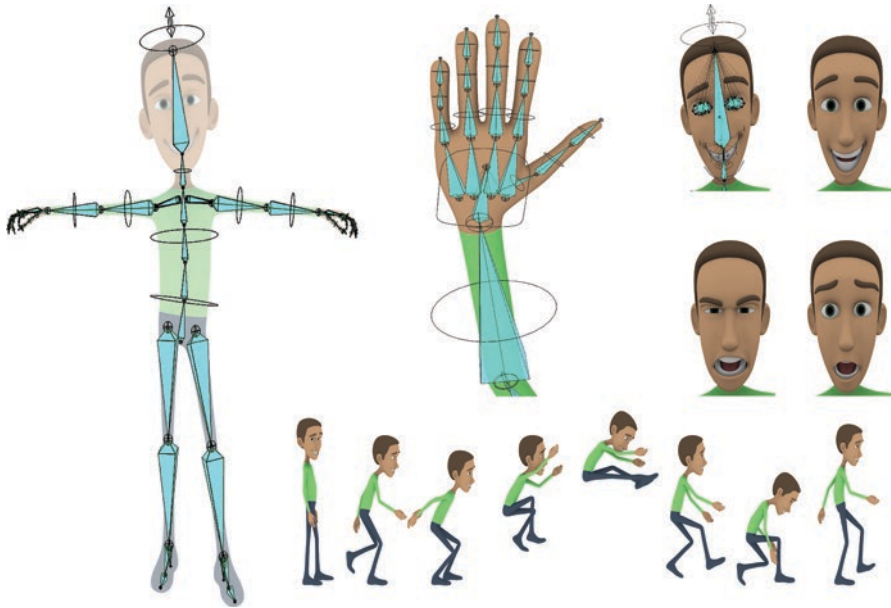


Fig. 3.14 Modeling and animation of virtual humans: by moving the skeleton bones, animations of bodies and facial expressions can be created

naming and hierarchical structure of the bones. An open standard is H-ANIM of the Humanoid Animation Working Group of the Web 3D Consortium (2019). In commercial tools such as the Character Studio of the modeling tool 3DS MAX different conventions may be used.

The animation of virtual humans is often based on the combination of different individual methods. Basic animations such as those for walking or running are typically created by means of *motion capture*. Motion capture data for typical basic animations can be found, e.g., on the internet or are supplied with 3D modeling tools. Goal-oriented animations, such as looking at a moving object, grasping an object or placing the feet when climbing stairs, however, must be calculated at runtime. *Inverse kinematics* algorithms can be used to compute skeletal postures such that the extremities (i.e., hands, feet, head) are placed at the intended target position (and in the correct orientation). Finally, virtual humans should be able to react to events in the virtual world or user interactions in a manner that is appropriate for the current situation. This is achieved by more or less complex control programs (called ‘*Game AI*’ in computer games; in the simplest case realized by means of state machines as described above in Sect. 3.4.3), which, among other things, choose between available basic animations and apply inverse kinematics procedures as demanded by the current situation.

The generation of realistic or believable behavior of virtual humans generally places many demands on the modeling and simulation of human abilities regarding perception, planning and action. These topics are far from being fully understood in

research. Research topics concern, for example, abilities to understand and generate natural language, including abilities for non-verbal communication, emotion, and personality. A comprehensive overview of the research area of virtual humans is given, for example, in Magnenat-Thalmann and Thalmann (2006).

3.6.2 Particle Systems

Particle systems enable the modeling of special effects such as fire, smoke, explosions, water drops or snowflakes in virtual worlds (Reeves 1983). In contrast to the 3D objects considered so far, which represent bodies with a firmly defined boundary, phenomena of fuzzy, continuously changing shapes can be represented. Accordingly, the underlying concepts for modeling and animating particle systems differ fundamentally from the geometry-based representations of solid bodies. Scene graph-based systems for modeling virtual worlds typically provide a special node type for particle systems. Figure 3.15 shows several visual effects that can be accomplished with particle systems.

A particle system consists of a multitude of individual particles: in real-time VR applications, for example, several hundreds or thousands. During the simulation of a particle system, each particle is understood as a point mass (i.e., a particle has no spatial extension but non-zero mass) whose position in 3D space is updated in each time step based on simple physical simulations of the forces acting on the particle. At each time step:



Fig. 3.15 Examples of particle systems. The animation of smoke and fire is accomplished using the physical simulations and particle-age dependent colorizations outlined in the text. Grass can be generated by a variant in which individual blades of grass are simulated by a fixed number of connected particles. The gushing water on the right is modeled with a hybrid approach consisting of a deformable geometry for the main water gush and a particle system for the water droplets splashing away

- new particles are inserted into the particle system via a so-called ‘emitter’,
- old particles are removed from the particle system if their lifetime has expired or if they leave a predefined area,
- for each particle, the forces acting on the particle (e.g., gravity, wind, damping) are used to update the position and velocity of the particle, and
- visualization attributes such as color and texture are updated for each particle and the particles are visually displayed.

There are various types of emitters that differ in the initial positions of the ejected particles. For example, depending on the type of emitter, all new particles may be ejected from a single point, along a line segment, 2D shapes such as circles or polygons, or 3D volumes such as cuboids. Emitters can also differ in their ejection direction, i.e., whether particles are ejected in all directions or only within predefined direction ranges. An essential feature of emitters is that all the parameters, such as number, initial position, and ejection velocity (i.e., direction and magnitude of velocity), of the newly generated particles are randomly varied within predefined ranges to achieve the desired irregular, fuzzy appearance of the simulated phenomena.

In each simulation time step, all the forces acting on a particle are calculated and accumulated. Typical forces considered are gravity, global wind fields or damping (a particle becomes slower with time). In some cases, spring forces are also considered. For example, in clothing simulations or in the modeling of strand-like objects such as hair and grass, springs are attached to particles that connect them with other particles. From the forces acting on the particle and its constant mass, its acceleration is calculated by simple Newtonian physics ($F = m \cdot a$). By integration over the time interval passed since the previous time step, the new velocity and position of the particle are then computed.

There are also various possibilities for the visual rendering of particle systems. In applications with strong real-time requirements, such as VR systems, particles are typically displayed as textured polygons, usually quadrilaterals, that are aligned to the viewing direction of the VR user (see Sect. 3.3.4 Billboards). The color and texture of the particles may change over time, e.g., from red-hot at emission to smoky-gray in later phases. Alternatively, to provide a better sense of the movement direction of the particles, particles may be rendered as line segments, e.g., with the current particle position as the starting point and the added velocity vector as the end point. Furthermore, particles may be rendered by more or less complex geometries, e.g., spheres, cylinders or, if a history of past particle positions is additionally stored, as line segments or ‘tubular’ extrusion geometries. *Solid particle systems* render each particle as a complex static polygonal mesh, e.g., for flying debris or shrapnel. However, the more complex the geometries used, the higher becomes the rendering effort, which may impair the real-time capability of big systems with a large number of particles.

3.6.3 Terrain

Fundamental to terrain modeling is a simple data structure, the so-called *height field*, sometimes also called the *elevation grid*. In essence, this is a two-dimensional grid where each grid point is assigned a height value. A height field in which all height values are equal would yield a completely flat landscape, for example. A realistic appearance can be achieved by texturing the height field.

To model more interesting and varied terrains with mountains, hills and valleys, suitable height values should be assigned to the elements of the height field. To avoid drastic discontinuities in the landscape, care should be taken to ensure that adjacent elements have similar height values. Since height fields often become quite large – e.g., with a dimension of 256×256 , more than 65,000 height values must be set – modeling ‘by hand’ is obviously not practical. In common 3D modeling tools, the creation of height fields is therefore supported by partially automated techniques. Here, the user defines the height values for selected areas, e.g., the highest elevations of hilly landscapes, whereupon the transitions to the surrounding terrain are automatically smoothed, e.g., by means of a Gaussian filter.

For the creation of fissured landscapes such as rock formations, which have a fractal structure, even more automated *procedural modeling* techniques are used. A simple algorithm is the *midpoint displacement* method, which is illustrated in Fig. 3.16. Starting from the height values at the four corners of the height field, first height values are calculated for the points in the middle between the corners. As shown in Fig. 3.16 (left), exactly five midpoints are considered. In a first step, the height values of the five midpoints are calculated as the mean value of the neighboring corner points. In a second step, the new height values are slightly varied by adding a random value. The addition of this random value is crucial for the generation of fissured structures, since otherwise only linear interpolation would be performed. The midpoints, for each of which new height values were just calculated, define a subdivision of the entire height field into four sectors. In the following iteration of the algorithm, these four sectors are processed (recursively) by assigning new height values to the midpoints in the four sectors. The recursive subdivision – each sector is split into four smaller subsectors – is repeated until all elements of the height field have been assigned new height values. The basic midpoint displacement algorithm can be modified in various ways to further increase the realism of the generated terrain shapes. For example, in addition to the recursive subdivision into four squares, the *diamond square algorithm* also considers diamonds rotated by 45° in intermediate steps (Fournier et al. 1982).

An optimization technique for very large areas is the spatial subdivision into so-called tiles (*tiling*). When the user moves through the terrain, only a small section of the terrain, one or a few tiles, needs to be loaded into memory.

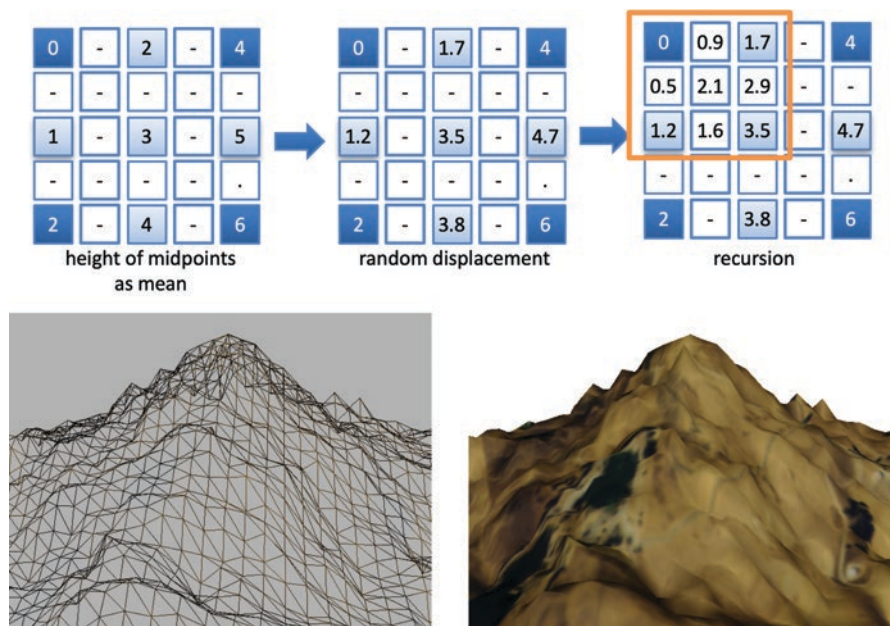


Fig. 3.16 Procedural generation of terrain using the midpoint displacement method. Top: Starting from the four corners of the height field, height values for the five midpoints are calculated by averaging over the height values at the corners and then adding a random displacement. Subsequently, the height values for four subsectors (upper left subsector highlighted) are recursively calculated using the same procedure. Bottom: Wireframe and textured rendering of a larger height field

3.6.4 Vegetation

Trees and other plants occur in nature in very complex, *fractal* shapes. In computer graphics, similar to rugged terrain, they are typically created using *procedural modeling* techniques. A comprehensive overview of common generation methods is given in Deussen and Lintermann (2005). The following example for the procedural modeling of a tree is based on the method of Weber and Penn (1995). In the first stage, the wooden parts, i.e., everything but the leaves, are generated. In the example, a three-level branching structure is assumed consisting of a trunk, the branches and the twigs. A configurable number of branches is randomly attached to the trunk, then several twigs are attached to each branch. The trunk, branches and twigs are each defined by line segments, which can later be wrapped by extrusion geometries in the final 3D model (Fig. 3.17a, b). In the second stage, the leaves are added to the branches (Fig. 3.17c). Instead of modeling the individual leaves geometrically as polygon meshes, which would result in an excessive number of polygons, the leaves are represented by highly simplified geometries, e.g., rectangular polygons (quadrilaterals) which are rendered with semi-transparent leaf textures mapped to them (Fig. 3.17d, e).

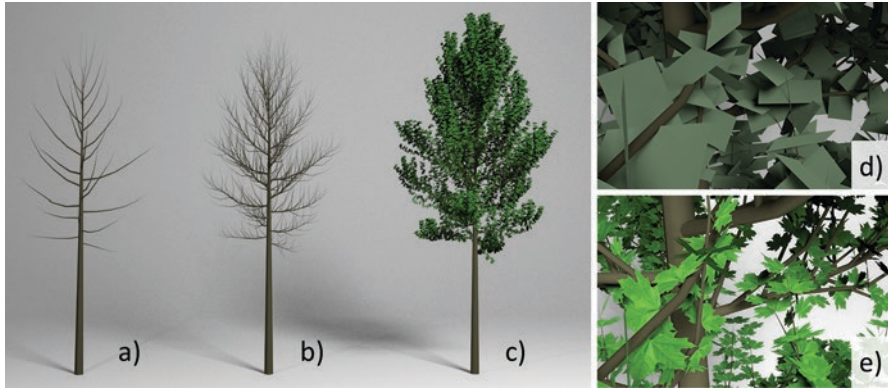


Fig. 3.17 Procedural generation of a tree. (a) Trunk and branches. (b) Trunk, branches and twigs (c) with leaves. (d) Leaves are modeled as quadrilaterals (e) with semi-transparent textures

In practice, the *procedural modeling* of trees and other plants is typically done within specialized modeling tools, some of which are also integrated into common general purpose 3D modeling tools. The procedurally generated trees can be exported as (textured) polygon meshes and stored in common 3D file formats. Since the complex structure of the trees typically results in a large number of polygons, however, not too many trees should be displayed in full resolution with regard to the real-time capability of the VR system. An often-used optimization technique is to represent more distant trees, which occupy only a few pixels on the screen, simply as semi-transparent textured billboarded quadrilaterals (see Sect. 3.3.4).

In addition to trees, other forms of vegetation can also be created using similar procedures to those described above. Bushes can usually be created by suitable parameterizations of the tree generators. For the automatic generation of ivy and similar climbing plants, the contact with surrounding objects such as house walls or columns is also taken into account. Grass, for example, can be realized as a variant of particle systems, where each blade of grass is made of several particles connected by line segments (Reeves and Blau 1985). Some *game engines* support the simulation of dynamic behavior of trees, grass and other plants under the influence of wind.

3.7 Summary and Questions

On the one hand, virtual worlds should often appear as realistic as possible, but on the other they are subject to strict real-time requirements. A main concern of this chapter was to show how virtual worlds can be optimized with respect to real-time aspects, both by clever modeling techniques and by the use of memory-efficient data structures. A general idea for increasing the rendering efficiency is to reduce the number of polygons and other visual details of the 3D objects in the virtual world. For this purpose, game engines and other scene graph architectures provide

a number of optimization options. For example, the hierarchical structure of a *directed acyclic graph (DAG)* used in many *scene graphs* allows the reuse of geometries that therefore only need to be loaded into memory once. The conversion of polygon meshes to *triangle strips*, which in scene graph systems can also be done automatically when loading the objects, significantly reduces the number of vertices per triangle compared to other polygon mesh representations. *Level of detail* techniques reduce the detail with which distant objects are rendered: an object is represented by multiple 3D models with different resolutions of geometry and textures, from which an appropriate one is automatically selected for rendering depending on the distance to the viewer. *Bump mapping* and *texture baking* are useful techniques for reducing the number of polygons in the modeling stage. Besides supporting the efficient rendering of virtual worlds, scene graphs also provide mechanisms for animation, simulation and user interaction with 3D objects. Supplementing the modeling of 3D objects ‘by hand’, *procedural modeling* methods are commonly used for the generation of complex, natural phenomena, e.g., fire and smoke (using particle systems) and rugged terrains as well as trees and other kinds of vegetation.

Check your understanding of the chapter by answering the following questions:

- What is the main purpose of a scene graph?
- Name five node types that typically appear in a scene graph.
- A person is standing on a tower and watches a moving car with a telescope. Sketch a scene graph that reflects this situation. Which transformation maps the vertex coordinates of the car into the coordinate system of the telescope?
- A triangle mesh consists of 15 triangles. By how many vertices are the triangles described if the mesh can be represented by a single triangle strip?
- Explain the basic principles of physics-based rendering (PBR) and the Phong illumination model! Why is the latter often called an ‘empirical’ model?
- Explain the difference between a color texture and a bump map. What is the difference between a bump map and a normal map?
- What types of light sources exist, and how do they differ from each other?
- What does LOD stand for? What is it used for and how?
- The behavior of a car driving autonomously over an (infinite) plane is to be modeled. The car should try to avoid collision with obstacles in front of it, if possible, by changing its direction (the direction of avoidance does not matter). If a collision with an obstacle nevertheless occurs, the car stops. From the initial state, the car should switch directly to the moving state. Sketch a simple state machine to model this behavior.
- A common method of animating virtual humans is to play back motion capture data. These define, for each time step or only for single keyframes, the virtual human’s pose (i.e., all joint angle values). How can a smooth transition between two subsequent animations, e.g., from walking to running, be achieved? How must a pre-recorded jump animation be modified so that the virtual human can land on platforms of different heights?

Recommended Reading

- Akenine-Möller T, Haines E, Hoffman N, Pesce A, Iwanicki M, Hillaire S (2018) *Real-Time Rendering*, 4th edn. Taylor & Francis – *Textbook on advanced topics in computer graphics, providing a comprehensive overview of techniques for real-time rendering of 3D objects.*
- Millington I, Funge J (2019) *Artificial Intelligence for Games*, 3rd edn, Morgan Kaufman, San Francisco – *The book provides a comprehensive overview of ‘Game AI’ techniques that are suitable for planning and controlling intelligent behavior of virtual humans, for example.*

References

- Akenine-Möller T, Haines E, Hoffman N, Pesce A, Iwanicki M, Hillaire S (2018) Real-time rendering, 4th edn. Taylor & Francis
- Berkhout AJ (1988) A holographic approach to acoustic control. *J Audio Eng Soc* 36(12):977–995
- Colledanchise M, Ögren P (2018) Behavior trees in robotics and AI: an introduction. CRC Press, Boca Raton
- Cook R, Torrance K (1981) A reflectance model for computer graphics. *Computer Graphics* 15(3):301–316
- Dachselt R, Rukzio E (2003) Behavior 3D: an XML-based framework for 3D graphics behavior. In: Proceedings of eighth international conference on 3D web technology (Web3D ’03). ACM, pp 101–112
- Deussen O, Lintermann B (2005) Digital design of nature: computer generated plants and organics. Springer Verlag, Berlin Heidelberg
- Fournier A, Fussell D, Carpenter L (1982) Computer rendering of stochastic models. *Commun ACM* 25(6):371–384
- Gerzon MA (1985) Ambisonics in multichannel broadcasting and video. *J Audio Eng Soc* 33(11):859–871
- Hartley J, Zisserman A (2004) Multiple view geometry in computer vision. Cambridge University Press, Cambridge
- Hoppe H (1996) Progressive meshes. In: Proceedings of 23rd conference on computer graphics and interactive techniques – SIGGRAPH ’96. ACM, pp 99–108
- Khronos Group (2017) glTF specification, 2.0. <https://github.com/KhronosGroup/glTF/blob/master/specification/2.0>. Accessed 6 Feb 2021
- Magenat-Thalmann N, Thalmann D (2006) An overview of virtual humans. In: Magnenat-Thalmann N, Thalmann D (eds) Handbook of virtual humans. Wiley, Chichester
- Pharr M, Jakob W, Humphreys G (2016) Physically based rendering: from theory to implementation, 3rd edn. Morgan Kaufmann, Burlington
- Phong BT (1975) Illumination for computer generated pictures. *Commun ACM* 18(6):311–317
- Reeves WT (1983) Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans Graph* 2(2):91–108
- Reeves WT, Blau R (1985) Approximate and probabilistic algorithms for shading and rendering structured particle systems. In: Proceedings of SIGGRAPH ’85, pp 313–322
- Schroeder WJ, Zarge JA, Lorenson WE (1992) Decimation of triangular meshes. In: Proceedings of SIGGRAPH ’92, pp 65–70
- Torrance KE, Sparrow EM (1967) Theory of off-specular reflection from roughened surfaces. *J Opt Soc Am* 57:1105–1114

- Vitzthum A (2005) SSIML/behaviour: designing behaviour and animation of graphical objects in virtual reality and multimedia applications. In: Proceedings of seventh IEEE international symposium on multimedia (ISM 2005), pp 159–167
- Web 3D Consortium (2013) X3D standards for version V3.3. <http://www.web3d.org/standards/version/V3.3>. Accessed 6 Feb 2021
- Web 3D Consortium (2019) Humanoid animation (H-ANIM). <http://www.web3d.org/working-groups/humanoid-animation-h-anim>. Accessed 6 Feb 2021
- Weber J, Penn J (1995) Creation and rendering of realistic trees. In: Proceedings of SIGGRAPH '95, pp 119–128

Chapter 4

VR/AR Input Devices and Tracking



**Paul Grimm, Wolfgang Broll, Rigo Herold, Johannes Hummel,
and Rolf Kruse**

Abstract How do Virtual Reality (VR) and Augmented Reality (AR) systems recognize the actions of users? How does a VR or AR system know where the user is? How can a system track objects in their movement? What are proven input devices for VR and AR that increase immersion in virtual or augmented worlds? What are the technical possibilities and limitations? Based on fundamentals, which explain terms like degrees of freedom, accuracy, repetition rates, latency and calibration, methods are considered that are used for continuous tracking or monitoring of objects. Frequently used input devices are presented and discussed. Finally, examples of special methods such as finger and eye tracking are discussed.

Input devices are used to record user interactions using sensors, as well as other objects and the environment. The data obtained in this way are summarized, if necessary, semantically interpreted and forwarded to the world simulation. There is a wide range of VR/AR input devices available and a classification of these can be done in different ways. The distinction can be made based on accuracy (fine or coarse) or range (from an area that can be reached with an outstretched arm, to an area where one can walk or look around). It is also possible to distinguish between discrete input devices that generate one-time events, such as a mouse button or pinch glove (a glove with contacts on the fingertips) and continuous input devices that generate continuous streams of events (e.g., to continuously transmit the position of a moving object). The physical medium used for determination (e.g., sound waves or electromagnetic fields) can also be used for classification (see Bishop et al. 2001). In the following, the fundamentals of input devices are presented. Then, in Sect. 4.2 tracking techniques are presented in general before a more detailed

Dedicated website for additional material: vr-ar-book.org

P. Grimm (✉)

Department of Media, Darmstadt University of Applied Sciences, Darmstadt, Germany
e-mail: springer@paul-grimm.de

© The Author(s), under exclusive license to Springer Nature
Switzerland AG 2022

R. Doerner et al. (eds.), *Virtual and Augmented Reality (VR/AR)*,
https://doi.org/10.1007/978-3-030-79062-2_4

discussion of camera-based tracking approaches in Sect. 4.3. Sections 4.4 and 4.5 give examples of finger and eye tracking to show how natural user interactions can be detected using specialized input devices. Afterwards (Sect. 4.6) further input devices are presented, which are often used in VR systems. Finally, the chapter is summarized and example questions as well as literature recommendations are given.

4.1 Fundamentals of Input Devices

The interaction of a user with a VR or AR system can be manifold. In a simple case, a conscious action of the user takes place in the form of a push of a button, which is recognized as a unique event by the system in such a way that it can react to it. More difficult are more complex interactions, such as hand movements (e.g., to point at something) or to direct the gaze at something.

This section explains the foundation to describe input devices in more detail. In the case of interactions, a distinction can be made between whether the interaction should be continuous (e.g., in continuous pursuit of a finger pointing at something) or whether part of a movement should be recognized as a gesture (e.g., when pointing at an object in the virtual world to select it). In both cases, however, the system must be able to track the user, as gestures can only be extracted from recorded data in a subsequent step. It must be determined what exactly is to be tracked by the VR system. Either interaction devices, such as VR controllers or a flystick (see Fig. 4.4), or the user directly can be used. In the latter case, it must then be determined what kind of movements a VR/AR system should detect, or which parts of the body should be considered for interaction (e.g., only the hand, the arm, the head or perhaps the movement of the whole body, as shown in Fig. 4.1 as an example).

Technically speaking, continuous tracking by an input device continuously determines the position and orientation of an object (e.g., hand or head, controller). This process is called tracking. For simplification, an object is usually regarded as a so-called rigid body that cannot be deformed.

The movement of a rigid body can be broken down into a displacement (translation) in space and a rotation around three perpendicular axes. Thus, the movement of a rigid body can be specified by giving six values (three coordinates as position and three angles to describe the orientation) for each time step. These independent movement possibilities are called degrees of freedom. Generally, a system with N points has $3 \times N$ degrees of freedom (each point in space has three degrees of freedom, N points in space corresponding to $3 \times N$ degrees of freedom), which in turn are reduced by the number of constraints. In the case of rigid bodies, where all distances between points are constant, there are always six degrees of freedom left (Goldstein 1980). As an example, a cube can be used that has eight vertices and thus $3 \times 8 = 24$ degrees of freedom. If the cube is considered to be non-deformable, the constraints are that the respective distances between the eight points remain unchanged. For eight points, this means $6 + 5 + 4 + 2 + 1 = 18$ constraints (4 + 2 for the distances including the diagonals of the flat base surface, 3 + 2 for the first side surface including the diagonals, two for the next side surface and one for the last side surface).

Fig. 4.1 Recording of body movements (© ART 2013, all rights reserved)



Degrees of Freedom (DOF) are the independent movement possibilities of a physical system. A rigid body has six degrees of freedom: three each for translation and rotation.

The goal of tracking is to determine or estimate the values corresponding to these six degrees of freedom (6DOF) of the tracked objects for continuous interaction. The data acquisition is usually performed in the reference system of the respective tracking system. If several or even different systems are used, the tracking data must be transferred to a common reference system.

Starting with mechanical tracking systems (see Sect. 4.6.2), through the use of strain gauges to camera-based approaches (see Sect. 4.3), data was recorded in different ways, as was data transmission by cable or radio. Correspondingly, very different input devices are available, which have different advantages and disadvantages. Input devices can be described by the following characteristics.

Number of Degrees of Freedom Per Tracked Object

The number of specific degrees of freedom per tracked object varies depending on the input device. Usually, the determination of all six degrees of freedom by an input device is desirable. However, it also happens that only the position – equivalent to the three degrees of freedom of translation – or only the orientation – equivalent to the three degrees of freedom of rotation – is determined. Examples of the limited determination of degrees of freedom are the compass (one degree of freedom, determination of the orientation in the plane) and GPS, which, depending on the number of visible satellites, determines two to three degrees of freedom of translation. It is also possible that the accuracy of the determination of individual degrees of freedom is different (in the case of GPS, the position on the Earth's surface is recorded more accurately than the height above it).

Number of Objects Tracked Simultaneously

Depending on the application, it is important to consider how many objects are to be tracked simultaneously. In addition to tracking the user or recording the viewer's point of view, other objects (e.g., one or more input devices) often need to be tracked. For the use of several objects it is helpful if they can not only be tracked, but they can be uniquely identified by an ID. It is helpful if these IDs are retained, even if individual objects are temporarily out of monitoring.

Size of the Monitored Area or Volume

The size of the monitored area or volume varies greatly depending on the type of input device used. It must be ensured that the selected input devices offer an area that is large enough for the requirements.

Depending on the application, this can mean that it is sufficient to cover an area that can be reached with the arm or that corresponds to the movements of a head in front of the monitor. There are also applications where it is necessary to be able to walk around. The reason for the size restrictions may be that the input device is wired, has a mechanical construction or (in the case of camera-based input devices) the resolution is too low. Depending on the technology used, the shape of the monitored area may vary (e.g., similar to a circle in the case of wired technologies or similar to a truncated pyramid in the case of camera-based technologies with one camera).

Accuracy

Not only because of the physical limitations of the input devices, high accuracy is not always achievable. Sometimes it is also a question of cost. For example, in optical tracking a change of camera can increase the accuracy. However, if an expensive industrial camera is used instead of a simple webcam, the price can easily increase by a factor of 10 or more. Depending on the application, it must be considered what accuracy is necessary or what budget is available. The usual range in spatial

resolution is between millimeter accuracy (e.g., optical finger tracking) up to an inaccuracy of several meters (e.g., when using GPS). The accuracy can also vary with different types of degrees of freedom (translation or rotation), e.g., as in the case of GPS, where altitude determination is not as accurate as position determination. The accuracy can also be position-dependent: for example, the accuracy may be lower at the edge of the monitored area than at its center. During digitization, the measured values are quantized, e.g., to 8 bits or 16 bits. With regard to the measurement technology, noise (addition of an interfering signal), jitter (temporal inaccuracy of the time of measurement or of the sampling time) or interpolation errors can also be assumed to be interfering influences.

Update Rate

The update rate describes the resolution of an input device in time. The degrees of freedom are determined in discrete time steps. The number of these measurement points per second is called the update rate. Thus, monitoring the real continuous motion of an object (shown as a black line in Fig. 4.2) results in corresponding measuring points. Basically, a time-discrete signal is obtained, which will usually have errors. Figure 4.2 shows some of the possible errors.

Latency

Each input device requires a certain amount of time to react (e.g., time until the next scan, due to signal propagation times in cables or due to the processing of data by algorithms), which causes a delay. This is called latency. An example of the effect can be seen in Fig. 4.2. The significance of latency for VR systems is discussed in more detail in Sect. 7.1.

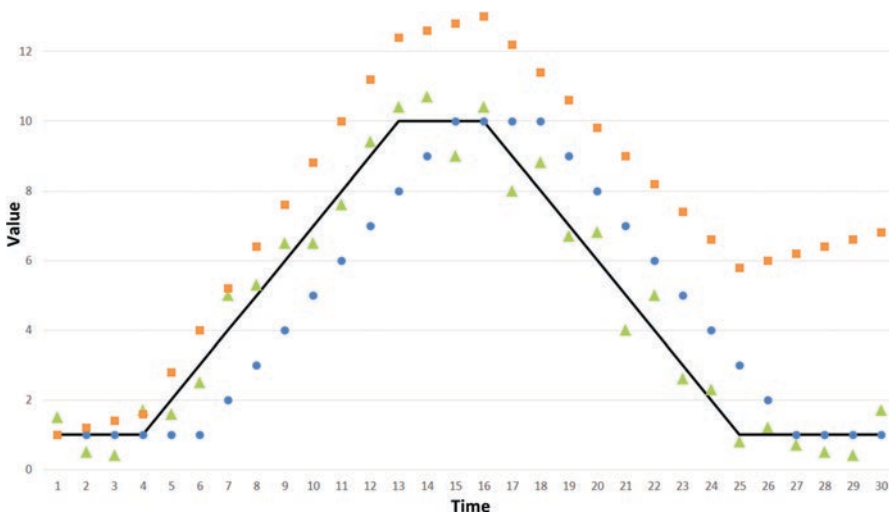


Fig. 4.2 Possible errors during data acquisition of the position of a moving object (black line): acquisition with latency (blue dots), with drift (orange squares) and with noise (green triangles), displayed over time (horizontal axis)

Drift

Errors that keep adding up can cause drift. If input devices record relative changes (e.g., change in position compared to the previous scanning or the previous measuring point), errors can increase over time. An example of drift is shown in Fig. 4.2.

Sensitivity to External Conditions

Depending on the technology used, the external conditions must be observed. Lighting or temperature can have just as much influence as the furnishing of the room in which the VR system is set up. Uniform lighting can be of great advantage, especially with optical methods, compared with hard transitions from direct sunshine to shaded areas. It would be annoying not to be able to use a tested application because the sun appeared from behind a cloud. A problem has often been reported to have arisen in trade fair construction, where before the opening usually only some working lights were used, but during trade fair operations there were often many other spotlights, which then led to disturbing influences.

With optical tracking systems it can be helpful to work in darkened rooms and to create the desired lighting situation with artificial light. It should be noted that direct light sources can interfere with camera sensors. Methods based on sound are often susceptible to different temperatures or different air pressures, as this changes the speed of sound (on which the measurement is based). Electromagnetic methods in turn react sensitively to (ferro-)magnetic materials and electromagnetic fields in the rooms (e.g., metallic table frames or the power supplies of other devices).

Calibration

Calibration is the adjustment of measured values to a given model. For both virtual reality and augmented reality, the measured values must be adjusted to the real objects used, so that the real movements that are tracked also correspond to the dimensions in the virtual world. With optical methods, this also includes the determination of imaging errors of the optics (e.g., distortions).

Usability

For the application it can be decisive to what extent a user is restricted by the input devices. For example, it may be necessary to put on glasses or shoes or hold VR controllers. It also makes a difference for the application whether the respective devices are wired or connected via radio technologies. The size of the room in which a user is allowed to interact also influences whether the user can immerse himself or herself in the application or whether he or she must constantly ensure that he or she does not exceed predetermined interaction areas. It may also be necessary that the user is always oriented towards the output device to enable good tracking. A detailed consideration of usability is given in the framework of the consideration of basics from the field of human–computer interaction in Sect. 6.1.

The obtrusiveness of an input device can be seen as a measure of the extent to which it is considered to be disruptive. For example, it makes a big difference whether a head-mounted display can be worn like sunglasses or whether it can be used like a bicycle helmet due to its weight and dimensions.

4.2 Tracking Techniques

As explained in the introduction, tracking is the continuous estimation of the position and orientation of an object. Generally, we may distinguish between systems in which the measuring sensors are located on the tracked objects themselves and determine their position and location in relation to their surroundings (inside-out tracking), and systems in which the measuring sensors are distributed in the environment and interact to measure an object from outside (outside-in tracking) (see Sect. 4.3 on camera-based tracking). The determination or estimation of the position of an object is carried out in a defined coordinate system. One possibility is the estimation in relation to individual objects. Here, the relative transformation between the user or camera coordinate system and the object coordinate system is determined for each object. Another possibility is that several objects use a common coordinate system. In this case, the transformations between the individual objects within the coordinate system must be known, and the transformation between the camera and this coordinate system is estimated. If only the position of some objects in a global coordinate system is known, while others can change their position and orientation within it, you get mixed forms of both scenarios.

In the following, different tracking techniques are presented with their advantages and disadvantages. Camera-based tracking techniques will be presented in Sect. 4.3 due to their diversity.

4.2.1 Acoustic Tracking

Acoustic-based input devices use the differences in the time of flight (TOF) or phase of sound waves. Ultrasound that is inaudible to humans (sound waves with a frequency of more than 20,000 Hz) is used. The measurement uses a transmitter and a receiver, where one of them is connected to the tracked object. This allows for the determination of the distance between them. By that, the position of an object can be limited to a spherical surface around the transmitter. By adding a second transmitter or a second receiver, the position can already be limited to a circular path (as an intersection of two spheres). Adding a third transmitter or receiver then limits the position to two points (as an intersection of three spheres or as an intersection of two circles). A plausibility check is then used to determine the actual position from these two points. A setup with one transmitter and three receivers (or three transmitters and one microphone) thus allows for the determination of all three degrees of freedom of the translation (3 DOF). If the orientation is also to be determined (6 DOF), three transmitters and three receivers must be used.

Compared to other 3D tracking systems, acoustic systems are rather cheap. A disadvantage of acoustic tracking is its sensitivity to changes in temperature or air pressure. Any change in temperature or air pressure requires a (re)calibration of the system.

4.2.2 *Magnetic Field-Based Tracking*

Magnetic fields can be used for tracking. However, a distinction must be made between artificial magnetic fields and the Earth's magnetic field. In mobile systems, so-called fluxgate magnetometers (also known as Förster probes) are usually used for electronic measurement of the Earth's magnetic field. Based on the individual sensor orientation, both the horizontal and vertical components are measured. This gives two degrees of freedom of the current position. Sensors for magnetic field measurement are disturbed easily by artificial magnetic fields in their environment. Especially indoors, electromagnetic fields (e.g., from installed cables) can falsify the recorded data to such an extent that they become useless for determining the position. In smartphones and tablets, three orthogonal magnetometers are usually combined with three linear inertial sensors and three angular rate sensors each (cf. Section 4.2.3) to compensate for measurement errors through redundancy.

For indoor systems, the use of the Earth's magnetic field is usually not possible due to disturbing influences. However, with the help of current-carrying coils, artificial magnetic fields can be created which can then be used for tracking. Coils are also used as sensors. Depending on whether a static magnetic field (direct current, DC) or a dynamic magnetic field (alternating current, AC) is used for the measurement, different measuring methods are used. With alternating magnetic fields, the magnetic field induces currents in the coils, which are used as a measure of the position and orientation in the magnetic field (or in space). In DC magnetic fields, a current flow through the receiver coils and a voltage drop can be observed perpendicular to both the direction of current flow and the magnetic field direction when the coils are moved through the magnetic field. This so-called Hall effect also allows tracking by measuring the Hall voltage. The combination of three orthogonal transmitters and three orthogonal receiving coils allows one to determine the position and orientation in space. The advantages of electromagnetic tracking systems are that the receivers are small and that they are insensitive to occlusion by the user or other non-conductive objects. The major disadvantage is that no (ferro-)magnetic materials must be used in the room (up to the use of plastic screws for fastening the sensors) and no electromagnetic fields should exist, as these interfere with the magnetic field, introducing measurement errors. Since interference influences, especially in a room with other electromagnetic components of a VR or AR system, can usually not be avoided, complex calibration procedures are necessary to compensate for disturbing interference. However, this assumes that the interference is exclusively based on static, permanently mounted objects.

4.2.3 *Inertial Tracking*

Inertial tracking is based on sensors that measure acceleration (called inertial sensors or acceleration sensors). Inertial tracking is primarily used to determine orientation. One area of application is, among others, the detection of the joint positions of a user by attaching appropriate sensors to the individual limbs.

Depending on the design, a distinction is made between linear inertial sensors, which measure the acceleration along an axis, and angular rate sensors, which measure the angular acceleration around an axis. Since the latter behave like a gyrocompass (gyroscope), they are sometimes also called gyro sensors. Together they form a so-called Inertial Navigation System (INS). Typically, three linear inertial sensors (translation sensors) and three angular rate sensors, arranged orthogonally to each other, are integrated into an inertial measurement unit (IMU). Such units often also include three magnetometers, which are also arranged orthogonally to each other (see Sect. 4.2.2).

Linear accelerometers can be used to determine the orientation, but only in the idle state. Then, the inclination to the vertical can be measured due to the direction of gravity. Since the orientation in the horizontal is perpendicular to gravity, this cannot be measured by linear inertial sensors. For input devices that can be moved freely, three orthogonal sensors are nevertheless installed so that at least two can be used for measurement at any time. However, linear inertial sensors may also be used for position determination. Based on the linear acceleration values in the three orthogonal sensors, the current speed can be estimated by integration and the change in position by a second integration. However, due to measurement inaccuracies (usually amplified by a relatively low accuracy in converting the analog measured values into digital values), drift effects often occur. This means that if, for example, a sensor is moved out of its resting state and then stopped again, the sums of the recorded acceleration values would have to add up to zero at the end, resulting also in zero velocity. However, this is usually not the case, so that the measurement results in a low residual speed even in the idle state. This leads to an increasing deviation between the measured and actual positions over time.

In the case of acceleration sensors for measuring angular velocity, the acceleration values are integrated twice analogously to obtain the angle of rotation. This also causes the problem of drift. It is therefore recommended to recalibrate in the idle state using the linear accelerometers. For the detection of rotations over all three axes, three sensors are usually installed orthogonally to each other, even with gyro sensors.

4.2.4 Laser-Based Tracking

In laser-based tracking, the tracked objects are equipped with several photosensors that detect laser beams emitted from a base by two rotating lasers. If only one base is used, the photosensors are often occluded, e.g., by the user's own body. Most systems therefore use several base stations. This also allows a larger tracking volume to be covered. For synchronization between the base stations and the objects, either additional infrared signals are used, or the sync signal is transmitted via the laser beam itself. The lasers rotate around a horizontal or vertical axis, whereby the laser beam is emitted only in one direction with a certain aperture angle (e.g., 120°). The position and orientation of the object can be calculated based on the time

difference between the detection of the laser light by the individual photosensors. At a defined rotation speed of the lasers (e.g., 1000 Hz), the position is determined by the time difference between the infrared flash, which is emitted before the start of a laser rotation, and the impact on one of the sensors. At a rotation frequency of 1000 Hz, an aperture angle of 120° and a time difference of $1/6$ ms from the infrared synchronizing flash, this results in a position at the center of the monitored space.

4.2.5 Outdoor Position Tracking

In the field of mobile outdoor applications, global satellite-based systems such as GPS, Glonass or Galileo are used for positioning. Mobile position tracking is especially relevant for AR, since VR applications are typically not used outdoors. However, in contrast to navigation applications, where satellite data can be compared with existing roads and paths, the position of an AR system is almost arbitrary. Thus, deviations of 10 m and more are not uncommon. Especially under poor reception conditions, the accuracy can be reduced even further. Global satellite-based systems usually require a view of at least four satellites to determine their position. While this is usually not a problem outdoors, reception inside buildings with conventional receivers is not suitable for AR. But even in forests and deep valleys the reception quality can be significantly impaired, so that positioning is not possible or only possible to a limited extent. A particular problem is the use in inner city areas. Due to high buildings and narrow alleys, the free view of the satellites may be so limited that proper positioning cannot always be guaranteed. Here, one also speaks of ‘urban canyons’ (see Fig. 4.3).

While conventional GPS signals are not sufficient for AR in most cases, the accuracy can be significantly increased by using differential methods. A distinction is made between Differential GPS (DGPS) and Satellite Based Augmentation System (SBAS). With DGPS, a correction signal is calculated based on a local reference receiver whose position is known. This correction (received by radio or via the Internet) is then applied to the locally received GPS signal, allowing accuracies down to a few centimeters. In SBAS, the reference system is formed by several geostationary satellites. These reference satellites each provide correction data for specific areas (WAAS in North America or EGNOS in Europe). Based on SBAS, accuracies of about one meter can be achieved. However, SBAS (in particular) in city centers is again sometimes problematic due to the often limited visibility to the south (geostationary satellites have an orbit above the equator). For outdoor AR applications, however, the use of SBAS is usually the only way to achieve an acceptable positioning accuracy. This is already sufficient for the augmentation of objects and buildings that are not in the immediate vicinity of the observer. If DGPS is used, augmentation can usually be achieved even at a short distance without any noticeable deviation from the actual position. However, the objectively perceived quality of the positioning strongly depends on whether the virtual object must fit seamlessly

Fig. 4.3 Buildings block GPS signals in so-called urban canyons



to a real object or can be positioned rather freely (for example, a virtual fountain on a real site).

In addition to DGPS and SBAS, Assisted GPS (A-GPS) and WLAN positioning are also frequently used, especially in smartphones and tablets. With A-GPS, an approximate position is determined on the basis of the current mobile radio cell (possibly refined by measuring the signal propagation times to neighboring mobile radio masts), whereas WLAN positioning uses known WLAN networks (these do not have to be open, but only uniquely identifiable). Neither method provides sufficiently accurate position data for AR. However, A-GPS can also significantly accelerate the start-up phase of an ordinary GPS receiver by transmitting satellite information (especially current orbit data and correction data). This is particularly relevant for AR applications if the users are frequently in areas where there is no satellite reception – for example in buildings.

4.3 Camera-Based Tracking

In recent years, camera-based tracking, also known as optical tracking, has become increasingly popular because it enables high accuracy and flexible use. In the field of optical tracking different techniques are used. They are based on the idea of using objects recorded in the video stream to determine the relative positioning and orientation of the objects to the camera (the so-called extrinsic camera parameters) (Hartley and Zisserman 2000).

Basically, techniques can be distinguished according to whether markers (see Fig. 4.8) are used for tracking which are easily recognizable in the recorded video stream (by their color, shape, contrast, brightness, reflective properties, etc.), or whether the method also works without markers (markerless). In the latter case, either lasers are used, or cameras capture features within the camera image (see Sect. 4.3.3). It is also possible to distinguish between methods in which the cameras are directed at the object to be monitored from the outside (outside-in), or whether the cameras are mounted to the object to be monitored and record the surroundings (inside-out). In most cases, outside-in methods combine several cameras with the aim of increasing the area of interaction or making it less susceptible to occlusion. The disadvantage of outside-in methods is that a (very) large number of cameras may be required to monitor large interaction areas and that the overall costs may rise rapidly, especially when using special cameras. The disadvantage of inside-out procedures is that the user must accept restrictions by carrying cameras around. Even though camera modules have become very small nowadays, the total package of camera and possibly battery and transmission or evaluation logic is relatively heavy. The advantage is that users are not restricted to a certain interaction space and can therefore move around more freely.

From the user's point of view, a markerless outside-in method would of course be desirable, as this is where the restrictions for the user are least severe. Users do not have to hold anything in their hands, do not need markers (e.g., on clothing) and can move freely and walk freely through the room. In practice, however, it has been shown that markerless tracking systems are more susceptible to interference (e.g., additional people in the room or changing lighting conditions) than marker-based systems, and that the accuracy of marker-based systems is often higher.

4.3.1 Marker-Based Methods

To reduce the complexity of calculations and to avoid errors in different lighting situations, optical tracking techniques often use clearly specified markers whose image can be quickly identified in the video stream via threshold filters. Basically, active and passive marker can be distinguished, depending on whether the markers passively reflect the light or they themselves actively radiate light. Figure 4.4 (top) shows an example of a six degrees of freedom controller with active markers (18



Fig. 4.4 (Top) VR controller with active marker; (bottom) cameras with infrared LEDs for illumination and flysticks with reflective markers

white LEDs arranged in a given pattern). Figure 4.16 shows a similar controller with active infrared LEDs.

When using RGB cameras, black and white markers with defined sizes are often used for this purpose. These are discussed in detail in Sect. 4.3.2. There are also different approaches with colored markers. However, due to the lighting situation and possibly also due to inferior cameras, even areas that are actually monochrome are usually no longer monochrome in the video stream, so that the susceptibility to errors increases when searching for a colored area. Better results can be achieved using color-based tracking with active markers, i.e., self-luminous markers. Electric lights (with the disadvantage of the power supply) such as the PlayStation Move controller or glow sticks (also known as bend lights, which use chemiluminescence) have proven to be very useful for this purpose.

To allow illumination of a scene without dazzling the users, infrared cameras are often used in VR. The markers used here are either passive reflectors in combination with infrared lights or active infrared LEDs such as the Nintendo Wii (see Lee 2008). Figure 4.4 (bottom) shows the infrared LEDs used for illumination. In the video stream, small very bright round areas can be seen for each marker. The

visibility of a marker in several camera views allows the three-dimensional position to be calculated.

Single markers are sufficient if tracking is only to provide the position (3 DOF). However, a rigid body (also called a target in some tracking systems) typically requires the calculation of its position and orientation. Consequently, a target is composed of several individual markers. In a calibration step, the geometric structure of the targets (e.g., the distances of the individual reflection spheres) must be communicated to the tracking system. If all targets differ in their geometric structure, identification can be made based on these characteristics. In Fig. 4.4 (right side) two input devices with targets are shown, which take over the function of a 3D joystick, and with which the user can indicate positions and orientations in 3D space (so-called flysticks).

To make the reflection of passive markers as efficient as possible, retroreflection is usually used. Retroreflection means that the beams of light are reflected specifically in the direction of the incident light and is based on two basic optical principles: in the case of reflection by triple mirrors, the mirrors are arranged with a right angle in between, as shown in Fig. 4.5 (left). When reflected by glass spheres, the spheres focus the incoming light approximately on the opposite surface of the glass sphere (see Fig. 4.5, right). A layer of microscopically small glass spheres applied to reflective material acts as a retroreflector. These foils can be produced on flexible carrier material and are therefore used to produce ball markers as shown in Fig. 4.6 and Fig. 4.7.

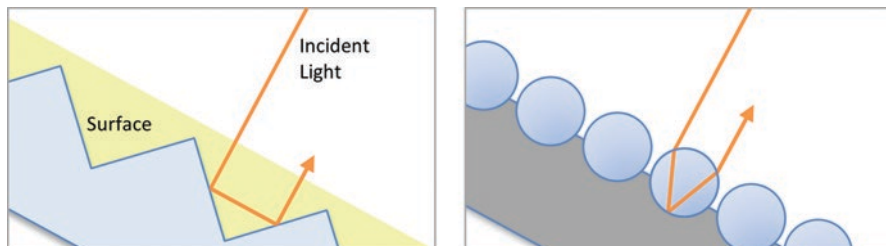


Fig. 4.5 Retroreflection of protected triple mirrors and glass spheres. (© ART 2013, all rights reserved)

Fig. 4.6 Tracking a target from two cameras

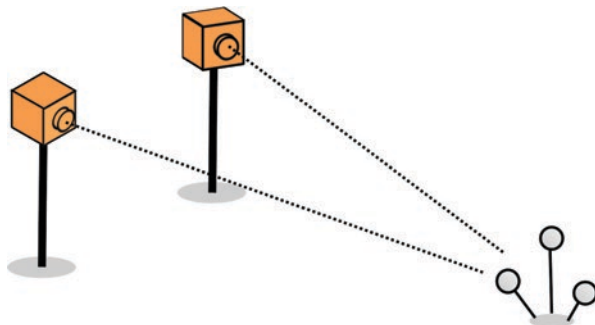




Fig. 4.7 Optical tracking of a person with reflective markers (the markers appear to be illuminated by the flashlight used) and several infrared cameras (infrared LEDs appear red)

Active markers often use infrared LEDs that must be synchronized with the cameras. This synchronization can be done with active markers via an IR flash. The cameras emit IR flashes that are reflected by the markers towards the camera lens. Due to the IR flashes, it is possible that opposite cameras are blinded. A common solution for this is to divide the cameras into so-called flash groups that work alternately, so that the opposite camera is inactive when taking the picture.

The tracking cameras that scan a specific area register the reflected radiation in a grayscale image. The pre-processing of this image data takes place in the camera and provides 2D marker positions with high accuracy using pattern recognition algorithms optimized for circular surface detection. To be able to determine the coordinates of a marker or target in space at all, it is necessary that at least two cameras scan the same area simultaneously (cf. Figure 4.6). Larger volumes are accordingly built up with more cameras, whereby it must also be ensured that partial areas of overlap are scanned by additional cameras. It is therefore important to ensure that the individual areas are linked.

The calibration of outside-in procedures with markers is usually carried out with the aid of test objects known in shape and size, which are moved in the monitored room. The test data obtained in this way allows the coordinate systems of the individual cameras to be aligned with each other such that tracked objects can be described in a uniform coordinate system.

The camera 2D data is transmitted to the central tracking controller, which calculates the 3D positions of the marker or the 6D data of the rigid bodies by triangulation and passes them on to the user. To enable the tracking software to perform this triangulation, the exact positions and orientations of the tracking cameras must be known. In a typical VR system, the accuracy requirement for this is less than 1 mm in position and less than 0.1° in angle. To determine the position and orientation of the tracking cameras with this precision, the tracking software provides a simple calibration step whose basic mathematics (bundle adjustment) is derived from photogrammetry (Hartley and Zisserman 2000) and which allows the calibration in a short time. To achieve a coverage of the tracking volume according to the

requirements, the tracking cameras are equipped with lenses of different focal lengths. This allows a variation in the field of view (FOV). To allow unrestricted working in front of power walls or in multi-side projections, wide-angle lenses for the tracking cameras are selected. It is important that the user can get close to the projection screens to achieve high immersion. Figure 4.7 shows an example where an optical tracking system is used to capture the movement of a user, so-called motion capturing.

Optical tracking in closed multi-sided projections (such as 5- or 6-sided CAVEs; see Sect. 5.4.2) presents a special problem. Optical tracking through projection screens is not possible because these screens have a highly scattering surface and optical imaging through a scattering surface is generally difficult. Therefore, tracking cameras must be installed inside the CAVE, which leads to an impairment of the spatial impression in the virtual environment by these camera bodies. For multi-sided projections in particular there are special cameras that are installed in the corners of the multi-sided projection, looking through a hole of about 40 mm diameter. This allows precise optical tracking in CAVEs to be used, whereby the optical interference caused by the holes in the corners is negligible according to the users.

4.3.2 Tracking Using Black and White Markers

Camera-based tracking using markers has been used for AR since the late 1990s and the technique is still in use today. In most cases, markers with black and white patterns are used (see Fig. 4.8). Compared to colored markers, these offer the advantage that they can be extracted from images with the aid of simple threshold values, even under varying brightness conditions.

The markers used are usually either square or round and bordered by a completely black or completely white border. Criteria for selecting one of the systems can be stability, recognition speed or the number of distinguishable marks. Some of the better-known marker-based tracking approaches include ARToolkit, ARTag, ARToolkit+ or the IS 1200 VisTracker. For a detailed comparison between different marker-based approaches, see Köhler et al. (2010).



Fig. 4.8 Typical markers as used for camera-based tracking

Use of Marker Tracking

For marker tracking, the pattern and size of the individual marker must be known in advance. While some methods (such as ARToolkit, cf. Berry et al. 2002) allow any black and white patterns for the inner part of the marker, the possible patterns are predefined in other methods (such as ARToolkit+). The latter prevents performance losses with many markers. As a rule, markers must be completely visible in the captured camera image to be recognized. With predefined patterns, however, redundancy can often still be used to detect a marker that is not completely visible. If markers are too large, it can also happen that only a part of the marker is visible when the camera is very close to it and tracking is therefore not possible or only possible to a limited extent. Conversely, if the marker is too small in the camera image, this leads to both faulty pattern recognition due to the too small number of detected marker pixels and to a significant reduction of the tracking accuracy, such that even with static objects and a practically motionless camera, transformation values can vary greatly. In addition to the size of the marker, the resolution of the camera is a decisive factor. If the AR application requires that users look at an object from very different distances, it can be advantageous to use markers of different sizes in parallel. A universal solution for this problem is the use of fractal markers (Herout et al. 2012). In addition to the distance, the angle between camera and marker as well as the current lighting situation have a major impact on the quality of the tracking results. If the angle becomes too flat, the calculated transformation values often start to vary greatly (Abawi et al. 2004). If the lighting is too bright (also due to reflections) or too dark (also due to shadows), white and black marker areas are ultimately no longer recognized sufficiently clearly from each other, making tracking no longer possible.

The main advantages of marker-based tracking are that the markers can be created quickly and easily by printing them out and can be applied to objects, walls and ceilings, or can be easily integrated into books and magazines. Even though AR markers may look similar in parts, they should not be confused with QR codes, which are used to encode strings of characters, especially URLs.

The main disadvantage of markers is that they usually must be applied directly to or on the object to be augmented. This is due to the fact that the markers would otherwise often not be visible when looking at the object (more closely) as well as because tracking inaccuracies have a much stronger effect on augmented objects if the distance from the marker to an augmented object get bigger. The markers are therefore often disturbing with respect to the real object. Another aspect is that it is not possible or not appropriate to place markers on many real objects (for example on a statue). An aggravating factor for smaller objects is that when interacting with the object (for example, by touching it), the markers are easily covered by the user's hand or arm, either completely or partially, so that tracking is no longer possible. There are numerous other factors that influence the quality of tracking. An essential aspect is the quality of the camera and the camera calibration (see Szeliski 2011). Another problem is that with some methods (such as ARToolkit) the performance decreases reciprocally quadratic with the number of patterns to be detected.

Basic Operation

In the following, the basic procedure of marker-based tracking will be outlined using ARToolkit (Kato and Billinghamurst 1999) as an example. The tracking is basically done in four steps:

1. Camera captures video image
2. In the picture, the system searches for areas with four connected line segments
3. It is checked whether the detected areas represent one of the predefined markers
4. If a marker was found, the position and orientation of the camera to the marker are calculated from the position of the vertices in the image

After obtaining the current camera image, it is first converted to a grayscale image. A black and white image is then generated based on a threshold value, whereby all values below the threshold value are displayed in black and those above the threshold value in white. All line segments in the image are now identified and then all contours are extracted from line segments with four lines. The parameters of the line segments and the positions of the corner points are temporarily stored for later calculation (see Fig. 4.9).

The region found within the four vertices is then normalized. As the surrounding black border has a uniform width of 25% of the edge length, the image to be compared can be easily extracted from the center of the image. The image is then tested for matching with the stored patterns (see Fig. 4.10). For the comparison of each stored pattern, the four possible orientations at three brightness levels each are used. The pattern with the highest degree of similarity is recognized if a defined threshold value for similarity is exceeded. It is therefore also important to select patterns with the lowest possible similarity between them to avoid false positives. Based on the orientation of the pattern, the recognized vertices can easily be assigned to the corresponding coordinates in the marker's coordinate system.

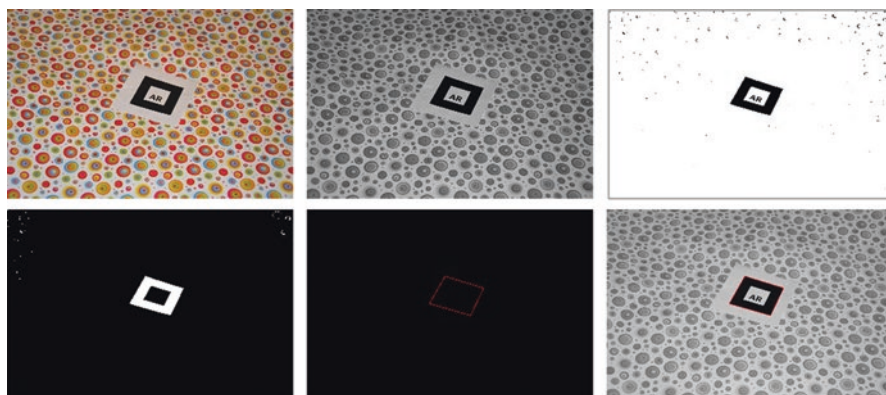


Fig. 4.9 Single steps in the recognition of the marker boundaries in the camera image: conversion to grayscale image, black and white image based on a threshold value, segmentation, identification of lines, identification of contours from four lines and storage of the corner points



Fig. 4.10 Recognized marker, normalized marker, normalized original

Intrinsic and Extrinsic Camera Parameters

The calculation of the pose of the marker in relation to the camera is based on the mapping of the marker’s corner point coordinates to pixels. The size of the marker must be known.

T_{cm} is the transformation matrix from the marker coordinate system M to the camera coordinate system C . The position of the camera corresponds to the optical center and thus the origin of the camera coordinate system. The viewing direction of the camera is along the negative z -axis of this coordinate system. \vec{v}_m is a coordinate in the marker coordinate system M and \vec{v}_c the coordinate transformed into the camera coordinate system C . For a detailed representation of the relationships see Fig. 4.11. Thus, the following applies:

$$\vec{v}_c = T_{cm} \cdot \vec{v}_m$$

and

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ z_m \\ 1 \end{bmatrix}$$

wherein the homogenous matrix T_{cm} is composed of a 3×3 rotation matrix R and a translation vector \vec{t} . Both components have three degrees of freedom each; the whole transformation thus has six. Camera calibration (cf. Szeliski 2011) yields the intrinsic camera parameters and thus the calibration matrix K , which determines the mapping of the camera coordinates to the image plane S . Here applies:

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where f is the focal length of the camera (distance from the image plane) and (c_x, c_y) is the optical center of the image in image coordinates. Strictly speaking, this is an idealized (pinhole) camera, where it is assumed that the focal length is the same in both sensor dimensions and that there is no distortion due to a non-perpendicular installation of the camera sensor (cf. Szeliski 2011, p. 47). Thus, the relation between a camera coordinate \vec{v}_c and an image pixel \vec{v}_s can be described by

$$\vec{v}_s = \begin{bmatrix} s_x \\ s_y \\ s_z \\ s_w \end{bmatrix} = \begin{bmatrix} \mathbf{K} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \cdot \vec{v}_c$$

where \vec{v}_s the must be normalized so that $s_z = 1$ (Fig. 4.11).

By inserting the detected pixels and using the calibration matrix \mathbf{K} and the known distance between the vertices, and taking into account the orientation known from the marker orientation, the 3×3 rotation matrix \mathbf{R} and the translation vector of \mathbf{T}_{cm} can thus be determined. These are called extrinsic camera parameters. For further details of the method see Kato and Billinghurst (1999) and Schmalstieg and Höllerer (2016).

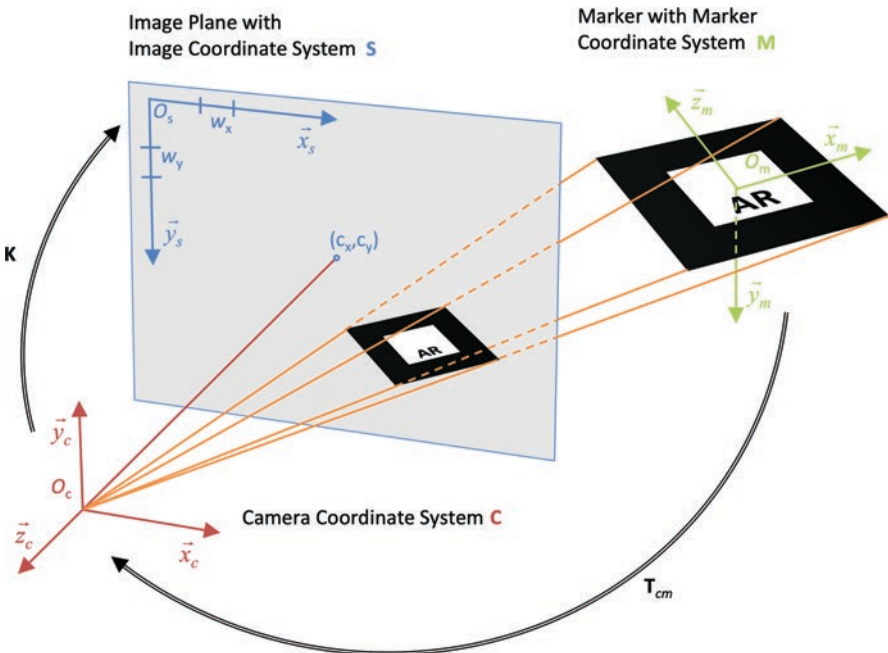


Fig. 4.11 Camera coordinate system C , image coordinate system S and marker coordinate system M (position of image plane flipped for illustration)

4.3.3 *Feature-Based Tracking Techniques*

In addition to marker-based tracking techniques, there are also camera-based tracking techniques that recognize features in the camera image and assign these to models. The models, which can be 2D or 3D, can be built on the fly or could be already known from a database. Feature-based tracking techniques represent a generalization of the marker-based approach.

Geometry-Based Tracking

In geometry-based tracking, features such as edges and/or vertices are extracted from the camera image. Based on an extrapolation of the transformation extracted from the previous camera image, the distances between the lines and corners of the calculated and the current image are used as the basis for the modification of the transformation.

As can easily be seen from the example of a cube with six identical sides, in many cases the individual features are not unique, i.e., there are often several valid poses for a current camera image. Thus, based on the last used pose, one of several possible transformations is always used: the one that has the smallest change to the previously calculated transformation. The correct initialization of the tracking is therefore crucial, as further poses are calculated incrementally. For a unique initialization, additional tracking techniques (such as the already described marker-based method) can be used. Neural networks are also increasingly used for matching with a given model (cf. Klomann et al. 2018).

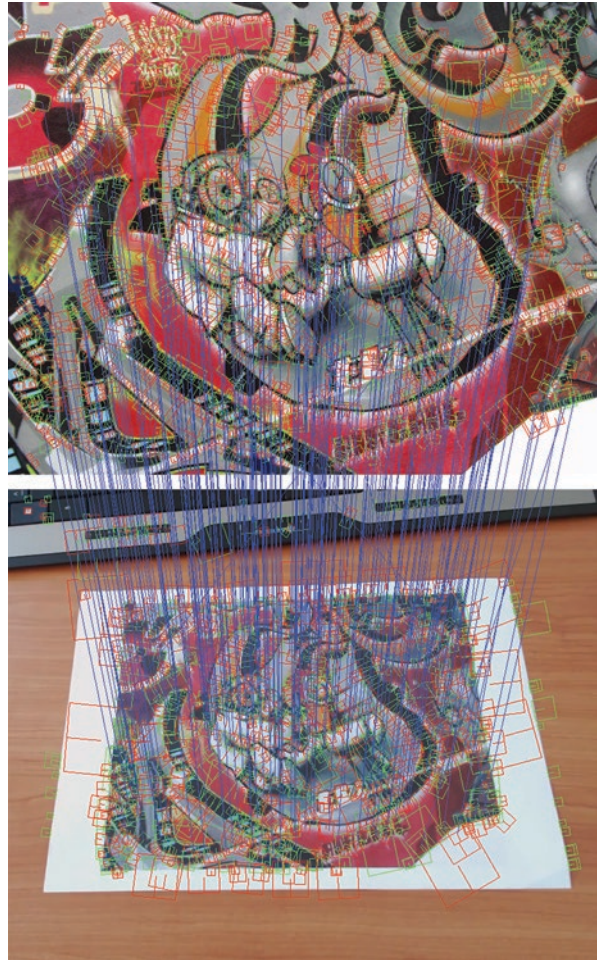
Feature-based approaches using edges and/or corners are particularly suitable in areas of uniform geometric shapes, especially when the areas have few other features for extraction.

Other Feature-Based Tracking Techniques

Unlike corners and edges, other visual features are often not easily recognizable to a human observer. However, they offer the advantage that they can be found quickly and reliably in a camera image using corresponding feature detectors. As far as is possible to extract enough of such features from the camera image, they will be compared with existing 2D or 3D descriptions of the features (the so-called descriptor). After outliers have been sorted out – usually using a RANSAC method (Fischler and Bolles 1981) – the pose of the camera in relation to the known feature groups can be calculated on the basis of the correct assignments (see Fig. 4.12).

Feature detectors differ significantly in their speed and reliability. Not all detectors offer corresponding descriptors. It is advantageous here if the detection of the features is independent of rotation (rotation invariance) and distance (scale invariance). If this is not the case, corresponding features must be calculated from

Fig. 4.12 Assignment of feature points in the current camera image to those of an existing feature map



different angles and in different resolutions. Detectors used for feature-based tracking include SIFT – Scale Invariant Feature Transform (Lowe 1999, 2004) – and SURF – Speeded Up Robust Features (Bay et al. 2006). A basic description of feature-based tracking for AR can be found in (Herling and Broll 2011). Figure 4.13 shows the robustness of feature-based methods using a SURF-based approach: despite numerous occluding objects, the remaining features visible allow for a stable pose estimation.

Another possibility to implement camera-based tracking is the combined use of color cameras and depth cameras in the form of so-called RGBD cameras. Here, the depth information can be used for tracking the camera position as well as for tracking user interactions. The latter is done by estimating to what extent skeletons can fit into the recorded depth data and thus allow the recording of user movements such as the movement of an arm. RGBD cameras usually use an infrared projected



Fig. 4.13 Tracking based on features is much more robust against interference than marker-based tracking: despite numerous objects obscuring the image used for tracking, the virtual object can be registered correctly



Fig. 4.14 Projected infrared pattern for depth detection of an RGBD camera. (© DLR 2013, all rights reserved)

pattern (see Fig. 4.14) or a Time of Flight (TOF) method for depth detection, where the travel times of the reflected light are determined. The technology of RGBD cameras has become particularly well known through the great success of the first generation of Kinect, which was sold as an input device for a game console.

4.3.4 *Visual SLAM*

In the tracking techniques presented so far, it was assumed that markers, images, or objects are known regarding their characteristics. This made it possible to determine the relative position and orientation of the camera. If either the position and location of the markers or the camera(s) in the surrounding (spatial) coordinate system was known (e.g., in the form of a map), this information could also be used for absolute location (position estimation) in the spatial coordinate system. But how to realize a tracking in an unknown environment, i.e., without known markers, images or objects and without any information about the arrangement these in space?

In this case, SLAM (simultaneous localization and mapping) – a method originating from robotics – is used. Initially, neither the position and orientation of the camera nor the environment are known. SLAM approaches primarily based on cameras observing the environments are also referred to as Visual SLAM. For SLAM-based tracking in the AR context, either features (SIFT, SURF, FAST, etc.) and/or depth information (e.g., Kinect, Intel RealSense, Google Tango, Structure.io) are used.

More recent handheld devices may also apply LiDAR (light detection and ranging), originally used in robotics and automated driving only, providing high-quality depth estimation of the environment. While the former produce sparse maps with comparatively few feature points (cf. PTAM, Klein and Murray 2007), the latter generally use dense maps of volume. Since initially no map exists, the coordinate system can be freely selected based on the starting position. The map is then successively created based on the movement of the camera, i.e., features found in the current camera image are compared with the existing map and new features are located in the map. Based on the already known parts of the map, the position and location of the camera are simultaneously reassessed based on detected features.

The simultaneous reconstruction of the environment in the form of a map as well as the estimation of the position based on this still incomplete information usually leads to increasing errors (both with regard to the quality of the map and the position estimation based on it) as long as new unknown areas are continuously added. It is crucial that known surrounding areas are reliably recognized, even if their position and location are different from the current map information. In this so-called loop closing, all map data must be adjusted to ensure that the current and stored information are consistent.

Dynamic objects represent an additional difficulty with SLAM methods. Since the resulting features change their position and location, they must be identified and then ignored in the processing, otherwise they lead to both a faulty map and faulty tracking.

4.3.5 Hybrid Tracking Techniques

For augmented reality applications it is common to use combinations of different tracking techniques. The reason for this is usually that the individual methods provide different results, depending on the situation. A typical example is a marker-based approach: this approach usually works well if the position and location in relation to the camera can be determined for all virtual content via at least one marker. However, if an occlusion occurs even for a short time, the marker is not recognized and registration of the virtual object(s) in the real scene is no longer possible. In order not to immediately lose the illusion of an augmented reality, it is therefore recommended to estimate the change of position and attitude based on alternative tracking techniques. If, for example, a tablet or smartphone is used, the change in position could also be determined by the integrated position sensors (see Sect. 4.2.3). This can be used to ensure that in situations where the brand tracking does not provide any information, a transformation can be specified that is correct at least regarding the position. If the user does not change his or her position until the corresponding marker is visible again, or only changes it slightly, the illusion can be maintained in this way.

Another way to compensate for short-term failures or even latency of the tracking technique is to use prediction techniques. While simple extrapolation methods are basically also suitable for this purpose, Kalman filters (cf. Bishop et al. 2001, p. 81) are a widely used and significantly better alternative. Depending on whether the position or the rotation must be estimated, ordinary or advanced Kalman filters are used. Another possibility is the use of particle filters (cf. Arulampalam et al. 2002).

Cloud-Based Tracking

Hybrid tracking techniques can also be used for multiuser experiences. The first step is to build a tracking reference (called an anchor) within a spatial environment or context. Feature maps in combination with additional information like GPS data (for outdoor applications) or room information (for indoor applications) can be used for this. The second step is to send to a cloud service. By downloading this cloud anchor, applications on other devices can align virtual objects to the same spatial context, enabling users to view the same content at the same location but from an individual perspective (see Fig. 4.15, left).

In visions of the near future of computing – coined as AR Cloud, Spatial Web, Mirror World or Digital Twin – a large amount of constantly updated digital content (e.g., construction, IoT, traffic, shops, artists) is spatially anchored and can be perceived and shared by many users as a persistent, dynamic overlay of the real world (see Fig. 4.15, right). Reliable, precise and easily functioning tracking and localization technologies are an essential part of the implementation of these concepts. Organizations are developing universal open standards to ensure open, free and

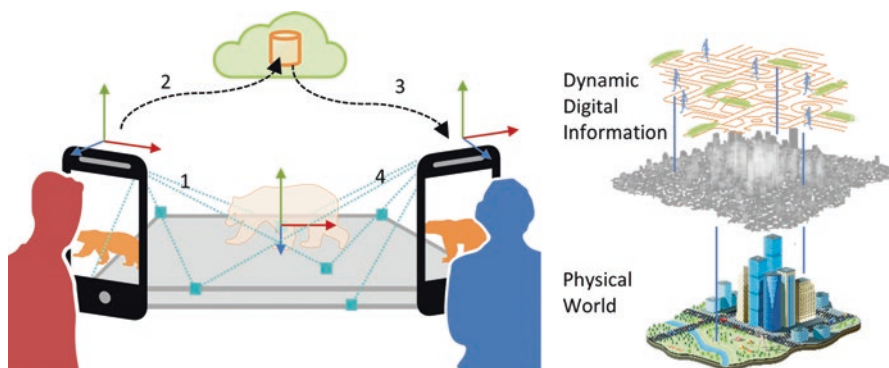


Fig. 4.15 (Left) Simplified concept of cloud anchors: 1) One device captures peculiar features from the environment. 2) It saves these and an object anchor in cloud storage. 3) Another device downloads this information and 4) tries to find the same features in its view to position a virtual object at the same anchor position. (Right) The AR Cloud concept: different layers of dynamic georeferenced information augment the real world

interoperable use of the deeply linked partial technologies. For example, the Open AR Cloud organization (OpenAR 2021) together with the Open Geospatial Consortium (OGC), is developing a standard for a geographically anchored poses with six degrees of freedom (GeoPose 2021) referenced to standardized Coordinate Reference Systems (CRSs). Since these tracking and immersive visualization technologies capture and operate with many personal and potentially protected private data, for long-term acceptance it is important to take care of privacy and data security issues and to respect possible ethical, legal and social impacts (CyberXR 2021) as part of development and operation.

Microsoft HoloLens Tracking

The SLAM approach used in Microsoft's HoloLens 2 (see HoloLens 1 in Fig. 5.10) has several special features regarding the combination of different hardware sensors. It uses a total of four cameras exclusively for tracking. The four cameras work with a comparatively low frame rate of only 30 Hz. This means that fast head movements cannot be detected without noticeable latency. To compensate for fast movements, the tracking data is therefore combined with those of an IMU (see Sect. 4.2.3) with an update rate of 1000 Hz. This allows not only the calculation of intermediate values between the determined camera poses at 240 Hz, but also compensation of color shifts (late-stage reprojection) due to the color sequential output (see Sect. 5.3.2). Instead of a global coordinate system, a graph of position estimations is used, whereby the individual local coordinate systems are connected by relative poses. If relative poses are not, or not yet, available, the graph may break up into several subgraphs. A loop closing does not take place, so that the graph is not necessarily globally consistent.

In addition, data from a depth camera (1-MP Time-of-Flight depth sensor) is used for spatial mapping with a framerate of 1–5 fps. If a user's hand is recognized the modus of the depth camera will change to high-frame rate (45 fps) near depth sensing, which is used for hand tracking in an area up to 1 m (see also Sect. 4.4). For power saving, it reduces the number of illuminations while doing the hand tracking.

Furthermore, the HoloLens has a high-resolution front camera with a FOV of 65°, a five-channel microphone array with noise cancellation to allow voice input even in loud environments, and eye tracking (see Sect. 4.5). The eye tracking is especially used for the rendering using the waveguide displays (see Sect. 5.3.2).

4.4 Finger Tracking

Although the interaction with standard input devices and the corresponding interaction methods are usually sufficient, these devices and methods hardly reproduce the natural interaction of a human being with the virtual world. New types of interaction (e.g., by pointing gestures) must first be explained to the user.

One example is the virtual assembly simulation. Using a standard interaction device such as a VR controller, a component can be easily moved from one location to another by detecting its position and orientation and by pressing a button. However, it is not possible (or very difficult) to check whether a user is able to install a component with only one hand or whether the user needs both hands for this action. Figure 4.16 left shows a user in front of a VR display during a virtual assembly simulation of a satellite. The user is equipped with optically tracked 3D glasses and a finger tracking device and tries to insert a module of the satellite with only one hand into the corresponding module slot. Other scenarios in the field of virtual assembly simulation are testing for the general tangibility of objects or the

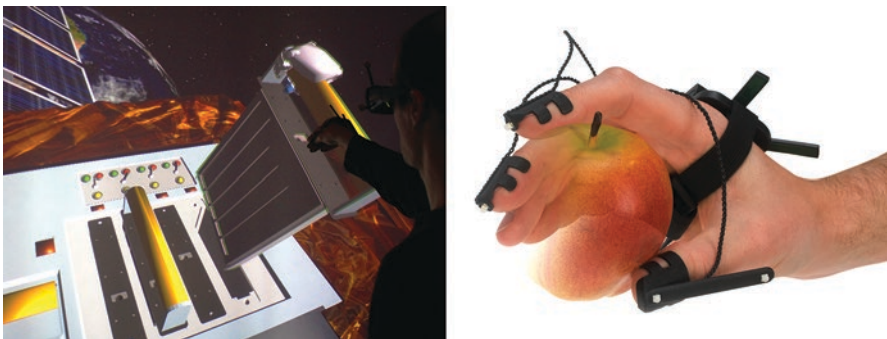


Fig. 4.16 (Left) User with tracked 3D glasses and finger tracking during an installation test of a satellite module in a virtual assembly simulation (© DLR 2013, all rights reserved). (Right) Grasping a virtual apple with a tracked hand (© ART 2013, all rights reserved)

transfer of objects from one hand to the other. The use of standard interaction devices like VR Controller is not suitable for this kind of applications.

In general, the direct interaction of users with their environment by tracking their hands and fingers in the virtual world is easier and more intuitive for them (Bowman et al. 2004). In contrast, interactions with VR are faster when using indirect interaction methods in combination with simple or standard interaction devices (Möhring and Fröhlich 2011; Hummel et al. 2012).

In general, the term finger tracking is used to describe the detection of the position and usually also the orientation of a hand and its fingers. Depending on the application, the required accuracy varies. Relatively low accuracy and only the detection of the position of the back of the hand or a finger is already sufficient to emulate a mouse or to interact with a user interface in a virtual world. However, low to medium accuracy and the relative position of individual fingers to each other is already necessary to recognize gestures. For application areas such as virtual assembly simulation in the automotive, aerospace and aviation industries, which require direct interaction, not only the position and orientation of the back of the hand and all fingertips are important for tracking, but also the lengths of the individual finger links and the angles of the corresponding finger joints. Only this accuracy enables a perfect image of the real hand.

There are two major challenges in finger tracking. First, the human hand has many degrees of freedom. The back of the hand is usually seen as a rigid body with six DOF: three translational and three rotational (see Fig. 4.17). Each finger has another four DOFs, two rotational DOFs at the root of the finger and one rotational DOF each for the joints to the middle and outer phalanx. The thumb has a special role because it has an additional DOF at the root. Therefore, five DOFs are required for the thumb, three rotatory DOFs at the wrist and one for each additional finger joint. Added up, this results in 27 DOF for one hand (Lin et al. 2000). Second, the tight position of the fingers in relation to each other is a great challenge for the tracking system. For optical systems in particular this is a non-trivial problem to

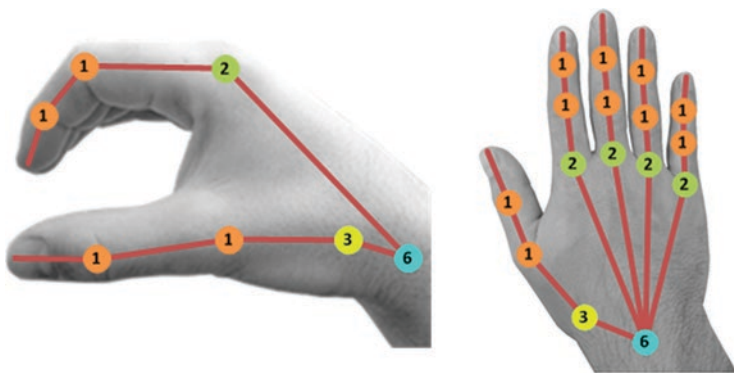


Fig. 4.17 Data model of a hand to implement finger tracking (the circles symbolize the joints of the hand and fingers with their respective degrees of freedom; the lines represent the skeleton)

solve because of the occlusion of markers, the small visual difference of the fingers and the 27 DOF per hand.

In addition, it should not be forgotten that each person's hands and fingers are different. This includes not only the length and thickness of the individual phalanges, but also the joints and joint angles between them. A physical handicap or even the absence of one or more fingers must not be ignored either. The respective tracking devices must take this into account and be adaptable to it.

Since finger tracking has high requirements on the tracking hardware, a wide variety of techniques are employed. In earlier days mechanical tracking techniques were most common, for example optical fibers, strain gauges or potentiometers (variable resistors). The Sayre Glove (DeFanti and Sandin 1977) has bendable tubes that run along each finger inside a glove. The Data Glove (Zimmermann et al. 1986) uses two optical fibers per finger. At one end of this fiber optic cable is a light source; at the other end is a photocell. Depending on the bending of the finger, a different amount of light hits the photocell. This allows the joint angles of the fingers to be approximately determined. The CyberGlove (Kramer and Leifer 1989) uses 22 thin, metallic strain gauges to measure the joint angles of the fingers. In the Dexterous Hand Master (Bouzit et al. 1993), an exoskeleton is pulled over the hand and fingers. Using cable pulls, potentiometers are then activated, from whose resistance values the positions of the fingers can be determined by analog/digital converters. With mechanical methods, however, only a relative measurement of the fingers to the back of the hand is possible. The position and orientation of the back of the hand must be measured using a different tracking technique.

More rarely, magnetic trackers are used for finger tracking. These can detect up to 16 individual 6-DOF sensors. This means there is one sensor for each of the three-finger links and one sensor for the back of the hand. The disadvantage of magnetic tracking is the slight susceptibility to interference from metallic or electromagnetic sources. In addition, most magnetic trackers are wired due to their design.

Optical finger-tracking devices predominate in the non-mechanical tracking techniques. The MIT LED Glove (Ginsberg and Maxwell 1983) is equipped with light-emitting diodes (LEDs), which are recorded by an external camera system. To distinguish individual fingers from each other, the LEDs flash alternately one after the other (Hillebrand et al. 2006). At a recording rate of 60 Hz, for example, the alternate flashing of the LEDs reduces the repetition frequency to 20 Hz for a three-finger system and to 12 Hz for a five-finger system. The use of optical tracking enables high accuracy and lightweight wireless interaction devices, but usually at least four expensive special cameras are required to ensure triangulation of each LED used. Some optical finger tracking devices are additionally equipped with inertial sensors to temporarily bridge any obscurations of the LEDs, which often occur due to the small distances between the fingers. In Hackenberg et al. (2011) a method was presented that is based on depth cameras and uses special feature detectors for finger phalanges and fingertips.

There are inexpensive camera-based finger trackers available, which nevertheless offer high accuracy and low latency and can be easily integrated into VR applications. Leap Motion, as an example, uses two cameras in combination with infrared



Fig. 4.18 3D model of a hand controlled by a VR controller with touch sensors

LEDs (wavelength 850 nm). The hardware covers an interaction space of up to 80 cm by 80 cm, with the brightness of the infrared LEDs being the limiting factor. The controller transmits two grayscale videos to the software, which in turn determines the finger positions from this data. Usually, the controller is used while lying on a table. With the help of an adapter, however, it is also possible to attach the controller to VR glasses to use finger gestures as input for VR applications.

Using touch-sensitive surfaces it is also possible to track fingers using a VR controller (see Fig. 4.18).

4.5 Eye Tracking

4.5.1 Eye Movements

Eye-tracking, or gaze registration, generally refers to tracking the movement of the human eye. The procedure is used to record and evaluate the course of a person's gaze.

If a user views an image, he focuses by changing the focal length of his lens and depicts the image onto light-sensitive cells of the retina. The amount of incident light can be varied through the iris. The iris works like an aperture and changes the diameter of the pupil. The eye muscles that move the eye in the eye socket are attached to the sclera. The types of movement of the eye are differentiated into drifting, following, trembling, rotating, fixing and saccades. However, only the last two are interesting for tracking the eye. During fixation, e.g., while reading, the eye concentrates on one point and collects information. Saccades are jumps that take place between fixation and last about 20 ms to 40 ms.

4.5.2 Methods

Various technical methods have been developed in recent decades to determine the direction of gazes. An overview of these methods and sub methods is given in Fig. 4.19. In principle, a distinction is made between invasive and non-invasive

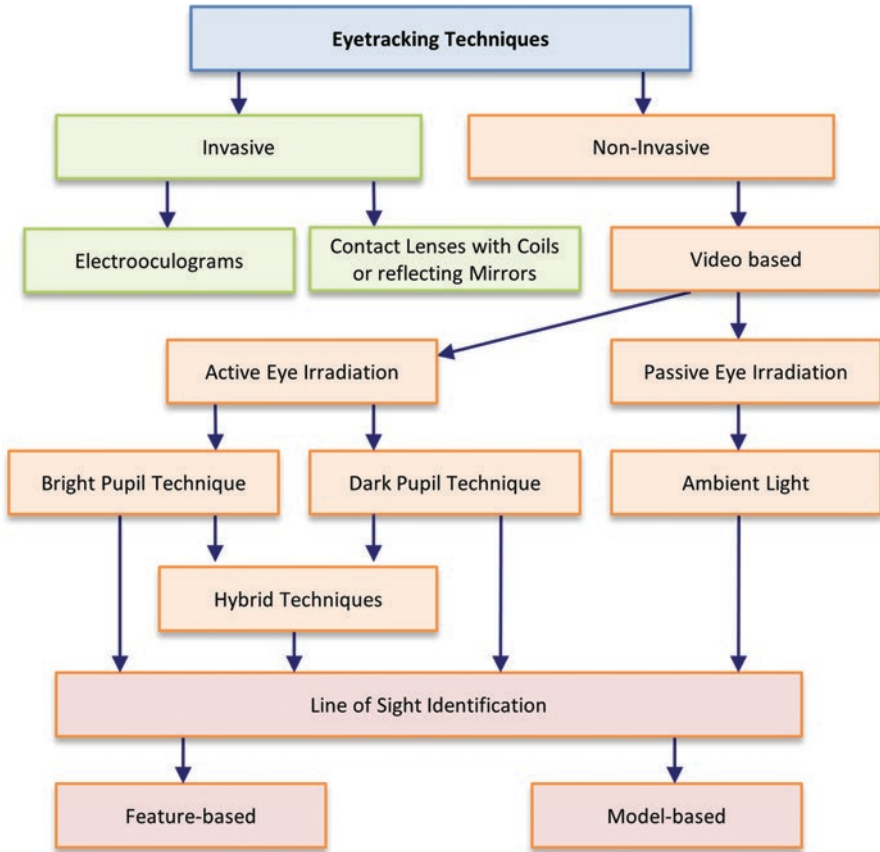


Fig. 4.19 Overview of methods for eye tracking

procedures. Invasive procedures always require a direct intervention on the user’s body, e.g., with electrodes.

With non-invasive procedures the user’s gaze can be followed without contact. The first developed eye-tracking techniques were purely invasive. Electrooculography was developed more than 40 years ago. In electrooculography, the electrical potentials of the skin around the eye are measured. These potentials range from 15 μV to 200 μV . The sensitivity for eye-tracking is about 20 $\mu\text{V}/\text{angle degree}$ (Duchowski 2007). With this technique the relative eye movement to the head can be recorded. However, it is not possible to determine an absolute point of view of the eye on an object. Another invasive eye-tracking technique is the contact lens method. Here, contact lenses are used either with small coils or with reflectors. For contact lenses with coils, the change of the magnetic field is measured, and from this the relative movement of the eye is derived. If there are reflectors on the contact lenses, the reflected light can be used to deduce the relative direction of vision.

In recent years non-invasive video-based eye-tracking techniques have been used. Here, the eye is captured by a camera and the gaze direction is determined by image processing algorithms. In video-based methods, a distinction is made between passive and active eye irradiation. Passive methods use ambient light to irradiate the eye scene. Due to the undefined irradiation conditions of an environment, there are high requirements for precise feature identification of the eye components.

With passive irradiation, the contour between the dermis and iris is used to identify features. A more precise method is the active irradiation of the eye scene by an infrared light source. Figure 4.20 illustrates the more favorable contrast ratios of the active method, which enables robust feature identification between pupil and iris.

Depending on the arrangement of the IR irradiation source, a distinction is made in active irradiation procedures between the light and dark pupil technique. If the irradiation source is located outside the optical axis of the eye-tracking camera, the radiation is reflected by the iris and sclera; thus, the pupil is the darkest object within the recorded eye scene. If the light source and the camera are arranged in the same optical axis, the radiation is reflected at the retina inside the eye, making the pupil the brightest object.

Hybrid processes require optics with different arrangements of the IR irradiation sources. Regardless of whether active or passive eye irradiation is used, the evaluation of the direction of gaze is based on features on the one hand and on models on the other. Combined methods are also used. Feature-based methods detect contours, e.g., the pupil geometry, and calculate the center point and the relative gaze coordinates. Side effects, such as reflections, can cause other features to be interpreted as the pupil; this property reduces the accuracy of feature-based methods. Model-based methods, on the other hand, compare the image information of the recorded eye scene with a corresponding model of an eye. By varying the parameters, an iterative attempt is made to adapt the model to the real eye scene. If the model could be adapted with a certain error, the relative gaze coordinates are obtained. Model-based procedures belong to the more precise, but also to the more computationally intensive, approaches. Video-based eye-tracking techniques not

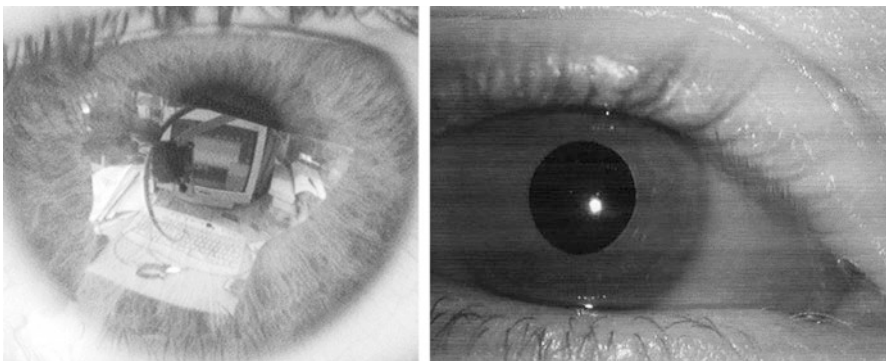


Fig. 4.20 Recorded eye scene with passive and active irradiation

only allow the relative direction of gaze to be determined. With calibration, the correspondence between the direction of gaze and regions in the virtual image (e.g., a button) can be found.

4.5.3 Functionality of an Eye Tracker

Figure 4.21 shows the basic procedure of an eye-tracking routine with active illumination and bright pupil technique. An eye-tracking camera, which is focused on the user's eye, captures a digital grayscale image. This image is passed to the eye-tracking image processing system. First, an adjustment of the gray values is applied and then the image is pre-filtered, e.g., to improve a noisy image. Furthermore, a

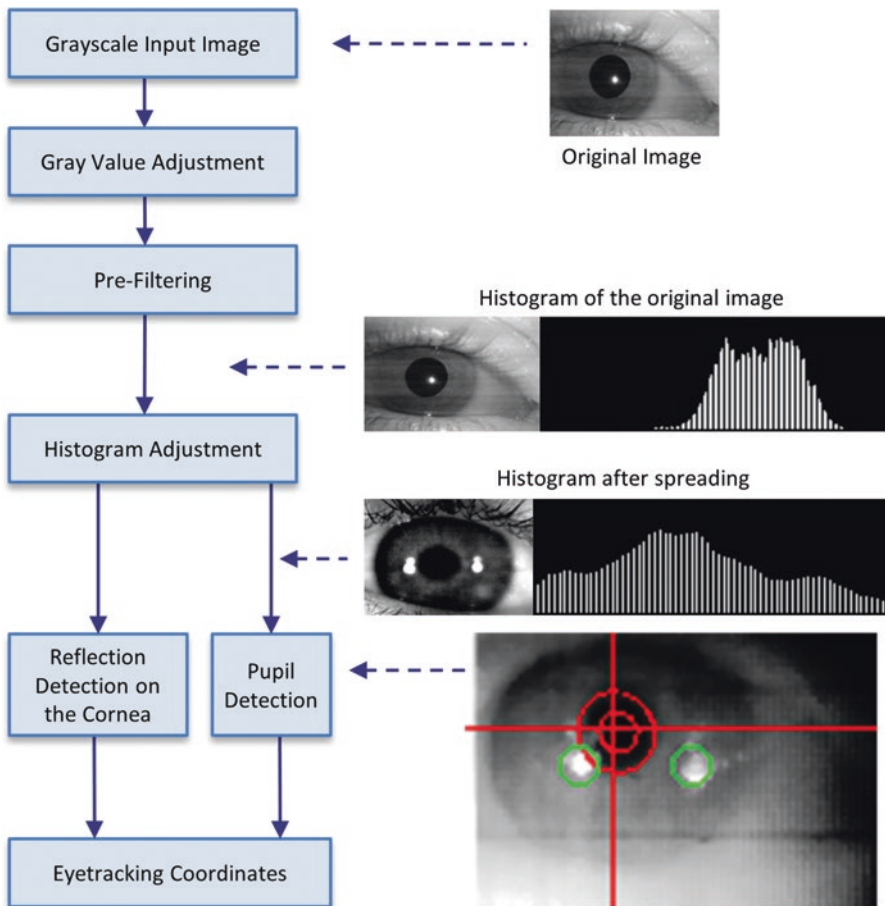


Fig. 4.21 Image processing process for eye tracking

histogram spread is performed to highlight the object contours of the eye such as the pupil or iris. In the next step, the contour of the pupil is detected by edge detection, and the pupil center is calculated. Furthermore, in the case of active illumination, the reflections at the cornea are used as additional information. With a Head-Mounted Display (HMD; see Chap. 5) with integrated eye tracking, these reflections are often used as a reference point. The eye-tracking image processing finally outputs the coordinates of the pupil center in horizontal and vertical direction. If the corneal reflections are also evaluated, the eye-tracking image processing outputs a difference vector between the pupil center and the center of the corneal reflection, from which it can be concluded where the user focuses.

4.5.4 Calibration

To enable user interaction with virtual objects in addition to the actual eye-tracking, an assignment between the camera's detection range and the displayed image is necessary.

Figure 4.22 shows the nested coordinate systems of the eye-tracking camera and the virtual image. To be able to establish a connection between the pair of coordinates in the camera coordinate system \bar{x}_c , \bar{y}_c and the coordinates of the virtual image \bar{x}_{virt} , \bar{y}_{virt} , there are various assignment methods. In Duchowski (2007) a simple linear analytical mapping function is presented. Equations (4.1) and (4.2) describe the linear mapping functions for the horizontal and vertical direction. In eq. (4.1) the horizontal coordinate \bar{x}_c is set by subtracting from \bar{x}_{c_min} to its origin. Then this coordinate is scaled to the virtual image by the horizontal resolution ratio between the virtual coordinate system and the camera coordinate system. Then the relative position in the virtual image is calculated by adding the minimum coordinate of the virtual image. For the vertical coordinate assignment, the calculation method described in eq. (4.2) is analogous to eq. (4.1).

$$\bar{x}_{virt} = \bar{x}_{virt_min} + \frac{(\bar{x}_c - \bar{x}_{c_min})(\bar{x}_{virt_max} - \bar{x}_{virt_min})}{(\bar{x}_{c_max} - \bar{x}_{c_min})} \quad (4.1)$$

$$\bar{y}_{virt} = \bar{y}_{virt_min} + \frac{(\bar{y}_c - \bar{y}_{c_min})(\bar{y}_{virt_max} - \bar{y}_{virt_min})}{(\bar{y}_{c_max} - \bar{y}_{c_min})} \quad (4.2)$$

In practice, more complex assignment procedures, such as the second and third-order polynomial procedure or the homographic procedure, are usually used. These assignment procedures require several parameters. The parameters are obtained by a calibration routine. In this calibration routine the user has to select points that are distributed over the virtual image (e.g., in the corners and in the middle). The user must look at these points one after the other. Using these parameters, the calibration routine can now determine the parameters for the complex assignment functions.

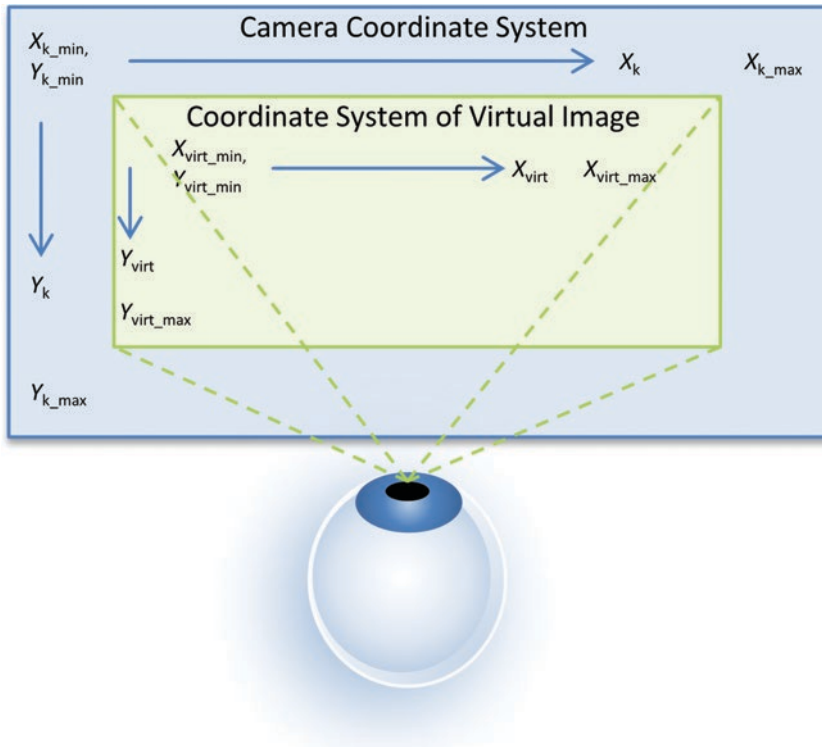


Fig. 4.22 Camera coordinate system of the virtual image and the camera

4.5.5 Eye Tracking in Head-Mounted Displays

If you want to use gaze control, you can use an eye-tracking HMD. Figure 4.23 shows the basic procedure of an eye-tracking HMD. As already mentioned in Sect. 4.5.2, a camera is required for a video-based procedure. The camera is attached to the HMD in a way that it can focus on the eye. The captured image of the eye scene is then transmitted to the computer or to the HMD electronics and an eye-tracking algorithm calculates the direction of the eye (see Sect. 4.5.3).

Eye-tracking HMDs evaluate either both eyes simultaneously or only one eye. If, for example, the gaze direction of both eyes is determined, the 3D viewpoint of the user can be determined from the intersection of both vectors.

As already explained in Sect. 4.5.4, there must be a correspondence between the coverage area of the camera and the display area of the virtual projection. Therefore, a calibration must be carried out. Compared to the remote eye trackers presented in Sect. 4.5.6, eye tracking HMDs have better conditions for recalibration due to the tight fit of the glasses. If the HMD moves only slightly, the calibration does not have to be repeated during operation.

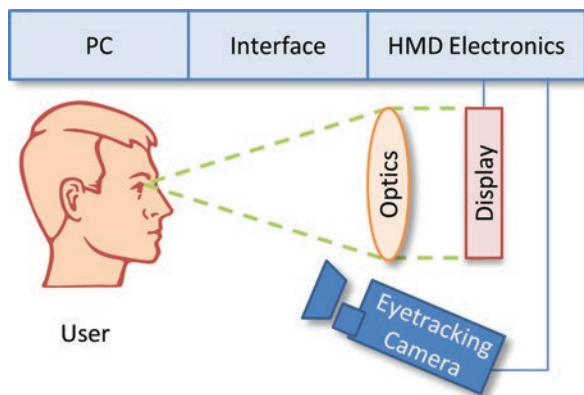


Fig. 4.23 Basic procedure of an eye-tracking HMD

4.5.6 Remote Eye Tracker

A remote eye tracker has essentially the same components as the eye-tracking HMD presented in Sect. 4.5.5. With a remote eye tracker, the user sits in front of a monitor. A camera mounted near the monitor focuses on the user's head. There are two methods to capture one or both eyes. On the one hand the camera captures a large area where the user's head is located. The image processing locates the area of the eye and calculates the position of the pupil in this section. With this method, only a few pixels are available to calculate the pupil position. This low resolution of the pupil area also reduces the accuracy. With a second method, the eye-tracking camera captures only a small area, but this area is captured with high resolution. This camera automatically aligns itself so that the current position of the eye is recorded. As mentioned in Sect. 4.5.4 a calibration must be performed for the remote eye tracker to assign the calculated coordinates of the gaze direction to the display area of the monitor. Unlike eye-tracking HMDs, remote eye trackers often need to be recalibrated during operation because the user changes their sitting position relative to the monitor and the eye-tracking camera.

4.6 Further Input Devices

In this section we will consider other input devices that are often used to build VR systems, in addition to standard PC input devices (such as 2D mouse, keyboard, microphone or touch monitors).



Fig. 4.24 Different variants of a 3D mouse

4.6.1 3D Mouse

One of the simplest input devices is the 3D mouse (see Fig. 4.24). This enables direct navigation according to the six degrees of freedom as well as interaction via freely assignable buttons. By shifting the mouse sideways and pushing and pulling it vertically, a translation in 3D space can be performed; by turning or tilting it, a corresponding rotation is achieved.

Versions of the 3D mouse differ not only in size but also in the integration of additional buttons, which are usually freely assignable. The advantage of a 3D mouse is its high accuracy. Because a 3D mouse is usually placed on a table, it is more suitable for desktop VR. Sometimes it is also used as a control unit permanently mounted on a column, which limits the user's working range.

4.6.2 Mechanical Input Devices

Mechanical input devices record the movements of a user via a mechanism (e.g., via a linkage or cable pulls). The advantage of mechanical input devices is that, on the one hand that they can be highly accurate, and on the other hand that they are well suited to provide haptic feedback to the user. The disadvantages are that the user always has something in his or her hand or has to be connected in some way to the mechanical input device and that the mechanics may be a disruptive object. Figure 4.25 shows an example of a mechanical input device where the user holds a pen in his or her hand. The fact that the user is used to holding pens means that the use of the device can become part of normal habits, provided that the actual application supports this usage scenario.

Mechanical input devices use angle or distance measurements at the joints to obtain users interactions. The high accuracy is achieved by correspondingly accurate angle measurements, which are usually carried out using gear wheels or gears, potentiometers, or strain gauges. In some cases, similar measuring methods are used as in computer mice, which are known to allow high resolution. The latency of mechanical input devices is low due to the direct measurement. Smooth operation is particularly important for use (Salisbury and Srinivasan 1997) in order not to be



Fig. 4.25 Mechanical input device in pen form with haptic feedback

restricted by the input device and thus to perceive it as disturbing. By integrating haptic feedback, a mechanical input device becomes an output device at the same time (see end effector displays in Sect. 5.5).

4.6.3 Treadmills for Virtual Reality

Due to the limited size of a VR system, it is difficult to allow walking or running around in a virtual environment. In most cases the user reaches the edge of the interaction area after a few steps. Accordingly, control techniques for navigation have established themselves, using different input devices such as VR controllers or a flystick (see Sect. 4.3.1). In addition, input devices have been developed that allow walking or a walk-like movement for navigation in virtual worlds. Many approaches are based on the idea of treadmills on which users move and whose speed is controlled by the VR system. By means of a mechanism for tilting, it is possible to walk uphill or downhill. The disadvantage of treadmills, which are used in a similar way in gyms, is that they only allow walking or running in one direction, which is a significant limitation for use in VR systems.

In recent years, so-called omnidirectional treadmills have been developed using different approaches. One possibility is to construct the treadmill from small treadmills that are arranged orthogonally to the main direction. This creates a surface on which the user can move in all directions. By tracking the user, the individual treadmills can be controlled so that the user always moves in the middle of the surface. The CyberWalk Treadmill (Souman et al. 2008) is an

example of this. Large balls, in which the user moves and which are themselves supported so that they remain in one place, are another possibility. The problem with this approach is that the perceived floor for the user is not flat but curved by the shape of the sphere. This can make walking more difficult. The Cybersphere (Fernandes et al. 2003) is an example of this type. Other variants are based on constructing the floor from appropriately arranged castors to allow walking around. More cost-effective approaches are based on the idea of holding the user in place by means of a retaining ring and allowing him or her to walk on a smooth or slippery floor. The Virtuix Omni (see Fig. 4.26) and the Cyberith Virtualizer are examples of this.

Fig. 4.26 User with VR glasses on an omnidirectional treadmill



4.7 Summary and Questions

In this chapter you have acquired basic knowledge in the field of tracking and VR/AR input devices. Starting from the consideration of how many degrees of freedom an object has, basic terms such as accuracy, repetition rates, latency and calibration were introduced with respect to their applicability in the fields of VR and AR. Following the presentation of different tracking techniques for the continuous determination of 3D data, further input devices were introduced.

Check your understanding of the chapter by answering the following questions:

- Why is high accuracy not sufficient as a requirement for VR/AR input devices?
- Which effects can cause problems during data acquisition?
- What is determined by a tracking system and what are the characteristics of tracking systems?
- What effects can interfere with a tracking system?
- What problems arise with outdoor tracking in city centers and what alternatives exist?
- Find an application example for hybrid tracking techniques.
- What is the difference between inside-out and outside-in tracking techniques and what are their advantages and disadvantages?
- What are the advantages of camera-based tracking?
- Why should you actively illuminate the eyes of a user during eye tracking and what should be considered?
- How many degrees of freedom must be determined for finger tracking?

Recommended Readings

- Bishop G, Allen D, Welch G (2001) Tracking: beyond 15 minutes of thought. SIGGRAPH 2001, Course 11, http://www.cs.unc.edu/~tracker/media/pdf/SIGGRAPH2001_CoursePack_11.pdf. Accessed 18 October 2021 – *The authors of this course at SIGGRAPH presented a good overview of the technical fundamentals of input devices.*
- Szeliski R (2011) Computer vision - algorithms and applications, Springer, DOI <https://doi.org/10.1007/978-1-84,882-935-0>. – *This book gives a good overview of the basics of computer vision, which is used for VR and AR.*

References

- Abawi FA, Bienwald J, Dörner R (2004) Accuracy in optical tracking with fiducial markers: an accuracy function for ARToolKit. In: Proceedings of the 3rd IEEE/ACM international symposium on mixed and augmented reality (ISMAR '04). IEEE Computer Society, Washington, DC, USA, pp 260–261. <https://doi.org/10.1109/ISMAR.2004.8>
- Arulampalam MS, Maskell S, Gordon N, Clapp T (2002) A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Trans Signal Process* 50(2):174–188
- Bay H, Tuytelaars T, Van Gool L (2006) SURF: speeded up robust features. In: *Computer vision—ECCV 2006*. Springer, Berlin Heidelberg, pp 404–417
- Berry R, Billinghamurst M, Cheok AD, Geiger C, Grimm P, Haller M, Kato H, Leyman R, Paelke V, Reimann C, Schmalstieg D, Thomas B (2002) The First IEEE international augmented reality toolkit workshop. IEEE Catalog Number 02EX632
- Bishop G, Allen D, Welch G (2001) Tracking: Beyond 15 minutes of thought, SIGGRAPH 2001, Course 11. http://www.cs.unc.edu/~tracker/media/pdf/SIGGRAPH2001_CoursePack_11.pdf. Accessed 18 March 2021
- Bouzit M, Coiffet P, Burdea G (1993) The LRP Dextrous Hand Master. Proceedings of Virtual Reality Systems Fall '93, New York
- Bowman DA, Kruijff E, LaViola JJ, Poupyrev I (2004) 3D-user interfaces: theory and practice. Addison Wesley Longman Publishing Co., Inc., Redwood City
- CyberXR (2021) Cyber-XR Coalition: Immersive technology standards for accessibility, inclusion, ethics and safety. <https://www.cyberxr.org>, Accessed 18 Mar 2021
- DeFanti IA, Sandin DJ (1977) Final report to the National Endowment of the Arts. US NEA R60–34–163, University of Illinois at Chicago Circle, Chicago, IL
- Duchowski A (2007) *Eye tracking methodology: theory and practice*. Springer, London
- Fernandes KJ, Raja V, Eyre J (2003) Cybersphere: The fully immersive spherical projection system. *Communications of the ACM*, 46(9), 141–146. ACM, New York
- Fischler MA, Bolles RC (1981) Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6), 381–395. ACM, New York
- GeoPose (2021) GeoPose Standards Working Group, <https://www.ogc.org>, Accessed 18 Mar 2021
- Ginsberg CM, Maxwell D (1983) Graphical marionette. Proceedings of SIGGRAPH Computer Graphics 18(1):26–27
- Goldstein H (1980) *Classical mechanics*. Addison-Wesley
- Hackenberg G, McCall R, Broll W (2011) Lightweight palm and finger tracking for real-time 3D gesture control. In Proceedings of IEEE Virtual Reality Symposium 2011 (IEEE VR 2011), pp. 19–26
- Hartley R, Zisserman A (2000) *Multiple view geometry in computer vision*. Cambridge University Press, Cambridge
- Herling J, Broll W (2011) Markerless tracking for augmented reality. In: *Handbook of augmented reality*. Springer, New York, pp 255–272
- Herout A, Zacharias M, Dubska M, Havel J (2012) Fractal marker fields: No more scale limitations for fiducial markers. In IEEE International Symposium on Mixed and Augmented Reality (ISMAR), pp. 285–286. IEEE
- Hillebrand G, Bauer M, Achatz K, Klinker G (2006) Inverse kinematic infrared optical finger tracking. 9th International Conference on Humans and Computers (HC 2006). Key 1045432
- Hummel J, Wolff R, Dodiya J, Gerndt A, Kuhlen T (2012) Towards interacting with force-sensitive thin deformable virtual objects. Joint Virtual Reality Conference of ICAT – EGVE – EuroVR, 2012 Eurographics Association, pp. 17–20. <https://doi.org/10.2312/EGVE/JVRC12/017-020>
- Kato H, Billinghamurst M (1999) Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR), pp. 85–94. IEEE

- Klein G, Murray D (2007) Parallel tracking and mapping for small AR workspaces. In 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, pp. 225–234. IEEE
- Klomann M, Englert M, Weber K, Grimm P, Jung Y (2018) Improving mobile MR applications using a cloud-based image segmentation approach with synthetic training data. In Proceedings of the 23rd International Conference on 3D Web Technology, Web3D 2018, pp. 4:1–4:7. ACM
- Köhler J, Pagani A, Stricker D (2010) Detection and identification techniques for markers used in computer vision visualization of large and unstructured data sets. In Applications in Geospatial Planning, Modeling and Engineering (IRTG 1131 Workshop)
- Kramer J, Leifer L (1989) The talking glove: An expressive and receptive ‘verbal’ communication aid for the deaf, deaf-blind, and non-vocal. Proceedings of the 3rd Annual Conference on Computer Technology, Special Education, Rehabilitation. California State University Press, Northridge
- Lee JC (2008) Hacking the Nintendo Wii remote. *Pervasive Computing* 7(3):39–45. <https://doi.org/10.1109/MPRV.2008.53>
- Lin J, Wu Y, Huang TS (2000) Modeling the constraints of human hand motion. In Proceedings of the Workshop on Human Motion (HUMO ‘00), IEEE Computer Society, Washington, DC, USA
- Lowe DG (1999) Object recognition from local scale-invariant features. In Proceedings of the Seventh IEEE International Conference on Computer Vision, Vol. 2, pp. 1150–1157. IEEE
- Lowe DG (2004) Distinctive image features from scale-invariant keypoints. *Int J Comput Vis* 60(2):91–110
- Möhring M, Fröhlich B (2011) Effective manipulation of virtual objects within arm’s reach. 2011 IEEE Virtual Reality Conference, 2011, pp. 131–138. IEEE. <https://doi.org/10.1109/VR.2011.5759451>
- OpenAR (2021) Open AR Cloud. <https://www.openarcloud.org/>. Accessed 18 March 2021
- Salisbury JK, Srinivasan MA (1997) Phantom-based haptic interaction with virtual objects. *Computer Graphics and Applications*. IEEE. <https://doi.org/10.1109/MCG.1997.1626171>
- Schmalstieg D, Höllerer T (2016) *Augmented reality: Principles and practice*. Pearson
- Souman JL, Robuffo Giordano P, Schwaiger M, Frissen I, Thümmel T, Ulbrich H, Bühlhoff HH, Erst MO (2008) Cyberwalk: Enabling unconstrained omnidirectional walking through virtual environments. *ACM Transactions on Applied Perception*. <https://doi.org/10.1145/2043603.2043607>
- Szeliski R (2011) *Computer vision - algorithms and applications*, Springer. <https://doi.org/10.1007/978-1-84,882-935-0>
- Zimmerman TG, Lanier J, Blanchard C, Bryson S, Harvill Y (1986) A hand gesture interface device. *Proceedings of SIGCHI Bulletin*, 17, SI(May 1987), 189–192

Chapter 5

VR/AR Output Devices



Wolfgang Broll, Paul Grimm, Rigo Herold, Dirk Reiners,
and Carolina Cruz-Neira

Abstract This chapter discusses output devices and technologies for Virtual Reality (VR) and Augmented Reality (AR). The goal of using output devices is to enable the user to dive into the virtual world or to perceive the augmented world. Devices for visual output play a crucial role here, they are of central importance for the use of VR and AR. First and foremost, Head-Mounted Displays (HMD) must be mentioned, the different types of which are discussed in detail here. However, VR also uses different forms of stationary displays, which are another major topic of this chapter. Finally, output devices for other senses are reviewed, namely acoustic and haptic outputs.

5.1 Introduction

How can virtual content be transformed into sensory experiences? What possibilities and alternatives exist to address individual senses? Output devices serve to present the virtual world to users or to expand the real world by generating appropriate stimuli. In this chapter, output devices for VR and AR are presented. A VR or AR system must react to user actions, which are recognized by the use of suitable input devices (see Chap. 4), and generate a corresponding representation, which in turn appeals to the senses of the users (see Sect. 2.1). Commercially available output devices address in particular the visual, acoustic and haptic senses. Here, we will focus on the visual output of the output devices, because it is of outstanding importance for VR and AR. Figures 5.1 and 5.2 show two typical representatives of the most important visual device categories: an HMD and a CAVE-like large

Dedicated website for additional material: vr-ar-book.org

W. Broll (✉)

Department of Computer Science and Automation / Department of Economic Science
and Media, Ilmenau University of Technology, Ilmenau, Germany
e-mail: wolfgang.broll@tu-ilmenau.de

© The Author(s), under exclusive license to Springer Nature
Switzerland AG 2022

R. Doerner et al. (eds.), *Virtual and Augmented Reality (VR/AR)*,
https://doi.org/10.1007/978-3-030-79062-2_5



Fig. 5.1 Typical consumer HMD with integrated head-tracking and accompanying controllers (© TU Ilmenau 2019, all rights reserved)



Fig. 5.2 CAVE-like multi-sided projection to visualize a design study (© RWTH Aachen 2013, all rights reserved)

projection. We will then provide an overview of acoustic output devices and some haptic output devices used in VR and AR. In addition, there is a multitude of other, sometimes very special, output devices in the form of prototypes and demonstrators,

which address further senses. For example, there are olfactory displays, acceleration simulators based on galvanic-vestibular stimulation and specific solutions such as the event-controlled generation of wind or the splashing of water, which will not be considered further here due to their limited popularity so far. Pure motion platforms, such as those used for driving and flight simulators, or in amusement parks, are also not discussed here, although one might consider them a large-scale VR output device.

Sect. 5.2 introduces basic aspects of visual output for better understanding of the following sections. Section 5.3 deals with Head-Mounted Displays (HMDs). This includes those for VR as well as for AR. Furthermore, part of it deals with the technical properties of HMDs. In Section 5.4 stationary VR systems are considered. This also includes multi-sided projections such as CAVEs and tiled displays, and their technical challenges as well as technologies for stereo presentation. Sections 5.5 and 5.6 deal with audio output and haptic output devices for VR and AR, respectively. The chapter concludes with a short summary, including a list of questions, a list of recommended literature and the list of references.

5.2 Basics of Visual Output

The basic goal of the visual output is to present the *virtual world* (in the case of a VR system) or the *augmented world* (in the case of an AR system) to users in such a way that they can perceive it in a similar way to the real world. The term *display* is used in the following as an umbrella term for monitors, *projection systems* (i.e., projector with projection surface or canvas) and *head-mounted displays (HMDs)*. Monitors and projection systems are used in stationary systems. HMDs refer to displays mounted on the head, which can usually be viewed by both eyes, or sometimes only by one eye. In the following, *HMD* is used as a generic term for both VR and AR glasses. Smart glasses, which are also HMDs, but are primarily used to display information in a small area of the field of view, should be distinguished from HMDs as they are not suitable for VR or AR.

The classification of visual output devices can be based on different criteria. Possible criteria include quality, brightness, field of vision or perception, size of the area of use, uniformity, freedom of movement, usability or location. The following aspects can thus be used to describe visual VR/AR output devices. Starting with viewpoint-related aspects, we will look into the technical parameters of such systems before discussing more user- or usage-oriented aspects.

Visual Field

The visual field is the area that can be perceived by the eyes of a user without moving the eyes or head. The human visual field is about 214° horizontally (see Fig. 5.3). Each eye covers an angle of approximately 60° towards the nose and 107° towards the outside. The area that can be perceived with both eyes, the so-called binocular cover field, is thus approximately 120° (horizontal). Vertically, the visual field is generally much smaller (approximately 130° – 150°).

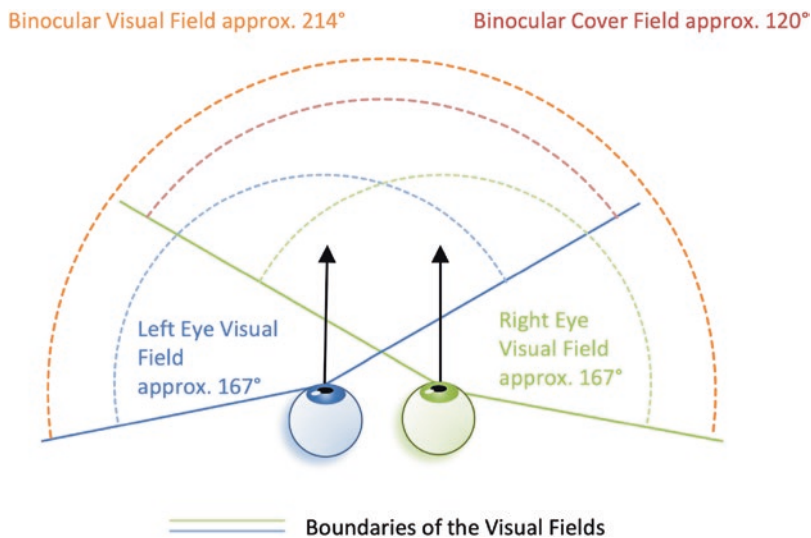


Fig. 5.3 Human visual field

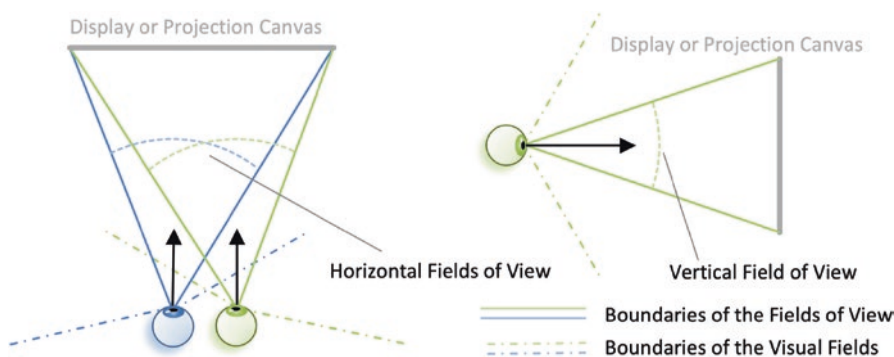


Fig. 5.4 Horizontal and vertical field of view (FoV)

Field of View

The field of view (FOV) is the angle of view that can be perceived using a technical device (e.g., an HMD). It is usually specified separately for horizontal and vertical angles (see Fig. 5.4); sometimes the diagonal angle of view is also used. When using the device, the visual field is either reduced to the field of view (e.g., in the case of HMDs) or the field of view covers only a part of the visual field. Thus, one criterion for evaluating visual output devices is the size of the field of view. Here, the absolute size of the display is irrelevant. For example, a smartphone has a very small display in relation to a large screen as a projection surface. However, if the smartphone is used in an HMD (see Sect. 5.3.1), the field of view can be much larger than when

standing several meters in front of a screen. The size of the field of view has a major influence on the degree of immersion and thus the sense of presence.

Frame Update Rate

The *update rate* describes the resolution of an output device in time. The output is done in discrete time steps and is specified either in *Hertz* [Hz] or in *frames per second* (fps). The repetition rate can be different depending on the sensory channel.

Latency

Each output device needs a certain amount of time to output the transferred data (e.g., time until the output is refreshed, due to signal propagation delays in cables or due to the processing of data by algorithms), causing a delay. This is called *latency* (see Chap. 7).

Brightness, Luminance and Dynamic Range

Brightness is a subjective measure of the amount of light a user perceives. It is therefore only of limited use for the evaluation of displays. For projectors, the *luminous flux* (in lumen) is usually specified. However, the resulting impression of the user is significantly influenced by the size and nature of the canvas. *Luminous intensity* describes the luminous flux per solid angle (measured in candela). A better way to describe the brightness of planar light sources is therefore the luminous intensity in relation to the area. This describes the *luminance* (measured in candela per square meter). *Contrast* is a measure to differentiate the luminance. The *dynamic range* (DR) or *contrast ratio* describes the ratio between the minimum luminance to the maximum luminance of a display. For visual output devices, luminance and dynamic range are crucial for the capabilities of VR and AR applications: if they are too low, they can only be used in darkened areas (e.g., in a laboratory without direct sunlight). If they are large enough, they can even be used in daylight.

Ambient Light

The ambient light represents the light in the environment of a user or a display. Here, this includes all light in the scene except that emanating from the display itself (i.e., the screen or projector), whether it is sunlight or lamps, directional or non-directional. A bright ambient light usually leads to reduced brightness of the display. Even though the luminance of the light source does not change as a result of this, of course, the perceived amount of light becomes less due to the lower contrast ratio. This more traditional view of ambient light should not be confused with ambient lighting in virtual worlds.

Color Reproduction

To evaluate the quality of a display in terms of colors that can be displayed, the CIE Yxy color system can be used. Figure 5.5 shows this *color space*. In order to describe the display colors, a triangle is drawn in the color system, where the vertices correspond to the three basic colors of the display. The triangle includes all colors that can be displayed (called the *gamut* of the display).

The gamut always covers only a part of the colors perceived by the human eye, which are represented by the area enclosed by the curve. Here, the points on the

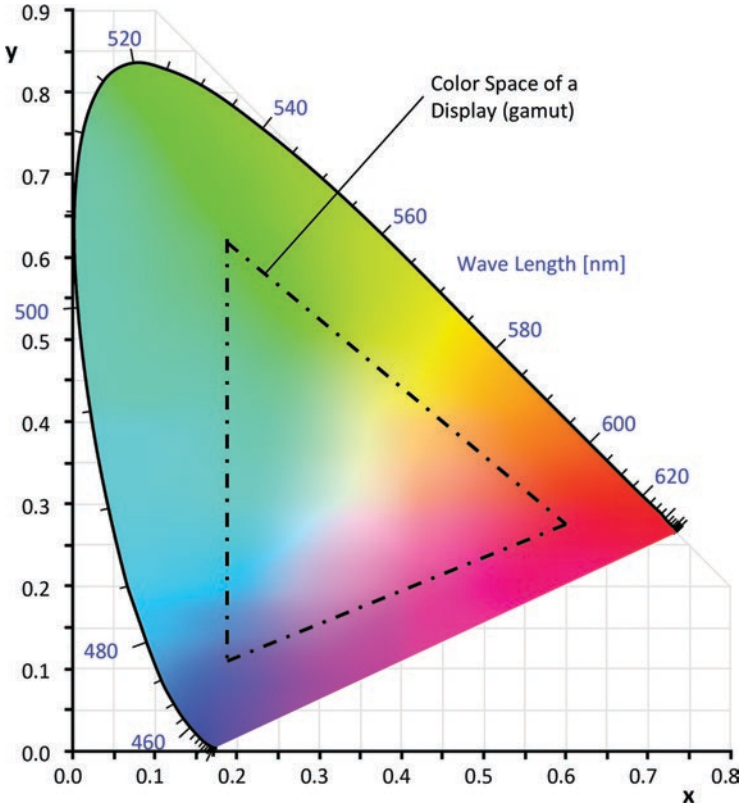


Fig. 5.5 The CIE chromaticity diagram of the colors perceived by humans with the indicated color space of a display (gamut)

curve correspond to the wavelengths of light visible to the human eye, e.g., the wavelength 555 nm corresponds to a bright green color. So you can describe a color in the color system by xy coordinates. Displays with more than three primary colors exist, covering a larger part of the visible color spectrum. These are not in common use though, and therefore will not be discussed further here.

Resolution

A visual display presents content using *pixels*. The resolution of visual displays is specified either by specifying the total number of pixels in (mega) pixels (similar to photos) or by specifying the horizontal and vertical number of pixels separately. The *resolution* of output devices is crucial for the details that can be displayed.

Homogeneity

Output devices should reproduce the virtual world or the virtual parts of an augmented world in homogeneous quality independent of position and direction. With regard to visual output devices, this means that brightness uniformity is maintained as well as that the image sharpness and color reproduction are of constant quality.

Location (in-)Dependence

Depending on their structure, VR/AR systems can be described as stationary, i.e., location-bound, or *mobile systems*. *Stationary systems* are usually permanently installed and cannot be used at another location (or only with substantial effort). An example of a stationary system would be a large multi-sided projection as shown in Fig. 5.2. Reasons for a stationary use can be, size, weight, dependence on connections (e.g., power supply), or the overall effort required for installation (e.g., a complex calibration process – see also Sect. 5.4.3). Mobile systems can be used independent of location. Systems that are used in stationary location, but can in principle be set up at another location with very little effort are called *nomadic systems*. An example would be a VR system consisting of an HMD, two controllers and a tracking system on tripods.

Personal output devices vs. multi-user output devices

Generally, a distinction can be made between personal output devices that can only be used by one person (e.g., HMDs or headphones) and multi-user output devices that can be used by several people at the same time (e.g., projections). However, real multi-user output devices further require consideration of each user's individual viewpoint, which typically is not the case for most projection-based systems, where at most the viewpoint of a single (tracked) user is considered. Additionally, software can be used to give multiple personal output devices access to a shared virtual or augmented world (Collabative Virtual Environment – CVE – or Collaborative Augmented Environment).

Usability

For the application it can be important to what extent users are restricted by the output devices. For example, it may be necessary to put on glasses or attach an HMD to the head. It also makes a difference for the application whether the respective devices are wired or connected via RF technology. The supported room size also has an influence on whether the user can immerse herself in the application or whether she must constantly take care not to exceed predetermined interaction areas. It can also be necessary for the user to always be oriented towards the output device in order to be able to see something. A detailed examination of *usability* is carried out in the context of the consideration of basics from the area of human–computer interaction in Sect. 6.1. The *obtrusiveness* of an output device can be seen as a measure of the extent to which it is considered disturbing. It makes a big difference whether a head-mounted display can be worn like a pair of sunglasses, or whether it can be used like a bicycle helmet due to its weight and dimensions. Ergonomics such as grip or weight distribution can also be critical.

5.3 Head-Mounted Displays (HMDs)

Head-mounted displays (HMDs) are generally understood to be personal displays that are worn on the head directly in front of the user's eyes. Depending on their design and weight, they are worn like glasses or more like a bicycle helmet. HMDs

often have an integrated tracking system or are combined with a tracking system (see Chap. 4) to continuously adjust the viewing direction and viewing position of the virtual camera according to the current position and orientation of the HMD. HMDs usually use binocular optics, so that the user can perceive the contents stereoscopically. A distinction is made between VR glasses, which isolate the user from the outside world and thus facilitate immersion in a virtual world, and AR glasses, which enrich the user's real environment by adding virtual content.

5.3.1 VR Glasses

This section deals with HMDs for VR applications. Figure 5.6 shows typical consumer VR glasses. VR glasses usually use a closed design so that the user is visually completely isolated from his environment, only allowing him to see the virtual world. The field of view here sometimes almost matches the natural visual field. This may result in complete immersion.

Consumer HMDs are often based on a simple magnifier design. Here, a simple magnifying optic is used for each eye, allowing the user to focus on the actual display (see Fig. 5.7). Depending on the individual design, a single display or two separate displays are used.

In a single display, the eyes see different areas of the same display, allowing stereo vision. In two separate displays, they are often slightly tilted towards each other to cover a larger field of view. Through the use of LCD or OLED displays, which are also used in smartphones, such displays now achieve high resolutions combined with high luminosity and reasonable prices.



Fig. 5.6 Typical representative of consumer VR glasses with integrated sensors for tracking (© TU Ilmenau 2019, all rights reserved)

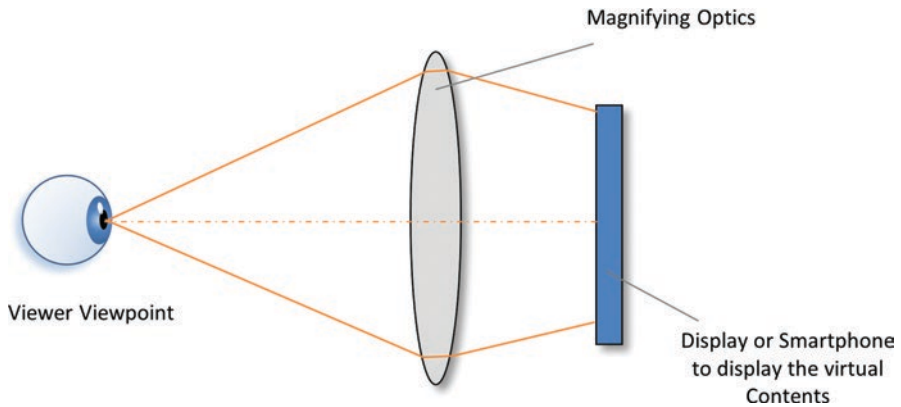


Fig. 5.7 Simple magnifier principle as typical design of current VR glasses

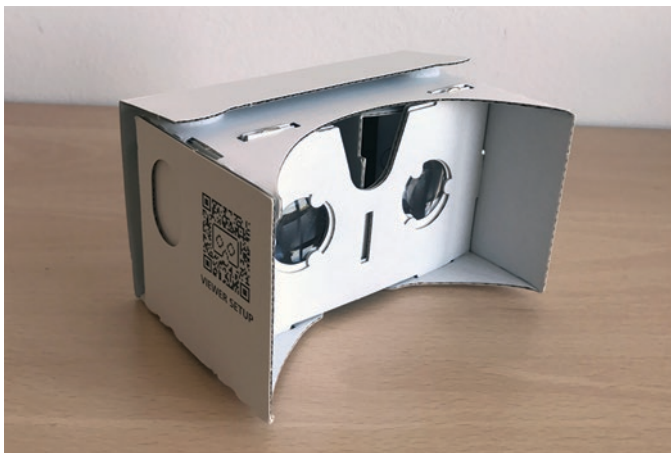


Fig. 5.8 Simple Cardboard HMD using the magnifying glass principle

Low-cost versions do not have their own display. Instead, a smartphone is used as the display, which is inserted into an HMD rack. Cardboard displays (see Fig. 5.8) use only a holder made of cardboard in which two lenses are inserted for the optics.

In contrast, high-end systems may even use multiple displays per eye. See also Sect. 5.3.4. for such an approach. Besides the simple magnifying glass design, alternative designs exist. For example, prism-based VR glasses allow a very compact design because the prism optics significantly reduce the overall depth (see also Fig. 5.11). However, commercially available smartphone displays cannot be used for this purpose. Therefore, displays of this type are not very common for more recent HMDs.

The classical design of HMDs often used optics with mirrors. Here, a mirror and possibly an additional semi-transparent mirror were used to make the virtual content of the display visible to the eye.

5.3.2 AR Glasses

There are two basic approaches to glasses for Augmented Reality, or *AR glasses* for short: *optical see-through displays (OST displays)* and *video see-through displays (VST displays)*. While the first type optically superimposes real and virtual images in the user's view, the latter type uses video cameras to capture the environment and then superimpose it with virtual content during rendering (see Chap. 8). In the following sections, we will first take a closer look at OST displays and their construction methods, before we go on to discuss VST displays in more detail.

Optical AR Glasses

When we talk about AR glasses, we usually mean such OST displays. While the view of reality is always direct and thus immediate, the virtual contents are only optically superimposed. Thus, in contrast to VST displays, there are no limitations in terms of quality and resolution when viewing the real environment. As with VR glasses, a virtual image is generated by a display and enlarged by a lens. However, here it is projected into the user's eye with the help of a beam splitter. A major problem is that the virtual image is projected at a fixed distance, which may differ from the distance of real-world content currently focused by the user.

A general problem of OST-HMDs is the insufficient background contrast ratio in bright environments. Due to the low luminance ratio of the display with respect to ambient light, virtual content is perceived only faintly, i.e., it appears increasingly transparent to reality. In practice, the transparency is reduced accordingly in bright environments to provide the required background contrast ratio for a given front luminance. The different superimposition methods usually lead to a significant reduction of the amount of incident light, so that the surrounding reality appears darkened to the observer. In comparison to viewing without a display, in some cases only about 25% of the light reaches the eye of the observer. This corresponds approximately to sunglasses with a medium protection factor (S2). Due to the low light intensity, most HMDs of this type are not, or are only partially, suitable for sunlight use, even if the transparency is reduced by means of appropriate filters. There are a large number of different designs of OST displays, the most important of which will be briefly explained in the following.

Waveguide Optics Glasses

Strictly speaking, this refers to a whole range of different approaches. These approaches have in common that the light is fed into a largely planar glass body, which acts as *waveguide optics*, and then travels through it as in a fiber optic cable by being reflected from the outside of the glass body. The decisive factor now is how the light enters the glass body (coupling-in) and how it exits it (coupling-out).

Special optical elements are used for this purpose. These result in the light being transmitted and radiated in a previously predefined direction. If the light is fed into the waveguide from the side, there is no need for corresponding elements for coupling-in (see Fig. 5.9). The elements for coupling-out are arranged in such a way that the light leaves the waveguide at the appropriate point in the direction towards the eye. In this way HMDs are possible that look more like conventional glasses due to their flat optics. For a full color display, three layers of light guides must be arranged one above the other, since the individual color channels must be transmitted separately due to the dependence of the refraction on the wavelength of the light. Holographic waveguides applying *holographic optical elements* (HOE) are among the best known representatives of this approach (see Fig. 5.9). By analogy with holograms, light beams impinging on the HOEs generate a secondary light beam in a predefined direction, while the ambient light passes through them without interference. Other approaches include diffractive, polarized, and reflective waveguides.

For AR displays like the Microsoft HoloLens (see Fig. 5.10) or HoloLens 2, the Meta 2 or the display from Daqri, waveguides are used. However, another, mostly curved glass is often used before the actual light guide for protection and shading.

Prism-Based Glasses

Prism-based glasses, which are also used for VR, enable a relatively high light output with a more compact design when compared to other designs. For use as a see-through display, the prism is complemented by a glass body with parallel outer surfaces, so that the ambient light can pass through the glass without being refracted (see Fig. 5.11).

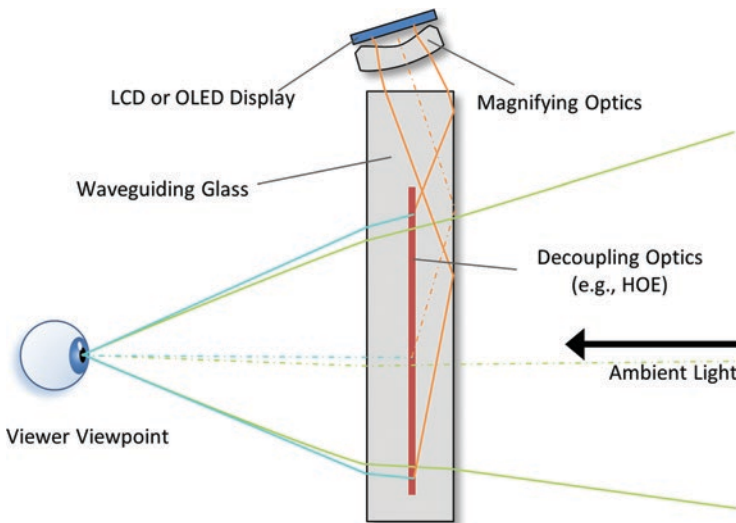


Fig. 5.9 Schematic principle of operation of an OST display based on waveguides (here applying holographic optical elements)



Fig. 5.10 HMD with clearly visible waveguides for the individual color channels (see magnification) (© TU Ilmenau 2019, all rights reserved)

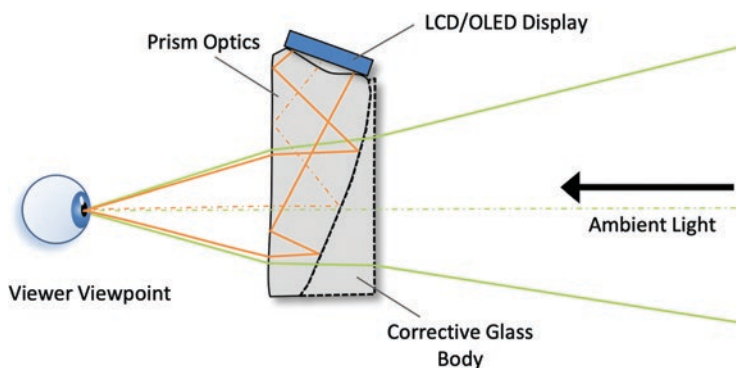


Fig. 5.11 Schematic principle of a prism-based OST display consisting of two glass bodies. The complement is shown as a dashed line

The best known representative of this type of display so far has been Google Glass, although as a so-called SmartGlass it was not really suitable for AR applications due to its small field of view.

Mirror-Based Glasses

The use of semi-transparent mirrors has been the preferred design for OST displays for a long time. The content was displayed on an LCD or OLED display and was magnified by means of magnifying optics located directly beneath or above the display. The classic approach used to apply two semi-transparent mirrors, one mounted vertically at the front and a second mirror at a 45° angle right behind it towards the eye. Thus, the ambient light simply passed both mirrors, while the light

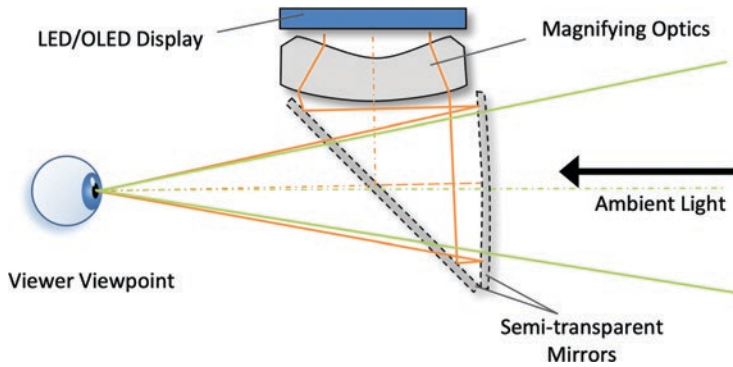


Fig. 5.12 Schematic structure of a classic mirror-based OST display

from the display was reflected by the diagonal mirror towards the front face, then reflected back towards the eye, passing the diagonal mirror (see Fig. 5.12). A simpler approach would use just a single semi-transparent mirror (with the back mirror flipped by 90°). While this also allows for better light output, it requires stronger magnification, which typically results in less compact HMDs.

More recently, mirror-based approaches have regained popularity, as they enable the realization of a simple OST display using a smartphone. The smartphone is usually inserted vertically into a holder in the area in front of the user's forehead. A mirror reflects the image downwards into the area in front of the eyes. Here, there is usually also an optical system in the form of one magnifying lens for each eye. There are also models without lenses, but as the eyes then have to focus on the smartphone display, they do not allow for a relaxed viewing position of the virtual image in space. In the area in front of the eyes there is a diagonally aligned semi-transparent mirror, which on the one hand reflects the image of the display towards the eyes and on the other hand allows a view of the environment. In cheap models a perspex panel is used for this purpose. There are also versions where the upper mirror is omitted so that the smartphone is inserted horizontally. Similar models, but without a semi-transparent mirror, use the smartphone camera to realize VST AR glasses.

Retinal Glasses

Retinal HMDs do not have a display in the actual sense, since the content is projected directly onto the retina (see Fig. 5.13). This is also called a *virtual display*. This approach offers two major advantages: on the one hand, a complex optical system is avoided, and on the other hand, despite an extremely compact design, very large fields of view can be generated because no optics in front of the eye have to cover the displayed field of view. Modulated laser light is used as the light source, which is directed into the eye via a semitransparent mirror or prism. Until now, only monochromatic OST glasses of this type have been commercially available. For a full color display three separate lasers (RGB) would be necessary.

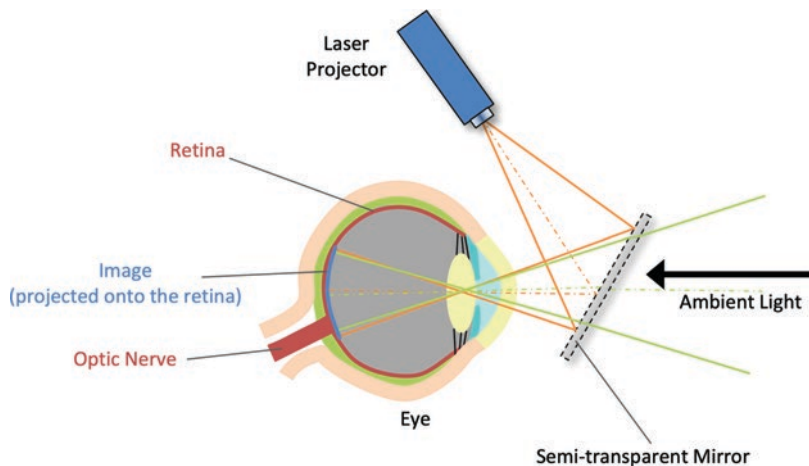


Fig. 5.13 Schematic principle of a retinal virtual OST display (here with mirror)

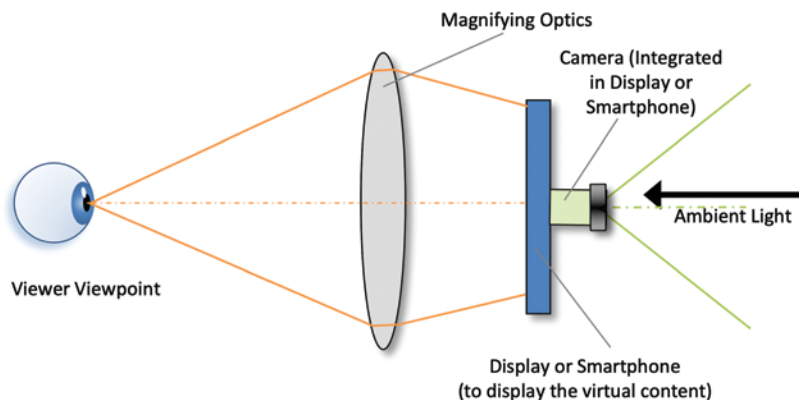


Fig. 5.14 Optical principle of recent HMDs with integrated cameras. Since the cameras are mounted right in front of the eyes, their point of view is almost identical

An alternative design approach replaces the laser projector by an RGB light source and the mirror by a DLP (Digital Light Projector) microdisplay.

Video AR Glasses

Video AR glasses, more precisely Video See Through (VST) displays are basically HMDs as they are used for VR (see Sect. 5.3.1). This means that the user is completely isolated from the environment, at least when the device is completely closed. In contrast to their use for VR, however, a video image of reality is inserted in such a way that the user has the impression that she can look at the world around her through glasses. For this purpose one or two video cameras are attached to the HMD or are directly integrated into it (see Fig. 5.15).

Fig. 5.15 Example of an HMD with integrated cameras



Since the human eye only sees the information projected by the display, the real and virtual content are always in the same focusing plane. Thus, those parts of the real world that are not in focus for the camera cannot be focused by the user either. Furthermore, the perception of the real world is in a reduced resolution and has limited dynamic range due to the camera as well as the display used, when compared to the direct view by the human eye.

The captured video image is correctly inserted as a background image when rendering the scene. Basically, the field of view of the camera must be larger than that of the HMD used. In most cases, it is not possible to position the video cameras directly in the area of the beam path in front of the eyes. Therefore, when correcting the perspective of the camera image, translational and/or rotational offsets often have to be deducted in addition to the rectification and restriction of the viewing angle. Without this, the user will have difficulty in estimating distances, and proportions correctly (at least temporarily until his visual system has adapted). HMDs in which the lens of the camera is positioned directly in front of the eye in the direction of vision, or the light rays arriving there are deflected into the camera, avoid this problem (see Figs. 5.14 and 5.16).

5.3.3 General Characteristics and Properties of HMDs

In this subsection some basic characteristics and properties of HMDs will be reviewed and discussed. Depending on the type of application planned, these can sometimes be decisive for the selection of an HMD to be used.

As already introduced in Sect. 5.3.1 the basic optical principle of VR glasses is that of a magnifying glass. Let us have closer look at its general characteristics according to Melzer and Moffitt (1997). The display, which the user looks at through the lens, is positioned at the distance of the focal length to the lens (see Fig. 5.17).

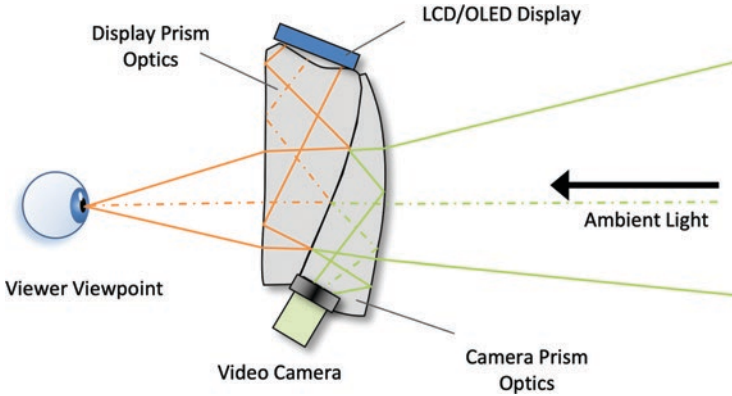


Fig. 5.16 Schematic principle of a prism-based video see-through display with cameras for recording in the viewing direction

Pupil Forming Vs. Non-pupil Forming HMDs

On the optical side, there are two basic approaches to realize an HMD. On the one hand, non-pupil forming HMDs are used, which are based on the principle of a simple magnifying glass. On the other hand, we have pupil forming HMDs, which are based on a projection (Cakmakci and Rolland 2006). An important parameter, which refers to the use of the HMD and is specified for non-pupil forming HMDs, is the so-called *eye motion box* (sometimes also called the head motion box or just eyebox) in its vertical and horizontal dimensions. This is the size of the optical opening of the HMD on the eye side. The larger the eye motion box, the further the position of the HMD can be shifted in relation to the user's eye without restricting the visibility of the virtually projected image. In pupil forming systems, however, a diameter is specified at the optical output of the HMD within which the viewer can see the virtual image. This parameter is called the *exit pupil*. In contrast to the eye motion box, this diameter remains constant regardless of the distance between the user's eye and the HMD optics.

Field of View (FoV)

Based on the optics shown in Fig. 5.17, the field of view can be calculated for the horizontal, vertical and diagonal by eq. 5.1, where F represents the focal length of the lens and S is the size of the display horizontally, vertically or diagonally, respectively.

$$FoV = 2 \arctan \left(\frac{S}{2F} \right) \quad (5.1)$$

Theoretically, the FoV calculated according to eq. 5.1 is independent of the diameter of the lens D . In practice, however, there is the problem that at a higher distance between the eye and the magnifying lens (the so-called *eye relief* L) not all light rays of the display can reach the eye via the lens. In this case, for technical

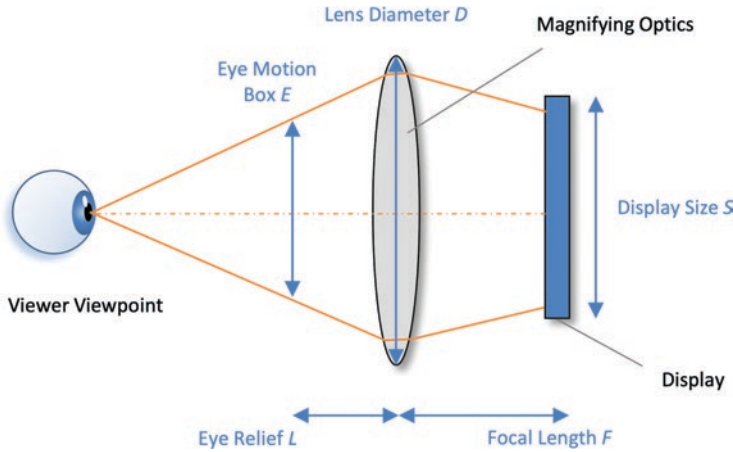


Fig. 5.17 Optical principle of VR glasses, where F represents the focal length, S the size of the display, D the diameter of the lens, E the eye motion box and L the eye relief

reasons, the lens diameter and the eye relief according to eq. 5.2 determine the maximum possible field of view.

$$FoV_{\max} = 2 \arctan \left(\frac{D}{2L} \right) \tag{5.2}$$

Eq. 5.2 is valid for $D < L (S/F)$. HMDs, which optically follow the simple magnifying principle, have an eye motion box E instead of an exit pupil. The size of the eye motion box for the horizontal, vertical and diagonal direction can be determined according to Eq. 5.3 (Melzer and Moffitt 1997).

$$E = D - \frac{LS}{F} \tag{5.3}$$

Accommodation Distance

The *accommodation distance* indicates the distance from the user’s eye at which the virtual image appears. Most optical see-through HMDs have a virtual image at infinity. For a simple HMD using the magnifying glass principle, the relation between the lens position and the distance of the virtual image D_{virt} can be described by eq. 5.4:

$$D_{\text{virt}} = \frac{dF}{F - d} + L \tag{5.4}$$

Here the parameter d is the distance between the lens and the display. If the display is within the focal length of the lens, as shown in Fig. 5.18, the denominator in

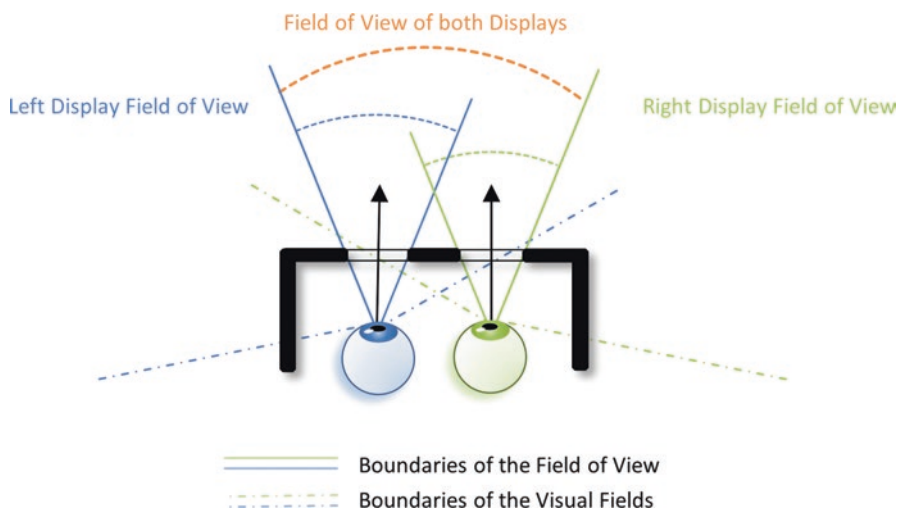


Fig. 5.18 Field of view of an HMD compared to the user's visual field when using binocular AR-glasses in a closed design

eq. 5.4 becomes zero and the virtual image is at infinity. If d is smaller than F , the virtual image is projected enlarged. This means that the projection is larger than the illuminated area of the display. If d is larger than F , the virtual image is projected scaled down.

Interpupillary Distance (IPD)

The interpupillary distance (IPD) is the distance between the two eyes of an observer. It is usually measured from pupil to pupil and in the range of 6 to 8 cm for an adult. Many, though not all, HMDs allow the eye distance to be adjusted to suit the individual user. Otherwise, especially in combination with a small eye motion box, parts of the displayed image may be cut off. The eye distance also has a direct influence on the perception of sizes and distances of the virtual content (see Chap. 2).

Monocular vs. binocular HMDs.

With HMDs, one can basically distinguish between monocular and binocular variants. *Monocular HMDs* have only one display with associated optics for one eye, while the other eye usually remains free. While this can be useful for certain AR applications, it drastically reduces immersion in VR. *Binocular HMDs* have separate optics for each eye, allowing different content to be viewed. Only this enables stereoscopic perception and thus a spatial impression. In contrast to binocular displays, there are also binocular displays in which both eyes look at the same image through separate optics. However, this does not allow stereoscopic perception. If both eyes look at different areas of one and the same display via separate optics (e.g., in smartphone-based HMDs), they usually see different images nevertheless.

Open Vs. Closed HMDs

The design of an HMD also affects the perception of the virtual environment (for VR) or the augmented environment (for AR). Basically, one can distinguish between open and closed designs of HMDs. While the closed design limits the visual field of the observer to the field of view of the HMD, the open design allows unrestricted perception of the environment outside the display. Figure 5.18 illustrates an HMD of closed design using OST AR-glasses as an example.

The illustration clearly shows how much the visual field of the observer is restricted by the field of view of the HMD. Stereoscopic vision is only possible in the area where the fields of view of the display for left and right eyes overlap. This is called the stereoscopic or binocular field of view. Its size in VR glasses depends on the distance at which the display appears to the viewer due to the optics. It can therefore vary between 0 and 100% of the individual fields of view.

Small fields of view are problematic for several reasons. With VR glasses as well as with closed AR glasses they lead to tunnel vision and thus to increased cybersickness due to the lack of peripheral perception. An additional complication is that closed AR-glasses shield the viewer from the perception of a large part of his real environment. This is particularly problematic when used in unprotected areas (such as mostly outdoors), since the perception of stairs, cars, bicyclists, etc. occurs much later than normal.

Monocular HMDs, i.e., those that only superimpose the vision of one eye, allow an unrestricted view of the surroundings, at least with the other eye. In the field of working environments and military application scenarios, such designs (see Fig. 5.19) are therefore strongly represented, whereby a largely open design is usually used here, so that only the display mounting causes a certain restriction of the visual field.

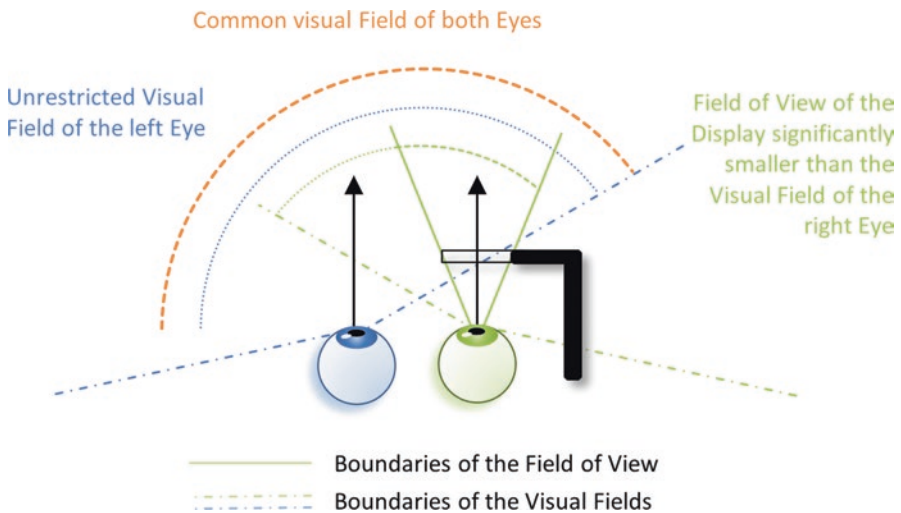


Fig. 5.19 Field of view and visual field for a monocular display (right) in a closed design

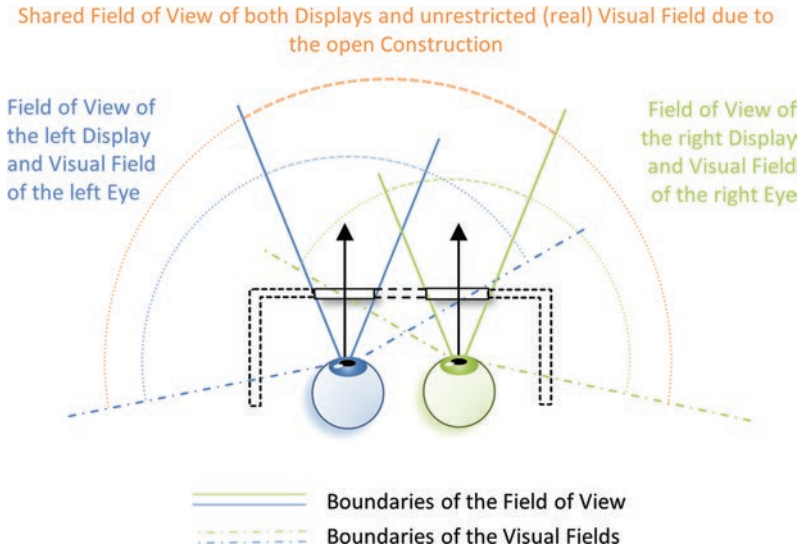


Fig. 5.20 Fields of vision of a binocular HMD with an open design

HMDs in an open design enable users to directly perceive the environment outside the HMD's field of view. Thus, the peripheral vision of the user is not restricted, although virtual content remains limited to the area of the HMD's field of view (see Fig. 5.20). With open AR glasses, it can be disturbing that the area covered by the HMD usually appears significantly darker than the part that is not covered. Furthermore, the limited field of view in comparison to the field of vision causes the problem that virtual objects leaving the field of view of the AR glasses, are only partially displayed at its edges, while the real background remains continuously visible (cf. Figure 5.21). This effect immediately destroys the viewer's impression of a correct registration of the corresponding virtual object in the real world (see frame cancellation, Chap. 2).

Contrast Ratio

As we have previously learned, the dynamic range or contrast ratio CR is the ratio between the brightest and darkest representation. For VR glasses, this is the ratio between the luminance of a maximally bright pixel and a completely dark pixel:

$$CR = \frac{L_{\max}}{L_{\min}} \quad (5.5)$$

For OST-AR glasses (see Sect. 5.3.2), however, the contrast ratio between the luminance of the display (the so-called front luminance) and the background of the real environment is of particular interest. The contrast ratio of the background CR_{back} is thus the ratio of the front luminance L_{front} minus the background luminance L_{back} to the background luminance:



Fig. 5.21 Problems with the display of virtual objects at the boundaries of the field of view with an open design

$$CR_{back} = \frac{L_{front} - L_{back}}{L_{back}} \quad (5.6)$$

When using OST-AR glasses outdoors, especially in bright sunshine (e.g., on an unclouded day), the brightness of the projected image must be correspondingly high so that the virtual content stands out sufficiently from the background (see also Sect. 8.1.2). Indoors, on the other hand, for example, AR-supported assembly work in a factory building, a significantly lower front luminance may be sufficient to provide the same contrast ratio with respect to the surroundings. With OST-AR glasses, the see-through transparency $T_{see-through}$ indicates how bright the user can perceive the real environment or by how much the brightness of the environment is reduced by the HMD, similar to sunglasses.

Distortion

Due to the highly distorting simple magnifying optics used especially in recent consumer VR glasses, the displayed images must be pre-processed (see Fig. 5.22). This is done by applying an appropriately parameterized equirectangular function to the images with the aim of providing the user with an undistorted image after being distorted by the lens. For this purpose, manufacturers often provide corresponding *distortion maps*.

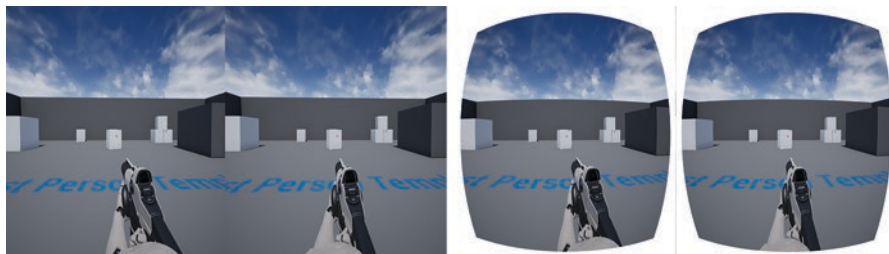


Fig. 5.22 Display of the two partial images for the left and right eyes of the viewer (left without, right with perspective predistortion for HMD)

In order to be able to judge the image quality of the virtually projected image, the horizontal, vertical and diagonal distortions of the virtual image in relation to the original image are compared. Distortions occur if the virtual image does not have the same projection scale in every area. Distortions become noticeable, for example, when the virtual image has the outer shape of a cushion.

5.3.4 *Special HMDs*

Eye Tracking for VR and AR Glasses

With the availability of VR glasses for the consumer sector, a need to capture where the user is looking in virtual worlds quickly arose. This information can be used for investigations of user behavior in user tests as well as for the fixation of virtual objects for selection and manipulation. A further application area is *Foveated Rendering* (see Sect. 7.1.4), in which different display areas are shown in different detail depending on the retinal area on which they are perceived. While a rigid division of the field of view can lead to disturbing effects when focusing on peripheral areas, in combination with eye-tracking it can be ensured that the rendering always takes place in the center of the current viewing direction at the highest quality.

On the one hand, various commercial eye-tracking systems are now available for direct installation in consumer VR glasses. On the other hand, HMDs are increasingly being delivered directly with integrated eye-tracking for user interaction. Examples are the HoloLens 2, the Magic Leap One (see Fig. 5.23) or the HTC Vive Pro Eye. Commercial systems are usually based on the fact that for each eye several infrared LEDs are arranged mostly in a ring around the HMD's optics. The positions of the reflections of the LEDs are then recorded by a camera, which is also mounted directly next to the HMD's optics. Based on the points identified in the camera image, the direction of vision of the eye can then be calculated (see also Sect. 4.5.5).

Fig. 5.23 Optical see-through AR glasses with integrated eye tracking (© W. Broll, all rights reserved)



Multi-Display Glasses

Some high-end systems, like the Varjo glasses, combine multiple displays to achieve very high perceived resolutions (Lang, 2018). The basic idea is to combine a regular, large field-of-view display with a much smaller foveal display that only covers the center of the field-of-view (the fovea). The approach extends *Foveated Rendering* to the usage of high-res foveal displays. The perceived quality is significantly better than one display systems, but the additional effort in design and production results in significantly higher prices. One approach to realize this is the application of a semi-transparent mirror in combination with eye-tracking (see Fig. 5.24).

Adaptive HMDs

All currently commercially available HMDs have the problem that due to static optical elements the virtual image always has a fixed distance to the eyes of the user. However, the distances of the real objects the user is looking at vary. Since the human eye cannot focus on different distances at the same time, one of them is usually out of focus. One possible solution is an adaptive HMD (Herold et al. 2015). Such adaptive HMDs are based on a liquid lens, which makes it possible to adjust the focal length and thus also the distance of the virtual image to the user.

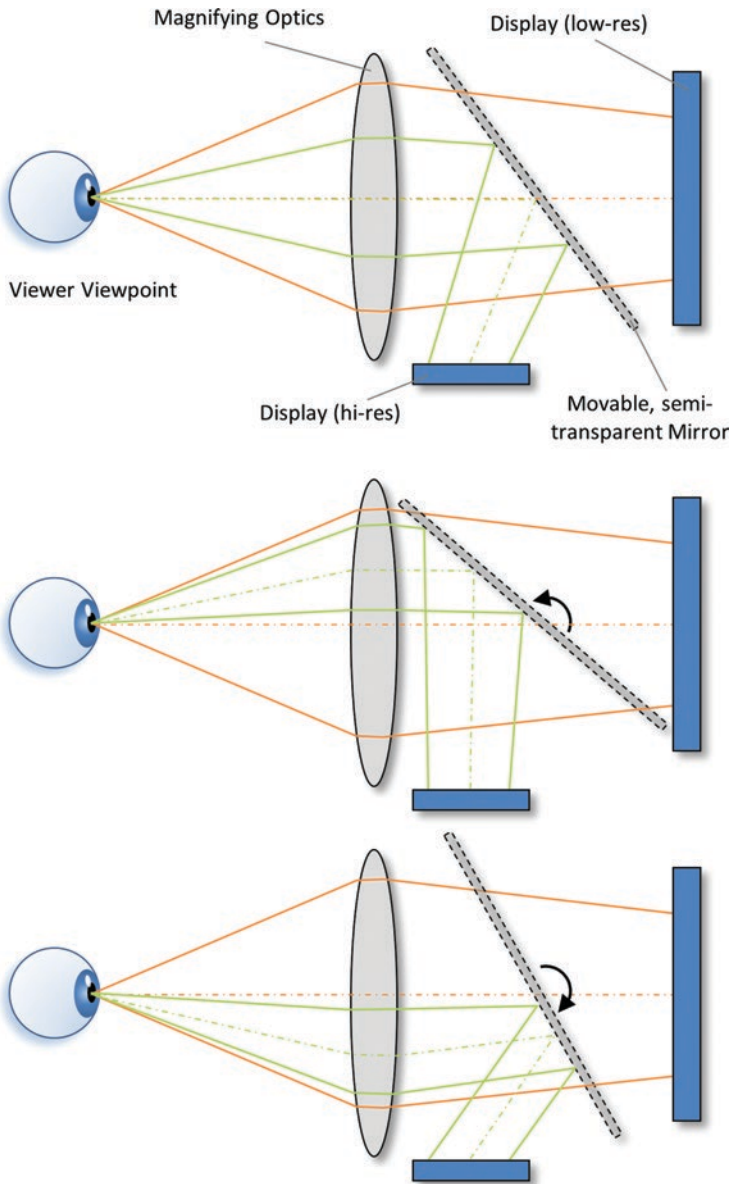


Fig. 5.24 Multi-display system supporting foveated rendering

5.4 Stationary VR Systems

Stationary VR systems use one or more mostly vertically oriented (i.e., standing upright) displays (projection screens or large monitors) for visual output. Depending on the type of system, alternatively or additionally horizontally oriented projection surfaces or monitors or even spherical projection surfaces are used. The output is usually stereoscopic. For the correct calculation of perspective, the user's head is usually tracked. The necessity is easily recognized by the following example: if the user bends to the right or left to look past a virtually represented column, the virtual world must be displayed accordingly. This requires an individual calculation of the images shown on the displays from the user's perspective. This is also the reason why even stationary VR systems are still single-user systems almost without exception (and despite the fact that they are often used by several users in parallel). The one exception is new systems that use extremely high-speed projectors that can display a sufficient number of images per second (a typical example would be 360 fps) to display separate stereo pairs for multiple users.

In principle, AR systems can also be stationary. In particular, spatial AR systems such as projection-based AR are usually stationary. While most of the technical aspects discussed here also apply to them, they are dealt with in Sect. 8.4.

5.4.1 *Single-Sided Displays*

Many stationary VR systems are simple single-sided displays, i.e., a single projection surface as large as possible is used on which the virtual world is displayed stereoscopically. In the simplest case this can also be just a large monitor.

It is crucial for a high level of user immersion that the display's field of view covers as much of the user's visual field as possible. The larger the field of view (FOV), the less often a virtual object from the user's perspective will reach the edge of the display, destroying the spatial (stereoscopic) impression by frame cancellation (see Sect. 2.4.3). This means that the smaller a display area is, the closer the user has to be in front of it, or the larger the display area, the further away the user can be (cf. Figure 5.25).

Vertical and Horizontal Displays

Single-sided displays are usually oriented vertically (upright) so that the user(s) can stand or sit in front of the display, comparable to a 3D cinema. Depending on the application, however, horizontal (lying/tabletop/floor) displays are also useful (as an example see *Responsive Workbench*; Krüger and Fröhlich (1994)). In tabletop systems, virtual objects usually appear to lie on the table or hover above it. Users have to stand very close to the display to avoid frame cancellation.

Both single-sided vertically and horizontally arranged displays can be used to view virtual content with multiple users at the same time (see Fig. 5.26). Usually, however, a perspective correct stereo view is only generated for one user. All other users see

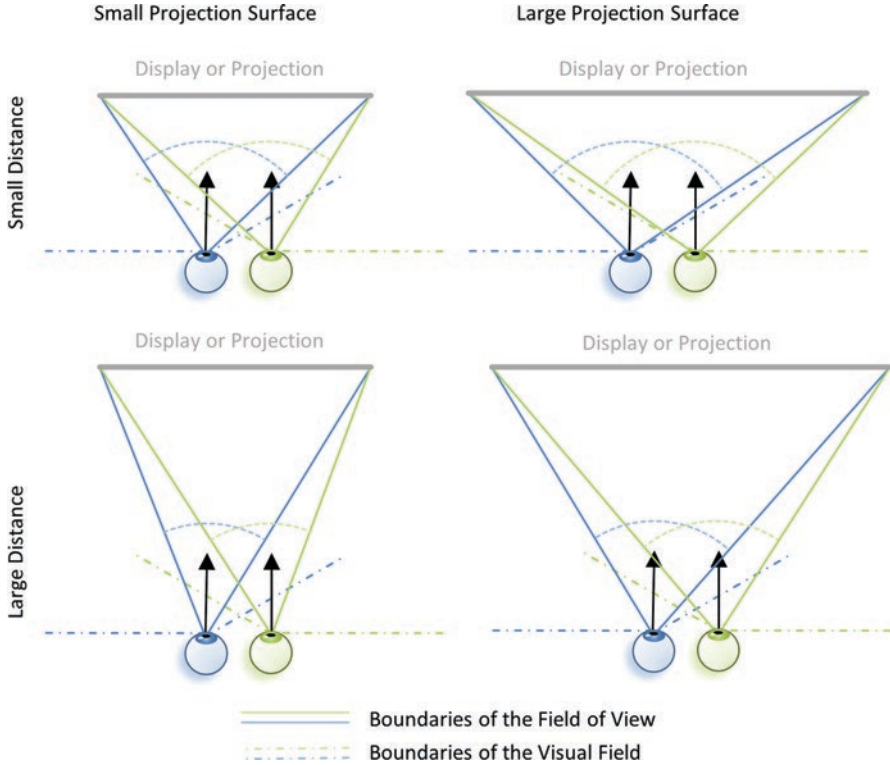


Fig. 5.25 Dependence of the field of view on the size and distance of the display or projection surface



Fig. 5.26 Vertically and horizontally arranged single-sided displays and projections

virtual content stereoscopically, but in a different position. For an interaction with virtual objects in particular, perspective correct stereo presentation is essential.

Front and Rear Projections

If no monitors but projection systems are used for a display, the projector can basically illuminate the projection surface (screen) from the user's side or from the side opposite to the user. If the projection is made from the same side from which the

user looks at the projection surface, this is called *front projection*. If, on the other hand, the projection is made from the opposite side, i.e., the rear side, it is called *rear projection*. With front projections, the user must maintain sufficient distance from the projection surface to avoid obstructing the beam path of the projector. The shadows cast on the projection surface can also lead to frame cancellation.

However, if the user has to maintain a greater distance from the display, this inevitably leads to a restriction of her interaction space (see Fig. 5.27) and at the same time increases the risk of frame cancellation. With rear projection systems, these disadvantages are generally avoided, but a correspondingly larger space is required for the beam path of the projector behind the projection surface. Furthermore, specific, usually more costly, canvases must be used for rear projection. By employing mirrors in combination with ultra wide angle projector lenses or ultra short throw (UST) projectors, the space required for both front and rear projections can be significantly reduced, whereby front projections also benefit from an increased interaction space.

5.4.2 Multi-Sided Displays

With a single flat display, it is difficult if not impossible to achieve complete coverage of the user's visual field by the field of view and thus a high degree of immersion. Accordingly, there are numerous approaches that combine several display surfaces or realize curved display surfaces. Well-known representatives of the first group are so-called *CAVEs* (*Cave Automatic Virtual Environments*) and *L-Shapes*; the second group particularly includes *spherical displays*.

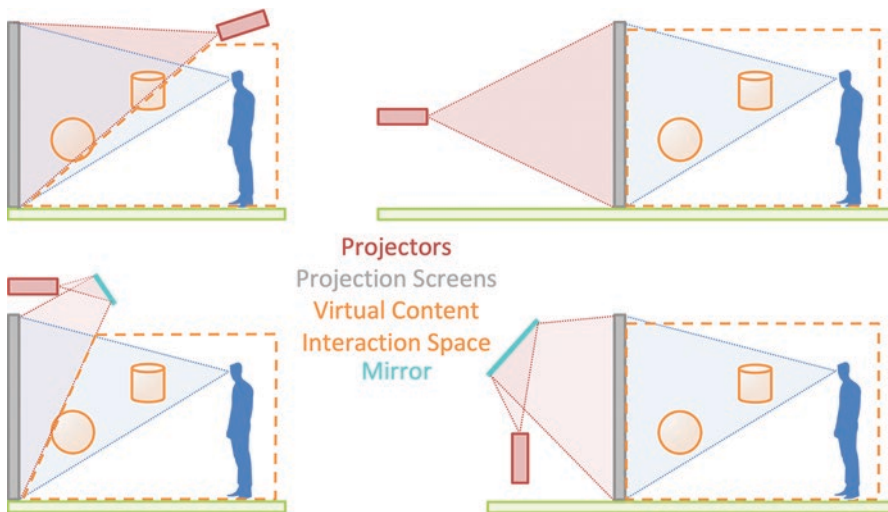


Fig. 5.27 While front projections limit the interaction space of the user, rear projections require considerably more space. Usage of mirrors or ultra short throw projectors can significantly reduce space requirements

L-Shapes

An L-shape uses two displays. One display is usually mounted vertically, while the second display is usually placed horizontally and has an edge directly adjacent to the first display (in side view, the two displays placed next to each other thus resemble the letter L; hence the name). L-Shapes offer the great advantage over single-sided displays, especially in stereoscopic presentation, that the volume for displaying virtual content is significantly larger. Thus, virtual objects can be displayed up to the immediate vicinity of the user, e.g., for hand-based interactions. Frame cancellation, which often occurs in the lower part of a vertically arranged display in single-sided displays, is thus effectively prevented by the second horizontally arranged display (see Fig. 5.28). Similarly, in the case of primarily horizontal displays (e.g., Responsive Workbench; Krüger and Fröhlich (1994)), a second vertically arranged display prevents frame cancellation when viewing virtual objects close to the opposite side of the horizontal display.

For larger L-shapes it may be necessary for the user to stand on the horizontal display. If this is a monitor or panel, the challenge is that the display must not only have sufficient optical properties but also sufficient static stability to reliably support one or even several users.

Spherical Displays

Spherical displays or curved screens (also known as *dome projection* when covering 360°) consist of a *curved screen* on which the image is usually displayed with the aid of several projectors (see Fig. 5.29). The projection surface has the shape of a sphere, a cylinder or a cone, or a cutout of these basic shapes. The projector image must be distorted according to the shape of the projection surface. If several projectors are used (see also the next section on tiled displays), their images cannot be projected without any overlapping. The overlapping image areas must therefore always be adjusted accordingly, i.e., masked by software or physical barriers, making these transitions appear seamless to the user.

Cave

A *CAVE* (*Cave Automatic Virtual Environment*) is a cube-shaped arrangement of displays with the user standing inside the cube (Cruz-Neira et al. 1992). Figures 5.2 and 5.29 show two installations of CAVE-like displays. Depending on how many sides of the cube are designed as displays, we speak of three- to six-sided CAVEs.

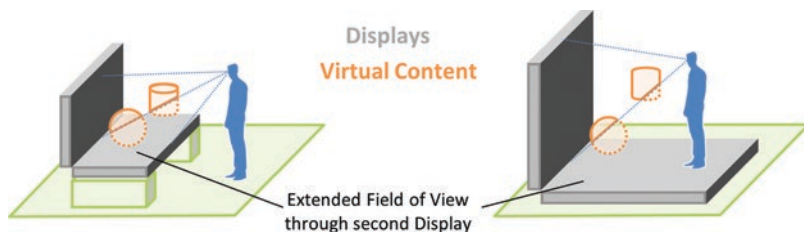


Fig. 5.28 L-shapes expand the working space available for stereo vision and reduce frame cancellation



Fig. 5.29 Example of a curved screen projection. (© Fraunhofer IFF 2013. All rights reserved)

In a six-sided CAVE the user is completely surrounded by the virtual world. In this case, only rear projections can be used, which not only requires a sufficiently large space behind each of the projection surfaces, but also, due to the ceiling and floor projections, equally large space above and below the CAVE (see Fig. 5.30). If a projection from above is used for the floor, this first implies that a ceiling projection is no longer possible and second that the users are right in the beam path of the projector. However, such floor shadows are often perceived as less disturbing, since users are used to casting a shadow on the floor in reality. Also, stereoscopic representation is sometimes omitted for floor projection. For stereoscopic representation, CAVEs mostly use active methods, i.e., shutter glasses (see Sect. 5.4.4). Case study 9.7 describes some of the challenges involved in the construction of a CAVE.

An advantage of a CAVE is that the user can move around in it as in reality (at least within the limits given by the surrounding projection surfaces). Another advantage of the CAVE compared to VR glasses is the self-perception of the user's own body. A fundamental problem of CAVEs is that the representation can generally only be calculated correctly only for the position of a single user based on their point of view. For all other users inside a CAVE, this results in a disturbing offset at the boundaries between the projection surfaces (i.e., the edges of the cube). In the best case, this will only lead to frame cancellation if parts of a virtual object extend over several projection surfaces. For many of these users, however, this increases the probability of developing symptoms of cybersickness (see also Sect. 2.4.7). One way to overcome this limitation is to use high-framerate projectors and custom stereo glasses (see also Sect. 5.4.4). In this configuration the projectors can display enough images for multiple (typically two or three) users to provide each user with their own pair of stereo images, in combination with tracking everybody's head

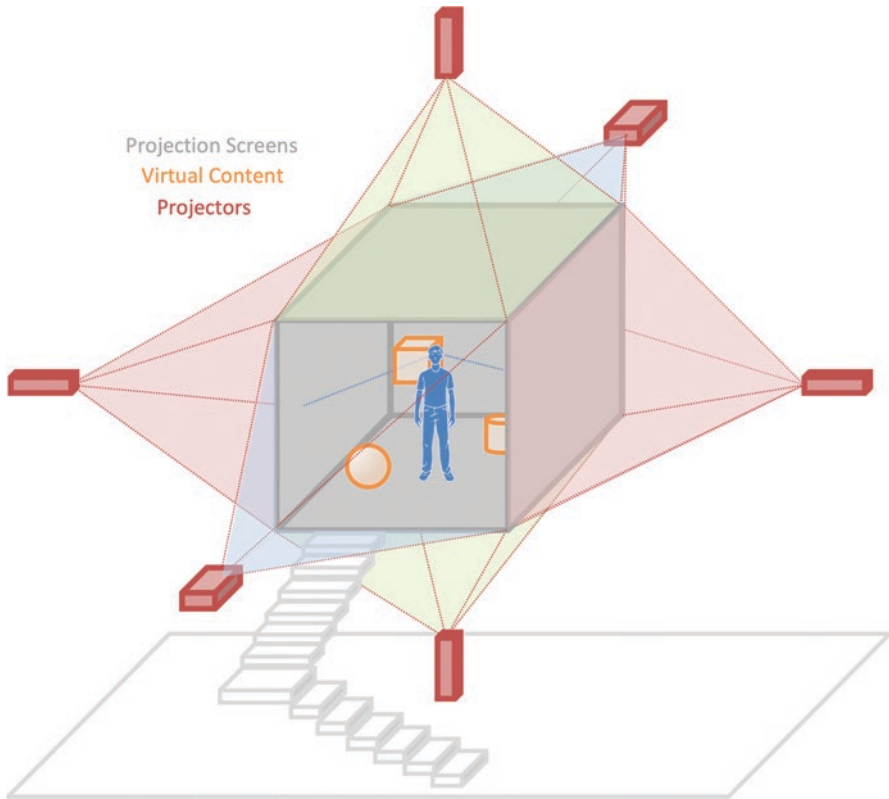


Fig. 5.30 Layout of a six-sided CAVE

resulting in the correct depth perception for each user. As of today, no CAVEs using these projectors have been built, but several are under construction.

5.4.3 Tiled Displays

Stationary VR systems often use displays that are as large as possible. The reasons for this are on the one hand that a large display offers a larger field of view at the same distance from the user and thus results in higher immersion, while on the other hand a larger number of users are able to use such a system at the same time. Since the resolution and brightness of projectors as well as the resolution and size of monitors cannot be increased at will, the size that can be achieved with a single display of a certain quality are limited.

To increase the resolution or to realize large projection and monitor surfaces with high resolution and/or high luminous intensity, a division into several displays (i.e.,

several projection systems or monitors) appears reasonable. We refer to this as *tiled displays*.

The main idea here is to combine several display systems in such a way that the user perceives them as a single, larger system. The idea as such is not new and has been used for a long time in military flight simulators (here to completely cover a dome projection) or for so-called video walls. With this approach, the limitations of a single display can be bypassed to achieve larger sizes and/or higher resolutions.

As the number of individual tiles usually quickly exceeds the number of outputs of a graphics card, tiled display systems usually use a cluster of computers to calculate the output images. Generally, the fewer tiles a computer has to serve, the higher the performance can be. Conversely, the synchronization effort increases with the number of computers used.

Tiled displays can occur with both projection systems and monitors. Both approaches are presented in more detail in the following. Also, tiled displays always have to be calibrated to create the impression of a single display surface. Basic calibration methods for geometric calibration and for achieving brightness and color uniformity are therefore also briefly discussed in the following sections. Various approaches to specific setups exist, e.g., Bajestani (2019) and Okatani and Deguchi (2009).

Tiled Projections

Figure 5.31 shows the C6, a six-sided CAVE built at Iowa State University in 2006 using *tiled projections*. It was built using 24 projectors with 4096×2160 pixels each. A 2×2 raster per side with two projectors per tile is used for the stereo display, which combined can display a stereo image with over 100 million pixels. Each



Fig. 5.31 CAVE C6 at the Iowa State University



Fig. 5.32 Tiled wall using the example of the HEyeWall with 48 projectors. (© Fraunhofer IGD 2013, all rights reserved)

individual pixel is only 0.7 mm in size, a size that is close to the resolution of the human eye at typical viewing distances of 1–5 m.

Instead of fewer very high-resolution and light-intensive projectors, smaller tiles can be used with a correspondingly higher number of projectors, but with lower resolution and light intensity. An early example of such an approach was the HEyeWall shown in Fig. 5.32, a system with 48 standard projectors installed at Fraunhofer IGD in Darmstadt in 2003. Figure 5.33 shows the view behind the screen so that the arrangement of the projectors is visible as a 6×4 grid with two projectors per tile.

Tiled Monitors

Tiled displays consisting of monitors can also be used to realize large display areas with a high resolution. Compared to projectors, monitors have a significantly lower price per pixel and, due to their small installation depth, allow high-resolution systems, even if there is significantly less space available. Figure 5.34 shows the Reality Deck at Stony Brook University, which was built in 2012. The system used 416 standard monitors, each with 2560×1440 pixels, so together the whole system can display 1.5 billion pixels simultaneously. As shown by the figure, it is important that the individual monitors have a seamless display. Otherwise, the impression is quickly created that the user is looking through a grid at the virtual world. With stereoscopic displays, a gap between the monitors, which is clearly perceived by the user, very quickly results in frame cancellation. Tiled monitors are suitable for single-sided display surfaces as well as for CAVEs, L-shapes and cylindrical spherical VR systems.

The tile approach is very well suited to overcoming the limitations of individual display systems in terms of resolution, brightness or price. But while the basic idea

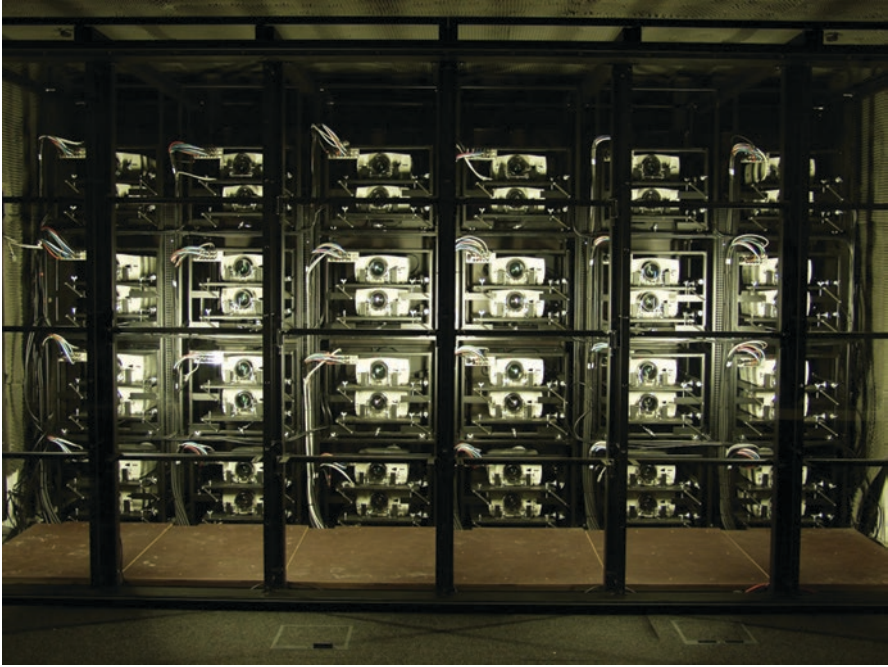


Fig. 5.33 HEyeWall setup. (© Fraunhofer IGD 2013, all rights reserved)



Fig. 5.34 Tiled wall of monitors illustrated by the Reality Deck at Stony Brook University

is very simple, the details require a lot of effort. As a result, the use of tile systems for high-quality applications is either limited or relatively costly. In particular, the calibration of the different display tiles in terms of geometric alignment as well as homogeneity and color representation can very quickly become a significant time and cost factor that is quickly overlooked, or at least underestimated. However, if the method is applied correctly and carefully, extremely impressive display systems can be developed, showing where the journey into virtual worlds may lead.

Geometric Calibration

As soon as several individual displays are to be tiled, geometric consistency is no longer automatically given. A horizontal line that is one pixel wide and runs across all display tiles is not automatically at the same height on each tile. This continuity must be explicitly established.

Under favorable conditions the geometric calibration can be solved purely mechanically. For this purpose, a fixture is used that allows exact mechanical positioning and orientation of the individual display tiles. This requires accuracies in the sub-millimeter range, corresponding to the pixel sizes for high-resolution displays. Obtaining this mechanical accuracy over a large display such as a HEyeWall is a considerable amount of work, which can cancel out a significant part of the price advantage due to installation costs.

This task is further complicated by the inherent assumption that the display tile is geometrically correct in itself. In a conference room it is virtually impossible to see whether the center of the projection is a few pixels higher or lower than the edges, or whether the left edge is a few millimeters larger than the right edge. When several projections are put together, such inaccuracies quickly become obvious. A purely geometric-mechanical calibration cannot always correct such errors, since many variables, such as image border size, squareness and line straightness, depend on each other and cannot be changed independently.

This is especially important if the projection is to be made on an uneven surface (e.g., for spherical displays). A mechanical correction is no longer possible here. The alternative is a correction in the image creation software. There are different approaches possible. The most common is the texture distortion method, in which the image to be displayed is first rendered into a texture and this texture is then displayed on a grid that corrects the geometric inaccuracies of the display. This method is extremely flexible and can correct a wide range of geometric problems. However, it also has some disadvantages. First, the correction must be done within the image creation software, i.e., only software that has knowledge about the display can be used. On the other hand, it involves a (slightly) increased rendering effort, since the image must first be rendered into a texture and then displayed. In many modern systems, however, this is done anyway to produce high-quality images (e.g., in High Dynamic Range Rendering), which is even possible without reducing the refresh rate. Due to the fact that the image is displayed using a texture, however, texture filtering must also be performed, which may result in a certain degree of inaccuracy and image blur.

The biggest challenge, however, is to create an appropriate correction grid. For small systems this can be done manually (and especially in flight simulators this is not uncommon). For larger systems, however, the effort quickly becomes unreasonably high. In such cases, image processing methods that automatically generate corresponding correction grids from test images can help. Nevertheless, this is not a trivial problem and corresponding calibration systems are a price factor that (again) should not be underestimated.

After all these steps the system is now geometrically correct. Straight lines are straight, objects of the same size on all tiles are the same size, etc. Nevertheless, there remain other problems that have to be solved to get a uniform display.

Brightness and Color Uniformity

Besides geometrical problems, projectors also have problems with the *uniformity* of their brightness distribution. These stem from the geometric properties of the light source-lens-screen system, such as vignetting, where the image becomes darker towards the edges. With a single projector, this effect is much less noticeable, since there is no comparison image past the edge of the screen. However, if several tiles are arranged next to each other, the bright-dark-bright transition becomes much more visible. Vignetting is only caused by the projector and lens: it is independent of the viewer's point of view.

Vignetting already occurs with a single projector. When several projectors are used together, production variations in the projectors and especially in the lamps are added. Two identical projectors placed next to each other with the same settings do not necessarily have to be equally bright (and they usually are not). To achieve the impression of uniform brightness, each projector must therefore be individually adjusted. While this is possible using the naked eye, it will not give very accurate results, because the eye can adapt very quickly to different brightness levels. Good results can only be achieved with special light meters.

Another brightness effect comes from the properties of the canvas. Most screens for projections are not perfectly diffuse, i.e., light coming from behind is not emitted uniformly in all directions (see Fig. 5.35). Almost all commercially used screens have a *gain factor* that ensures that more light is emitted to the front than to the sides.

Since the viewer of a normal projection practically never looks very oblique from the side, this arrangement makes sense, because more light reaches the viewer. In the case of tiled projections, however, this ensures that in the transition area between two tiles there are very clear differences in brightness, even if both projectors emit exactly the same amount of light. In Fig. 5.35, the viewer looks directly into the left projector and therefore sees an area of the screen with high gain. The area of the right projector is seen at a much larger angle and therefore in an area of the canvas with low gain. Thus, at the point where the projection areas meet, a clear difference in brightness becomes visible. To make matters worse, this difference is dependent on the angle of viewing: when the viewer moves in front of the screen, one area becomes brighter while the other becomes darker. This makes a uniform image impression practically impossible, the only solution is to use extremely diffuse canvases, which then result in a rather dark projection.

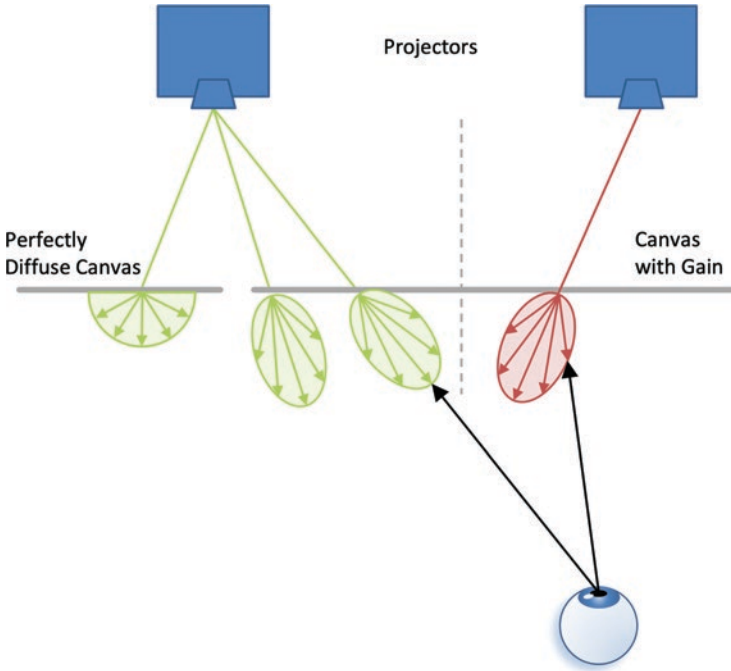


Fig. 5.35 Brightness discrepancy in the transition area between the images of two projectors due to non-diffuse projection canvas

The transition area between tiles is also critical for another aspect, that of overlapping. There are two alternatives for creating the transition between two tiles: either without overlap (*hard edge*) or with overlap (*soft edge* or *blending*). With hard edge, the projectors are arranged in such a way that the transition from one projector to the next is hard: the last pixel of one projector is immediately followed by the first pixel of the other projector. To make this possible, all components of the system (projectors, projector mounts, canvas, etc.) must be extremely stable. Even the slightest movement in the sub-millimeter range can cause a gap to appear between the two projections, which is clearly visible as a black line, or the projectors can overlap and the result can be seen as a bright line in the image. The HEyeWall (Figs. 5.32 and 5.33) was a hard edge system, so special attention had to be paid to the stability of the screen. For this purpose, precisely adjustable baffles were installed, which made it possible to avoid overlapping.

The alternative is to allow overlapping of the projection areas. This creates an area where both projectors beam onto the canvas. To prevent this area from appearing artificially brighter, the displayed image must be adjusted so that one projector is increasingly faded in and the other is faded out in the overlap area. This adjustment is usually achieved by a blend mask that is placed over the image after the rendering process. The C6 is a soft-edge system in which the two projectors per side overlap by approximately 220 pixels.

The overlap prevents the formation of gaps when the canvas is deformed or moved, and reduces the gain problem. In the transition area, the user no longer sees only the image of a single projector, as the projector images merge seamlessly. The main problem with overlap is when dark images or backgrounds are displayed. Modern LCD or DLP projectors cannot display true black because they rely on filters that attenuate the light from the lamp. These filters are never perfect, so a certain amount of residual light always penetrates. In the overlapping areas a double (at the inner corners a quadruple) residual light is therefore visible. As long as only bright images are displayed, this can be masked, but as soon as darker areas appear at the edges/corners, the overlapping and thus the tiling becomes clearly visible, which considerably disturbs a uniform image impression.

While brightness is only a one-dimensional problem, color uniformity requires three dimensions to be matched. This is already apparent within a single projector. LCD projectors in particular often show significant color differences between different areas of an image. If color differences already occur within an image, it is not surprising that massive color differences often occur between several projectors. To achieve a high quality result, these differences must be compensated. This is a much more complex process than brightness calibration and is practically impossible to do effectively manually.

5.4.4 Stereo Output Methods

To support stereoscopic vision (see Sect. 2.2.1) with the goal of making a virtual world stereoscopically experienceable for the user, each eye of a user must be provided with an individual view. While in binocular HMDs this is done by separate optics for each eye, in monitors or projection systems both eyes basically see the same display. Therefore, additional methods for channel separation between the left and right eyes have to be applied. The individual methods used for this purpose are therefore briefly presented below, whereby individual advantages and disadvantages in each case will be highlighted.

Anaglyphs

Anaglyphs are an approach to stereo imaging in which the two partial images are colored differently for the left and right eye – in the original approach, one in red, the other in green. Both images are then combined into one image by superimposition. Red-green glasses are used for viewing. Here, a red filter is placed in front of one eye and a green filter in front of the other, so that each eye only sees its respective partial image. The two colored partial images complement each other to form a stereoscopic grayscale image. For the observation of a usually colored virtual world the approach is therefore not suitable in this original form.

However, the approach can be extended to color image pairs. In this case, red-cyan glasses are usually used instead of the red-green glasses mentioned above. In a display or projection, each individual pixel usually consists of one red, green and

blue subpixel (RGB). For the color anaglyph process, the two images are now divided according to their subpixel assignment (see Fig. 5.36). For one image, only the red channel is used, while the green and blue channels (green + blue = cyan) are used for the other image. The problem is that objects whose color values are only displayed in one subframe cannot be perceived stereoscopically (see also Fig. 5.36). The problem of a unilateral representation can be reduced by a suitable color selection of the objects.

The division does not necessarily have to be along the subpixel boundaries, but a color image can be divided along any complementary colors (e.g., yellow/blue or green/magenta). Of course, the corresponding color filters must then be available for the glasses (see Fig. 5.37). To calculate the partial images, the RGB color value is linked to the respective filter color by a bitwise AND operation with each pixel of the corresponding image.

Polarization

Polarization is a widely used approach to realize stereo vision using channel separation. The method is used in the majority of 3D cinemas. The approach uses the characteristic of light waves to oscillate in different directions. Polarization filters allow only light waves with a certain oscillation direction to pass through. In

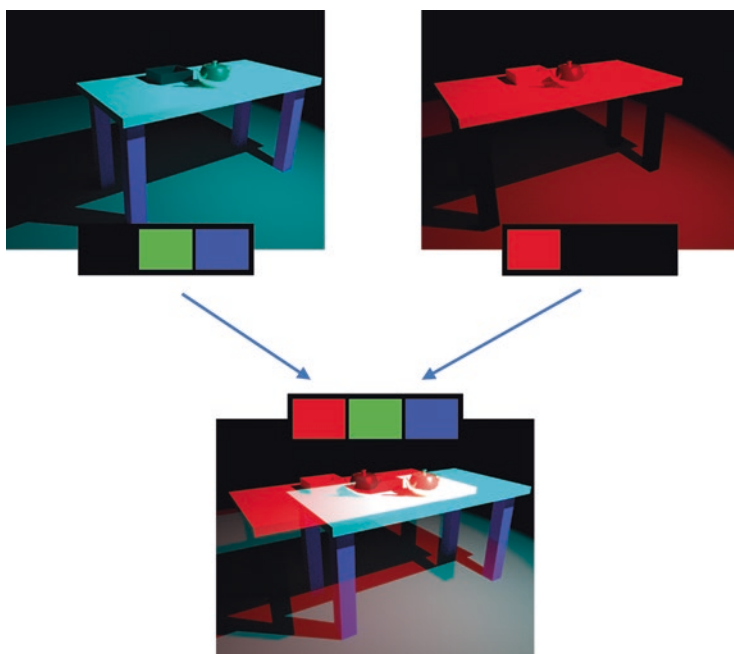


Fig. 5.36 Distribution of pixels along the RGB subpixels (left) and color anaglyph display of a 3D scene with red-cyan channel separation. The problem with this approach can be seen in the dark blue table legs, which are only present in the left subframe. (© Rolf Kruse, FH Erfurt 2019, all rights reserved)

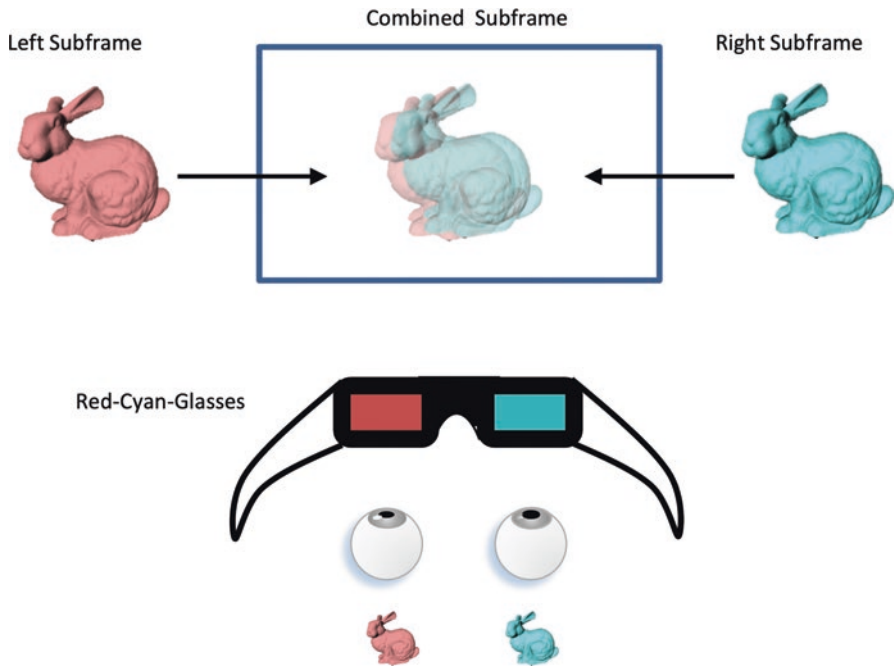


Fig. 5.37 Superimposition of partial images and channel separation by means of color filters for the anaglyph method

general, there is a distinction between systems that use linear polarization filters and those that use circular polarization filters.

For *channel separation*, typically two projectors (or possibly one projector with two lenses) are required per screen. A polarizing filter is mounted in front of each lens, whereby these are rotated 90° to each other. Thus, one polarizing filter, for example, only allows the horizontally oscillating part of the light to pass, while the second only allows the vertically oscillating part to pass. Since the two partial images overlap on the projection surface, they are perceived simultaneously by the viewer. Polarization glasses are now used to separate the channels of the images for the left and right eyes. Here, the two polarizing filters in front of the eyes are aligned in the same way as those on the lenses. If the polarization axes are exactly the same, each eye only sees the corresponding partial image, enabling stereo vision (see Fig. 5.38).

Due to this approach, however, the procedure is very susceptible to *crossstalk*. If the user tilts the head just a little to the side, the polarization axis changes and a *ghosting* of the other channel results. Instead of a horizontal and vertical alignment, combinations of $45^\circ/135^\circ$ are often used, which, however, has no advantage with regard to the problems described above.

The use of *circular polarization filters* solves this problem. Here, a distinction is made between left- and right-turning light waves. This is not influenced by the head

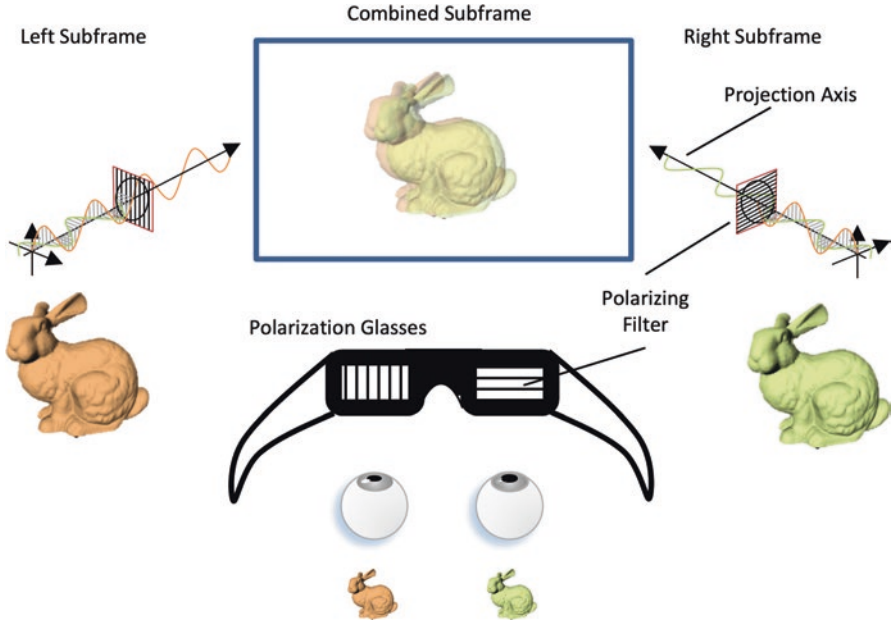


Fig. 5.38 Superimposition of the partial images and channel separation by means of linear polarization filters

tilt, so that crosstalk can be avoided as far as possible. However, circular polarization filters are much more expensive. Therefore, only linear polarization filters are used for “disposable” (cardboard) glasses. A further disadvantage of polarization-based approaches is that the projection surface must retain the polarization. This is only the case with high-quality metal-coated screens, which limits their use and, like the requirement for two projectors per projection surface, increases the costs even further. Since polarization filters generally filter out at least half of the light, only 50% of the light from a projector reaches each eye.

Wavelength Multiplex

The *wavelength multiplex* method, also known as *interference filter* method, uses dielectric interference filters for channel separation. Each filter is based on several coupled resonators, which filter out three very narrow frequency ranges in the three primary colors red, green and blue (see Fig. 5.39).

By mixing the respective primary colors, full color images can be created. By using different frequencies for the three primary colors, the superimposed partial images can then be separated again into two channels, i.e., one for each eye. Similar to a polarization-based channel separation, a filter pair is used for each of the two projectors (or a projector with two lenses) and an identical filter pair for one pair of glasses. In contrast to polarization methods, the wavelength multiplexing method does not require any special characteristics of the projection surfaces. A further advantage is the low susceptibility to crosstalk. A disadvantage is the color shift

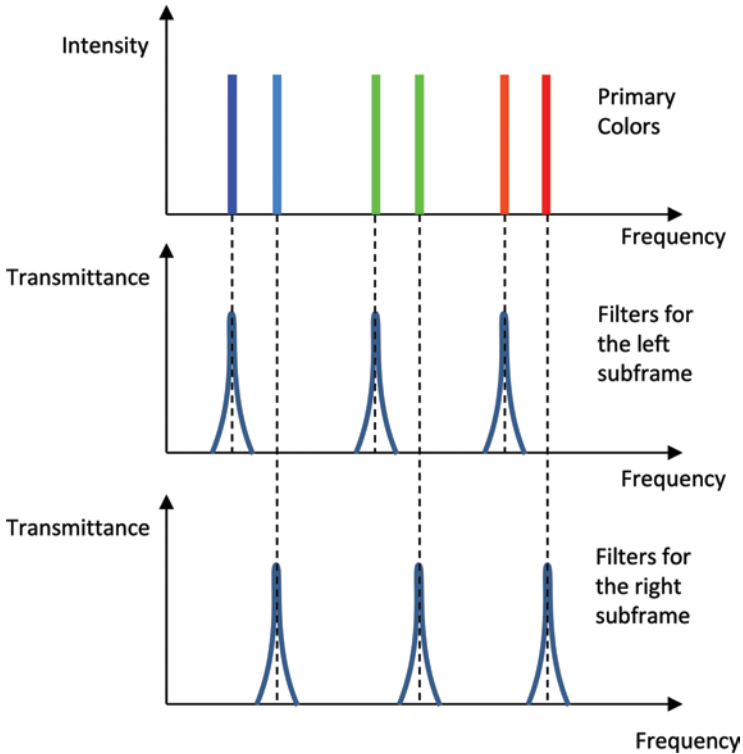


Fig. 5.39 Composition of the two partial images from three different base colors each

between the image for the left and right eyes due to the three different primary colors. To avoid this, the color of the images to be output can be adjusted so that they lie exclusively within the range that can be displayed with both primary color triples (see Fig. 5.40).

However, this further limits the total color space available. The wavelength multiplexing method also allows the independent display of more than two channels. In this case, there are always two channels used for each user, which allows n users to see a stereo image correctly calculated for their individual point of view. For these n users, $2 \times n$ different basic color triples are required and thus a corresponding number of filter types as well as projectors and n different types of glasses with two different filters each.

Shutter Glasses

Besides polarization glasses, *shutter glasses* are another widely used method for stereo output. They are also partly used in 3D cinemas and for 3D TV sets. While the methods presented so far were based on superimposition and subsequent channel separation based on filters, shutter glasses display the partial images in time sequence. Here, the left eye sees its partial image for a short time and shortly afterwards the right eye sees its corresponding partial image. Because the change occurs

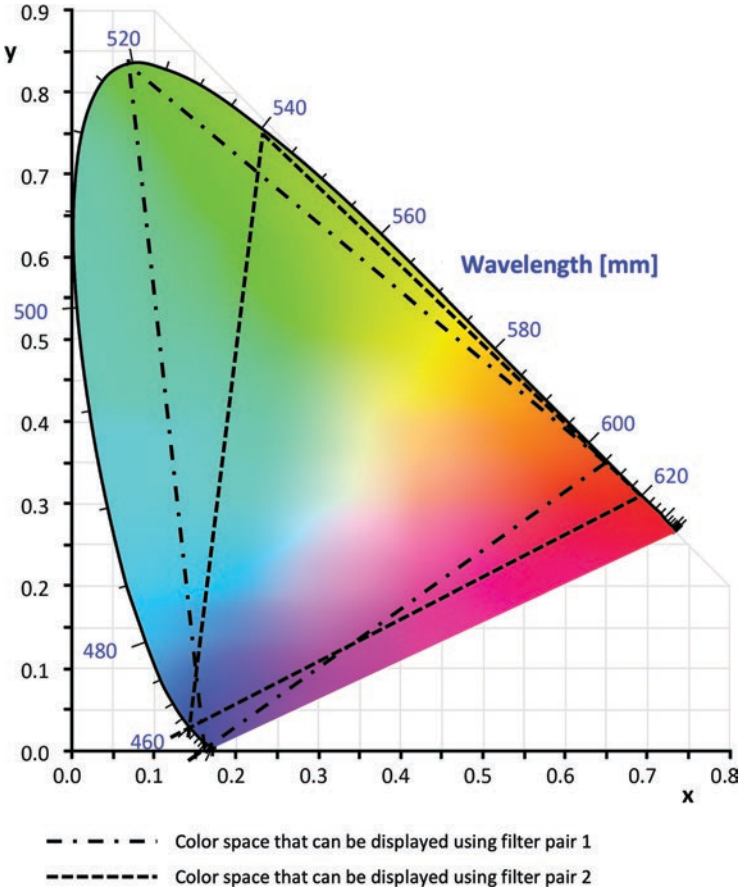


Fig. 5.40 CIE standard valence system with the primary colors of two filters using the wavelength multiplexing method

at a high frequency, the brain is still able to fuse the two partial images into one stereoscopic image, even though they are not perceived simultaneously at any time. To ensure that each eye only sees the partial image intended for it, shutter glasses use two LCD shutters (hence the name). This always covers the eye whose partial image is currently not displayed, so that it cannot perceive any image (see Fig. 5.41). Synchronously to the change of the partial image, the corresponding LCD shutter is opened while the shutter of the other eye is closed. Due to the active switching, this is called an *active stereo method* (in contrast to the *passive stereo methods* using filters). Due to the time-sequential display of the two partial images, their frame rate has to be twice as high to achieve the same overall frame rate as with the passive methods.

Proper synchronization between the shutter glasses and the image output is crucial for channel separation. While earlier systems were primarily synchronized via

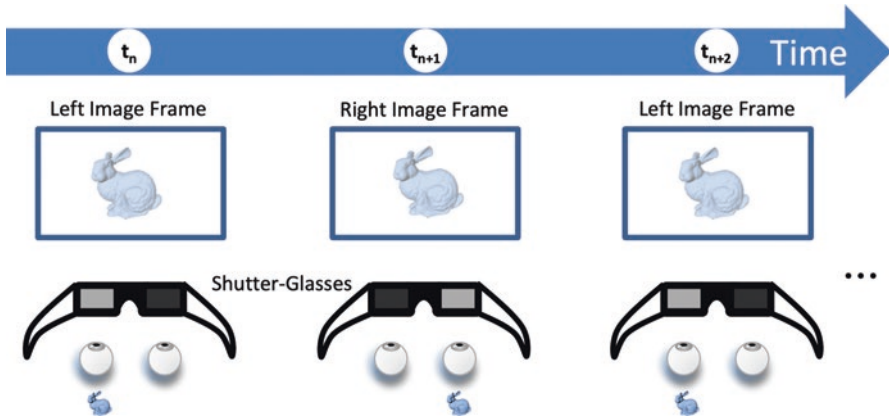


Fig. 5.41 Time-sequential display of the left and right partial image synchronously to the alternating opening and closing of the shutter LCD in front of the left and right eye, so that each eye only sees the partial image intended for it

infrared, which is in principle susceptible to occlusion or further interference, nowadays synchronization via radio-based procedures, especially based on Bluetooth, has become generally accepted. An alternative is synchronization via a white flash (known as *DLP link*). Here a very short, completely white image is shown, which is detected by a photo diode attached to the shutter glasses and used for synchronization. The duration is so short that the user is not consciously aware of this white flash.

Lenticular Lenses

Lenticular lenses are a method to make different (partial) images visible depending on their direction of view. The simplest variant of this are so called “wobble images”, which allow you to view simple animations consisting of very few frames. 3D postcards are based on the same principle. In both cases a prismatic grid consisting of lenticular lenses arranged in vertical rows is used. Each prism covers at least two pixels. Depending on the viewing angle, one or the other pixel becomes visible (see Fig. 5.42).

For stereoscopic output, a prism foil is glued to a screen with pixel accuracy. If the observer is vertically in front of the display at the correct distance, he sees one subframe with one eye and the other subframe with the other eye (see Fig. 5.42).

The advantage of this method is that it does not require any form of glasses, which is why it belongs to the so-called *autostereoscopic methods*. A disadvantage of this method, however, is that the resolution of the display is reduced by half horizontally. Another problem occurs when the user moves to the side or changes their distance to the display. This can lead to the channel separation not working or only working in a limited way (the already mentioned crosstalk). Tilting the head may also cause crosstalk. In principle, the procedure also works for several users at the same time. However, if several users are grouped around such a display, it must be ensured that an individual view is also possible with each eye from other viewing angles. This is achieved by using larger prismatic grids in which each lenticular lens

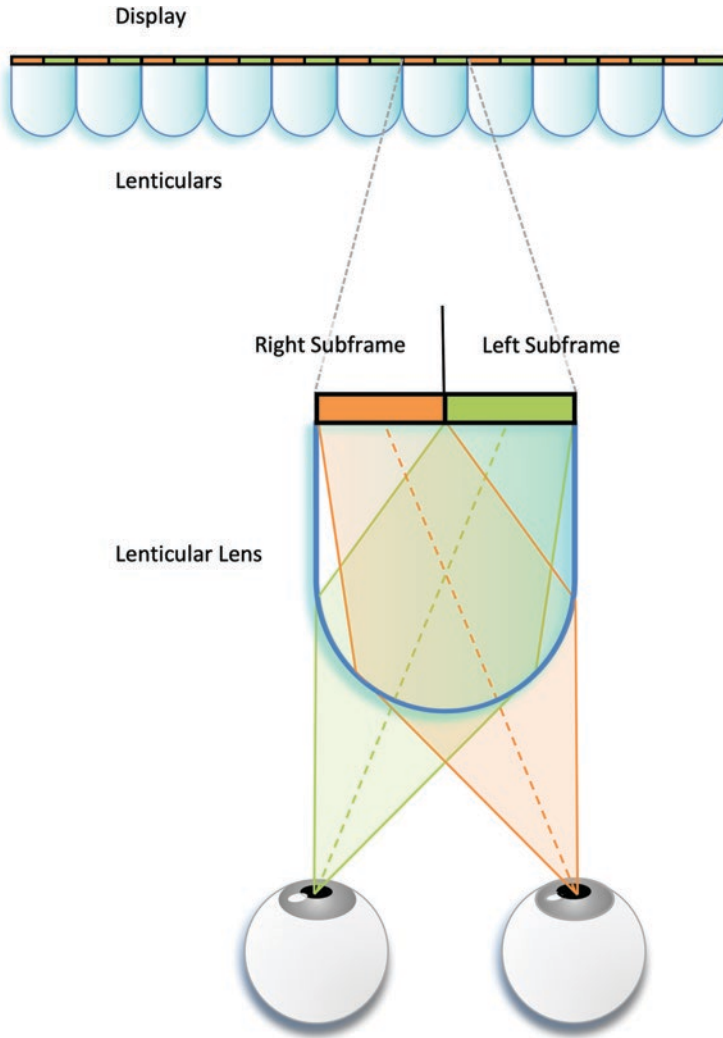


Fig. 5.42 Prismatic grid of lenticular lenses with detail magnification of a lenticular lens for channel separation

covers more than two pixels. For example, seven different stereo views can be created by using eight pixels. The disadvantage here is that the horizontal resolution is reduced even more (in this case to one eighth!).

Parallax Barriers

Parallax barriers represent another autostereoscopic method. Here, a shadow mask is placed in front of the display or the projection surface. Due to the arrangement of the holes in the shadow mask, each eye of the observer sees different pixels, which

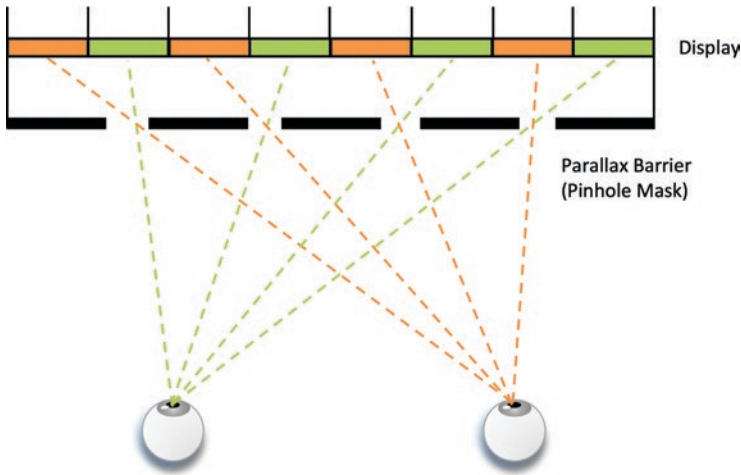


Fig. 5.43 Parallax barrier for channel separation

are then used to display the different subframe for the left and right eyes (see Fig. 5.43).

While rigid parallax barriers also only work within a certain distance from the display, movable parallax barriers allow for adjustment to the distance of the viewer. Either two slit masks are mechanically moved against each other so that the position and size of the holes change accordingly, or an additional LCD layer is used for this purpose. Parallax barriers also reduce the resolution by at least a factor of two and are susceptible to head tilting. In LCD-based systems, if detected, this can be solved by software. Software parallax barriers are used as masks to generate the two subframes, so that ultimately only the pixels visible to the respective eye from each perspective are included in the overall image to be displayed. In principle, the approach can also be extended to more than one user, as Ye et al. (2010) have shown with a display in which they used randomly distributed holes in well known locations.

5.5 Audio Output Devices

The goal of acoustic or audible output is to reproduce the sounds and tones of the virtual world in such a way that the user can perceive them in the same way as in the real world. Even though the spatial resolution of human audio perception is lower compared to the visual sense (see Sect. 2.3.1), it clearly supports spatial orientation. A simple audio system at least is also important for the temporal assignment of events that happen in the virtual world: for example, the user can be given audible feedback when selecting objects or controlling a menu.

Stereo Speakers

A simple spatial audio model, such as the one used in X3D, only influences the volume of the output for the left and right ear depending on the distance of the sound source to the virtual position of the ears (whereby this is only taken into account due to the position and orientation of the virtual camera, i.e., the position of the loudspeakers to the real ear has no influence here). For more realistic effects, the individual signal delay due to the distance to the sound source can be included. With this form it is not possible to distinguish between audio sources in front of or behind the user, which is usually not sufficient for VR or AR. Also sound sources above or below the user cannot be determined from their direction.

Multi-Channel Systems

For better spatial orientation the use of more complex audio installations is necessary. Often, *multi-channel audio systems* are sufficient to provide orientation for the user of a virtual world. In multi-channel systems one or more main loudspeakers are usually available as the actual sound source, while several additional loudspeakers are used to support the spatial effects. When installing multi-channel audio systems, it is important to ensure that appropriate loudspeakers are also installed behind the user. The disadvantage of multi-channel audio systems is that spatial perception is really good only in a small area (the so-called sweet spot). If the user is able to move, the limits of such a system are quickly reached. Furthermore, although the horizontal direction of a sound source can be well simulated, the height of the sound source usually cannot be reproduced.

Binaural Sound

One way to achieve a more realistic audio impression is *binaural sound*. This is an attempt to imitate natural, spatial hearing. The output is only possible via headphones. For an optimal hearing impression, the *Head Related Transfer Function (HRTF)* of the user must be known. If this is not known, the HRTF of a standard head is usually used, which can provide very good or even bad results depending on the individual user. The advantage of binaural sound is that with correct HRTF not only audio sources in front of and behind the user, but also below and above the user, can be clearly identified with regard to their direction.

Ambisonics

Ambisonics usually uses four channels to record and play back three-dimensional sound sources in the form of a sound field. Although the technology is over 50 years old, it has only recently gained some popularity through its use in conjunction with 360° video and VR. In the meantime, relatively inexpensive commercial microphones and software for mixing ambisonic recordings are available. The four channels represent the sound pressure gradients in the X, Y and Z directions and the sound pressure. For these second-order ambisonics, eight capsule microphones are used. Higher-order ambisonics are rarely used.

Wave Field Synthesis

Another way to create more realistic spatial sound is *wave field synthesis* (Bertino and Ferrari 1998; Brandenburg 2006). The goal of wave field synthesis is to record

the wave field of a real event (e.g., the playing of an orchestra) and to be able to reproduce it at any time as a synthetic wave field. Thus it is possible to position sound sources freely, within certain physical limits. For this purpose, the wave field is generated by a large number of loudspeakers that have to be arranged around the playback area. These loudspeakers are operated by a central computer, controlling the reproduction of the sounds together with their positioning.

Application to Stationary VR Systems

Often it seems to make sense to position the speakers of an audio system behind the display. In principle, this is possible for projection systems with permeable screens. However, with multi-sided projection systems such as CAVES, the sound is partially reflected by the projection surfaces, reducing the quality of the audio simulation. A further problem with loudspeakers located behind the projection surface is that the sound causes the canvas to vibrate. This can sometimes have a negative effect on the quality of the visual impression. When using glass panels as projection surfaces and in the case of monitor-based solutions (see Fig. 5.32), it may be necessary to place the loudspeakers under, above or next to the displays.

A comprehensive overview of sound, especially in the context of VR, is given by Vorländer (2008).

5.6 Haptic Output Devices

Haptic output devices make virtual objects tangible for the user by means of mechanical, pneumatic or electrical stimuli, vibration or the application of force. Haptic output is mostly integrated into input devices like gloves (see also Sect. 4.4) and mechanical input devices (see Sect. 4.6.2).

Generally, haptic output devices are divided into those that generate *tactile feedback* and those with *force feedback*. Tactile feedback generates a haptic sensation for the user when touching a virtual object, without necessarily corresponding to the sensation when touching a similar real object. Force feedback usually requires an external skeleton structure (*exoskeleton*), which restricts the freedom of movement of fingers or other limbs.

Haptic Loop

While haptic output basically uses the same information about the virtual world as graphical output, haptic rendering differs significantly from normal (graphical) rendering. A haptic output device is usually combined with an input device, since the haptic output is typically the result of a movement of the user. Of course, a virtual object can also move towards the user and thus initiate a haptic output. An example would be a virtual bullet in a VR game. If the initiation is done by the user, this results in a so-called *haptic loop* (or *haptic rendering loop*) (see Fig. 5.44). For the user's movement to result in a haptic output, it must first trigger a collision between the user's representation (i.e., their avatar) and an object of the virtual world. This collision is detected by collision detection (see Chap. 7). The resulting collision

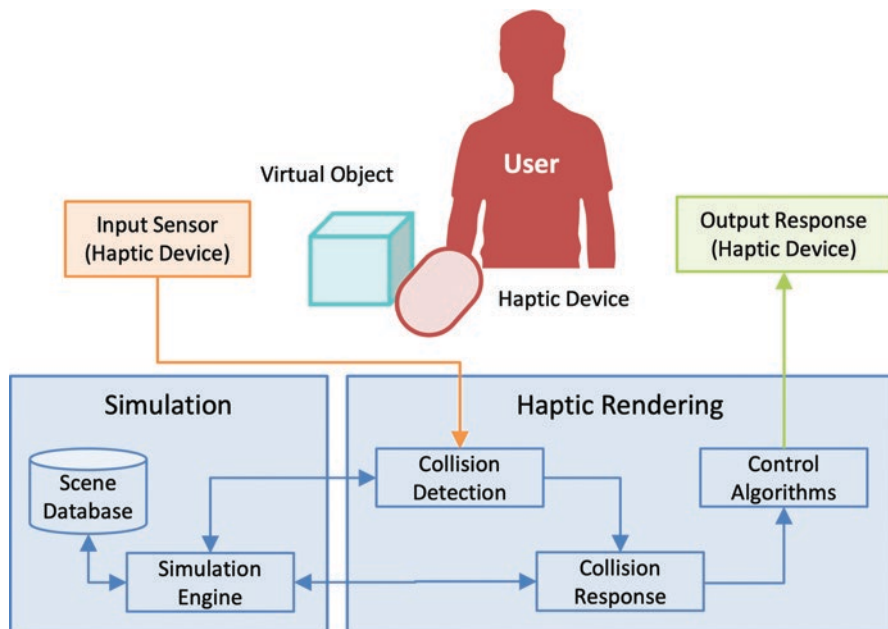


Fig. 5.44 The haptic rendering loop

response induces on the one hand a modification of the VR simulation, while on the other hand the ideal force feedback is calculated as response. The extent to which this can now be (re)transmitted to the user depends in particular on the specific haptic output device used. Appropriate control algorithms are used to convert the ideal force into an actual output reaction. Correspondingly, a massive resistance may end up in a rather soft tactile perception. In contrast to the graphics render loop, which should reach at least 60 frames per second, the haptic render loop typically operates at a much higher frequency. 1000 Hz (or 1000 fps) is not uncommon at this point. An overview of haptic rendering can be found in Salisbury et al. (2004).

Data Gloves with Tactile Feedback or Force Feedback

While *data gloves* are primarily input devices, a number of models have been developed over time that involve a haptic output component. However, most of them are limited to simple tactile feedback. For example, small vibration motors are placed on the fingertips, the fingertips are mechanically contracted by small bands or electrical impulses are generated. However, some also use external skeletal structures to actually restrict the freedom of movement of the fingers. The *HGlove* uses an exoskeleton for the thumb, index finger and middle finger. The *HaptX* glove (see Fig. 5.45), for example, uses a combination of a miniaturized pneumatic actuator consisting of 12 elements at each fingertip and an exoskeleton.



Fig. 5.45 Glove with pneumatic finger actuators and external skeletal structure. (© HaptX Inc. 2021. All rights reserved)

Air and Ultrasound-Based Systems

Vortex rings use air over a certain distance to create a short haptic stimulus. If a grid of actuators is used to generate the stimuli, this allows the simulation of even more complex virtual objects.

Similar in effect are haptic output devices based on ultrasound. Those devices use up to several hundred sound generators arranged in arrays. At update rates of up to 40 kHz, a haptic output can be generated at a distance of up to 70 cm.

Vests and Suits

In games and military simulations especially, vests and whole suits for haptic output have been developed. The *Teslasuit* full body suit is an example of an individual, haptic output device that uses electrical impulses to perform *transcutaneous electrical nerve stimulation* and *electrical muscle stimulation*. Furthermore, the suit can influence the temperature sensation of the user. A motion capture system is also integrated for input. Haption's *Able* is another example, providing an exoskeleton for shoulders, arms and hands (see Fig. 5.46).

End Effector Displays

An *end effector* is a device that is typically mounted on a robot arm (see Fig. 5.47). *End effector displays* are haptic output devices for tactile stimulation, which a user can grasp or otherwise manipulate with hands or feet. In contrast to an input device, an end effector display is not passive, but reacts actively through resistance or force feedback. The best-known representative of this device class is the *Phantom Omni* as a desktop device. Larger versions can easily cover volumes of several cubic meters.

Fig. 5.46 Example of a vest-based system. (© Haption SA/Laval Virtual, 2020. All rights reserved)

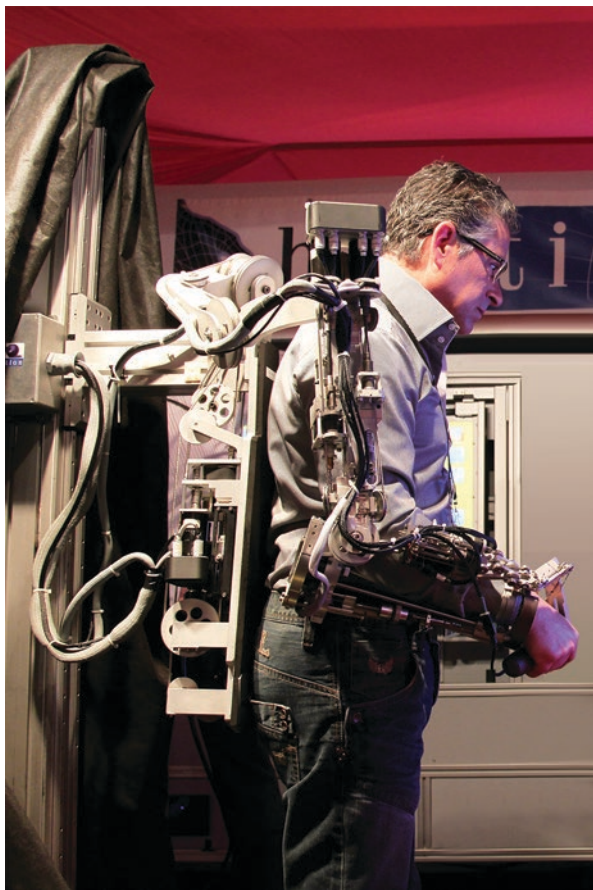


Fig. 5.47 Example of an end effector display. (© Haption SA, 2020. All rights reserved)

5.7 Summary and Questions

VR/AR output devices are used to present the virtual world or the augmented world to the user via appropriate stimulus generation, i.e., to convert the virtual content into something that can be experienced by the user's senses. The visual output can be done with HMDs, monitors or projection systems. To cover the field of vision of stationary VR systems as much as possible, display systems are often composed of several individual displays, which are arranged in different forms: examples are walls, L-shapes, curved screens or CAVEs. The tiling of displays (in the form of tiled displays) is especially used to improve resolution and luminous intensity, but leads to a considerably higher calibration effort. Using active or passive stereoscopy methods, monitors and projection systems can also spatially represent virtual worlds. Although the optical sense is the most important one, a high degree of immersion can only be achieved by addressing additional senses. While there are almost no VR or AR systems without acoustic output and therefore the question of the quality of the effects to be produced is primarily concerned here, haptic output devices are much less common.

Check your understanding of the chapter by answering the following questions:

- What difficulties can occur when using tiled displays and what are the solutions?
- What advantages do multi-sided displays offer compared to single-sided displays?
- When should optical AR glasses (OST) be used for an AR application and when should video AR glasses (VST) be used?
- Explain the basic differences between active, passive and autostereoscopic stereo methods.
- Which audio technologies are suitable for realistic surround sound?
- Explain the difference between tactile and force feedback using a simple example

Recommended Reading

- Bajestani SA, Pourreza H, Nalbandian S (2019) Scalable and view-independent calibration of multi-projector display for arbitrary uneven surfaces. *Machine Vision and Applications*, 7–8, Springer.
- Burdea GC, Coiffet P (2003) *Virtual reality technology*, John Wiley & Sons, Hoboken, New Jersey.
- Okatani, T., Deguchi, K. (2009) Easy calibration of a multi-projector display system. *International Journal of Computer Vision*, 85, 1–18. <https://doi.org/10.1007/s11263-009-0242-0>
- Sherman W, Craig A (2019) *Understanding virtual reality*, second edn. Morgan Kaufmann.
- Vorländer M (2008) *Auralization – fundamentals of acoustics, simulation, algorithms and acoustic virtual reality*. Springer.

References

- Bajestani SA, Pourreza H, Nalbandian S (2019) Scalable and view-independent calibration of multi-projector display for arbitrary uneven surfaces. *Machine Vision and Applications*, 7–8, Springer
- Bertino E, Ferrari E (1998) Temporal synchronization models for multimedia data. *TKDE* 10(4):612–631
- Brandenburg K (2006) Digital entertainment: Media technologies for the future. In *Second International Conference on Automated Production of Cross Media Content for Multi-Channel Distribution (AXMEDIS '06)*, 4–5
- Cakmakci O, Rolland J (2006) Head-worn displays: a review. *J Disp Technol* 2(3):199–216. <https://doi.org/10.1109/jdt.2006.879846>
- Cruz-Neira C, Sandin DJ, DeFanti TA, Kenyon RV, Hart JC (1992) The cave: audio visual experience automatic virtual environment. *Commun ACM* 35(6):64–72
- Herold R, Weidenmüller F, Penzel M, Ebert M (2015) Data-glasses for improved user interaction in 3D. In: *Proceedings of SID 2015 International Symposium*, pp. 189–191
- Krüger W, Fröhlich B (1994) The responsive workbench. *IEEE Comput Graph Appl* 14(3):12–15
- Lang B (2018) The key technology behind Varjo's high-res 'Bionic Display' headset. <https://www.roadtovr.com/graphic-illustrates-key-technology-behind-varjos-high-res-bionic-display/>. Accessed 28 Mar 2021
- Melzer J, Moffitt K (1997) *Head-mounted displays: designing for the users*. McGraw Hill, New York
- Okatani T, Deguchi K (2009) Easy calibration of a multi-projector display system. *Int J Comput Vis* 85:1–18. <https://doi.org/10.1007/s11263-009-0242-0>
- Salisbury K, Conti F, Barbagli F (2004) Haptic rendering: introductory concepts. *IEEE Computer Graphics & Applications*, Jan/Feb, pp 24–32
- Vorländer M (2008) *Auralization – fundamentals of acoustics, simulation, algorithms and acoustic virtual reality*. Springer
- Ye G, State A, Fuchs H (2010) A practical multi-viewer tabletop autostereoscopic display. *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR) 2010*. Seoul, South Korea, October 13–16

Chapter 6

Interaction in Virtual Worlds



Ralf Doerner, Christian Geiger, Leif Oppermann, Volker Paelke,
and Steffi Beckhaus

Abstract In Chap. 1, VR and AR have already been introduced as innovative forms of human–computer interaction. This chapter deals in detail with the design and realization of interaction and the resulting *user interface* of a VR/AR system. A user interacts with a virtual world to select (*selection*) and change (*manipulation*) virtual objects and to control the position and viewing direction in the virtual environment (*navigation*). In addition, the user interacts with the system itself (*system control*) to perform functions outside the virtual environment on a meta-level (e.g., loading a new virtual world). These basic tasks of system control, selection, manipulation and navigation are each dealt with in a subsection. Solutions for the realization of corresponding interactions are presented. It is essential to achieve good *usability*. This is a core issue of human–computer interaction in general. Therefore, the basics of human–computer interaction are discussed at the beginning of the chapter. Moreover, a subsection considers special design processes that guide a developer in the design and realization of VR/AR interactions. An essential aspect here is the repeated validation of interactions with users in the form of *user tests*. Methods for the execution and evaluation of user tests are therefore dealt with separately in a subsection. Interactions with VR/AR systems always have effects on the user. The related ethical and legal aspects are discussed in the last subsection.

Dedicated website for additional material: vr-ar-book.org

R. Doerner (✉)
Department of Design, Computer Science, Media, RheinMain University of Applied
Sciences, Wiesbaden, Germany
e-mail: ralf.doerner@hs-rm.de

6.1 Fundamentals of Human–Computer Interaction

A virtual world or a world extended with virtual elements employing AR can be made interactive for users. This means enabling users to interact with this environment under real-time conditions. It is thus about an exchange of information between the human users and the computer, which controls the virtual part of the user’s environment; in other words, it is about communication between humans and computers. Technically, this is known as *Human–Computer Interaction (HCI)*. HCI is concerned with the design, evaluation, and implementation of interactive computer-based systems and the phenomena involved. An essential aspect is the user-oriented design of interfaces based on findings in computer science, but also in other fields such as psychology and cognitive science, ergonomics, sociology and design.

An important concept of HCI is *usability*, which is most aptly described as “fitness for purpose” and defined according to ISO 9241.

“*Usability* is the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use.” (Source: DIN EN ISO 9241,11: Software Ergonomics)

Aspects of usability include usefulness, efficiency (effort in relation to the achieved goal), effectiveness (achievement of goals, avoidance of errors), learnability, or training effort as well as subjective satisfaction. For some time now, HCI research has also been looking at interaction with a technical system in a wider context and taking into account all the experiences a person has had when using an interactive product. In addition to classic usability, this *user experience* includes, for example, the elegance and aesthetics of the interface or the *joy of use*.

Human–computer interaction aims at supporting the user to utilize a technical system to perform the tasks they are pursuing well, e.g., effectively and efficiently. In doing so, information is explicitly exchanged between humans and computers. Besides, there are context knowledge and assumptions that implicitly provide information for communication with the computer. HCI uses metaphors and mental models to support this implicit knowledge. A *metaphor* is a linguistic image that is used to explain complicated relationships. One uses knowledge from a known area, e.g., waterways in nature, to explain an unknown area, e.g., the flow of data in computer programs. Metaphors are used so that a user can get an idea of the technical system. The reaction of the system to an action of the user should be predictable or at least explainable. Such a mental simulation model that a person’s brain uses to make predictions about the system behavior is also called a *mental model*.

While classical user interfaces based on the *WIMP (Windows, Icon, Menu, Pointer) paradigm* have been established for many years and guidelines for their

effective development are available, nothing comparable exists in VR/AR. It is, therefore, necessary to develop prototypical solutions for the respective tasks of the users and to evaluate their suitability. HCI's popular approach of designing hardware-independent user interfaces by abstracting the available hardware to its function, e.g., as *logical input devices* (Foley et al. 1993), is of only limited use in VR/AR user interfaces. Due to the wide range of VR/AR hardware and a broad range of interaction techniques that a single piece of hardware can support, such an abstraction is difficult. Nevertheless, classical user interfaces are often taken as a starting point for VR/AR interaction. Also, other classes of user interfaces that are not based on the WIMP paradigm have become classical user interfaces due to their widespread use: for instance, user interfaces based on voice or touch-based user interfaces. These can be useful since users have usually already acquired significant competencies with classical user interfaces. This is also true for developers, who should rely on their own real experience in dealing with computers when implementing interaction techniques in virtual environments (Winograd and Flores 1986).

When designing the “best” interaction technique, it is important to consider whether a technique should be as natural as possible or can also be magical. A *natural 3D interaction* in a virtual environment tries to simulate the interaction known from the real world as exactly as possible. For example, users move through a virtual city at walking speed by real walking and can only manipulate objects within reach of their own arms. A *magical 3D interaction*, on the other hand, allows teleporting to any position or modifying objects that are far away by extending the arms at will. If one follows the approach that a virtual environment should reflect reality as closely as possible, one is inclined to make the 3D interaction more natural. However, a magical 3D interaction allows more possibilities and new functionalities. Here, the context of use, the user experience and the degree of naturalness play a role (Bowman et al. 2004).

Even if one chooses a high degree of naturalness, the interaction of humans with virtual objects is never as direct as with real objects, mostly due to technical intermediate layers. User interfaces are said to support *direct manipulation* if the user can modify a graphical representation of an object with input devices and receives immediate and continuous visible feedback about these actions (Shneiderman et al. 2016). Direct manipulation is a key concept in the design of interaction techniques in VR. In AR, direct manipulation enables the equal treatment of interaction with real and virtual objects.

6.2 System Control

The system control of a VR/AR system triggers actions that change the interaction mode or the system state. For example, these can be commands that cause the system to load a new scene, to change the navigation mode through the virtual environment or to set up the display. Conventional graphical user interfaces mainly use elements such as menus, buttons or toolbars to execute such commands. *Drag &*

Drop, text commands and double-clicks are also common techniques. These techniques from 2D user interfaces can only be transferred to virtual environments to a limited extent (for example, on which 2D surface in the virtual world does a button appear and how is it operated?). In the following consideration, use cases in which the system control is carried out by another exterior user (e.g., the instructor in a flight simulation) will not be considered further, since techniques from 2D user interfaces can be used for this purpose.

A conceptual problem of system control by the user of a virtual environment is the inherent conflict with the intended “willful suspension of disbelief” (cf. Chap. 1), because the commands often have no equivalent in the real world or a realistic 1:1 implementation is not practicable. Particularly in the early phase of VR, when a faithful reproduction of reality was key, the proper development of techniques for system control was therefore neglected. Many systems work with *ad hoc* solutions because selection techniques in combination with a representation of the possible actions in a (3D-) menu are a possible solution for system control, and thus the development effort can be minimized. In some systems, for example, a large button suddenly floats in the middle of the virtual environment and is activated by moving a hand to select it. There is only limited scientific knowledge about the usability of such approaches. The development of powerful algorithms for speech and gesture recognition expands the spectrum of available techniques and allows for better adaptation to user requirements. A good source of inspiration for techniques for system control is often computer games, in which many interesting implementations of menu techniques can be found. Five concepts for system control are widely used: menus, 3D widgets, tangibles, voice commands and gestures.

Menus are the most widespread technique. Menu techniques can be structured systematically, e.g., by their positioning (or spatial reference system), the way they are presented and the selection technique used. For example, the position of a menu in the virtual environment can be fixed or it can be linked to the position of an object in the virtual environment (context menu)—or it can be linked to the user (e.g., to the hand) or real objects. The representation can be structured in one dimension (e.g., list, ring), in two dimensions (e.g., color space, table), or in three dimensions (e.g., matrix). Depending on the positioning and representation, different techniques can be used to select a menu item. Dachsel and Hübner (2007) give an overview of corresponding 3D menu techniques.

3D widgets are closely related to menu techniques. 3D widgets are 3D objects in the virtual environment that are coupled with interaction behavior. Their 3D geometry makes interactive functionality visible for the user. Moreover, their 3D geometry provides affordances for using the underlying functionality. These 3D objects do not represent the content of the actual virtual world to be displayed but are additionally inserted to control the VR system. 3D widgets can be inspired by real objects (e.g., some widgets are based on a 3D representation of light sources or cameras—these are manipulated as 3D objects and influence the representation of the scene accordingly). Alternatively, 3D widgets can be abstract objects whose function the user must learn.

Tangibles (sometimes called *props*) are real objects that the user can use as tools in the virtual environment. If the user reaches for such a “tool”, a new mode of interaction is chosen and the tangible itself can give the user immediate physical feedback about this interaction. An example: the user wants to position a power tool in the virtual environment and has the handle of a real tool available for this purpose. However, the *number* of tangibles that can be used in an application is limited and their assignment to interaction tasks is less flexible.

Voice commands can be used hands-free. Thus, they can be combined well with other interaction styles. An advantage of voice commands is that no part of the virtual environment is hidden by additional objects that are needed for facilitating interaction. However, the user has to learn the possible commands, since there is no direct representation of the possible interactions in the virtual environment. Advances in speech recognition make the use of voice input increasingly attractive, but the developer of a virtual environment should also keep in mind that the permanent use of voice input can be tiring. Environmental noise and use in collaborative work environments can also be problematic.

Gestures provide another powerful technique for system control and can be combined with voice input and other techniques. As with voice input, no part of the scene is hidden, but the available functionality becomes more difficult for the user to “discover” and must be learned. Often there is also no graphical representation in the user interface that can serve as a memory aid. The availability of inexpensive sensors and improvements in recognition algorithms make the use of gestures for controlling VR/AR applications interesting.

6.3 Selection

Selection is one of the essential tasks in the interaction of a user with a virtual world or the real world augmented with virtual elements.

Selection means that the user determines a point, area or volume in the surrounding world (e.g., to insert an object there) or selects a semantically meaningful subset of the surrounding world (e.g., a specific virtual object or sub-object to move it).

This task is much more difficult for the user to perform in a 3D context than with 2D user interfaces. First, there are more degrees of freedom in the input (it can be especially difficult to perform a selection in 3D space with a 2D input device). Second, occlusion can become a severe problem. Third, most users have less experience with specific VR interaction techniques. Fourth, there can be more usability problems undiscovered by developers because user interfaces are often not as standardized or tested as in the 2D case. To mitigate these difficulties, one can limit the

selection to an interaction surface (parallel to the image plane or spatially embedded on a 2D surface in the virtual world), which is often even more effective due to the similarity to the usual computer operation. However, VR/AR also allows us to break away from tradition and to introduce new interaction techniques that are more oriented towards our everyday real experiences – or even go beyond.

In the first subsection, pointing devices, their classification and targeting are discussed in more detail, since pointing is often used for selection in VR/AR (and not, for example, naming by voice input or typing in coordinates of the object to be selected). When designing the corresponding interaction technique, one basically has the choice of either restricting the user's degrees of freedom or working with different modes. These choices are discussed in the second subsection. Finally, the last subsection contains examples of selection techniques frequently used in VR/AR.

6.3.1 *Pointing in Virtual Worlds*

A common feature of many interaction techniques for selection is that they require a *pointing device* with which the user can make the selection. This can be the index finger or a special input device such as a hand controller or a 3D mouse (see Chap. 4). With this pointing device, the user must aim at the target to be selected and make the selection. In the VR/AR system, corresponding algorithms from Computer Graphics have to be implemented which identify the selected 3D entity from the user's input. This task is not trivial. On the one hand, it may be necessary to calculate back from a 2D input into 3D space. On the other hand, which object can be found at the calculated 3D position must be determined. This basic task of deducing the selection in the displayed 3D space from a 2D interaction of the user with the image of a 3D scene is called *picking*. A simple solution is to create an image of the 3D scene (which is not shown to the user) where each object is displayed with a different color and no lighting calculation is performed. You determine the pixel in the image that the user points to and determine the color of the pixel, which allows you to draw conclusions about the selected object ("*color picking*"). An alternative method, often used today because of its higher accuracy, calculates the intersections of a beam with the 3D geometry of the objects in the scene (*ray-casting*). The ray can emanate from the eyepoint of the observer through the pixel selected in the image. Alternatively, the ray can be an extension of the user's index finger (see Fig. 6.1, left). The object that has the point of intersection closest to the observer's eye is selected. By clever optimization, e.g., by employing bounding volume hierarchies (cf. Chap. 7), this picking can be realized in real time through ray-casting. Like collision detection, ray-casting has become a fundamental technique to realize interaction in VR and AR in general.

Picking can be complicated by the fact that the VR/AR system does not perform it with the selection granularity desired by the user because of semantic ambiguities (example: the user points to the head of a virtual person—does the user want to select the whole person, only the head or even a part of the head, such as the left eye?). In

AR, knowledge of the real objects in the user's environment is required to enable the selection of real objects. Here, methods of digital image processing and 3D scene reconstruction are usually applied which are based on the analysis of current video recordings of reality.

It is essential to support the user during the selection process with visual feedback. This can be realized by highlighting the selected object (or point, area, or volume) or in the form of a target point (*cursor*) (see Fig. 6.1). In VR, the *3D cursor* is the counterpart of the mouse pointer of the two-dimensional desktop metaphor and allows pointing to a virtual object, even if it is further away. Typically, suitable input devices like a flystick, 3D mouse or magic wand are used and their real position and orientation are mapped to the values of the 3D cursor (cf. Chap. 4). A different technique is the *virtual hand*, which allows a direct touch of the objects to be selected near the user. For this purpose, a 3D representation of the user's hand in the virtual world is used to select objects (see Fig. 6.1, right). Selection utilizing a virtual hand is a more natural selection technique, while the use of a 3D cursor tends more towards magical interaction techniques.

The physical pointing devices of human–computer interaction can be divided into the categories “direct” and “indirect”. Direct pointing devices (e.g., a pointing stick) can be used to position a 3D cursor directly (e.g., at the tip of the stick). Direct pointing devices are therefore able to define absolute coordinates. Thus, the operation is easy but may be tiring or inaccurate over time. In addition, the user may cover parts of the virtual world relevant to the selection task with a hand or arm when operating the device. Indirect pointing devices (such as a mouse) can reduce these disadvantages. They change the position of a cursor using direction vectors, i.e., its position is determined relative to the previous position. Indirect pointing devices, however, require a period of familiarization, as they require hand–eye

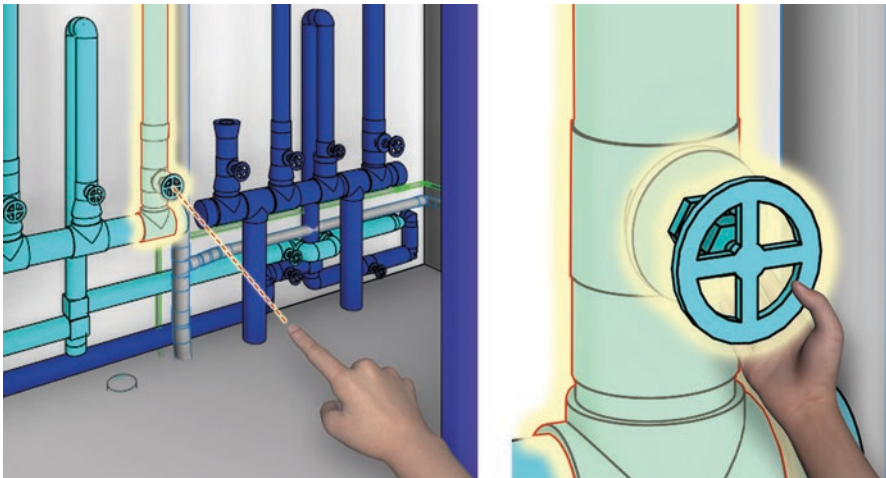


Fig. 6.1 Selection in VR and visual feedback. The selection process is realized (left) with a beam from the finger using ray-casting, or (right) by a virtual hand using collision detection

coordination training from the user. The user's attention is always limited to only one part of the overall space, the *focus*. The interaction activities, on the other hand, take place in another sub-area, which will be called the *nimbus*, following Benford and Fahlen (1993). In direct pointing devices both areas coincide and hand-eye coordination is therefore easy. With indirect pointing devices, however, the intersection of both areas can be empty. If the activity in the nimbus is disturbed or does not lead to the desired result, the attention is distracted and directed to the device itself. The focus of the user is then no longer on the actual task but turns to the interaction device. An everyday example of this is a mouse that encounters a physical obstacle during operation and has to be repositioned by the user. This is also a good example of a phenomenon formulated by Winograd and Flores (1986), namely that people only consciously perceive basic technologies and devices when they simply refuse to work.

When selecting in virtual environments, it is also possible to distinguish near interaction techniques, *local interaction techniques* and *remote interaction techniques*. The close interaction enables users to orientate themselves quickly based on their everyday experiences, which can often be helpful. In VR systems, however, it is also possible to realize interactions over a virtual distance that would normally be beyond human reach (as special cases of magical 3D interactions). When designing these interactions at a distance, it is now mainly a question of their manageability or the accuracy that the user can achieve with them. In HCI, accuracy in the selection of targets in connection with the time required for this and the size of the target could be related in the form of *Fitts' Law*. Although it is mainly used in the evaluation of traditional 2D graphical user interfaces, there are also references to the 3D context. Fitts' Law states that the smaller the target is and the further it is from the current cursor position, the longer it takes to select it. A logarithmic function is used to mathematically model this relation.

6.3.2 Interaction Design

To control interaction in virtual environments, input devices are required that cover the necessary degrees of freedom. In three-dimensional environments, there are six degrees of freedom: three for positioning along the x-axis, y-axis, and z-axis and three additional degrees of freedom for rotation around these axes. In many cases, however, fewer degrees of freedom can be used. For example, not all six degrees of freedom are required to aim at any point *B* from any point *A* (for example, because the imaginary connecting line from *A* to *B* can be rotated around itself without changing the target, this axis of rotation is not significant). It might also be desirable to deliberately limit the number of degrees of freedom in interaction by introducing *constraints*, e.g., to avoid accidentally changing the orientation of a 3D cursor when it is moved.

The Midas Touch Problem

One could come up with the idea of making the selection with the eyes, especially over longer distances; after all, one can look further than one can grasp and can focus very quickly. However, it has turned out that a general selection only with the eyes is not comfortable for the user, because if a VR system tries to attach importance to these glances by continuously pointing them towards the selection of virtual objects, the user would not be able to look anywhere without unintentionally selecting something. This is the classic *Midas touch problem* (Jacob 1990). The name for this problem comes from mythology. According to legend, King Midas of Phrygia received the supposed gift of being able to turn everything he touched into gold. However, this proved to be a hindrance when it came to eating, and he also supposedly turned his daughter into gold by mistake. To work around the problem, you can work with modes – the “system responds at glances” mode and the “system does not respond at glances” mode.

The Midas touch problem shows that different modes should be distinguished in the design of some interaction techniques for selection. However, this has the drawback of increased complexity. The users – based on their own previous experience – often have certain expectations regarding the operation of a system. If the operation corresponds to their expectations or the general conventions, the system is in line with expectations. If it does not, the users are typically disoriented because they have unintentionally triggered actions or cannot trigger desired actions. If there are several modes, a user can inadvertently switch the system to another mode or does not notice a change of mode, which can lead to confusion. Problems of this type are referred to as *mode errors*. They are not errors in the traditional sense of software technology, but errors in interaction design. They require different measures for detection and correction than the usual debugging of software. When designing interactions, it is advisable to limit the use of modes. Some designers suggest that modes should be removed from user interfaces completely and that, only if necessary, temporary *quasimodes* knowingly activated by the user should be used (Raskin 2000). Interaction designers in the VR area should be sensitized to these and similar problems, e.g., about the frustration of users when they are doubtful what is not selectable and they can only find out by unsuccessful trial and error. (Bellotti et al. 2002) posed five questions that designers of interactive systems should always ask themselves:

- If I address the system, how does the system know that I am addressing it?
- If I call the system, how do I know it will listen to me?
- If I give a command, how does the system know what it is referring to?
- How do I know that the system understands me and will carry out the action I want?
- How can I correct a mistake?

6.3.3 Examples of Selection Techniques

In the following, some examples of selection techniques are considered: ray-casting, the flashlight technique, the go-go technique, the HOMER technique, the image plane technique and the world-in-miniature technique.

In *ray-casting*, objects are selected using a beam that points from the 3D cursor into the environment. The position and orientation of the beam are controlled by the user, although degrees of freedom in control can be deliberately limited by setting constraints. All objects cut by the beam are candidates for selection. If there is more than one candidate, the object closest to the user is selected. The manageable accuracy of ray-casting decreases with distance because the angle to be set on the virtual hand becomes smaller and smaller and possibly falls below the resolution to be achieved by the input method. Ray-casting is considered the most important and effective selection technique. However, it is less suitable for longer distances.

A variation of ray-casting is the *flashlight technique*. Here, instead of a beam, a cone is projected that resembles that of a flashlight. Again, all objects that intersect the geometry are collected as candidates. As an additional selection criterion, the distance from the center of the cone is also considered.

Possibly inspired by the television series *Inspector Gadget*, the *go-go technique* allows the infinite extension of a virtual arm to which a virtual hand is attached. It thus allows the hand to be moved to the place of interest. Within the normal interaction distance, i.e., within arm's reach, the virtual hand behaves analogously to the real hand, i.e., the movement is scaled linearly. Beyond this distance, the movement of the real hand is mapped to the movement of the virtual hand by a usually non-linear scaling in such a way that with increasing distance from the user, increasingly larger distances are bridged with the same hand movement. Similar to ray-casting, however, it is only partially suitable for selection at a distance due to its angle dependence.

In the term *HOMER technique*, HOMER stands for "Hand-centered Object Manipulation Extending Ray-casting". With HOMER, a beam is also extrapolated from the current hand position. However, if the ray hits an object, the object is not manipulated as the endpoint of the ray, but the virtual hand is moved to the position of the object. This eliminates the dependence on angular accuracy and allows finer selections and manipulations of the target object.

The *image layer technique* uses virtual image layers on which the users make their selection, similar to a mouse pointer. The objects behind the image plane are projected onto them, just as they are projected onto the screen plane. The distance between users and the image plane is reduced for interaction. With their pointing device, the users now control a 2D cursor on this virtual plane. While this plane is within their reach, they are also able to select out-of-reach objects behind this plane, as these objects are projected on the plane. Because a user only has to control two degrees of freedom and the metaphor is known, this technique allows for easier control during selection.

An alternative approach to changing the reach of the user is the *world-in-miniature (WIM)* technique. This involves scaling down the entire virtual environment to such an extent that it fits into the user's field of view as a miniature model.



Fig. 6.2 Example of the selection of remote objects by a world-in-miniature. With this technique, users can select objects even if they are not in their field of view

The user can now select interaction targets in the model. Since the user leaves their own, egocentric perspective within the environment, this technique is also called an *exocentric* technique. An example of WIM is shown in Fig. 6.2. In contrast, techniques like ray-casting or the flashlight technique are *egocentric* techniques. There exist also mixed forms between egocentric and exocentric techniques, which are called *tethered*.

6.4 Manipulation of Objects

After a suitable object has been selected, its properties can now be changed by manipulation. Selection and manipulation techniques should not be addressed individually when designing a specific VR or AR interaction, but should be matched to each other. Already presented techniques like the HOMER technique or the WIM technique are suitable not only for selection but also for manipulation. For instance, using the go-go technique, objects can be moved very well. Moreover, the insights about interaction design discussed in Sect. 6.3.2 and illustrated with the example of a selection technique also extend to manipulation techniques.

Manipulation of a virtual object in a VR/AR environment is defined as an interactive change of the parameters characterizing the object, such as its location, its orientation in space, its size, its shape, its weight, its velocity or its *appearance* (which is determined by object parameters such as color, texture or shading).

Manipulations of virtual objects may not have a direct equivalent in the real world. For example, virtual objects can be manipulated by any affine mapping (e.g., shearing or scaling). To implement the manipulation, developers can fall back on a wide range of techniques that cover the entire spectrum from realistic interactions oriented on the user's everyday experiences to magical techniques that can only be realized in a virtual environment. Therefore, the choice of a suitable manipulation technique should be made by the developers with regard to the desired functionality and the concept underlying an application.

In simulations and training applications, for example, it is often desirable to have a reference to reality that is also supported by a realistic interaction technique. This means that the virtual objects should behave like real objects as far as possible and the manipulation actions of the user should be based on the corresponding actions in a real environment. If, on the other hand, the focus is on simple interaction with the presented content, e.g., in visualization and entertainment applications, interaction techniques can also be used that are not possible with real objects, e.g., manipulation of objects that are beyond the user's reach.

The manipulation of remote objects has great potential to improve the effectiveness of a user interface, as it decouples the target-oriented part of the interaction (e.g., changing the spatial orientation of an object) from the preparatory actions that are indispensable in a real environment (e.g., positioning the user within reach of the object). Ideally, users can then limit their actions to the target-oriented part of an interaction. A variety of manipulation techniques have been proposed and developed to try to realize this potential advantage of interaction at a distance in virtual environments. However, since these techniques are not directly based on the users' everyday experience, they typically have to be learned. The lack of haptics has also often proved to be problematic (De Boeck et al. 2005). Therefore, the aspects of intuitive usability and effective interaction must be weighed up in the development process.

Similar to selection techniques, manipulation techniques can distinguish between egocentric and exocentric interaction. In egocentric manipulation, the user is conceptually part of the virtual environment, and perception takes place in the first-person view. This egocentric perspective on content and interaction is particularly useful if the user is to feel as present as possible in a VR or AR. An interaction employing pointing gestures can extend the manipulation to more distant objects (*action at a distance*). In the literature, there are a multitude of interaction techniques based on pointing gestures for manipulating distant objects. Interaction techniques based on a virtual hand or pointing gestures are characterized by the fact that they can be applied generically to any content. However, one downside may be that objects can only be repositioned within arm's reach, resulting in clumping, i.e., an aggregation of objects close to the virtual hand that can be perceived as annoying. Besides, for specific applications (e.g., in simulations and training applications) the use of dedicated input devices such as a cubic-mouse has become common, which as "tools" support a specific interaction task.

With exocentric manipulation, the user is conceptually outside the virtual environment. The perception of the content is "from outside" and is also called "god's

eye view”. This exocentric perspective on content and interaction is particularly useful when simple interaction with complex spatial content is the main focus, for instance in visualization applications. Typical examples are the world-in-miniature (WIM) techniques already mentioned. In the following, some frequently used techniques will be presented here.

Arcball: The arcball technique is an example of a manipulation technique that allows solely the manipulation of the orientation of an object. The object to be manipulated is conceptually enveloped in a sphere and user interactions are transferred to rotations of this sphere around its center, which in turn is translated into a new orientation of the object in space. Thereby, a 2D interaction can also be mapped to the rotation of the sphere. Such a restriction to two degrees of freedom can be perceived as helpful by users.

Virtual Hand: The virtual hand technique strives to have users interact with virtual objects in a form that is similar to the interaction with real objects. Since the interaction is based on everyday experience, such techniques are easy to use and appear “natural” to the user. User interfaces that use such direct interaction, such as tapping or wiping, are called *natural user interfaces*. Here, the operation of artificial input devices such as the mouse does not have to be learned first. They are particularly suitable for applications where a high degree of realism is desired. This can be extended beyond reality to worlds, e.g., fantasy worlds, that the user is familiar with and hence can also appear “natural” or rather “supernatural”. Accordingly, *supernatural user interfaces* can be conceived, e.g., interaction techniques that are inspired by wizard spells. However, such natural techniques may also be subject to numerous restrictions: for example, users are confined to manipulate objects within their direct reach if it would be unnatural otherwise. In contrast to the 3D cursor, the virtual hand can be used to perform gestures with the fingers, which can be mapped to the manipulation of object parameters. This mapping must be learned by the user. An example is the *pinch gesture*, the bringing together of thumb and index finger, which can be mapped to object parameters, e.g., size.

Pointing Gestures: Techniques based on pointing gestures are suitable not only for selecting distant objects but also for manipulating them. For this purpose, the pointing gesture is typically interpreted as a pointing beam (cf. ray-casting) or as a pointing cone (cf. flashlight technique). Pointing gestures are often used in everyday experiences to select objects, e.g., in a discourse. Thus, they are an intuitive way for many users to select objects. The extension from selection to manipulation is then easy to learn. However, a direct transfer of gestures to manipulation is often difficult. The obvious option to use the pointer beam as a “lever” for manipulation makes precise positioning and orientation difficult.

Transmission of Hand Movements: A simple way to increase precision is to first make the selection using a pointing gesture, and for the subsequent manipulation to interpret the movements of the user’s hand as if the user had grasped the object. Conceptually, this can be done in such a way that the object moves into the hand of the user, is manipulated and returns to its starting point after the interaction is completed. Alternatively, the user is “teleported” to the location of the selected object and can then manipulate the object there using the techniques of the virtual hand.

Voodoo Dolls: Another example of an exocentric technique is voodoo dolls. Similar to the WIM technique, it is based on scaling. The user can interact with scaled copies of individually selected objects. In contrast to WIM or techniques that directly transmit hand movements, scaling ensures that the user can effectively manipulate objects of different sizes.

6.5 Navigation

Navigation is a fundamental and often challenging task, as anyone will discover who is looking for a gas station in an unknown city and does not have a satellite navigation system available.

Navigation in the real world can be defined as finding one's way in space by determining one's position and calculating a route to reach the desired location as well as the necessary activities to accomplish this.

In HCI, navigation is also an important user task: users navigate websites, complex text documents or tables, and stroll through computer game worlds. In a virtual environment, navigation is a universal interaction task and of central importance. Presence in VR requires that the user can move around the world as easily as possible. In this context, a distinction is made between two sub-areas, *wayfinding* and *traveling*.

Wayfinding is the cognitive component of navigation. On a higher level of abstraction, it comprises analysis, planning and decision about paths in the virtual world. This requires spatial knowledge of the environment, techniques for planning and deciding on routes, and the use of appropriate tools such as landmarks, signs, or maps.

The goal of wayfinding is always to generate a *cognitive map* of the virtual world, i.e., a simplified mental representation of virtual space. The process of wayfinding is usually unconscious, and the resulting cognitive map can be different for each user. Therefore, it is difficult to develop targeted computer-based support to enable the user to acquire the necessary spatial knowledge. This knowledge can be divided into three types. *Landmark knowledge* includes knowledge about prominent, often unique reference points in space (landmarks), which are easier to remember than other points and can be used for locating points in space. Landmarks are easier to remember the longer a user is present in the virtual environment. Thus, they are an important tool for the development of a cognitive map. Landmarks can

be integrated easily into virtual worlds, but they have to be distinguishable clearly from other objects in the environment and should be positioned in a suitable place. In AR, pathfinding is based on the usual pathfinding in reality, since the usual spatial navigation is available at any time. Nevertheless, this wayfinding in AR can be modified, e.g., by using virtual objects as landmarks, which can also support wayfinding in reality.

Route knowledge is also called *procedural knowledge* and describes the knowledge about the sequence of points in a scene that form a route and what actions are necessary to follow this route. Thus, route knowledge is an action-driven concept and does not necessarily require extensive visual information. In a virtual environment, tools such as a digital compass, signposts or waymarks can support the acquisition of route knowledge.

Knowledge about the topology of the environment is called *overview knowledge*. This knowledge is qualitatively the most extensive of the considered types and the acquisition usually takes the longest. Often existing landmark knowledge and route knowledge are used to get an overview of the virtual environment. For example, different routes and different reference points are utilized to get an overview of the virtual world through a comprehensive cognitive map. This knowledge acquisition is supported by interactive overview maps or the world-in-miniature technique already described in Sect. 6.3.3.

In virtual environments, the focus is usually on supporting the user's abilities through the technical parameters of the system. *Field of view*, depth and motion cues (see Chap. 2), and multimodal input/output techniques that appeal to different senses can support the user in generating a mental map of the environment.

Traveling is the motor component of navigation, i.e., only the basic actions needed to change the position and orientation of the virtual camera are considered.

Interaction techniques for traveling are considered of particular importance because almost every virtual environment must allow the user to move around the world or at least look around in it. User movement is also a necessary prerequisite for other basic 3D interaction techniques, such as manipulation or system control. Without being able to reach a certain place in the virtual world, the hero in a computer game cannot open the treasure chest and the engineer cannot virtually view the engine compartment of the new electric vehicle. Bowman et al. (2004) define three tasks for traveling: exploration, search and maneuvering.

In *exploration*, the user does not have a concrete goal but explores the virtual environment by way of investigation. This is especially used in architectural visualizations, 3D computer games and information visualization. Typically, this task often occurs at the beginning when an initial orientation is necessary. Direct control of the virtual camera is helpful to explore the environment interactively and supports the creation of cognitive maps.

During the *search*, the user has the goal of reaching a defined position. Without additional information this form is called “*naïve search*”; otherwise this targeted search is called “*primed search*”.

Maneuvering is about finding an exact position in the immediate vicinity of the user. It is characterized by short and precise movements. Maneuvering is an interactive task that often has to be solved between two other tasks. For example, when reading a sign in a virtual environment, you will first roughly approach the position before aligning yourself exactly. Then another task can be solved, e.g., manipulating an object based on the instructions shown on the sign.

In the following subsections, some examples of interaction techniques facilitating traveling in virtual environments will be presented. The last subsection deals with design recommendations for navigation techniques.

6.5.1 Control Techniques for Traveling

While most AR systems rely on the usual motion control in real space and control techniques for traveling are always implicit, VR systems can employ implicit as well as explicit techniques. Even a combination of both is feasible.

Many VR systems use *virtual reality locomotion* where the virtual camera is controlled by specifying a direction vector. Established 3D input devices such as flystick, wand and 3D controllers of HMD systems like Oculus or HTC Vive are especially well suited for hand-based control, as their 3D position and orientation in space are efficiently detected by a tracking system. The user starts the movement of the virtual camera through the handheld input device and often uses a vehicle metaphor for this type of movement. This means that a movement in the virtual environment is explained by a device like a car or an airplane that the user controls. Hand-based techniques are easy to implement but have the disadvantage that one hand has to be used for movement control and is therefore tied.

Eye-directed control is the basic principle of many first-person shooters and other 3D computer games. The player rotates a virtual *avatar* (a graphic representation as an agent of the user in the virtual world) in the first-person perspective with an input device in a certain direction and then moves forward in this direction at a certain speed. In desktop systems, this direction vector is determined and normalized as a beam from the virtual camera through the center of the screen. The user or the virtual camera in the first-person perspective is then moved along this vector until the user stops or changes direction again. Moving the virtual camera orthogonally to the viewing direction results in the movement technique known from computer games called “*strafing*” (originally a military practice for attack), where the user moves sideways out of a hiding place to fight the opponent. In an immersive environment with user tracking, the gaze vector can be determined directly, e.g., relying on head tracking that might already be used in the virtual environment. The gaze control is natural and easy to use, but has the disadvantage that users can only move in their viewing direction.

A decoupling of the view vector and direction of movement is obtained by using the body or hand to determine the direction. The latter is also the basis of the “camera-in-hand” technique, in which a physical object is equipped with appropriate sensor technology that serves as an exact reference for the virtual camera. A disadvantage, however, is that it takes some effort to get used to moving the camera by hand for an egocentric camera perspective.

A special case of motion control is *teleportation*, in which a user can move abruptly to any position. This is achieved by an immediate change in camera position and orientation. With the proliferation of consumer HMDs and VR games, teleportation has become a standard technique for traveling. A separate Sect. 6.5.4 deals with teleportation in more detail.

Altogether, there is a wide range of motion-based control techniques that can be used in the interaction design of a VR application. Boletsis (2017) analyzed 36 studies on traveling techniques and categorized them based on the typology shown in Fig. 6.3. A further overview of techniques with selected example applications can be found in (Reddit 2018).

Control techniques are generally easy to implement and established in VR. For example, teleportation is directly available as functionality in game engines, which are also used for the development of VR systems. However, the movement in the virtual world is often only perceived visually by the user. This is in contradiction to the various body perceptions such as the sense of balance and proprioception (perception of one’s own movement) in case the user does not move. The use of certain metaphors such as “driving” or “flying” in the user interface can only partially diminish the contradictions in these impressions. The use of natural gestures such as typing, pulling/pushing or arm swinging has proven to help improve user-friendliness (Ferracani et al. 2016; Wilson et al. 2016). Improved visual feedback during movement, e.g. by dynamically changing the field of vision or displaying a virtual nose in the peripheral visual area, reduces cybersickness (Fernandes and Feiner 2016).

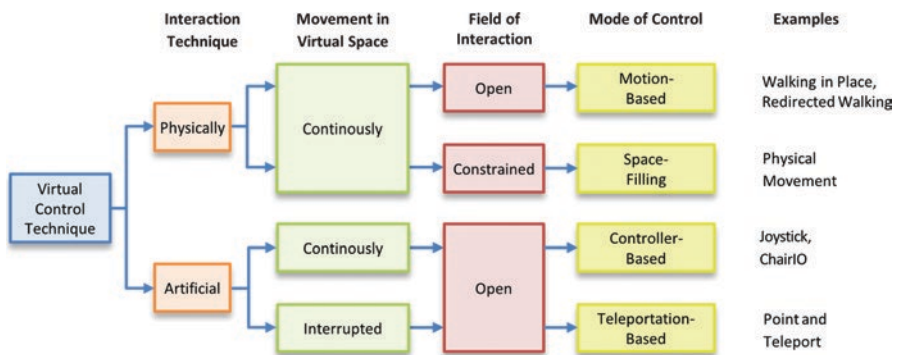


Fig. 6.3 Categorization of control techniques in virtual worlds

6.5.2 Walking Technique for Natural Movement Control

The obvious technique for motion control is physical *walking*. The advantages of this natural technique are the vestibular movement cues provided by the human organ of balance during real movement. However, since many VR systems do not have the necessary large interaction space, alternative mappings of the real user movement to changes in the virtual camera position must be found. A simple approach is to scale a small user movement to large virtual changes but then one gets quite strong fluctuations for small changes due to tracking inaccuracies. Another approach has become known as “*walking in place*” (see Fig. 6.4). The user moves on the spot and is tracked by a suitable tracking system. Advanced approaches based on movement platforms also allow for manual or automatic reorientation of the user. On corresponding devices (such as Omni Virtuix), the user glides back to the starting position on a concave running surface.

Studies have shown that walking in place increases the feeling of presence compared to a purely virtual technique without body movement. However, real movement in space offers an even greater sense of presence (Usoh et al. 1999), although the risk of increased cybersickness (cf. Sect. 2.4.7) must be accepted (Suma et al. 2009).



Fig. 6.4 The figure shows a simple “walking in place” method where a user is tracked by a mobile motion capturing system. The body movements are detected by inertial sensors at the joints and control movement and orientation in the virtual environment. On the bottom left, the user’s pre-distorted view (see Chap. 5), which is generated for both eyes, is shown. (© Christian Geiger, HS Düsseldorf. All rights reserved)

With HMD-based virtual environments, the problem often arises that the user can quickly move out of the spatially limited tracking area. Here the technology of *redirected walking (RDW)* (Razzaque 2005) offers a solution. While the user physically moves in one direction, the scene is manipulated in a way that the according changes are hardly noticeable to the user. The user unconsciously adapts to these changes, so that redirected walking credibly simulates a straight movement in the virtual environment for the user, although the user has been walking in circles in the real world. This effective technique is based on the fact that visual feedback influences navigation more than a physical sensation. Spatial sound can be even more supportive in this respect since research has shown that the simulation of a rotating sound source creates the illusion of self-rotation when visual and acoustic stimuli coincide (Bowman et al. 2004).

Redirected walking has been a scientifically intensively studied control technique for more than 15 years. According to Nilsson et al. (2018), the ideal RDW technique should have four characteristics. Firstly, it must be imperceptible and not allow the user to detect the manipulation. Secondly, it must guarantee security and prevent the user from leaving the tracking area or colliding with objects or other persons. Generalization to multiple users or any virtual environment is the third requirement. Finally, an RDW technique must not have any undesirable side-effects, such as cybersickness or distraction from the primary task. These properties also depend on static parameters such as room size, the number of users or the size of the tracking area, as well as dynamic parameters such as the previous positions of the users and their targets.

Most RDW approaches change the rotation, translation or movement on a pre-defined path in the virtual world in relation to the real environment. For individual cases of this mapping, it has been observed in relevant studies that users do not notice a deviation of the virtual rotation in the range of -20% to $+50\%$ of the real rotation angle. For RDW movement along a curve, Langbehn et al. (2017) were able to simulate a virtual area of $25\text{ m} \times 25\text{ m}$ with a physical tracking area of $4\text{ m} \times 4\text{ m}$. All previous research results confirm the advantage of smaller, more subtle, and frequent modifications over larger, less frequent modifications. Therefore, the idea of using unconscious blinking or saccades – the fast jumps between fixations during which the visual system is blind – as an opportunity for manipulation of positioning is interesting. Sun et al. (2018) describe a system that allows dynamic saccadic repositioning in real time and even evades moving elements, e.g., other users.

However, if the user reaches the edge of the tracking area or an obstacle too quickly, the cautious countermovement is not fast enough to avoid tracking or collision problems. In Razzaque (2005) it was suggested that in such cases, the system should “interrupt” users in navigation by forcing them to turn their heads briefly. Since after such a distraction one has to re-orientate oneself in the virtual environment, the system can rotate the virtual scene so that the user then moves away from the obstacle or the tracking border.

Peck et al. (2011) have presented a three-stage system RFED (“Redirected Free Exploration with Distractors”), which is designed to prevent users from moving beyond tracking limits or colliding with real obstacles during free exploration in a

virtual world. In each frame, the system determines the expected user direction and rotates the scene unnoticed by the user in such a way that the next step really goes into the middle of the tracking space. If the user gets too close to the tracking limits by fast movements, a distraction in the VR environment is generated as a second step. In Peck's example, a hummingbird flies close in front of the user and provokes the required head movement. If even this distraction is not enough, a virtual barrier is faded in, which makes it clear that there is no further movement feasible.

An alternative approach by Suma et al. (2011) subtly changes the virtual architecture in the scene. This approach makes use of *change blindness*, a phenomenon of visual perception in which sometimes large changes in a visual scene are not perceived by the viewer. Specifically, the position of doors and passageways behind the users was dynamically changed in the work, so that almost 220 m² of virtual space in an 18.5 m² area was accessible. Only one person out of 77 users noticed this manipulation.

6.5.3 *Leaning Interfaces for Movement Control*

In connection with the movement technique of steering, there are special interaction techniques that stimulate the sense of balance more strongly than movement on the spot through walking in place. The user leans in the desired direction of movement and the system calculates the locomotion. This "leaning" is comparable to steering a motorbike or moving while skiing or skateboarding. *Leaning-based interfaces* often provide hands-free, easy-to-learn, space-efficient and economical motion control that uses the sense of balance as physiological feedback (Wang and Lindeman 2012). Different types are distinguished depending on the input device used and the type of force applied.

- *Isometric interfaces* require a holding force, i.e., the muscle tension is not converted into movement. The Wii Balance Board is an example of an isometric leaning interface because it does not move when in use.
- *Isotonic interfaces* have practically no noticeable counterforce during use, i.e., there is no resistance and the input device moves effortlessly. An example is the Tony Hawk RIDE game board, which allows users to move in any direction without feeling any resistance.
- The combination of both approaches is called an *elastic interface* in VR and offers a better user experience and a higher presence than a purely isometric interface (Wang and Lindeman 2011).

An example of such an elastic interface is the input device ChairIO (Beckhaus et al. 2007; see Fig. 6.5). Such "leaning-based" interfaces have the advantage that they do not require much space. In contrast to purely virtual techniques, they stimulate the sense of balance and thus enable higher presence. Utilizing inexpensive tracking technologies, unusual interfaces can be realized that enable an attractive user experience.



Fig. 6.5 The ChairIO allows navigation in a virtual environment by leaning in the desired direction of movement. To implement this concept, a special chair was equipped with additional sensors according to Beckhaus et al. (2007). (© Steffi Beckhaus. All rights reserved)

6.5.4 Teleportation for Movement Control

One of the simplest approaches to traveling is teleportation in VR. This approach reduces the susceptibility to cybersickness, as the path to the new position is not perceived. However, at the same time, the user's orientation is limited as it is difficult to figure out their location on their own cognitive map. On the other hand, users are used to cuts from traditional film, in which there is also an abrupt change of environment based on the narrated content. The only essential difference is that in VR the user actively causes the abrupt change and also chooses the target. While classical film editing has rules such as the 180° rule (the orientation of the camera before and after the cut should avoid crossing an imaginary line that would result in the “swapping” of right and left), comparable design guidelines are still lacking in teleportation.

A special form is the *point & teleport method*, where users are teleported to a point in the field of vision, which they have previously selected by a selection method like pointing (Bozgeyikli 2016). To do this, the user points to the desired point, activates the process, and is immediately at the chosen location. The orientation remains the same. One reason for the popularity of this method is that the user can move freely within a limited tracking area (room-scale VR, approx. 10 m²–20 m²). In addition, the point & teleport method, in combination with walking, in room-scale VR allows both exact movements near the user's position and movement over long distances. In studies comparing different methods, such as teleportation, joystick and redirected walking for room-scale VR, subjects attest that teleportation is intuitive and user-friendly, while the use of joysticks often leads to cybersickness (Langbehn et al. 2018).

Another special form is the *speed teleporting method*. It was added to the VR version of the well-known first-person shooter Doom because the speed of the game poses special challenges to the player's movement. In speed teleporting, the camera is not changed abruptly nor does the user see a black intermediate image when changing location. Instead, images are shown of how the user moves at high speed on the path from the start position of the teleportation to its end position. Therefore, the user can continue to act during the teleportation. However, this form of teleportation is more prone to the occurrence of cybersickness.

6.5.5 Route Plan, Goal-Based and Guided Movement Techniques

The interaction techniques discussed in this section differ from the direct motion controls mentioned so far because the users partly give up control of their movement. In *guided navigation*, a user makes use of a moving entity (e.g., uses a train, steps on an escalator, hops on a conveyer belt) and is moved by it. In route plan or goal-based movement techniques, users specify only one path to the target, which is then followed. This two-step approach of *path planning* and execution of movement is less common in virtual environments. The user defines the path by directly specifying the path on a map, by specifying waypoints through which a path is interpolated or by specifying a destination while the system automatically determines the optimal path to that destination. An advantage of losing control over one's own movement is that the path animation can be optimized by motion smoothing. This also includes goal-based techniques where the user only specifies the desired endpoint and the system determines and executes the path to that point itself. For an easy selection of possible targets, 2D maps or the three-dimensional *world-in-miniature* (WIM) technique already presented in Sect. 6.3.3 can be used. For large virtual worlds, especially, a three-dimensional miniature is a good approach to select the desired target. The manipulation of the WIM must be done with suitable tracking technologies and the WIM must be able to be displayed in the virtual environment properly. In large installations such as a CAVE or a powerwall, only a display surface can be utilized, which is far away from the user. With the availability of mobile devices and tablets with good position sensors, this hardware is particularly suitable for use as a WIM, as the necessary movements can be easily registered, direct touch with finger, pen or keystroke is possible, and the WIM can be rendered directly on a screen close to the user.

6.5.6 *Criteria for Navigation Techniques*

At this point an overview of important design recommendations is given. These are essentially based on Bowman et al. (2004).

- Virtual landmarks should stand out clearly in the scene and be located at a suitable, clearly visible position.
- The motion control should use techniques and input devices that support physiological movement cues.
- Maps support orientation very well if they are readable, represent the environment with the current position of the user, and are suitably oriented. It is important to choose the right size so that the map does not obscure the surroundings.
- Maneuvering techniques must first be easy to use to facilitate rough positioning and later also allow for exact alignment.
- The motion control should be selected according to the application, the goal of the user and the technical conditions (e.g., I/O devices) of the virtual environment.
- Natural and magical interaction techniques can be equally helpful. Therefore, one should always consider both possibilities in interaction design. Compatibility with other techniques (e.g., for manipulation) should be considered.
- Different interaction techniques may also be useful for different motion control tasks. It should be taken into account that users may have different abilities. It is helpful to offer simple and complex navigation techniques when the user profiles differ greatly.
- For exploration and search, steering techniques and walking are well suited; for goal-based tasks (“Go to X”), procedures based on route plans are better.
- If navigation is only a secondary user task, the interaction technique should be as simple as possible so that the user can focus on the important tasks.
- In the case of complex interaction techniques, users should be trained in order to generate overview knowledge.

6.6 Processes for the Design and Implementation of Interaction

It is essential for the success of a VR/AR system that the applied interaction techniques allow for a system with good usability. How can this be achieved during the development of the system? How should one proceed when designing VR/AR interaction techniques? How can one ensure that not only technical requirements are considered in the development processes? Here it makes sense to draw on previous experience and results in the development of human–computer interfaces in general. VR/AR can be regarded as a special case. Therefore, the following section will highlight the special features of VR/AR, before the next section focuses on the

general software engineering concept of *human-centered design* as a starting point for a successful approach to the design and development of VR/AR interaction.

6.6.1 *Characteristics of VR/AR User Interfaces*

Human–computer interaction is important for many software systems, for example in the desktop or web area. Over time, design processes have been established in these domains that are also applicable in principle to the design of user interfaces in VR. In practice, however, some special features of VR need to be considered in the design process. A crucial difference is the lack of standardization. For desktop applications and the design of websites, the available hardware (and the interaction techniques that can be used based on it) has long been assumed to be largely standardized. However, these prerequisites do not apply to the field of VR/AR. In particular, standardized hardware platforms have been developing only recently. Experience has shown that careful coordination between hardware and software is necessary to arrive at highly usable solutions. Thus, the development or the selection of interaction hardware in the design process must be considered equivalent to software development. Specifically, the following differences arise for VR/AR applications compared to other domains:

- **Development processes:** The established human-centered design processes can in principle be transferred to VR/AR applications. Differences arise primarily for individual design activities within these processes. Furthermore, the development/selection of suitable hardware must be included in the process.
- **Authoring kits:** In the desktop and web area, developers have access to established and largely standardized authoring kits of interaction and presentation elements (widgets or controls). Since their visual design and function has been optimized over the years, the developer can concentrate on the problems arising from the interaction of several widgets in a user interface. In the field of VR/AR, on the other hand, even basic interaction techniques often have to be reimplemented, so that problems can already be expected here.
- **Tools:** In the desktop area, rapid prototyping tools are widely used to quickly create user interface designs. They allow authors to compare different designs at an early stage and involve end-users in the design without programming effort. Comparable tools are only available to a limited extent for VR/AR applications. Furthermore, there are hardly any special test tools that help developers to evaluate and test.

Due to these specifics, different approaches have been developed in the past to support the design of VR/AR interactions. A common concept is to develop complex interactions based on simple building blocks provided by an appropriate toolkit. A systematic approach to this was presented e.g. by Card et al. (1990). The basis is formed by the available sensor data and a series of operators for linking, which

together span a design space for interaction techniques. This concept of the design space supports the developer in the systematic consideration of different design options. The physical properties that can be detected by input sensors (e.g., absolute and relative position, absolute and relative force) are combined with linking operators (e.g., merge, layout, connection). An interaction technique is realized by combining physical sensor data with logic operators and mapping the resulting data into the application domain. In practice, this approach is well suited to specify interaction techniques but does not provide much support for (creative) design and practical implementation. Another well-known approach to systematize the development of VR/AR interactions goes back to Bowman and Hodges (1999). It is based on a taxonomy of typical recurring interaction tasks (such as selection, positioning or manipulation of 3D objects). Based on these general interaction tasks, a division into individual subtasks is made. The technical implementation of these subtasks can then be carried out by one or more technical components. For example, the interaction task “coloring a 3D object” can be divided into the following subtasks: selection of an object, selection of a coloring tool and application of the tool to the object. The taxonomy is supplemented by various metrics that describe the suitability of a specific interaction technique in a concrete application context. In the above example, the metrics can then, for instance, provide developers with information on the advantages and disadvantages of different potential components for implementing the subtask “selecting an object”.

6.6.2 *Human-Centered Design of VR/AR Interactions*

A systematic approach that is central to *human-centered design* is suitable for developing both individual interaction techniques and complete systems with good usability. Iterative procedures have established themselves as “best practice”, which divide the development into several phases and are iterated taking the results of user tests into account. In the literature, there are various iterative process models, some of which have the status of ISO standards (e.g. DIN EN ISO 9241-210, often referred to as its outdated predecessor ISO-13407, or ISO/PAS 18152). In practice, VR/AR projects often use a procedure adapted to the specifics of the current project.

Iterative development processes are based on a cyclical sequence of *design activities* (see Fig. 6.6). The sequence of these activities is repeated until a satisfactory result is achieved. The goal is to obtain feedback from users as early and repeatedly as possible and to be guided by this feedback in the development process. The actual iterative design process is often preceded by project preparation. The following points should be addressed during project preparation:

- Defining the development goal
- Specifying a (possibly modified) development process
- Putting together the development team
- Selecting the development tools

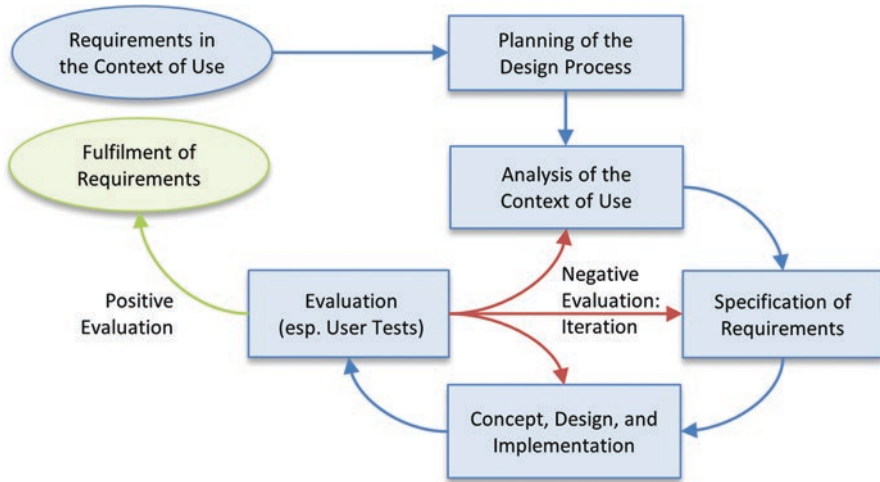


Fig. 6.6 Iterative development process according to ISO 9241-210

- Planning of user participation
- Defining quality criteria, e.g. learnability, efficiency, effectiveness, error rate, user satisfaction, user experience

In DIN EN ISO 9241-210, the procedure is structured into four central design activities (see Fig. 6.6), which are considered in more detail below:

- Analysis of the context of use
- Specification of requirements
- Concept, design, and implementation
- Evaluation (especially user tests)

Analysis of the Context of Use

The analysis and documentation of the context of use, for example through interviews, field studies and user workshops form the basis for the following development. In this activity, the user groups, the tasks to be supported and the application environment are analyzed and documented. It is important in the development of VR/AR applications that the technical environment is also analyzed here, for example, to identify the available sensors and input modalities. The specification of the usage context is not a static document in an iterative process, but is continuously checked, updated and refined during the development. Especially in VR/AR applications, in which new interaction techniques based on additional sensors are integrated, significant changes can occur during development.

Specification of Requirements

In this design activity, specific requirements for the system are identified taking into account the context of use. In addition to the requirements of the customer and the end-user, other framework conditions such as goals regarding usability, regulations for occupational safety, etc. must also be considered. As the development of VR/AR systems often involves breaking new technical and thematic ground, explorative techniques such as *scenario-based design* (Carroll 2000) have proven effective. Here, short stories are used as “scenarios” to describe a hypothetical interaction. These quickly created and easily modified prototypes enable potential users and content experts to provide feedback on the planned processes early in the design phase, even if the system (and possibly even the necessary technology) is not yet available. Since these prototypes can be created and modified quickly, it is possible to explore a wide range of different concepts at low cost. More formal approaches, such as *functional decomposition* and *task analysis*, are often more difficult to apply in VR, as with novel interaction techniques and applications the detailed requirements can often only be identified iteratively during development.

The concept of *use cases* is closely related to the scenarios in scenario-based design. Use cases also describe the interaction of users with a system. The term *use case* is sometimes used differently. One view (particularly common among designers) interprets use cases in terms of goals that are supported by an application. A VR/AR application of a complex technical system could support, for example, the use cases “trade show presentation” and “interactive training”, each of which contains different functionalities. A scenario then describes an interaction sequence in one of these use cases, for example, an interactive sequence for presenting content in the “trade show presentation” use case.

The second view (especially in software engineering) employs use cases to define an interactive system in detail. In this view, a use case consists of a list of steps (both user and system) that lead to the achievement of a goal. A central difference is the stronger formalization. Use cases in software engineering are often formulated in a formal system, e.g., in a *UML use case diagram*. This view of use cases is of particular interest if the initial exploration is already completed and an interaction concept is to be implemented.

Concept, Design and Implementation

This activity is concerned with creating designs. For the development of novel VR/AR systems, it is useful to apply a rapid prototyping strategy in an iterative process. In the first iterations, the designs are created as sketches, storyboards or mock-ups without implementation. Then, they are evaluated in the next step of the iteration (Buxton 2007). *Sketches* represent a simple graphical representation of the interface, while *storyboards* (originally used in film production) represent a dynamic interaction process as a comic-like sequence of sketches. The concept of *mock-ups* was taken from industrial design, where scale models have a long tradition. In the

context of user interfaces, the term *mock-up* can refer to both purely visual dummies and partially functional prototypes. The aim is to explore a wide range of design alternatives at a reasonable cost. This approach is particularly important for new types of user interfaces as less experience can be drawn on and design decisions should be based on user feedback – and users can only give limited feedback on a purely textual description of a user interface. In later iterations, the design representation is then increasingly refined until the implementation has become mature enough to be released.

“The purpose of the prototype is to make real the conceptual structure specified, so that the client can test it for consistency and usability.” Frederick P. Brooks Jr. (1995)

A *prototype* of a software application is a working software program that simulates some aspects of this software application while being less complex (e.g., by being less robust, by supporting only one specific type of hardware, or by omitting the implementation of data security requirements). Nielsen (1994) distinguishes between *horizontal prototypes* that simulate a broad range of the software application’s features and *vertical prototypes* that focus on simulating a smaller subset of features more in-depth with increasing functionality.

Evaluation (Especially User Tests)

In the following activity, the designs are evaluated or the implemented solutions are tested with real users. Based on the results, all or individual design activities are then iterated to improve and refine the design. Since evaluation in the form of user tests is of central importance for the development of attractive VR/AR applications, it is discussed in detail in the following section.

6.7 User Tests

Testing interactions in virtual environments is essential, especially because the complex behavior of humans cannot be modeled mathematically in such a way that the results of these tests are predictable. Therefore, user interaction in the virtual environment, as well as the user interface as a whole, is mostly designed and developed iteratively. Each iteration ends with a test. The evaluation of the test gives hints about what to change in the next iteration. Therefore, tests are not carried out only with the completely developed VR/AR system but in all development phases. The earlier that problems are detected by tests, the easier they can be solved. Testing

usability is of particular importance. Other aspects of *software ergonomics* can also be checked by tests, e.g., the user's exposure to unnatural body poses in a virtual environment and the resulting fatigue.

Participants (also called *test subjects*) must be carefully selected so that they represent the later users of the VR/AR system well. An alternative to relying on participants is *heuristic evaluation*, in which a system is evaluated by at least two experts working separately from each other based on guidelines (general guidelines such as standards and norms, for example, DIN EN ISO 9421 "software ergonomics", or product-specific guidelines).

A *test plan* must be drawn up so that the individual tests can be carried out effectively and are comparable. For this purpose, the test procedure is divided into phases. The first phase is *test preparation*, which should take place before the participant appears. After a *test introduction* (greeting, providing information on the purpose, procedure and duration of the test, and obtaining each participant's consent to take part in the test), the actual *test execution* is conducted. A participant can execute several tests one after the other (*within-group design*). This has the advantage that fewer participants are required and individual differences do not have such a strong impact. However, the participant tires more quickly, and learning effects occur, for example, if a task needs to be solved several times in different variations. Here, a test in which each participant tests only one variation (*between-group design*) is better suited. A between-group design is unavoidable if one wants to take into account characteristics (e.g., age, handedness, gender) of the users in the test – after all, a participant cannot take the test once as a young child and then minutes later as an adult. Assignments in a user test must be made *randomly*. For instance, in a within-group design, the order of the variants is determined randomly or according to a fixed variation scheme (e.g., a *Latin square*). The test instructions should be specific and should not leave room for interpretation or be sub-specified (for example, is the participant standing or sitting?). The last phase of the test is the *debriefing*, where, in addition to thanks and possible rewards, the participant should be asked for free comments. In total, the test duration should not exceed 45 min. Before the test is conducted with a large number of participants, a *pilot* (also called *pre-test*) with two to three participants should be conducted. This serves to better estimate the time required and to detect problems in the test plan (e.g., test instructions are ambiguous) at an early stage. Throughout the entire test, the test conductor should bear in mind that *ethical aspects* must be taken into account. After all, this is a test involving human beings as test subjects, so it is essential to protect privacy, be friendly and allow the test to be stopped immediately at the request of a participant. Some organizations require that a user test be approved by an ethics committee. In general, it is necessary to sign a *declaration of consent* or *informed consent*, which contains information on confidentiality, anonymization, utilization of data or possible risks, such as cybersickness, among other things.

Errors occur during the execution of the test and during measurement that can hardly be avoided. However, one should work towards minimizing *systematic errors* (*bias*). The mere fact that the participants are aware that they are being tested changes their behavior and falsifies the test results (*Hawthorne effect*). This can be

counteracted by creating a calm and relaxed test atmosphere. To avoid bias, tests should always be performed in the same way – it helps to adhere strictly to the test plan and to keep the environmental conditions (brightness in the room, temperature, volume, presence of an audience etc.) constant during all tests. The test leader should be neutral so that his or her opinion does not influence the participants. Thus, comments such as “I have worked very hard on my virtual world for three months, you will find it awesome” should be avoided. Bias can also be caused by learning effects that occur during the test. Over time, the participant becomes more and more familiar with the VR/AR system. Since the learning curve is usually steep at the beginning and then flattens out, it is advisable to first plan an introductory training session during the test – the first learning effects will then take place in this phase, in which measurements are not yet taken.

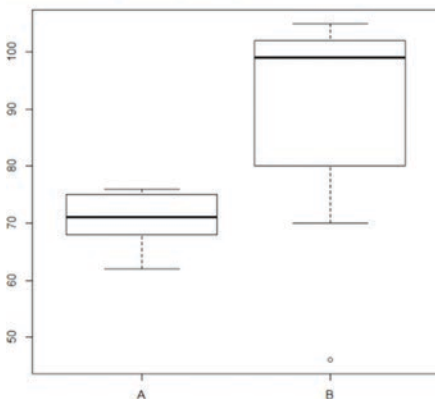
A simple test is to set tasks for participants and watch them perform them (record them on video if necessary or use logging tools like Morae and analyze them later). It is particularly effective to have the participants constantly say what they are thinking and what they are about to do (*thinking aloud test*). This helps to understand the thoughts of the participants. The test leader does not comment on the participants’ statements but just reminds the participants to verbalize. It is also helpful if the test leader demonstrates the verbalization at the beginning of the test so that it does not seem awkward or embarrassing to the participants. During the evaluation, phenomena observed throughout the interaction of participants with the VR/AR system can be described (*anecdotal data*) or *qualitative data* can be collected. A technique for *qualitative analysis* is *coding*, where a code is assigned for an incidence (e.g., “participant is frustrated”). In the end, the occurrences of the codes are analyzed. If several people independently code the same recording of a user test and the results are consistent, the observations have been objectified. *Cohen’s kappa*, a statistical measure, is used as a measure of agreement. The analysis of qualitative data is often based on the *grounded theory* of Glaser and Strauss (1967).

Interviews and *questionnaires* are further techniques to collect more data about a VR/AR system in a test. For this purpose, already existing carefully designed questionnaires can be used. Examples are the *ISONORM* questionnaire for usability (Prümper 1993), the *AttrakDiff* questionnaire for measuring user experience (pragmatic as well as hedonistic quality), the *QUIS* questionnaire (*Questionnaire for User Interaction Satisfaction*), the *task load index* of NASA (*NASA-TLX*), which deals with the stress and load users feel, or questionnaires for assessing presence such as WS (Wittmer and Singer 1998) and SUS (Slater et al. 1994). Questionnaires allow the test to be performed without the presence of a test leader. Special attention must be paid to the clear formulation of test instructions and questions (e.g., no double negations) because further questions are not possible. Questions that characterize the participant (age, gender, previous knowledge, etc.) should be asked at the end of the questionnaire, as even a participant who is tired at the end of the test can still answer these questions easily. Open-ended questions (such as “What did you find disturbing when using the VR system?”) are often not answered or not answered in detail. Nevertheless, at least one open question of the type “Do you have any further comments or remarks?” should always be included. If a test leader is

present, he or she can also ask the open questions in the form of an interview, as this gives better chances for a detailed answer. *Multiple-choice questions* are often used in questionnaires. A special form is a *Likert scale*. Here statements are made (e.g. “The navigation was easy to learn.”) and the participants can choose one of several alternative answers to express their degree of agreement with the statement (e.g., 1 – “agree fully”, 2 – “agree”, 3 – “don’t know”, 4 – “don’t agree”, 5 – “don’t agree at all”). Another special form is the *semantic differential scale*. Here, pairs of opposites are used in a statement and the users can indicate their position on a scale (usually 5 or 7 parts), for instance, “The learning of the navigation was: easy _ _ _ _ _ difficult”.

Scales such as the Likert scale in a questionnaire allow *quantitative data* to be collected. Quantitative data can also be obtained by taking further measurements in the test, such as the time taken to solve a task or the number of errors made. Three cases are distinguished for the evaluation of these data. In the first case, *nominal data* are available, i.e., data that cannot be ordered (e.g., the participant was right-handed/left-handed). In the evaluation, these data are described by their ratio (e.g., in the user test 12% of the persons were male, 88% female). They can be visualized with a pie chart. In the second case, the data is either *ordinal*, i.e., the data can be put in a sequence (e.g., interaction technique A is rated better than interaction technique B – but no statement is made as to whether it was better by a short margin or by lengths), or *rational*, i.e., here differences between the values have a meaning and the values can be put into a ratio (e.g., a participant needed twice as long (80 s) to complete a task with interaction technique A than with interaction technique B, where it took 40 s). The data is evaluated by determining *q% quantiles*, where *q* lies in the interval from 0 to 100. For example, the 30% quantile is the observed value *w* where 30% of all values are smaller than *w*. The 50% quantile, the *median*, is always determined. The data can be visualized in a bar chart or a *box plot* (also called a *Tukey box plot* or a *box-and-whisker diagram*); see Fig. 6.7. The values of scales

User Test Example 1: Visualization of the Results as a Tukey Box-Plot:



Reading a Tukey Box-Plots:

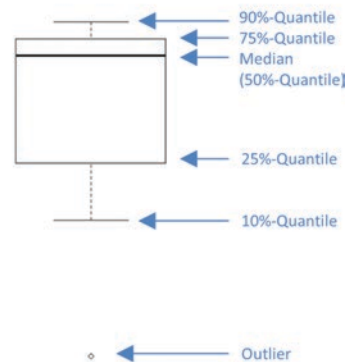


Fig. 6.7 Visualization of the test results with a box plot

from questionnaires such as the Likert scale are ordinal. It is controversially discussed whether they can also be treated as rational data – this would mean, for example, that the difference in agreement between “fully agree” and “agree” is as great as the difference between “don’t know” and “don’t agree”. The third case is when data is both rational and *normally distributed*, i.e. the distribution function of the data is in the form of a *Gaussian bell curve*. In this case, it is sufficient to calculate the *arithmetic mean* μ and the *standard deviation* σ . As a rule of thumb, a normal distribution can be assumed if the number of measured values is large (usually greater than 50) or if more than 99.7% of all measured values lie in the interval $[\mu - 3\sigma, \mu + 3\sigma]$. The presence of a normal distribution can be checked more precisely with a statistical test, the *Shapiro-Wilks test*.

User Test Example 1: In a user test, nine users had the task to perform a task once with VR system A and once with VR system B (whereby it was randomly determined which VR system was used first). The times for task completion were measured. The following results were obtained (the first number in the tuple indicates the time in seconds for A, the second the time for B): (66,102), (75,80), (62,81), (74,46), (71,105), (76,70), (70,100), (68,99), (75,102). We formulate our null hypothesis: “There is no difference in task processing time between the two systems”. Our hypothesis to be tested, “A is faster than B, or B is faster than A” comprises two possibilities. Hence, we must perform a *two-tailed* test. There is a within-group design (connected sample) with two groups and rational data. Therefore, we carry out a Wilcoxon signed rank test and obtain a *p*-value of 7.422%. The *p*-value is higher than our threshold of 5%. Due to the small number of participants, we cannot assume normally distributed data. In fact, the Shapiro-Wilks test shows that our data is not normally distributed. As a result, we must not use the paired *t*-test. All in all, we cannot make any statements regarding the hypothesis in our user test.

Since usually not all potential users of the VR system are tested, but only a *sample* instead of a *complete survey*, the question remains how meaningful the test result is. Let us assume that four tests have been carried out and all participants rate the VR system A better than B. Can we conclude that A is actually better? Assuming that both systems are equivalent on average (*null hypothesis*), 50% of all users would have to prefer A. Of course, it could be that we have accidentally caught four people among our participants who prefer A – just as a perfect coin can be flipped four times in a row and show “heads” four times. The probability for this case, if the null hypothesis holds, is $(0.5)^4 = 6.25\%$. In other words, if we answer the initial question with “yes”, i.e., reject the null hypothesis, the probability is 6.25% that we are wrong (and A is not better in reality after all, and we were unlucky in our sample

and happened to test an unrepresentative selection of users). As a rule, a *probability value* (*p-value*) of at most 5% is required. In our example, based on the test, we cannot prove that A is better than B in a *statistically significant way* at the *significance level of 5%*. No statement can be made. There are several statistical methods to calculate this *p-value*, which are shown in Table 6.1. Different tests are used depending on the type of data and the question. A distinction is also made as to whether a within-group design (paired groups) or a between-group design (unpaired groups) was used in the test. If you have more than two groups (e.g., five interaction techniques) that you want to compare, you can compare two groups in pairs. However, the significance level should be divided by *n* when statistically evaluating test data *n-fold* (*Bonferroni correction*). Therefore, there are special tests such as *ANOVA* (*analysis of variances*) which look at several groups simultaneously. The *p-values* for the individual statistical tests can be calculated with software, e.g., with spreadsheet programs such as *Microsoft Excel* or with statistical packages such as the commercial *SPSS* or the free software *R* (r-project.org).

User Test Example 2: In a user test, 10 participants had tested VR system M and 12 further participants had tested VR system N. All participants evaluated the system in a questionnaire with school grades (A to F). As a result M received the following grades: 3 × “A”, 2 × “B”, 3 × “C”, 2 × “D”, N received the following grades: 1 × “A”, 3 × “C”, 5 × “D”, 3 × “E”. The data are ordinal and there is a between-group design (unpaired groups). We apply the Mann-Whitney U-Test (also called the Wilcoxon *rank sum test*) as a *one-tailed* test. To conduct the test, we code grade “A” as “1”, grade “B” as “2” and so forth. We formulate the following hypothesis: “M was rated better than N”. We obtain a *p-value* of 0.789%, which is less than 5%. As a result, we can say that our test confirmed the statement “M was rated better than N” in a statistically significant way.

If one wants to determine whether two variables are related in measurements (e.g., the evaluation of an interaction technique with the handedness of a person), *correlations* can be calculated (see Table 6.1) to quantify this relationship. If there are more than two variables, a *regression analysis* is carried out. Especially in scientific studies, one is not only interested in correlations, but also in *causations* (and correlation does not imply causation). For this purpose, the methodology of the *controlled experiment* is used. Here, all *factors* are identified that can influence a measurable result, the *score*. In the experiment, all factors except one, the *treatment factor*, are kept constant. In this way, cause–effect relationships between changes in the treatment factor and changes in the score can be examined.

User Test Example 3: In a user test with two interaction techniques A and B, the following results are obtained: 12 right-handed persons prefer A, 23 right-handed persons prefer B, 18 left-handed persons prefer A, and 9 left-handed persons prefer B. Our hypothesis is: “There is a connection between handedness and preference of an interaction technique”. We have nominal data available with two unpaired groups. We run Fisher’s exact test and get a *p*-value of 2.036%. This is less than 5% and our data show statistically significantly that there is a correlation between handedness and interaction technique. By calculating the contingency coefficient (according to Pearson) we can quantify the strength of the correlation. In our example, it has a value of 0.306. A value of 0 would mean no correlation, a value of 1 would mean maximum correlation.

Table 6.1 Application of statistical methods in the evaluation of user tests

Task	Nominal	Ordinal or rational	Rational and Normally distributed
Statistic description	Relative frequencies	q% quantiles, especially median	Arithmetic mean, standard deviation
Compare 1 group with hypothetical value	Chi-square test	Wilcoxon signed rank test	One-sample <i>t</i> -test
Compare 2 groups, unpaired	Fisher’s test	Mann-Whitney-U-test (Wilcoxon rank sum test)	Unpaired <i>t</i> -test
Compare > 2 groups, unpaired	Chi-square test	Kruskal-Wallis test	One-way ANOVA
Compare 2 groups, paired	McNemar’s test	Wilcoxon signed rank test	Paired <i>t</i> -test
Compare > 2 groups, paired	Cochran’s Q test	Friedman test	Repeated measures ANOVA
Quantify the relationship between two variables	Contingency coefficient (after Pearson)	Spearman correlation	Pearson correlation

6.8 Ethical, Social and Legal Aspects of VR/AR

Why is it essential to actively deal with ethical or legal aspects when designing or providing interactive VR/AR systems to users? Why can it have moral or legal consequences if you fail to do so? An essential answer to these questions is that VR/AR can have significant effects on users. This becomes very obvious when users of a

VR system vomit due to cybersickness or when users of AR stumble and fall down because they perceive reality only partially. There are also less direct and visible effects. It is easier to distance oneself emotionally from a movie when watching it sitting at a certain distance to the screen than being fully immersed in a virtual environment. Users experience themselves *as part of* the virtual world (as opposed to *as an observer* of movies) and as part of the events. The more actively the user can interact with the virtual world and the more physical stimuli are experienced congruently with the virtual environment, the more credibly and “real” this world is experienced. Thus, the experiences potentially have a stronger effect on the psyche and self-model (Madary and Metzinger 2016) and thus also, often unconsciously, on current thoughts, feelings and behavior.

Behavioral changes can be induced, for example, by choosing an avatar with a certain size, skin color, gender and body dimensions. Research has shown the effect of *priming* through *stereotypes*, i.e., ideas and images typically associated with certain social groups (e.g., defined by race or gender) such as their behavior and abilities. People react differently to such priming in social contexts. The extent to which people are confident in themselves and perform also depends on this. The choice of a certain type of avatar in a virtual world can be a form of priming and can lead to a change of behavior later on in reality, even if the virtual world has already been left. Peck et al. (2013) have shown in an immersive experiment that after a stay in a virtual world in the body of a dark-skinned avatar, implicit racial prejudices decreased significantly, in contrast to the control groups without a dark-skinned VR avatar. Piryankova et al. (2014) were able to show the effect of experienced altered versions of one’s own body (thicker, thinner) on the self-image.

The term “*rubber hand illusion*” describes the phenomenon of triggering a subjective perception with VR/AR in which we are no longer our own body, but identify with an avatar, at least in partial aspects (Lenggenhager et al. 2007; Metzinger 2014) – just as one can trick people to integrate a rubber hand into their own body awareness. This transformation is created, for example, when a user feels a stimulus on his or her own back but can only see it on the distant avatar (see Fig. 6.8). In doing so, the sense of self can, so to speak, extend in the direction of the avatar. This is called a *whole-body illusion*. Other methods, for example, showing your own physical heartbeat as a pulsating aura around the avatar, can lead to similar effects. If one investigates the virtual environment from the perspective of an avatar, tactile sensations can be induced with visual methods. With these methods, events that happen to the avatar in the virtual world can be experienced more strongly as “happening to ourselves”, with all the possible consequences that can result from this.

These and other presentations of illusory content and contexts that cannot be experienced in the real world can lead to a change in the inner models of “how the world works” (Madary and Metzinger 2016) and thus to manipulation of internal assumptions and models. These modified internal models, no longer learned in the “real” world alone, may then possibly determine the user’s actions in the real world.

Fig. 6.8 Synchronous stroking of the user's and avatar's backs can cause a "(partial) transfer of the sense of self" to the remote avatar. (© Steffi Beckhaus. All rights reserved)



An example of this can be found in clinical psychology. Rizzo and Koenig (2017) show the progress made since the mid-1990s in the use of VR to treat post-traumatic stress disorders and many types of other psychological, motor and functional impairments. It becomes clear that such “manipulations” of people by VR can later lead to dysfunctions in the real world if VR therapy is not professionally accompanied, used responsibly and in full knowledge of possible consequences. The deceptive presentation and misleading of senses can have dire consequences, such as depersonalization. Cases of cyberbullying, immoral assaults or violence are known from non-regulated social VR worlds. A further problem area is an escape from reality (*escapism*).

Consequently, VR/AR can have diverse effects on users, including long-term effects, both obvious and not so obvious. This raises a lot of questions: How is the physical and mental health of users guaranteed? What are the effects of permanent residence in an immersive environment? If a user of AR can no longer distinguish between real and virtual content, what ethical and legal rules should apply in such an environment? What responsibility do developers, operators and users of VR/AR systems take? How do you make sure that this responsibility is accepted? Technology in general, and VR/AR in particular, is neither good nor bad. It is the effects of use on people and society that can become problematic, that must be estimated in time – and that must be considered when implementing and using VR/AR. It must be

considered that people not only create new technologies such as VR/AR and thus change their environment, but that they themselves are also changed, individually and as a society (e.g., with regard to self-perception, world view and judgment).

Values: internalized ideas in a socio-cultural unit that are recognized as desirable (e.g., sincerity, justice, loyalty).

Morality: The actual principles of action of a community, based on common values and traditional customs and traditions; they enable an intuitive distinction to be made between “good” and “evil” and provide practical guidance on alternative courses of action.

Ethics: Scientific discipline of philosophy that deals with the recognizability of values and the argumentative substantiation and precise formulation of moral principles. In everyday use, it is usually erroneously used synonymously with “morality”.

Law: External obligations of a party to act, which are usually sanctioned in case of violation. Law often has a moral quality, but does not regulate all ethical questions and also considers other issues (e.g., right/left-hand driving in road traffic).

ELSI/ELSA: Ethical, legal and social implications/aspects. Research activities that consider the non-technical aspects as an integral part of project work, especially in human–technology interaction projects.

Technology Assessment (TA): A research area in the philosophy of technology that considers developments in science and technology. Ideally, TA should support the public to form an opinion on the implications and consequences of using technology. TA should also formulate recommendations for action to avoid risks and to exploit opportunities for society.

First indications of how to answer the questions raised above are available from various sources. The *ethics guidelines* of renowned computer science and engineering associations, such as ACM (2018), point out the essential aspects of ethical and social development of technology and its applications. Legal regulations that are significant for information technology systems can be found in many different places in the legal system, including data protection law (including employee data protection), freedom of information law, computer criminal law, intellectual property law, IT security law, telecommunications law, media law, youth protection law and consumer protection law. Certification standards attempt to ensure the safety of commercial products. For example, IEC 62366-1:2015 obliges manufacturers of medical products to prove through evaluation that users are not jeopardized by incorrect operation. This means that there are implicit regulations for VR/AR applications, but these are usually only formulated in general terms and do not cover all

aspects. For example, effects regarding possible changes in the personality of users are currently not regulated at all.

There are further indications from initial research articles on ethics and VR/AR, for example in the area of possible legal guidelines concerning VR (Spiegel 2017) or the ethical aspects of the use of VR in clinical psychology already mentioned (Rizzo and Koenig 2017). A largely unexplored aspect is the question of *long-term consequences* for the individual, but also society as a whole. This includes questions like:

What happens if we repeatedly do not experience real things through our senses but only perceive simulated stimuli? We know that humans adapt quickly to sensory inconsistencies. What happens if we are successful with actions that were previously impossible in reality? How do we deal in the real world with the habit acquired in VR that virtual actions have no real consequences?

How does the self-image change through the possibility/experience of one or many *alternative self-representations* (avatars) each with their own physical appearance, movement and behavior (identity tourism)? What effect do incongruent, distorted, or for the real world wrong, physiological and psychological stimuli have on the internal models of the human being?

What are the consequences of excessive use (*overuse*), escapism, physical neglect and self-image change on society, e.g., follow-up costs in the health care system?

What is the influence on people and societies through the realistic experience of fictional cultures, behavior and moral standards?

In this field of guidelines, laws, opportunities, risks and open questions, it is crucial that researchers, developers and product manufacturers should now be able to assess the advantages and risks of their developments in an informed and comprehensive manner – to meet their responsibilities. Otherwise, they could harm their users and possibly be sued for this. This is not an easy task when given the complexity of the subject matter. In addition, the effects of VR/AR on people and society are not always obvious. It is also often difficult to causally attribute observed actions of users to VR/AR.

The five *perspectives* shown in Fig. 6.9 help to get an overview of the topics to be considered, the complex interrelationships, and possible effects of VR/AR for own research and project developments. A forward-looking risk assessment, a technology impact assessment and a social cost-benefit assessment are also necessary. Contents, procedures, forms of presentation, interaction possibilities, explanations and consent must be considered responsibly from each of these perspectives. The overarching view should be that VR/AR technologies are not developed and used as an end in themselves but should clearly serve people and their interests in the context of use. In certain cases, the deliberate violation of such guidelines may be necessary, for example in the course of research on humans. However, this must be appropriately reflected and documented. Ethics committees, such as those that exist at most universities, can provide support in this regard.






View	Aspects	Fields of Action (Examples)
 Person	Physical and Psychological Integrity of the Users	<ul style="list-style-type: none"> Ensure physical and psychological integrity, e.g., avoiding accidents or cybersickness, preventing harassment Ensure distancing to the virtual avatar (user representation) or agent (artificial identity) that might be harmed Support a sense of security, a sense of well-being, and a sense of community Ensure that users have full control over their virtual experience – they can decide to abort or continue anytime Ensure a clear distinction between virtual and real contents/human-generated and computer-generated contents, e.g. avatar vs. agent.
	Personal and Social Well-being	
	Autonomy, Freedom, Control with regard to own experiences	
 Law	Transparency	<ul style="list-style-type: none"> Ensure that users are fully conscious about the manipulation of senses, about contents, about short and long term consequences etc. by providing advance information or by providing opportunities for a debriefing Obtain consent explicitly (e.g., Which data may be recorded such as tracking data, movement data, physiological data? Who will be able to access the data? When is data deleted?) Preserve the users' rights, e.g., concerning their avatar, their performance, their creation of contents, their social interaction, their sensory data, their physiological data, their inputs provided in an application context
	Protection of Privacy and Data Security	
	Intellectual Property Rights	
 VR/AR as Tool	Cybersickness	<ul style="list-style-type: none"> Ensure consistency in multimodal techniques for interaction and presentation, eliminate latency. Assess the consequences of excluding the real world and the consequences of discrepancies in perception between the real world and a virtual world Increase the quality, benefits and usability of the technological components utilized in the virtual environment Assess the costs, benefits, safety and security of the components used. VR/AR should not be viewed as an end in itself. Carefully consider cables, tripping hazards, ergonomics etc.
	Consequences of Immersion, 3D	
	Costs and Safety	
 Implications	Consequences for the Users	<ul style="list-style-type: none"> Consider potential consequences of the manipulation of (1) the senses, (2) the users' self-image, (3) the contents with regard to the users' behavior in the real world after the VR/AR experience Consider the consequences for society, e.g., with regard to social interaction, to the health system (e.g., costs for therapy), to the image of human beings in society
	Consequences for Society	
 Contents	Violence	<ul style="list-style-type: none"> Prevent physical and psychological violence – protect yourself and others. Prevent assaults (e.g., obscene activities) and violations of the private sphere. Take extra precaution in unregulated and unsupervised environments. Avoid stereotyping. Support diversity and fairness (e.g., avatars should be designed on an equal footing – male avatars, for instance, should not look normal while female avatars are exaggerated). Do not manipulate users by letting them make experiences that are impossible in the real world and might negatively alter their behavior in normal life – or at least make it clear and explicit that there are deviations from the real world. Do not use subtle manipulation, nudging, and deception. Do not mislead users in their perception of authenticity. Be responsible when using psychological manipulation and manipulation of the body (e.g., manipulation of the user's body-image, psychological integrity and stability)
	Private Sphere	
	Stereotyping	
	Manipulation of Senses	
	Manipulation of Contents	

Fig. 6.9 Five perspectives that can show the effects of VR/AR and corresponding fields of action

6.9 Summary and Questions

To ensure that users do not feel like apathetic or even paralyzed persons in a virtual environment, it is essential to enable interaction between the users and the virtual world in VR or virtual elements in AR. Thus, designing and creating a user interface is a standard task. This interface uses interaction techniques. With some of them, the user might already be familiar, as they stem from traditional user interfaces. Some interface techniques, however, are specific to VR, such as the world-in-miniature technique or the camera-in-hand technique. Another novel aspect of interaction in VR and AR is that it opens up unusual design possibilities: for example, interactions can be natural or magical. In every VR and AR, there are typical basic tasks that can be solved by suitable interaction techniques: selection (of objects or places, surfaces, volumes), manipulation, navigation and system control. The individual basic tasks can be refined so that navigation can be separated into wayfinding and traveling – and traveling again can be separated into exploration, search and maneuvering. Overall, the selected interaction techniques are to be integrated into an overall concept of a consistent user interface and design decisions are to be made, e.g., whether constraints or modes are to be introduced. Here, results from the field of human–computer interaction can be helpful. Thus, especially in the field of HCI, process models developed for the design and realization of user interfaces can generally be transferred to interaction in VR and AR. Early involvement of the users, the execution of user tests (and their careful evaluation, especially with statistical methods) and an iterative procedure can also be advantageously applied to VR/AR systems to achieve a high level of usability. So far, there are hardly any empirical values on how VR/AR affects the individual user. Ethical, legal and social implications have to be considered continuously with their different perspectives.

Check your understanding of the chapter by answering the following questions:

- Find examples of interaction techniques that can be classified as *tethered*!
- In a virtual operating room, the user has the task of selecting the right instruments from a shelf with surgical instruments and handing them to the surgeon in a suitable orientation. Select a suitable selection technique and a suitable manipulation technique. How can feedback be given to the user (as required for direct manipulation)?
- The limitation to five instead of six degrees of freedom in interaction in virtual environments, as mentioned in Sect. 6.3.2, can also be seen in the first-person shooter computer games available on the market. Which degree of freedom is usually not used here? With regard to the input devices used: Why is this the case and how would you integrate the sixth degree of freedom into such games? Which input devices would you use for the realization?
- Develop a navigation technique based on the metaphor of a “flying carpet”. What constraints are you introducing? Which modes could be useful here? How do you classify this navigation technique? Which navigation tasks can be covered?

- You are developing a VR application for spatial planning. The users should be able to freely position furniture in a room and manipulate its dimensions and materials. Which questions should be answered in the analysis of the usage context? Develop the first specification of requirements!
- Two important aspects of your VR application for spatial planning are loading room geometry and placing furniture there. In which categories of basic interaction tasks are you looking for suitable techniques? Which techniques would be your first choice to implement these tasks?
- How would you change the concept of the spatial planning application if you use AR instead of VR, i.e., you place virtual furniture in a real room?
- To move to a very distant destination in a virtual world, one does not want to ask the user to walk for 20 min. One considers (a) displaying a selection list permanently at the bottom of the screen from which the user can choose a destination or (b) recognizing a gesture (crossing the arms behind the head) of the user and interpreting the following voice input as a destination. What are the advantages and disadvantages of these two alternatives? Suggest your own implementation alternative (c). How can you find out which of the three alternatives (a), (b) and (c) least impairs the feeling of user presence in the virtual environment?
- How high would the p -value be in user test example 1 if the values did not come from 9 users, but from 18 participants, of which 9 users tested system A and 9 users system B? [Answer: 2.412%]
- Which p -value would have been obtained in user test example 1 if the hypothesis “With A you are faster than with B” had been chosen (i.e., if a one-sided test had been performed)? [Answer: 3.711%]
- You are designing a VR application in which users control an avatar on a catwalk in order to show self-designed fashion to other users, who are also represented in the virtual world as avatars. Based on the five views in Fig. 6.9, consider which ELSI aspects are to be considered here, for what reason, and how. To what extent does it make a difference whether mirrors are present in the virtual world or not? What changes when AR is used to show an avatar on a real catwalk?

Recommended Reading

- Bowman DA, Kruijff E, Laviola JJ (2004) *3D user interfaces: theory and practice*. Addison-Wesley, Amsterdam – Standard textbook that discusses user interfaces in 3D.
- Lazar J, Feng JH, Hochheiser H (2017) *Research methods in human–computer interaction*, 2nd edn. Morgan Kaufmann, San Francisco – Detailed presentation of various research methods relevant to VR interaction, including controlled experiments and ethnography.
- Rubin J, Chisnell D (2008) *Handbook of usability testing*, 2nd edn. Wiley, New York – Practice-oriented book that shows how to plan, conduct and evaluate usability tests.

Shneiderman B, Plaisant C, Cohen M (2016) *Designing the user interface: Strategies for effective human–computer interaction* (6th revised edn). Addison-Wesley Longman, Amsterdam – Standard textbook for the field of human–computer interaction.

Tullis T, Albert W (2013) *Measuring the user experience*, 2nd edn. Morgan Kaufman, San Francisco – Book that focuses on measuring in the field of human–computer interaction and presents a variety of metrics.

Further information on the topic of interaction in VR can be found on the numerous websites of research institutions and especially in the conference proceedings of the corresponding conferences and workshops, e.g., IEEE Virtual Reality (IEEE VR), IEEE Symposium on 3D User Interfaces (3DUI), ACM Symposium on Virtual Reality Software and Technology (VRST), ACM Symposium on User Interface Software and Technology (UIST), ACM SIGCHI Conference on Human Factors in Computing Systems (CHI), IEEE Symposium on Mixed and Augmented Reality (ISMAR), Eurographics Symposium on Virtual Environments (EGVE), and the EuroVR Conference.

References

- ACM (2018) The ACM code of ethics: guiding members with a framework of ethical conduct. Association for Computing Machinery. <https://www.acm.org/about-acm/code-of-ethics>. Accessed 1 Apr 2018
- Beckhaus S, Blom K, Haringer M (2007) ChairIO – the chair-based interface. In: Magerkurth C, Rötzer C (eds) *Concepts and technologies for pervasive games: a reader for pervasive gaming research*. Shaker Verlag, Aachen
- Bellotti V, Back M, Edwards WK, Grinter RE, Henderson A, Lopes C (2002) Making sense of sensing systems: five questions for designers and researchers. In: *Proceedings of CHI 2002*, pp 415–422
- Benford S, Fahlen L (1993) A spatial model of interaction in large virtual environments. In: *Proceedings of ESCW 1993*, pp 109–124
- Boletsis C (2017) The new era of virtual reality locomotion: a systematic literature review of techniques and a proposed typology. *Multimodal Technol Interact* 1(4):24
- Bowman DA, Hodges LF (1999) Formalizing the design, evaluation, and application of interaction techniques for immersive virtual environments. *J Vis Lang Comput* 10:37–53
- Bowman DA, Kruijff E, Laviola JJ (2004) *3D user interfaces: theory and practice*. Addison-Wesley, Amsterdam
- Bozgeyikli E (2016) *Locomotion in virtual reality for room scale tracked areas*. Graduate theses and Dissertations, University of South Florida, Scholar Commons. <http://scholarcommons.usf.edu/etd/6470>. Accessed 31 Aug 2018
- Brooks Jr FP (1995) *The mythical man-month* (Anniversary edn). Addison-Wesley, London
- Buxton B (2007) *Sketching user experiences: getting the design right and the right design*. Morgan Kaufmann, San Francisco
- Card S, Mackinlay J, Robertson G (1990) The design space of input devices. In: *Proceedings of CHI 1990*, pp 117–124
- Carroll JM (2000) *Making use: scenario-based design of human–computer interactions*. MIT Press, Cambridge

- Dachselt R, Hübner A (2007) Virtual environments: three-dimensional menus: a survey and taxonomy. *Comput Graph* 31(1):53–65
- De Boeck J, Raymaekers C, Coninx K (2005) Are existing metaphors in virtual environments suitable for haptic interaction. In: *Proceedings of VRIC 2005*, pp 261–268
- Fernandes AS, Feiner SK (2016) Combating VR sickness through subtle dynamic field-of-view modification. In: *IEEE symposium on 3D user interfaces*, pp 201–210
- Ferracani D, Pezzatini D, Bianchini J, Biscini G, Del Bimbo A (2016) Locomotion by natural gestures for immersive virtual environments. In: *Proceedings of 1st international workshop on multimedia alternate realities*. ACM, New York, pp 21–24
- Foley JD, van Dam A, Feiner SK, Hughes JF (1993) *Computer graphics: principles and practice*. Addison-Wesley, Boston
- Glaser BG, Strauss AL (1967) *The discovery of the grounded theory: strategies for qualitative research*. Transaction Publishers, Rutgers
- Jacob RJK (1990) What you look at is what you get: eye movement-based interaction techniques. In: *Proceedings of CHI 1990*, pp 11–18
- Langbehn E, Lubos P, Bruder G, Steinicke F (2017) Bending the curve: sensitivity to bending of curved paths and application in room-scale VR. *IEEE Trans Vis Comput Graph* 23(4):1389–1398
- Langbehn E, Lubos P, Steinicke F (2018) Evaluation of locomotion techniques for room-scale VR. Joystick, teleportation, and redirected walking. In: *Proceedings of virtual reality international conference (VRIC)*, pp 1–9. <https://doi.org/10.1145/3234253.3234291>
- Lenggenhager B, Tadi T, Metzinger T, Blanke O (2007) Video ergo sum: manipulating bodily self-consciousness. *Science* 317:1096–1099
- Madary M, Metzinger TK (2016) Real virtuality: a code of ethical conduct. Recommendations for good scientific practice and the consumers of VR-technology. *Front Robot AI* 3:3
- Metzinger T (2014) *Der Ego Tunnel*. Piper, München
- Nielsen J (1994) *Usability engineering*. Morgan Kaufmann, San Francisco
- Nilsson NC, Peck T, Bruder G, Hodgson E, Serafin S, Whitton M, Rosenberg ES, Steinicke F (2018) 15 years of research on redirected walking in immersive virtual environments. *IEEE Comput Graph Appl* 38(2):44–56
- Peck TC, Fuchs H, Whitton MC (2011) An evaluation of navigational ability comparing redirected free exploration with distractors to walking-in-place and joystick locomotion interfaces. In: *Proceedings of IEEE virtual reality*, pp 55–62
- Peck TC, Seinfeld S, Aglioti SM, Slater M (2013) Putting yourself in the skin of a black avatar reduces implicit racial bias. *Conscious Cogn* 22(3):779–787
- Piryankova IV, Stefanucci JK, Romero J, de la Rosa S, Black MJ, Mohler BJ (2014) Can I recognize my body's weight? The influence of shape and texture on the perception of self. *ACM Trans Appl Percept* 11(3):1–18
- Prümper J (1993) Software-evaluation based upon ISO 9241 part 10. In: Greching T, Tschegli M (eds) *Human computer interaction*. Springer, Berlin
- Raskin J (2000) *The humane interface. New directions for designing interactive systems*. Addison-Wesley Longman, Amsterdam
- Razzaque S (2005) *Redirected walking*. Dissertation, University of North Carolina at Chapel Hill
- Reddit (2018) List of VR locomotion techniques. https://www.reddit.com/r/Vive/wiki/locomotion_methods. Accessed 31 Aug 2018
- Rizzo A, Koenig ST (2017) Is clinical virtual reality ready for prime time? *Neuropsychology* 31(8):877–899
- Shneiderman B, Plaisant C, Cohen M (2016) *Designing the user interface: strategies for effective human-computer interaction*, 6th revised edn. Addison-Wesley Longman, Amsterdam
- Slater M, Usoh M, Steed A (1994) Depth of presence in a virtual environment. *Presence* 3(2):130–144
- Spiegel JS (2017) The ethics of virtual reality technology: social hazards and public policy recommendations. *Sci Eng Ethics* 24:1537–1550

- Suma E, Finkelstein SL, Reid M, Ulinski A, Hodges LF (2009) Real walking increases simulator sickness in navigationally complex virtual environments. In: Proceedings of IEEE VR 2009, pp 245–246
- Suma E, Clark S, Krum D, Finkelstein S, Bolas M, Warte Z (2011) Leveraging change blindness for redirection in virtual environments. In: Proceedings of IEEE virtual reality, pp 159–166
- Sun Q, Patney A, Wei LY, Shapira O, Lu J, Asente P, Zhu S, McGuire M, Luebke D, Kaufman A (2018) Towards virtual reality infinite walking: dynamic saccadic redirection. *ACM Trans Graph* 37(4):1–13
- Usuh M, Arthur K, Whitton MC, Bastos R, Steed A, Slater M, Brooks FP Jr (1999) Walking > walking-in-place > flying, in virtual environments. In: Proceedings of SIGGRAPH 1999, pp 359–364
- Wang J, Lindeman RW (2011) Comparing isometric and elastic surfboard interfaces for leaning-based travel in 3D virtual environments. In: IEEE symposium on 3D user interfaces, pp 31–38
- Wang J, Lindeman RW (2012) Leaning-based travel interfaces revisited: frontal versus sidewise stances for flying in 3D virtual spaces. In: Proceedings of VRST 2012, pp 121–128
- Wilson PT, Kalescky W, MacLaughlin A, Williams B (2016) VR locomotion: walking > walking in place > arm swinging. In: Proceedings of 15th ACM conference on virtual-reality continuum and its applications in industry, vol 1, pp 243–249
- Winograd T, Flores F (1986) *Understanding computers and cognition: a new foundation for design*. Addison-Wesley, Boston
- Wittmer BG, Singer MJ (1998) Measuring presence in virtual environments: a presence questionnaire. *Presence* 7(3):225–240

Chapter 7

Real-Time Aspects of VR Systems



**Mathias Buhr, Thies Pfeiffer, Dirk Reiners, Carolina Cruz-Neira,
and Bernhard Jung**

Abstract The term *real-time* refers to the ability of computer systems to deliver results reliably within a predictable – usually as short as possible – time span. Real-time capability is one of the most difficult requirements for VR systems: users expect a VR system to let them experience the effects of interactions without noticeable delays. This chapter deals with selected topics concerning the real-time capability of VR systems. In the first section, an overall view of VR systems shows which types of latencies occur between user input and system reaction. It also discusses how latencies of the sub-components of VR systems can be estimated or measured. The second section presents common methods for efficient collision detection, such as the use of bounding volumes, which are important in real-time simulation of dynamic virtual worlds. The third section deals with real-time aspects when rendering virtual worlds.

7.1 Latency in VR Systems

A fundamental characteristic of VR systems is their interactivity. Realistic immersive experiences in a virtual world are only possible when users can immediately perceive the consequences of their actions and relate them to their own behavior. For example, when a user pushes a real button of an input device or a virtual switch in the simulation, the effects of this action must be experienced within a response time that corresponds to the user's expectations. The time span between action (input) and reaction (system response) is called *latency*. The greater the latency of the system, i.e., the greater the time interval between an action and its perceivable consequences, the less likely it is that users will associate the new world state with their own actions. This effect can also be observed in the real world: when

Dedicated website for additional material: vr-ar-book.org

B. Jung (✉)

Institute for Informatics, Technical University Bergakademie Freiberg, Freiberg, Germany
e-mail: jung@informatik.tu-freiberg.de

© The Author(s), under exclusive license to Springer Nature
Switzerland AG 2022

R. Doerner et al. (eds.), *Virtual and Augmented Reality (VR/AR)*,
https://doi.org/10.1007/978-3-030-79062-2_7

energy-saving light bulbs were first introduced, they had a rather long latency. In the transition phase, it happened quite often to the author of this section that after flipping a light switch and observing no immediate reaction, he flipped the switch off and on again – this of course had the opposite effect: the waiting time for the light to turn on increased significantly, and thus also the frustration with the system.

In the context of this book, the frequently used term *real-time capability* also describes this relationship. A system is called *real-time capable* if it is able to deliver results to an input reliably within a predictable time period. In VR systems the latency should be below the human perception threshold. For the visual sense, for example, 1/60 of a second is usually considered sufficient. In some other areas of information technology, the term “real-time” is interpreted more strictly, in that a guaranteed reliability is demanded: a system is considered to be real-time capable if it guarantees to be able to respond to an input within a defined period of time. Although this interpretation would also be desirable for VR systems, constant latencies cannot usually be guaranteed.

An example of an undesired effect in VR caused by latency occurs when moving a virtual tool that is coupled to the user’s hand movements via a tracking system: due to latency, the tool is not directly carried along with the hand, but rather, especially in the case of fast movements, is pulled at a greater or lesser distance. In this case, the total latency is made up of delays from the tracking system, network communication and graphics output. For the graphical output part, real-time capability means, for example, that images can always be rendered and displayed at such speed that the user cannot perceive any single image sequence. However, this state is difficult to achieve in practice, as a simple change of perspective by the user can lead to situations in which the graphical system (the graphics hardware) is no longer able to compute the next image fast enough because the complexity of the now visible scene is too large or the required data is not directly available.

The graphics system and the communication network of the tracker are only two of many parts of a VR system where latencies occur. In order to operate an interactive VR system, it is important to be aware of and, ideally, quantify all latencies that occur. Knowing the potential sources of latencies and how to determine these latencies should already inform the design of VR systems and applications but is also useful for optimizations in later stages of development. This section first discusses the concept of latency in the context of VR systems by addressing the requirements, sources and methods for estimation and measurement of latencies. Sections 7.2 and 7.3 show possible solutions for VR-related subproblems, which can be used to design real-time capable, and thus low-latency, VR systems.

7.1.1 What Are the Requirements on Latency?

A specific feature of VR/AR systems is view-dependent image generation based on head tracking. Here, strong requirements exist on latency, especially when head-mounted displays (HMDs) are used. As users can only see the virtual world but no

longer their own body, high latency has a particularly negative effect on the users' well-being. This can lead to dizziness and *cybersickness* (see Chap. 2). Meehan et al. (2003), for example, found a significantly higher number of people suffering from *vertigo* when they increased the latency of an HMD from 50 ms to 90 ms. A latency below 50 ms is recommended for HMDs (Brooks 1999; Ellis 2009). In stationary projection systems, such as CAVEs, latency requirements are not as hard compared with HMDs. Here, when users turn their head, an image with the approximately correct perspective is already displayed, thus reducing the dissonance between the expected image and the presented image. A more detailed analysis of the interaction between different parameters of a simulation and the still perceivable latency can be found in Jerald et al. (2012).

When it comes to VR and AR, latency is fundamental – if you don't have low enough latency, it's impossible to deliver good experiences, by which I mean virtual objects that your eyes and brain accept as real. By "real," I don't mean that you can't tell they're virtual by looking at them, but rather that your perception of them as part of the world as you move your eyes, head, and body is indistinguishable from your perception of real objects. [...] I can tell you from personal experience that more than 20 ms is too much for VR and especially AR, but research indicates that 15 ms might be the threshold, or even 7 ms. (Abrash 2012).

The blog post by Michael Abrash quoted above was written at the time (December 2012) when the Oculus Rift was first announced. The article received a lot of attention and inspired several extensive comments and discussions. Among others, John Carmack (co-founder of id Software, in a leading position at Oculus VR since 2013) reacted and discussed in a blog post of his own (Carmack 2013) problems and possible solutions in the areas of *rendering* and *displays*.

That such low latencies are called for may be surprising at first. A latency of 20 ms corresponds to an update rate of 50 Hz. One often hears that a rate of only 24 Hz is needed to display moving images, and this is still the most common capturing rate in the movie industry today. Typically, however, images are projected in the movie theater at 48 Hz (i.e., each image is displayed twice). Actually, an update rate of as little as 14 Hz already suffices, for humans, for the illusion of continuous motion from individual images to appear. However, this does not mean that we cannot perceive or distinguish between images at higher frequencies. At this point, it becomes useful to differentiate between the *refresh rate* (even of the same images) and the *update frequency* or *frame rate* (different images). The critical refresh rate at which one can no longer perceive the individual images of an image sequence starts just below 50 Hz, but depends on external factors (Bauer et al. 2009). Only with a refresh rate above 100 Hz is an image considered to be truly flicker-free. With HMDs, the frame rate plays a greater role, since the pixels of LCDs, for example, do not need to be refreshed as frequently as is the case for projectors and CRTs. Here, it is more important that the latency of the screen is low, so that the content can be updated in the shortest possible time. Also important, although often overlooked, are issues with fluctuations in the frame rate. 100 Hz (i.e. frames per second) are of little use if 99 of the frames are rendered and displayed within the first 5 ms and the last frame is only displayed after another 995 ms.

Besides the effects of latencies that can be perceived consciously, unconscious effects also play a role. In a simulation, different latencies can arise in different presentation channels, e.g., visual, auditory and haptic. The presentation can then become asynchronous. Such incongruencies can, however, be perceived by humans and may lead to discomfort. The *vestibulo-ocular reflex* ensures, for example, that the eyes are automatically moved to counter a head movement (intentionally or unintentionally) while looking at objects to enable continuous perception. If the image generation in a head-mounted display has too much latency, the learned motion reflex of the eyes no longer fits and a refixation must be performed. This effect occurs similarly under the influence of alcohol or narcotics. In some people, it is precisely this incongruity that causes discomfort or even nausea.

7.1.2 Where Do Latencies Actually Arise?

Figure 7.1 shows the structure of a typical VR system. Various input sensors, shown on the upper left of the figure, capture the user's behavior. *Tracking latency* occurs between the time of the user's movement and the availability of the movement data as an event for the world simulation. The transport medium exhibits another latency to be considered separately, the *transport latency*. An important task of sensor fusion is to make provisions regarding the latency differences between multiple

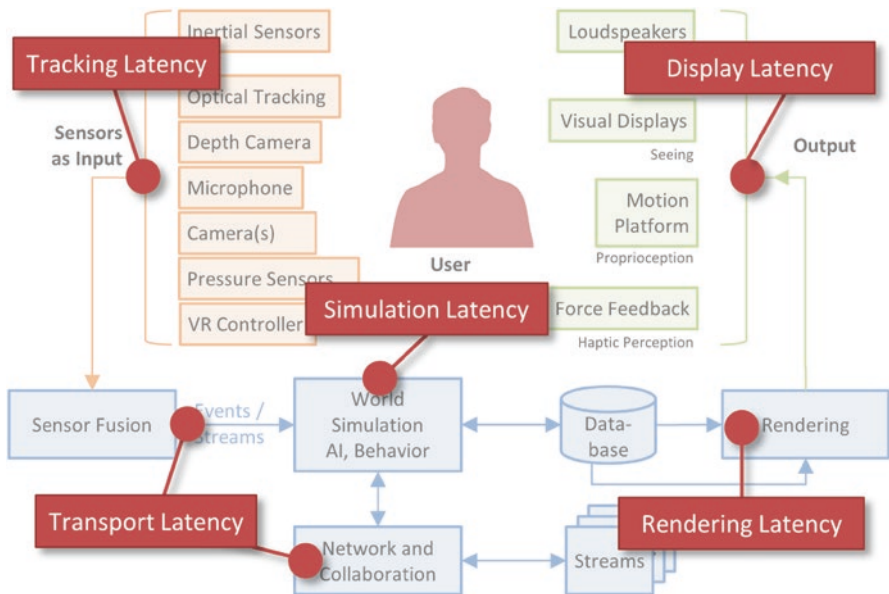


Fig. 7.1 Latencies occur at various points in a VR system

tracking systems. Often the weakest link, i.e., the slowest tracker, determines the overall latency of tracking as a whole.

In the world simulation, incoming tracking events are processed to simulate the effects of user interactions. The *simulation latencies* that occur here result from the necessary calculations and possible waiting times, e.g., for incoming tracking data. Simulation latencies may vary widely depending on the application.

After a new world state has been calculated by the simulation, the new state must be rendered into a format suitable for the respective output device. Rendering occurs not only for visual but also for other kinds of displays, such as auditory and haptic displays. The time necessary for rendering induces the *rendering latency*. Finally, the rendered data is displayed on the output devices, which also does not happen instantly and thus induces a *display latency*.

The total latency of the system is also known as *end-to-end latency* or, when focusing on visual displays, *motion-to-photon latency*. A similar categorization of latencies is proposed by Mine (1993).

When the virtual world, which has changed due to interaction, is (finally) presented to the user, a certain amount of time has already passed and the presentation is therefore already outdated. Depending on the frame rate of the system, it will now take a further amount of time until the currently presented content is overwritten with new content (*frame-rate induced delay*).

To assess the total duration of an interaction, it may be appropriate to also measure the reaction time of users, i.e., the time users need to recognize a newly presented stimulus, plan their reaction to it and, for example, respond to it with body movements. Here major fluctuations of latencies between users (e.g., age) but also within one and the same user (e.g., fatigue) may occur. Of course, the reaction time of a user is also a relevant factor for interactions in the real world. However, the following explanations refer exclusively to technology-induced latencies of VR systems.

7.1.3 Is Latency in a VR System Constant?

The combined latency of the entire VR system depends on, among other things, the update rates of the involved processes. For example, if a tracking system has a sampling rate of 60 Hz, the individual recording times are 16.7 ms apart. On average, a latency of 8.35 ms is already generated, as physical events (e.g., movements) that occur up to 16.7 ms later are not detected or passed on until the tracking system detects them. The same applies to the frame rate. If a projector is able to update the image at 100 Hz, a change that was not fully rendered until shortly after the last update will be displayed up to 10 ms later (on average 5 ms).

In a complex VR system with many concurrent subprocesses, update rates may vary a lot between its individual components. Therefore, the latencies of the overall system can be subject to significant fluctuations. Thus, in addition to minimizing the latency of the individual subsystems or the overall system, there is also the goal of

ensuring that latency is as constant as possible. Strong fluctuations in the overall latency can easily be perceived by users as jerking and can have a more disturbing effect than an overall higher but constant latency.

7.1.4 What Are the Approaches to Determining Latency?

Various approaches to *latency determination* are presented below. First, it is discussed to what extent the latency of a system can be *estimated* from datasheets of the individual components. This approach is primarily helpful in the planning phase of VR systems, but it can also give hints for potential optimizations later on. Then, various methods are presented with which the latency of a running system can be systematically *measured*.

Latency Estimation from Datasheets

To measure latencies, the VR system must already be operational and all relevant components accessible. However, this cannot be achieved in the planning phase of new installations. In this phase, the system designer must therefore rely on the information provided by manufacturers, on data from comparable systems and on expert experiences.

Table 7.1 shows examples of the *tracking latencies* for different types of tracking systems. The listed examples are based on real system data and are exemplary for commercially available systems. The data in the table are based either on statements

Table 7.1 Overview of frame rates and latencies of various existing tracking systems, the manufacturers were anonymized

Type	Frame rate	Latency
Optical Tracking Systems		
Example System A	30 Hz	90 ms–300 ms
Example System B	60 Hz	15 ms–20 ms
Example System C	Up to 10,000 Hz with reduced field-of-view	4.2 ms
Example System D	30–2000 Hz, depending on spatial resolution	> 2.5 ms
Electromagnetic Tracking Systems		
Example System E, wireless	120 Hz	< 10 ms
Example System F, wired	240 Hz	3.5 ms
Inertial Tracking Systems		
Example System G	60–120 Hz	10 ms with USB ^a
Hybrid Tracking Systems		
Example System H	180 Hz	1–2 ms RS-232; 5–8 ms USB

^aSkogstad et al. (2011) report a latency difference of 15 ms between the fast USB connection and the slower but mobile Bluetooth connection.

by professional users or on information provided by the manufacturers on websites or in product brochures. A similar, somewhat older, list can be found in (Ellis 1994). The concrete values are mainly to be understood as rough reference points, since there is no exact specification of how the measurement process should be designed and, for example, how many objects were measured simultaneously to collect the values.

Transport latencies occur during network communication between input devices, computers with VR software and output devices. In collaborative or multi-player applications, further, hardly predictable transport latencies occur during communication with remote computers. With wireless transmission technologies such as Bluetooth and WLAN, which are often used for communication between input devices and control computers, transport latencies of >1 ms occur for individual messages. With wired transmission, e.g., via Ethernet or InfiniBand, the transport latencies are generally lower, for example in the range of 0.001 ms to 0.03 ms. The actual transport latencies depend on the data volume to be transmitted: a network level event is a single data packet sent from A to B. In the best case, for example, a message describing a 6 DOF movement event fits into a single such data packet. Generally, however, this is not the case because some tracking systems send much larger amounts of data per time step, for example, 3D point clouds. For calculating the transmission time for all data, the number of packets that are sent over the network would then have to be known. Depending on the network topology, a parallel transport may be possible but also collision with other data services, e.g., file server accesses (best to use different network channels here). The actual latency at the network level is therefore difficult to estimate. For example, in scientific visualization very large amounts of data have to be moved. Here VR systems should be designed whose network components feature transmission rates in the multi-digit gigabit range, which then usually also offer very good latency characteristics.

Simulation latencies and tolerable threshold values strongly depend on the respective application and are therefore excluded from this analysis.

Rendering latencies are closely related to the complexity of the scene to be rendered (visually, acoustically, haptically). If the time needed for rendering dominates the overall latency of the VR system, *multi-GPU systems* may be considered. Hardware approaches for multi-GPU rendering include Nvidia SLI and AMD Crossfire, but software solutions also exist. For an overview of multi-GPU rendering see Dong and Peng (2019). For *stereoscopic rendering*, two images must be calculated per time step. If the images for the left and right eyes are computed one after the other, i.e., in two independent passes, the rendering frame rate is effectively halved compared with *monoscopic rendering* (as a counter measure one may need to halve the geometric complexity of the scene). A rendering technique known as *single pass stereo* reduces the computational effort for stereoscopic image generation (Hübner et al. 2007). This method takes advantage of the fact that the positions of the left and right eyes are close together and therefore see largely identical sections of the virtual world. By parallel geometry processing during rendering for the left and right eyes, scenes can be rendered almost as fast as in the monoscopic case. *Single pass stereo* can also be extended to more than two displays (*single pass*

multi-view rendering or *multi-view rendering* for short), e.g., to support tiled displays with multiple projectors (see Sect. 5.4.3) or multi-display HMDs (see Sect. 5.3.4). Another optimization possibility arises when VR or AR is used in combination with *eye tracking*: *foveated rendering* draws high-resolution images only in regions that the user is looking at. Other regions can be displayed in low resolution, as there is no detailed visual perception possible anyway (see Sect. 2.2). Section 7.3 discusses further methods for *real-time rendering* in more detail.

At the end of the 1990s, when CRT screens were still standard, *display latency* was unproblematic, at least on the desktop, as refresh rates of up to 200 Hz were achievable. This also made it possible to display content in stereo on the screens using *shutter technology*. However, the success of flat screens has largely pushed CRT screens out of the market – unfortunately without initially being able to offer similarly high refresh rates. In the meantime, however, flat screens have reached a comparable level of performance in terms of refresh rates, with current models exceeding 200 Hz. However, stereo solutions based on shutter technology are not offered broadly on the consumer market for desktop systems. In addition to the worse refresh rate, some LCD screens also have an input delay, which can sometimes be reduced by turning on a special low-latency gaming mode.

A precise determination of the latency can ultimately only be made on the real system. Therefore, in the following, different approaches are presented for how latency can be determined by experimental measurement.

Measuring the Latency of Tracking Systems

Most VR systems include some type of spatial tracking system. Viewer-dependent rendering, for example, relies on head tracking and many spatial interactions are based on 3D tracked controllers. *Tracking latency* is the time needed by the tracking system to detect and report the position and/or orientation of the tracked user or devices.

A very simple way to *measure latency* exists for the widely used *marker-based optical tracking* systems. These markers are usually attached to the user's body or an interaction device and either reflect or actively emit infrared light (see Sect. 4.3.1). The tracking latency can be easily determined with a setup where an infrared LED is placed in the tracking area. A computer that is connected to the tracking system controls the LED. The time difference between a strobe pulse of the LED and the reception of a corresponding event by the tracking system is the tracking latency.

While this method is very easy to implement, it also has a disadvantage: a robust tracking system may include filter mechanisms to eliminate short-term disturbances, e.g., due to reflections from clothing or jewelry. If these filters cannot be switched off in the system to be measured, the measured latency will be higher than later in the running system, where reflective markers usually move continuously and thus more predictably. A reasonable extension is therefore the use of an LED array,

where the LEDs can be controlled individually and thus any movement pattern can be simulated.

Instead of simulated movements, real movements can of course also be used for latency measurement. Periodically oscillating physical systems, such as pendulum systems, have proven to be particularly suitable (see Liang et al. 1991; Mine 1993). The basic setup could look like Fig. 7.2, where two pendulums are installed centrally in the tracked area. One pendulum serves as a reference for the direction of gravity. A tracking marker is attached to the second pendulum. This pendulum is made to swing during the measurement.

The measured position data of the marker and the current time stamp are displayed on a separate monitor (the monitor should feature a low display latency). The whole installation is recorded by a camera, which is positioned in such a way that the two pendulums are aligned in rest position (one occludes the other) and the monitor is also in view.

If one now starts the video recording and sets the pendulum in oscillation, it is later easy to navigate to the video frames where either the displayed y -position (y -axis in the direction opposite to gravity) is at a minimum or the two pendulums are fully aligned. The time difference between pendulum alignment and the subsequent minimum of the y -position is the latency of the tracking system. As an alternative to a purely visual comparison, and provided that a temporal synchronization between video camera and tracking system has been established beforehand, one

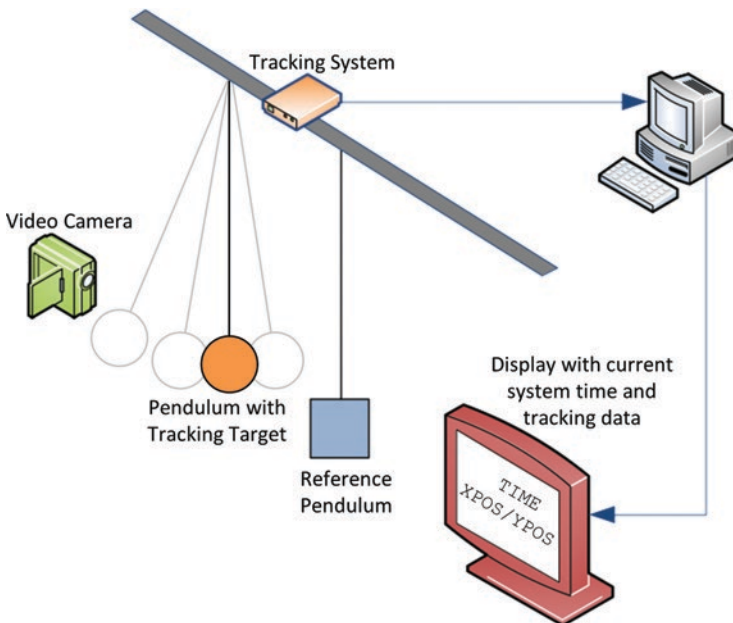


Fig. 7.2 Typical setup of a pendulum system for measuring the latency of an optical tracking system

may also analyze the recorded (and time-stamped) tracking data directly instead of their display on the monitor. This is advisable, for example, if the video camera has a significantly slower capturing rate than the tracking system.

With this setup, it must be considered that latencies for camera recording and displaying the time stamp and tracking data on the monitor may influence the result.

If one has more technology available, such as a precisely controllable robot arm, the measurement can also be carried out in an automated closed-loop setup where visual inspection of video recordings by a human is no longer necessary and thus larger quantities of data can be generated and analyzed. The idea is to attach a tracking marker to the robot's end effector. Tracking data then can be compared easily to the positions calculated from the robot's joint angle data (the robot in this sense acts a *mechanical tracking system* with close to zero latency). For example, Adelstein et al. (1996) used a motorized swing arm – a simple robot arm with one degree of freedom – that swings back and forth in the horizontal plane to evaluate the latencies of different tracking systems. Modern industrial robot arms with six degrees of freedoms also offer high precision and the additional advantage that they can perform movements resembling those of human VR users. Further, such robots have also been used to evaluate the *inside-out tracking* capabilities of mobile XR devices, such as AR-enabled smartphones and certain HMDs. Inside-out tracking uses a combination of visual camera images and other internal sensors (particularly the *IMU – inertial measurement unit*) to track the movement of the device. Instead of attaching a marker to the end-effector, the XR device is attached to – or simply held by – the robot arm (Eger Passos and Jung 2020).

Measuring End-to-End Latency

Uniform and very well controllable periodic motions can also be produced with a record player (Swindells et al. 2000). The idea is similar to that of the pendulum (cf. preceding section). An infrared LED is placed on a physical turntable to generate a live animation of a virtual turntable. The virtual turntable is projected onto the physical turntable. From the angular differences between the real and virtual rotating turntables, the latency of the entire setup, i.e., the *end-to-end latency*, can then be determined.

He et al. (2000) pursue a similar idea with their approach to determining the end-to-end latency in CAVEs and similar projection-based setups. A tracked input device (they used a *wand*) is moved by hand back and forth directly in front of one of the CAVE's projection screens. The tracked position is displayed on the screen as a virtual cursor along with a grid. During controller movements, the virtual cursor may lag the physical controller by several grid cells, depending on the speed of movement. A video camera records the whole setup. During video analysis, the field differences between the physical input device and the virtual cursor are counted from which the end-to-end latency is determined by simple calculations.

This method can also be easily combined with a pendulum to eliminate the need for manual movement of the physical controller (or marker). It is also easier to

determine the speed of the pendulum. Steed (2008) describes two approaches for determining the latency between the real and virtual pendulum. In the one approach, he counts the number of video frames between the extreme positions of the real and virtual pendulums. In the other variant, he analyzes the trajectories of the two pendulums by means of image processing methods and tries to find the most accurate mathematical approximation of the respective oscillation. Once this has been done, the phase shift and thus the latency can be easily determined. Steed reports that he achieves greater accuracy with the analytical method than by counting video frames.

7.1.5 Summary of Latency

In VR systems, low latency is a decisive factor for the creation of believable experiences of virtual worlds. Low latency is especially important when HMDs are used, since the scene portion to be displayed depends on the current head orientation of the user. In projection-based VR systems, where the displayed scene portion does not depend on the head orientation, latency requirements are less strict but still high. AR applications have even higher latency requirements, as virtual objects need to be anchored in the real world and the real world always has zero latency.

If the latency of an optical tracking system, as often found in VR installations, is too high, a combination with a low-latency inertial tracking system may be advantageous (You and Neumann 2001). Between phases of stable position tracking by the optical system, the inertial system can provide the necessary data for extrapolation of the new positions and orientations until stable data from the optical tracking system are available again. In this way, gaps can be bridged, e.g., when optical markers are occluded from the tracking cameras' views.

In practical operation, *network management* in particular has a major influence on *transport latencies*. For example, the VR system should be operated in a separate subnet to avoid collisions with other applications. Frequency range and channel of wireless access points should be selected in such a way that, if possible, no interactions with other wireless networks in the environment occur.

7.2 Efficient Collision Detection in Virtual Worlds

Where one body is, there can be no other. This simple physical fact poses a serious problem for VR/AR systems and real-time computer graphics in general. Virtual objects may in principle be placed at arbitrary locations in the virtual world and therefore may also penetrate each other if no precautions are taken. In the case of statically arranged objects, the programmer, or designer, can take the necessary care to ensure that no penetrations are visible to the observer of the scene. For a realistic and immersive representation of dynamic content, however, it would be helpful if the objects in the scene showed (approximately) physically correct behavior. Objects

should therefore be able to collide with and exert forces on each other. In the case of simulating the physics of the real world, not only the mere question of whether a collision occurred or not is relevant. To simulate a suitable reaction to a collision event, further properties of the collision must often be determined such as penetration depth, exact penetration locations and exact collision time. In many gaming applications it often suffices that the simulation provides a plausible approximation of the real world. In contrast, e.g., CAD, virtual prototyping, scientific applications and robotics problems usually place higher demands on collision detection and handling. In these cases, aspects such as numerical stability and physical correctness are often more important than the real-time capability required by VR applications.

The need for efficient collision detection is, however, not limited to physics simulations in the virtual world, but also occurs in many other areas of VR and AR systems. Even seemingly simple user interaction tasks like the selection of a scene object (see also Sect. 6.3) lead to related problems: to detect which object the user is pointing at, a ray may be generated from the user's pointing device. The scene objects are then tested for collision with the pointing ray and the object with the shortest distance to the user is chosen as the selected one.

Modern 3D computer graphics scenes achieve remarkable visual quality. Which techniques are used to render these scenes? Part of the reason can be found in the high performance of modern GPUs. However, the high quality could not be achieved if the GPU had to process all objects of the virtual world for each image to be generated. If an object is not at least partially in the *view volume* (or in other words, if there is no overlap or collision between the object and the view volume), the object does not contribute to the result of the image generation and therefore does not need to be processed further. This process is also called *view volume culling* and is described in more detail in Sect. 7.3.1. Given the desired graphical complexity of modern applications, the removal of non-visible objects based on efficient collision testing makes an important contribution to maintaining real-time capability of rendering.

The above-mentioned application areas of collision detection essentially require that the necessary calculations can be performed “in real time”, i.e., once per image frame (at least 25 Hz, ideally 60 Hz). View volume culling inserts a new processing step into the rendering pipeline that requires additional computation time. To justify the use of this technique, this computation time must be less than the rendering of the entire scene would otherwise require.

Real-time requirements on collision detection may even be much higher for VR systems that make use of *haptic interfaces*: according to Weller (2012), refresh rates of 1,000 Hz are required to ensure realistic haptic feedback for the user. In this case, less than 1 ms is available for collision detection.

Efficient algorithms and data structure are key for all the above-mentioned use cases of collision detection to ensure the central real-time requirement of VR and AR systems.

Following this introduction, Sect. 7.2.1 introduces common bounding volumes used for efficient collision detection. Section 7.2.2 then deals with their arrangement in hierarchical or spatial structures before collision detection methods in large

virtual world are discussed in Sect. 7.2.3. Then, in Sect. 7.2.4, the collision detection techniques are summarized and advanced topics in are addressed.

For more in-depth reading on the topic, we recommend the books by Akenine-Möller et al. (2018), Lengyel (2002) and Ericson (2005).

7.2.1 Bounding Volumes

Scene objects are constructed from primitives, typically in the form of triangle or polygon meshes. In a naive collision test between two polygon meshes, each polygon of the first mesh would have to be tested against each polygon of the second mesh. For example, if the two meshes consist of 500 and 1,000 polygons each, $500 \times 1,000 = 500,000$ tests would have to be performed between pairs of polygons. Considering that virtual worlds can consist not only of two objects but perhaps thousands of objects, it becomes clear that such a naive collision test is not practical for large virtual worlds.

Bounding volumes (BV) approximate the shape of the actual scene objects to facilitate efficient collision testing. Bounding volumes are stored in addition to the visible object geometry but are not rendered in the visual image. The additional storage requirements of bounding volumes, however, are usually justified by the reduced computational effort for collision testing. When scene objects are moved or otherwise transformed (e.g., translation, rotation, scaling), their bounding volumes must be updated too. The computational costs for such updates must also be considered when choosing appropriate bounding volumes. Generally, it is often desirable for a bounding volume to tightly fit a scene object such that the number of falsely reported collisions is minimized.

For some applications, e.g., gaming, bounding volumes may already provide for sufficiently precise collision testing. This is especially the case when the bounding volumes closely approximate the scene objects' shapes.

Even if approximated collision testing based on bounding volumes alone is not sufficient for the demands of the application (e.g., CAD, virtual prototyping, haptics, robotics), bounding volumes can still be used advantageously. In most virtual worlds, only a few objects will actually collide at a given time. Fast approximate collision tests based on bounding volumes can be used to determine that collisions between two objects do not occur. Only in cases where the approximate bounding volume-based test reports a collision is exact collision testing on the polygon meshes necessary.

Furthermore, hierarchal data structures may be used to quickly exclude large groups of scene objects from further collision testing. Examples of such data structures are *Bounding Volume Hierarchies (BVH)* and *Binary Space Partitioning (BSP)* discussed in Sect. 7.2.2.

Summarizing the above, desirable properties of bounding volumes include:

- simple and fast collision testing

- tight fit/good approximation of the detail geometry (otherwise false positives are possible)
- easy update in case of dynamic objects
- memory efficiency

These properties are partly contradictory. For example, two spheres are easy to test for collision and the memory requirement is minimal (position and radius). However, if you look at the fit, it is easy to see that not every object can be approximated as a sphere in a meaningful way.

The following typical bounding volume and their most important properties are discussed in the next sections:

- *Axis-Aligned Bounding Box (AABB)*
- *Bounding sphere*
- *Oriented Bounding Box (OBB)*
- *(k-dimensional) discrete oriented polytope (k-DOP)*

The text mostly discusses these bounding volumes for the two-dimensional case. This can easily be extended to three dimensions.

Axis-Aligned Bounding Box (AABB)

An AABB is a rectangle or cuboid whose edges are parallel to the axes of the global coordinate system and which encloses a given object with a minimum area. For three or more dimensions, this body is also called an axis-parallel (hyper-) cube. The orientation of the AABB is independent of the enclosed object and always the same (i.e., aligned to the global coordinate system). When the enclosed object changes its position, the new position must be applied to the AABB too. When the enclosed object is rotated or scaled, it is also necessary to update the shape of the AABB.

Memory space is required for four values in the two-dimensional case:

- positions (x,y) of two opposite corners; or
- position (x,y) of one corner + width and height; or
- center point + (half) width and height

To test two AABBs for collision, the boxes are projected onto the axes of the global coordinate system. For each axis, the projection intervals are tested for overlap separately. A collision occurs only if projections overlap on all axes. Conversely, the collision test can be aborted if a non-overlapping axis is found. Fig. 7.3 shows different configurations for AABBs and illustrates the collision test between two AABBs.

An AABB can be constructed in different ways. A simple approach is to determine the minima and maxima of all corner point coordinates along each axis. However, if the AABB needs to be updated frequently due to rotations of the enclosed object, this simple approach is inefficient for large meshes. In principle,

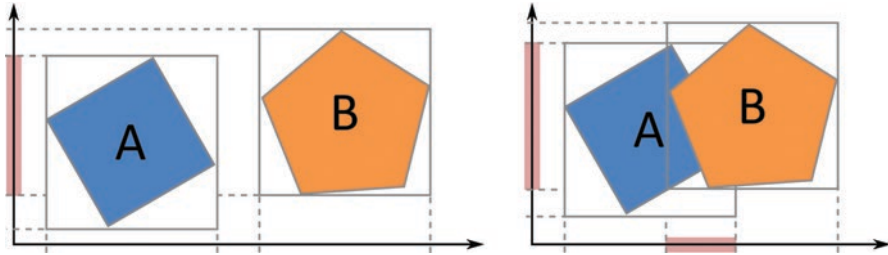


Fig. 7.3 Collision testing with AABBs. Left: 2D objects A and B with overlap on one axis only (no collision). Right: A & B with overlap on both axes (collision)

only the vertices of the mesh that form its convex hull need to be considered for the construction of the AABB. This fact can be exploited, for example, by calculating the vertices of the convex hull once and saving them. To update the AABB it is then sufficient to consider the convex hull only. For further details see Ericson (2005).

Bounding Spheres

Bounding spheres are very simple, easy-to-implement types of bounding volumes. They can be stored very efficiently (center point and radius) and collision testing can be carried out in a few steps: if the distance between the two center points is less than the sum of the two radii, then the two spheres collide. Otherwise, there is no collision.

A bounding sphere can be constructed by constructing an AABB first. The center of the AABB equals the center of the sphere and the distance to one of its corners gives the radius of the sphere. Alternatively, the sphere's center can be calculated by averaging of all vertex positions of the enclosed object's mesh. However, this approach does not necessarily result in a minimal envelope for any polygon mesh. In the worst case, the resulting bounding sphere could have twice the radius of a minimal variant and would therefore not be an optimal fit. The determination of a minimal bounding sphere from a point set has been the subject of various research. Welzl (1991) presents an algorithm for determining minimal circles and spheres from point clouds.

Due to the rotational symmetry of spheres, rotations of the enclosed object do not have to be transferred to the bounding sphere. Scaling and translations can be applied directly to the bounding sphere.

Oriented Bounding Boxes (OBBs)

OBBs can be seen as an extension of AABBs. However, the edges of the bounding cuboid, or in the 2D case bounding rectangle, are not aligned to the global coordinate system but oriented in such a way that the object is minimally enclosed. In

contrast to AABBs, the orientation of an OBB must therefore be saved explicitly. In the 2D case, this can be done using one of the following variants:

- positions of three corners (the fourth corner can be calculated from the three others)
- one corner + two orthogonal vectors
- center point + two orthogonal vectors
- center point + rotation (e.g., as rotation matrix, Euler angles or quaternion) + (half) edge lengths

These variants differ not only in terms of memory requirements but also in the amount of work required for collision testing. To save memory space in the two variants involving two orthogonal vectors, one of the vectors may be determined at runtime (using the cross product, see Chap. 11). However, in this case it is still necessary to store the length of the vector explicitly.

Collision testing for OBBs can be performed based on the *Separating Axis Theorem (SAT)*. This theorem states that two convex sets have no intersection exactly when a straight line/plane can be placed between them in such a way that one set lies in the positive half space and the other in the negative half space. The orthogonal projection of both sets onto an axis parallel to the normal of this line/plane is then called the *separating axis*, because the projections onto this axis do not overlap (see Fig. 7.4). If a single separating axis can be found, a collision of the two sets can be excluded.

To apply the theorem in practice, it is obviously necessary to clarify how a separating axis can be found. For three-dimensional OBBs it can be shown that 15 candidate axes have to be tested:

- The six axes orthogonal to the side faces of the OBBs (see Fig. 7.4, axes of the coordinate systems of the OBBs).
- The nine axes created by the cross product of one of the coordinate axes of each of the two OBBs.

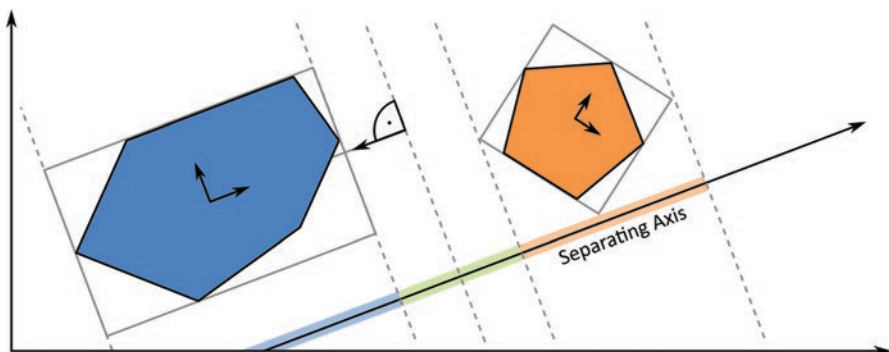


Fig. 7.4 Collision test of two OBBs and a separating axis

Similarly complex as the intersection test calculations is the generation of OBBs with a good fit. Exact algorithms for generating a minimal OBB typically belong to complexity class $O(n^3)$ and are therefore hardly applicable in practice. For this reason, algorithms are often used that only provide an approximation of the minimal OBB but can be calculated easily and at runtime. In Ericson (2005) different approaches to the solution are discussed. The update costs for OBBs are lower as compared to AABBs (and k -DOPS), as in addition to translations and scaling, rotations can also be applied directly to OBBs.

Discrete-Oriented Polytopes (k -DOPs)

Discrete-oriented polytopes (k -DOPs) or *fixed-directions hulls (FDH)* are a generalization of AABBs, as they are also always aligned to the global coordinate system. The term *polytope* refers to a polygon in the 2D case and, respectively, a polyhedron in the 3D case. A k -DOP is constructed from k half-spaces whose normals each take one of k discrete orientations. Opposite half-spaces are antiparallel, i.e., their normals point in opposite directions. The normals are usually formed from the value range $M = \{-1, 0, 1\}$. Since only the direction of the normals but not their length is relevant for further calculations, the normals do not have to be in normalized form (unit vector).

For the two-dimensional case, a 4-DOP (6-DOP for 3D) corresponds to an *AABB*, where the normals are parallel to the axes of the coordinate system. Different two-dimensional k -DOPs are shown in Fig. 7.5.

As the normals are identical for all k -DOPs of different objects, the memory requirements per object are reduced to the extension along each normal. For an 8-DOP, for example, eight values must be stored.

Collision tests between two k -DOPs are again performed based on the separating axis theorem. Since the normals are known and are the same for all objects, the great advantage of k -DOPs over OBBs is that only $k/2$ candidate axes must be considered as separating axis (opposite normals are antiparallel and thus yield the same axis). Accordingly, a maximum of four potentially separating axes must be tested for an 8-DOP. Collision tests can therefore be performed very quickly and easily.

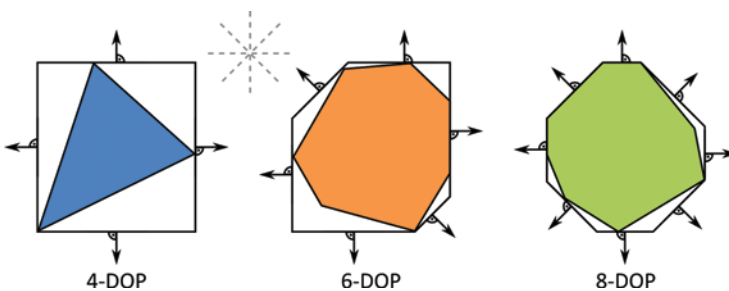


Fig. 7.5 Two-dimensional k -DOPs in different variants

The construction of a k -DOP is similar to that of an AABB: along each of the $k/2$ axes, minimal and maximal extensions of the object must be found. Although in principle any axis (or orientation) could be used, in practice the normals/orientations are usually chosen from the discrete number of values mentioned above. For collision testing it is only important that the same orientations of the half spaces must be chosen for all objects.

A disadvantage of k -DOPs is caused by the time-consuming updates that become necessary when the enclosed polygon mesh is rotated (translations can be transferred directly to the k -DOP), as the minima and maxima along the $k/2$ axes must be recalculated. To avoid cost-intensive iterations over all vertices of the enclosed polygon mesh (or its convex hull), additional optimizations are often applied at this point (e.g., hill climbing and caching; see Ericson (2005)).

Summarizing, k -DOPs offer efficient collision testing and low memory requirements without sacrificing a good fit. However, the high update costs imply that k -DOPs are often only of limited use for dynamic objects.

7.2.2 Bounding Volume Hierarchies and Space Partitioning Techniques

Although the creation of bounding volumes will simplify and accelerate collision testing between two objects, the total number of collision tests required (object against object) remains unchanged. For a scene consisting of n objects still $n(n-1)/2 \in O(n^2)$ collision tests must be performed in the worst case. To reduce the number of tests, several methods may be applied as discussed in the following.

Bounding Volume Hierarchies (BVHs)

Bounding volume hierarchies (BVHs) are created by arranging bounding volumes in trees. The hierarchies are created by calculating new bounding volumes for several geometric objects (or their bounding volumes). These new bounding volumes can in turn be combined with neighboring objects (or their bounding volumes). The parent nodes do not necessarily have to completely surround the hulls of the child nodes. It is sufficient that the geometric objects at the leaf nodes are completely enclosed. However, the construction of BVHs is often easier in practice if the bounding volumes are used for this process at each level of the tree. The granularity or depth of the tree is application-specific and can in principle be managed to such an extent that individual polygons and their bounding volumes are stored at the leaf nodes.

Examples of BVHs are *AABB trees*, *OBB trees* and *sphere trees*. An example of a sphere tree is shown in Fig. 7.6. The runtime gain of BVHs is due to the fact that the tree is tested against other objects, starting from the root. As an illustrative example, imagine a complex vehicle simulator that can display high-resolution

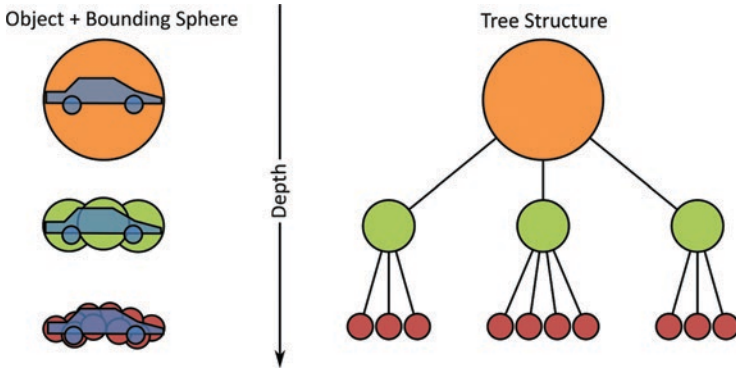


Fig. 7.6 Sphere tree for a complex object. Left: Geometric data and corresponding bounding spheres. Right: Hierarchy of bounding spheres (sphere tree)

models with several million polygons. The user points at the scene and the system has to quickly determine which component of the vehicle intersects with the pointing ray. To do this, the root node of a sphere tree could be placed around the entire vehicle (the user may miss the vehicle while pointing). If the vehicle was hit, bounding spheres of large components such as side/doors, rear/boot, front/engine compartment and tires may be tested on the second level of the tree. On the third level, individual parts of the respective branch could then be tested (e.g., for front/engine compartment: lights, air filter, battery, etc.). On each level, the collision test must be carried out only against a small number of bounding spheres, whereby the set of enclosed polygons becomes smaller and smaller. If no collision has been detected on one level (i.e., in all branches), the test can be aborted without testing lower levels. If necessary for the application, the remaining part of the polygon mesh (i.e., enclosed polygons of leaf BVs) can be used as a last step for exact collision determination.

BVHs require extra memory space whose size depends on the depth of the tree and the type of bounding volume. For static objects, BVHs can be calculated once at the beginning of the simulation. If dynamic objects come into play, updating the tree can become a problem. In these cases, it is advisable to manage dynamic and static components separately to avoid the need for updating where possible.

Space Partitioning Techniques

Space partitioning aims to minimize the number of collision tests required by assigning scene objects to spatial regions. With well-chosen partitioning strategies, collision testing can be reduced to objects known to be in the same or a close spatial region.

World space can be divided in different ways. Quite common are *regular grids*, as they are easy to implement and grid cells can be addressed with simple modulo operations. Space partitioning into a regular grid is also called *spatial hashing*.

The choice of a good spatial resolution depends strongly on the application. Fig. 7.7 depicts three cases for different cell sizes. If the cell size is chosen too small, objects must be assigned to multiple cells. This case results in high update costs when the object is moved. In contrast, if the cell size is too large, many objects will be assigned to the same cell, which is the very situation that the space partitioning actually tried to avoid. In the ideal case, each object can be assigned to exactly one cell. The cell size should be chosen in such a way that there are always only small numbers of objects in a cell. Nevertheless, it should be noted that multiple assignments (at most four cells per object in 2D) cannot be avoided, even with favorable cell sizes.

The practical applicability of spatial hashing therefore depends strongly on the cell size and the memory space required for the necessary data structures. The method is less suitable for scenes with objects of very different sizes or resizable objects. A positive feature of spatial hashing is that it can be implemented quite easily.

In addition to regular grids, space partitioning hierarchies or trees can be constructed. One method is the *binary space partitioning tree (BSP tree)*. Here, the space is recursively cut into two half spaces by a hyperplane at each recursion level. The two half spaces are also called positive and negative half spaces. When applied in two or three dimensions, the hyperplane is a straight line or, respectively, a plane. The space is usually recursively subdivided until only one primitive (triangle or polygon) can be assigned to a node. If a cutting plane intersects an individual polygon, the polygon must be split into fragments. Fig. 7.8 shows an example of how a space containing one polygon could be partitioned by a BSP tree. Each inner node of the tree defines a cutting plane that partitions the space associated with the node into two halves and, thus, also the set of vertices enclosed by the node. During the subdivision process, new vertices/polygons may also be created. In Fig. 7.8, for example, the orange vertices are newly created during the subdivision. The original polygon in Fig. 7.8 could be further divided by additional half spaces. However, this has been omitted in favor of better readability. It should also be mentioned that other partitions are possible and could be considered for optimization of collision testing.

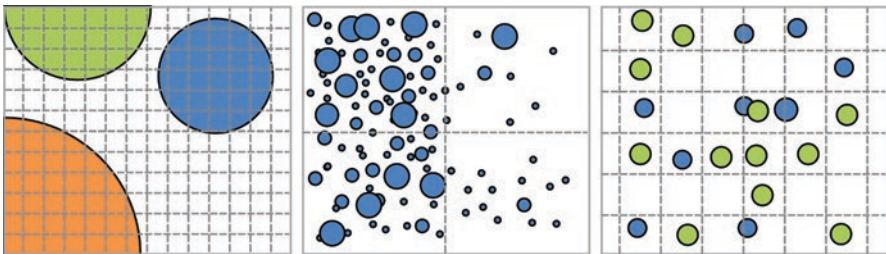


Fig. 7.7 Regular grids with different cell sizes. From left to right: grid too fine, grid too coarse, good grid size for the given objects

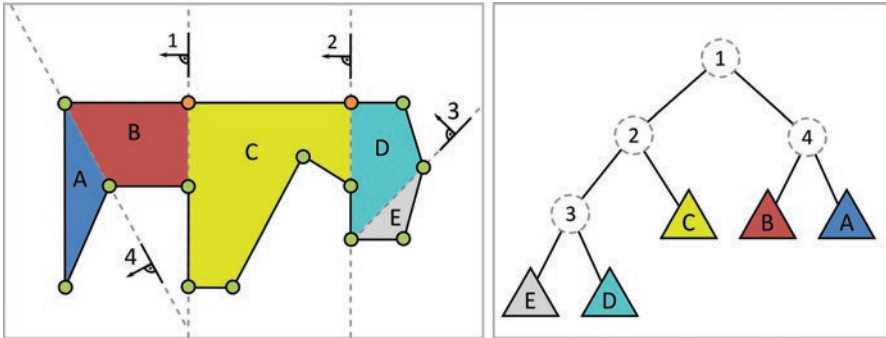


Fig. 7.8 BSP tree. Left: Binary space partitioning of a space containing one complex polygon (green: vertices of the original polygon, orange: newly created vertices during decomposition). Right: Binary tree defining half-spaces 1, 2, 3, 4 with fragments A, B, C, D, E of the original polygon

The positions of the hyperplanes and the depth (granularity) of the tree can be freely chosen in the case of general BSPs. If all cutting planes are chosen to coincide with one side of the object (edge of the polygon), the tree is also called *autopartitioning*, since there is no explicit calculation of the cutting planes.

Depending on the intended use, different forms of the tree are conceivable. For example, individual polygons or larger groups of polygons may be stored in the leaf nodes. Also, geometry data may be stored exclusively in the inner nodes of the tree (*node-storing BSP trees*). However, *leaf-storing BSP trees* are more relevant for collision testing. As the name suggests, they store geometry data in the leaf nodes. The BSP tree shown in Fig. 7.8 is an example. This form of data storage leads to a tree structure in which the positional relationships of the geometry data are reflected in the arrangement of the tree nodes. This property is particularly useful for collision queries.

In general, the cutting planes should be chosen in such a way that the following requirements are fulfilled as well as possible:

- The result is a balanced tree (all branches have equal or similar depth; for leaf-storing BSP trees each leaf node contains a similar number of objects).
- The number of half planes that cut through individual polygons (thus creating new vertices and polygon fragments) is minimal.

BSP trees can be constructed in various ways. The determination of the cutting planes according to the above requirements is often a non-trivial problem. Although the autopartitioning variant is easy to implement, it does not necessarily yield optimal results. In addition to collision detection, BSP trees are also used to determine visibility, among other things (see Ericson (2005) for details).

BSP trees can be understood as a generalized form of a *k-d tree* (see Fig. 7.9). A *k-d tree* is also a binary tree that subdivides a space recursively. In the variant of a *k-d tree* presented in the following, the spatial subdivision is driven by the input data, a *k*-dimensional set of points. All inner nodes of the tree define a dividing

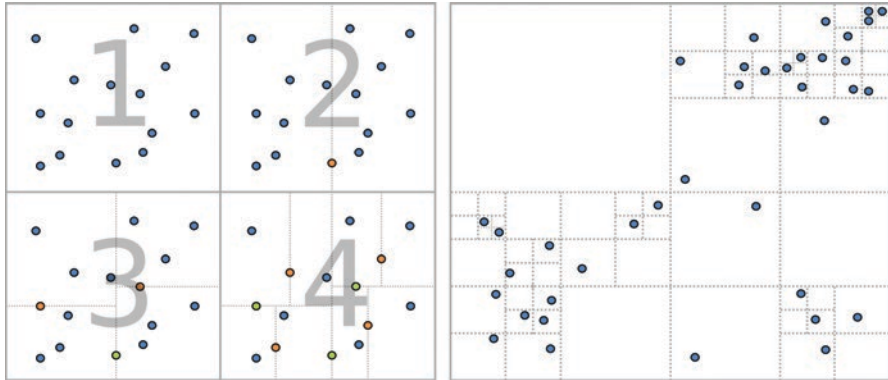


Fig. 7.9 k-d tree and quadtree. Left: The first four levels of a k-d tree. Right: Complete quadtree for a given point set

hyperplane (straight line for 2D case, plane for 3D case). Fig. 7.9 (left) illustrates the construction of a k-d tree: (1) A set of k -dimensional ($k=2$ in the example) points serves as input data. At each level of the tree one dimension – here: x or y – is selected for spatial partitioning. The cutting plane is perpendicular to the selected dimension. (2) An element of the input data, shown in orange in Fig. 7.9 (left), is now stored as the inner node of the tree and defines the position of this cutting plane by its coordinate value. (3) and (4) The newly created half spaces are subdivided further. At each tree level, a dimension different from the dimension in the level above is chosen – in the example, alternately x and y . To create a balanced tree, the position of the cut is chosen such that the same amount of data (approximately) remains in the positive and negative half spaces. Other k-d tree variants create the cutting planes explicitly and store data only in the leaf nodes.

When traversing a k-d tree from the root to a leaf node, only a single value needs to be compared at each level of the tree. For example, if a node of the tree defines a cutting plane orthogonal to the x -axis, then only the x -coordinate of the requested point needs to be compared with the value stored in the node. This process is therefore much easier to implement than for a BSP tree. Since the subdivision dimension can be anchored in the traversal algorithm, for example, $dimension = depth \bmod k$, it does not have to be stored explicitly.

Quadtrees (or *octrees* for 3D) use two (or three) axis aligned cutting planes per recursion level and thus create four (or eight) child nodes each. This decomposition is usually done in such a way that a given maximum number of objects is assigned to a quadrant. Fig. 7.9 (right) shows a two-dimensional quadtree for a given set of points.

The discussed variants of space partitioning trees differ in their memory requirements, their update costs and the computational effort for collision queries. In the case of BSP trees, for example, the position and orientation of the cutting planes must be stored, whereas for a k-d tree only a single value (position of the plane, orientation is implicit) must be stored. Similarly, for a query in the k-d tree, only a

single comparison has to be made at each tree level (is the queried coordinate in this dimension greater or less than the stored value?).

It is quite common that dynamic objects are not integrated into the space partitions discussed above, as the computational effort for updating them would be too large. Dynamic objects are usually managed separately.

7.2.3 Collision Detection in Large Environments

The collision detection methods presented so far may or may not be sufficient for a given application and use case. Whereas in a simple bowling simulation it might be possible to test polygon meshes directly against each other, a complex vehicle simulator likely requires both bounding volumes and space partitioning – and possibly additional methods – to ensure real-time capability. In large environments with very high numbers of objects, the task of collision detection is often split into two phases: a global *broad phase* and a local *narrow phase*.

Broad Phase Collision Detection

In a virtual world with thousands of objects, the vast majority of objects may collide with one or a small number of other objects but not with thousands. For any given pair of two objects, it is often easy to establish that they do not collide with each other, for example, because they are located far away from each other.

The goal of the *broad phase* is thus to quickly determine which objects certainly do not collide with each other. The result of the broad phase is a set of potentially colliding object pairs. As the tests are not exact, non-colliding object pairs can still be contained in the set.

Besides bounding volume hierarchies and space partitioning, depending on the granularity and size of the object set, the use of bounding volumes may also be considered a method of the broad phase. Only when the bounding volumes of two objects collide is it necessary to examine this object pair more closely in a detail phase. The classical algorithms of the broad phase, however, include *spatial hashing*, bounding volume hierarchies, and especially the *Sort & Sweep* (or *Sweep & Prune*) algorithm by David Baraff (1992). All techniques except for the latter have already been explained in the previous sections.

Sweep & Prune first projects the extents of the AABB of each scene object onto an axis, say the x -axis, of the global coordinate system. Since the axes for AABBs are aligned with the global coordinate system, this process is trivial. For each object i this yields an interval on this axis with the start value S_i and the end value E_i . The start and end values generated in this way are inserted into a list, which is then sorted by value (*Sort*). Two objects only form a potential collision pair if the projected intervals overlap. These collision candidates can be easily read out from the list by iterating over the list from left to right (*Sweep*). If a start value is encountered

during the sweep, object i is marked as “active”. The object becomes inactive when the end value E_i is encountered. If a second start value S_j is encountered while object i is active, the objects i and j form a potential collision pair. This procedure – project objects’ extents onto an axis, sort, sweep – is then repeated for the other axes of the global coordinate system. Only if the projections of objects i and j intersect on all axes will the algorithm report the two objects as potentially colliding. The result set of the algorithm is therefore a list of potentially colliding object pairs, which can be examined more closely in a subsequent detail step that uses more complex methods (exact polygon test or *GJK* for convex hulls; see the subsection below on the narrow phase). Fig. 7.10 shows a schematic diagram of the Sweep & Prune algorithm.

A key idea of Sweep & Prune is the exploitation of *temporal coherence*. Under the reasonable assumption that objects do not move erratically but will be roughly at the same position as in the previous time step, the sort orders from the previous time step can be reused. That is, after initial and one-time sorting for the first time-step, the lists are already presorted for the next time step. Certain sorting algorithms can update the list very efficiently when a presorted list is available as an extra input. *Insertion Sort*, for example, exhibits basically linear runtime behavior in these “best case” situations and is therefore particularly suitable.

However, it is precisely this temporal coherence that may also cause problems when scene objects form heaps. In these situations, small object movements can cause the list items of the intervals to be subject to major changes. As a result, sorting operations often have to be performed in full, and temporal coherence can hardly be exploited. In Fig. 7.10 this situation occurs on the y -axis.

Narrow Phase Collision Detection

After potential collision pairs have been found in the broad phase, the *narrow phase* performs exact collision tests on the objects’ detail geometry. Pairwise testing of all polygons of the two objects, however, has an algorithmic effort of $O(n^2)$ and would become inefficient for complex geometries. One possible measure is the insertion of an additional *middle phase* using bounding volume hierarchies, where parts of the

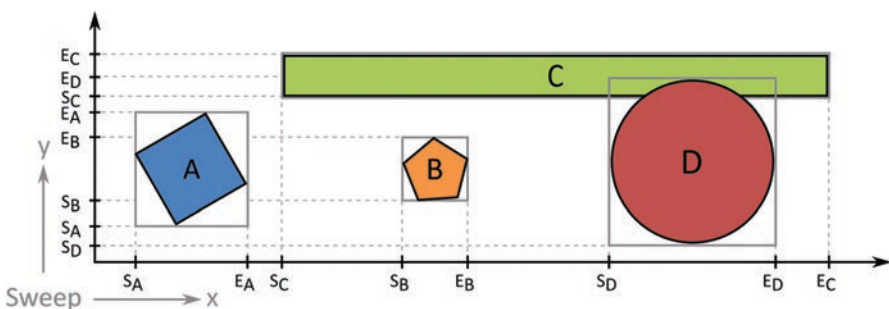


Fig. 7.10 Sweep & Prune: Objects A, B, C and D with AABB and projected intervals on the x - and y -axes

polygon meshes are approximated by bounding volumes. In this way, the set of polygons to be tested can be quickly limited to the relevant parts. However, depending on the type and objective of the application, other strategies may also be useful.

The near phase of collision detection can be broken down into subproblems:

- Removal of all false positives reported by the broad phase.
- Determination of the application-relevant collision parameters (e.g., contact points, penetration depth).

In practice, a third subproblem should be considered: objects may be in a state of permanent contact or collision. This state may occur, for example, when a thrown object comes to rest on the virtual floor. As long as no external forces other than gravity are applied, this state remains unchanged and, consequently, the two objects will be reported by the broad phase as a potential collision pair in all future time steps. Therefore, object pairs with similar contact information as in previous time steps should be marked as inactive, so that they are not examined by narrow phase collision detection over and over again.

A method often associated with the narrow phase is the *GJK algorithm*, named after its authors, Gilbert, Johnson and Keerthi (Gilbert et al. 1988). This algorithm determines the minimum distance between the convex hulls of two given point sets. If this distance is less than or equal to zero, the point sets collide with each other.

Minkowski Sum and Difference

The *Minkowski sum* is defined as: $A + B = \{\vec{a} + \vec{b} \mid \vec{a} \in A, \vec{b} \in B\}$,

the *Minkowski difference* is defined as: $A - B = \{\vec{a} - \vec{b} \mid \vec{a} \in A, \vec{b} \in B\}$,

where A and B are two subsets of a vector space.

The result of the Minkowski sum is thus a set which contains the sum of each element from A with each element from B . The result set does not contain any element twice. Under a graphical interpretation, the result is obtained by moving B along the border of A . In Fig. 7.11 a graphical interpretation of both the Minkowski sum and the Minkowski difference is given. The latter is often used in the field of collision detection, where one of the properties of the Minkowski difference turns out to be especially useful: the Minkowski difference contains the coordinate origin if and only if the intersection of the two sets is not empty.

The *GJK algorithm* exploits the useful property of a Minkowski difference (see Fig. 7.11), i.e., that it contains the coordinate origin exactly when the convex hulls of the objects overlap. In this way, the collision detection between two point sets of size n and resp. m (number of vertices in the convex hulls of the two polygon meshes) can be reduced to calculating the distance of a single point set (the Minkowski difference of size $n \times m$) to the coordinate origin. The explicit calculation of this large point set is avoided by iteratively checking whether the difference

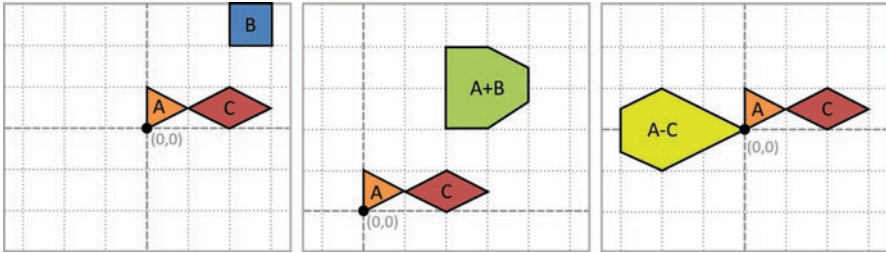


Fig. 7.11 Minkowski sum and difference: (from left to right) Objects A , B , C defined in a 2D coordinate system; Minkowski sum $A + B$; Minkowski difference $A - C$

can contain the coordinate origin. For this purpose, starting from any point of the difference a new point is searched for in each step, which is closer to the coordinate origin. If a point set containing the origin is found, a collision can be confirmed and the algorithm can be terminated. This method can further be used to determine the Euclidean distance of the convex hulls of the two polygon meshes as well as the points where the distance is minimal. This information can be used to determine contact points and collision depth. There are many publications around this algorithm in the scientific literature. Some address improvements of particular aspects of the original algorithm, e.g., hill-climbing for vertex search (Cameron 1997; Lin and Canny 1991) while others examine the case of moving objects (Xavier 1997).

Thus, the GJK algorithm cannot only be used to answer the question of whether a collision has occurred. It can also provide the contact parameters for generation of a suitable collision response. The method is very efficient and can be used for a wide range of object configurations.

The result of the narrow phase is a list that contains definitely colliding object pairs and associated contact information. These results can then be used to resolve the collisions. This process is called *collision response*. However, not every application area of collision detection requires the calculation of a collision response. For example, in the case of view volume culling, objects colliding with the view volume are displayed visually. All other objects are not rendered. Here, a spatial separation of the objects is not necessary. A physics simulation could, however, use the contact information to determine the forces necessary to separate the colliding objects.

7.2.4 Summary and Advanced Techniques

This section has examined basic procedures and strategies for collision detection between rigid bodies. Different types of bounding volumes were presented and their properties were discussed. Furthermore, it was shown how space partitioning and bounding volume hierarchies can be used to reduce the total number of collision tests required. In Sect. 7.2.3, the basic collision detection methods were put into the context of large environments with potentially thousands of objects.

The preceding subsections give an idea of how broadly the subject of collision detection can be approached. In the discussion it was always assumed that the simulation of the objects is carried out time step by time step (i.e., discretely). Although this process is easy to understand and implement, it involves some risks. If the movement of an object in one time step is larger than its extension, situations may arise where a “tunnel effect” occurs. As a practical example a soccer shot at the goal can be used: in the time step t the soccer ball is in front of the goal. However, due to the high speed of the ball, there is a high probability that the ball is already completely behind the goal in time step $t + 1$. The presented methods for collision detection do not report a collision for either time step. The ball has “tunneled” through the goal. To avoid this effect, various solutions may be pursued:

- Smaller time steps (= more computational effort at runtime).
- Determine the motion volume or motion vector and test for collision.
- *Continuous collision detection*.

The latter approach takes a completely different perspective on the problem: instead of examining objects present in each time step for collision testing, continuous collision detection calculates the exact place and time of a collision. An implementation of this technique can be found in the freely available 2D physics engine *box2d* (Catto 2020). Continuous approaches are also called *a priori* while discrete approaches are called *a posteriori*.

Modern applications and simulations increasingly require methods that can handle not only rigid bodies but also soft bodies such as clothes and fluids. These objects pose completely different challenges. For example, bounding volume hierarchies are rarely applicable for deformable objects, because costs for their initial creation and repeated updates at runtime would be too high. However, this problem can be addressed with the help of powerful, programmable GPUs. Research work on this topic has already existed for some time, for example (Sathe and Lake 2006). The Nvidia Flex simulation framework provides collision detection methods for soft bodies as well as support for popular game engines such as Unity and Unreal Engine.

7.3 Real-Time Rendering of Virtual Worlds

The visual sense is the most important one for human perception. Consequently, VR systems place particularly high demands on the *real-time rendering* of virtual worlds. In the literature it is generally assumed that the temporal resolution of our visual system is 60–90 Hz. A visual rendering system should therefore be able to provide at least 60 frames per second, so that the user is not able to perceive a sequence of individual images.

At present, typical display devices have resolutions of at least 1920×1080 pixels. If these are to be redrawn 60 times per second, almost 125 million pixels per second must be computed. This requires very powerful hardware to be able to

output the high-resolution content in real-time. The basic problem is to fill the pixel matrix in short time intervals. As this problem can be solved mostly independently for each pixel, special parallel computers are used for this task: the *graphics processing unit* (GPU). Today's GPUs often exceed the performance of CPUs many times over.

A naive program for the representation of virtual worlds could follow the following procedure:

1. Load the scene objects and build the virtual world.
2. As long as the program is not terminated:
 - (a) read the user input
 - (b) change the virtual world according to the user input
 - (c) pass the scene to the GPU
 - (d) draw the scene on the GPU

In this naive approach, for each image to be drawn, the entire content of the virtual world must be manipulated, transferred to the GPU and drawn. Despite the impressive computing capacity of today's graphics hardware, it is not capable of providing an appropriate amount of visual detail at sufficiently high frame rates with this approach. A part of the VR system's design should therefore include methods that support the rendering of visual images for high-resolution content, high-resolution displays, and in high temporal resolution, i.e., in real time.

General approaches for making the visual rendering as efficient as possible include:

- Draw only necessary, i.e., visible and perceptible, data.
- Use compact representations of the graphical data and avoid memory movement of the data whenever possible (time and energy costs).
- Use the available hardware as effectively as possible.

This section presents several methods for how these approaches can be implemented.

7.3.1 *Algorithmic Strategies*

Concerning the computational load of the graphics hardware, the best scene objects are those that do not need to be drawn at all. In the naive method above, all scene content is passed through the entire rendering pipeline, regardless of whether or not it can be seen by the viewer. For large virtual worlds with high-detail content, this is neither necessary nor efficient. At any time, large parts of the virtual world will be outside the user's field of view, occluded by other objects, or simply too far away to be seen in full detail. *Visibility testing* of objects and graphics primitives and the subsequent removal of invisible ones from the rendering pipeline is called *culling*.

View Volume Culling

During rendering, a *view volume* is specified for each eye which describes a mapping of 3D coordinates to 2D image coordinates. In the case of the common perspective projection this visual volume is called a *frustum* (see Fig. 7.12 left). The basic idea of *view volume culling* (or *view frustum culling*) is that only objects that are at least partially inside the view volume have to be drawn.

Different approaches and methods exist to determine which objects are in view and which are not. The graphics hardware provides support for this process at the level of graphics primitives (i.e., points, lines, triangles, polygons, ...) where it is called *clipping* (in addition to testing if a primitive is visible, partially visible primitives are cropped – or “clipped” – to the view volume). At this point, however, parts of the graphics pipeline, namely the vertex, tessellation and geometry shaders, have already been executed. Thus, it might seem like a good idea to perform the clipping on the CPU. However, graphics processors are able to draw polygons much faster than it takes the CPU to clip them, so no speed-ups are to be expected.

A more useful level of abstraction for performing many visibility tests is the object level. As the object level is coarser than the polygon level, some polygons will be sent to the graphics hardware that will not contribute to the resulting image. However, culling costs are usually amortized easily as large amounts of polygons must not be transferred to the GPU. An optimal balance between the computational costs for culling and the savings in terms of polygons not sent to the GPU depends on the scenario and the application. It is important, however, that visibility testing should always be designed conservatively: it should be guaranteed that objects marked as invisible are truly not visible. Otherwise, there is a risk of removing content that is relevant for the resulting image.

Section 7.2 has already introduced most of the tools needed to implement view volume culling efficiently, particularly bounding volumes and bounding volume hierarchies. Since the view volume is generally not a cuboid but a truncated pyramid (a frustum), special methods for efficient collision testing with common bounding

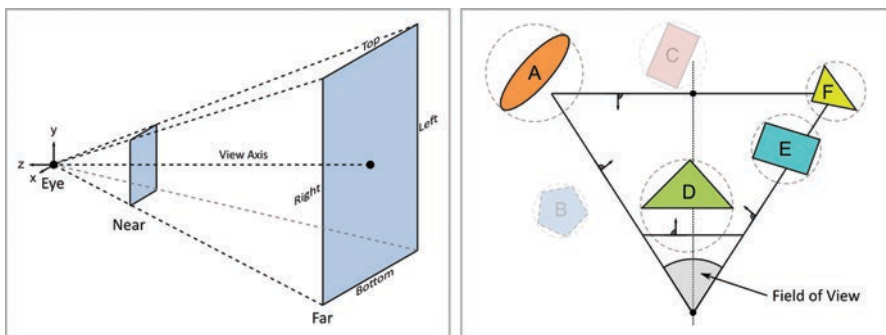


Fig. 7.12 View volume culling. Left: View frustum for perspective projection. Right: View volume culling with objects and bounding spheres (objects A, D, E and F are determined as visible)

volumes (spheres, boxes) are required. Gregory (2009) sketches a simple test for bounding spheres: for each bounding sphere of an object in the virtual world to be tested, each plane that defines the frustum is shifted outwards by the radius of the sphere (the normal directions for the frustum planes are indicated in Fig. 7.12 right). If the center of the bounding sphere is now in the positive half space for all six planes (or four planes in the 2D case), the bounding sphere is at least partially within the view volume. Fig. 7.12 (right) illustrates the process of view volume culling, where the scene objects are enclosed by bounding spheres. The approximation of objects with bounding volumes may yield results where an object is marked as visible while actually being outside the view volume. An example for this is object A in Fig. 7.12 (right).

For bounding volumes other than spheres the following method can be used for conservative view volume culling (Assarsson and Möller 2000): the six planes defining the frustum can be specified by a transformation matrix. This matrix is called a *projection matrix* and describes the mapping of the view frustum content onto a unit cube. The inverse matrix of the projection matrix is applied to the bounding volumes of the scene objects. For example, by applying the inverse projection matrix, a bounding box is “deformed” to the shape of a truncated pyramid (i.e., a frustum). For this “bounding frustum”, a new AABB (axis-aligned bounding box) is then constructed and used for intersection testing with the view volume (which is now a unit cube, after applying the projection matrix). In this way only AABBs have to be compared against each other.

Hierarchical View Volume Culling

Hierarchical view volume culling is an extension of view volume culling that takes bounding volume hierarchies (BVHs) into account. When a separate bounding volume is used for each scene object, view volume culling may make up a significant part of the available compute time for large scenes with thousands of objects. The hierarchy-building techniques presented in Sect. 7.2.2 can lead to significant improvements in such cases. For example, instead of a list of all scene objects, a tree can be constructed that structures the scene objects (or their bounding volumes) in bounding volumes of increasing size. This requires a suitable method for identifying suitable object groupings, and, in turn, groupings of groupings. Ultimately, the whole scene should be enclosed by a single bounding volume, i.e., the root of the BVH. In hierarchical view volume culling, the root node of the BVH is tested first. If it is not visible, no scene object is visible and the culling process finishes. Otherwise, deeper levels and branches of the tree can be tested recursively to determine the visible objects.

Other kinds of hierarchies, such as k-d trees and octrees, are also applicable and widely used for hierarchical view volume culling. Fig. 7.13 illustrates the hierarchical view volume culling method in 2D using a quadtree as example.

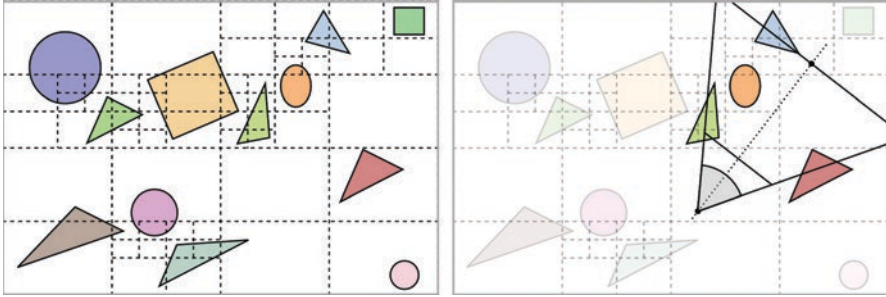


Fig. 7.13 Hierarchical view volume culling. Left: A scene and its quadtree. Right: Hierarchical view volume culling using the quadtree (highlighted objects are determined as visible)

Occlusion Culling

View volume culling provides a coarse test whether an object is potentially visible or not. However, just because an object (or its bounding volume) is within the view volume, this does not mean that it is actually visible in the rendered image: it may be *occluded* by other objects, such as walls, that are closer to the viewer. Filtering out objects that are within the view volume but hidden from the view by other objects is called *occlusion culling*.

Implementing occlusion culling in 3D object-space based on the objects' geometries could provide exact solutions, but is usually too costly. Instead one usually prefers an image-space solution that exploits a feature of modern GPUs: without special precautionary measures, the scene objects can be sent to the graphics hardware in arbitrary order where they are automatically drawn with correct occlusions. A simplified description of the standard rendering pipeline is:

1. Projection of the three-dimensional input data (primitives: triangles, quadrilaterals, etc.).
2. Rasterization of the primitive and generation of a fragment (fragment: data for one pixel, e.g., depth; also, but not used in simplified pipeline: interpolated color, normal, texture, etc.).
3. Fragment-based calculations and writing the pixel to the output buffer.

Without any further mechanism, this pipeline could lead to situations where scene objects that are close to the viewer are drawn early only to be overwritten, falsely, by other objects drawn later. To avoid this effect, the so-called *Z-buffer* (or *depth buffer*) of the GPU can be used. For each pixel, this buffer stores the z -coordinate of the last drawn fragment. If the fragment to be drawn next has a higher z -value, it lies "deeper" in the scene from the viewer and must not be transferred to the output buffer. Transparent objects must be handled separately and are usually sorted according to their depth before drawing. Other, more effective, techniques based on programmable GPUs are possible.

This Z -buffering can now also be used for occlusion culling. For this purpose, the scene is rendered once in a pre-processing step, whereby the computationally

expensive steps of the pipeline are deactivated beforehand (illumination, texturing, blurring, post-processing, etc.) and only the depth buffer is filled. The subsequent actual drawing process does not manipulate the Z-buffer, but only tests against the values in the buffer. The advantage of this procedure is that cost-intensive operations (e.g., illumination) are only carried out for fragments that contribute to the final image. In the literature, the described occlusion culling procedure is also referred to as *early Z rejection* or *Z pre-pass*.

An alternative occlusion culling method, which is also supported by the hardware, is the so-called *occlusion query*. For an occlusion query, the primitives of the object geometry are not sent through the pipeline, but only the primitives of the associated bounding volume. Visual effects need not be calculated. Without manipulating color or depth buffers, the graphics hardware counts the pixels that would be drawn for the bounding volume. The early stages of the rendering pipeline performed on the CPU can request this value from the GPU after the request has been executed. If the number of pixels covered by a bounding volume is zero, it is occluded by another object and the actual scene object does not need to be drawn. The problem with this technique, however, is that the CPU has to wait for the processing to finish for each request. In addition to sole processing time, a delay due to the comparatively slow communication channels to the GPU must also be expected. Fortunately, these requests can also be transferred asynchronously to the hardware, so that several tests can be processed in the GPU at the same time. Also, the CPU can process other tasks while waiting.

Occlusion culling is particularly interesting for applications whose runtime behavior is dominated by the computation time of the fragment shader (texturing, illumination, postprocessing).

Backface Culling

When polygon meshes are rendered, it is usually possible to specify if a polygon should only be visible when seen from one side (*one-sided polygon*) or when seen from either front or back (*two-sided polygon*). *Backface culling* deals with the removal of polygons from the rendering pipeline that face away from the viewer. In general, associated normals are stored for each polygon. If the normals are not explicitly stored, the direction of the normals can also be derived by using a convention whereby the vertex order (clockwise or counterclockwise) determines the orientation of the normal (see also Sect. 7.3.2). For backface culling the locally defined polygons and their normals are transformed into the camera's coordinate system. Now the normals of the polygons are compared with the camera's view direction. If the scalar product of a polygon's normal with the view direction is smaller than zero, the two vectors point in opposite directions, meaning that the front face of the corresponding polygon is visible from the camera. Otherwise, a backfacing polygon is encountered and culled from the rendering pipeline. Backface culling is nowadays almost exclusively performed on the GPU, since the transformation step is an integral part of the graphics pipeline.

Small Feature Culling

In many cases, details of a scene can be omitted without the viewer noticing that they are missing. The basic idea of *small feature culling* is that very small or very distant objects affect only a few pixels in the resulting image. To determine whether this applies to a given object, its bounding volume can be projected and its size measured. If the size is below a specified threshold, the object is not drawn. This process is particularly easy to solve in connection with the occlusion query (see occlusion culling).

If small feature culling is enabled, the rendered output image will be slightly inaccurate. However, especially in dynamic scenarios (also including fast viewer movements, head tracking), the probability is high that the error will not result in noticeable differences but will give an improved frame rate.

Portal Culling

The *portal culling* method is particularly suitable for virtual worlds that simulate closed rooms or buildings. For this purpose, the world is divided into sectors (rooms). The user can move from one sector to the next through defined portals (doors/passages). The sectors do not necessarily have to be spatially connected to each other. For portal culling it is only important that the polygon describing the portal is marked as such.

At a given time, the user (the camera) is in a sector. This sector is drawn as usual according to the camera's viewing frustum. In addition, a new viewing frustum is determined for each portal in the field of view, which is defined by the viewer position and the edges of the respective portal. With this new viewing frustum the sector on the other side of the portal is drawn (Fig. 7.14).

Thus, the number of sectors required for rendering is automatically limited to sectors that are actually visible through a portal. Furthermore, in these sectors, using

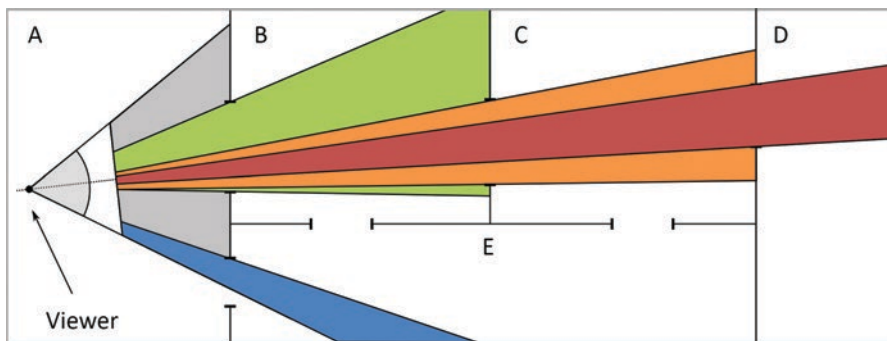


Fig. 7.14 Portal culling: the viewer is located in sector A (view volume/frustum of the viewer drawn in grey). For each visible portal the view volume is highlighted in color

view volume culling, only those objects have to be drawn that are located within the view volumes generated by portal culling.

As this method is very similar to view volume culling, these techniques can be combined without much effort. This makes portal culling not only easy to implement, but also very efficient for virtual worlds that are divided into different sectors or rooms.

Level of Detail (LOD)

Small feature culling removes small – and therefore hardly visible – objects from the scene. However, the technique does not solve a problem that quickly arises with high-resolution objects: with increasing distance to the viewer, the details become less and less perceptible. Without further measures, possibly millions of polygons and high-resolution textures must be transferred to the graphics hardware and drawn completely, even if the object covers only a few pixels in the rendered image. This situation can be avoided by introducing replacement objects according to the *level of detail (LOD)* method (see also Sect. 3.3.4 and Luebke et al. 2003).

According to the LOD method, several simplified versions of decreasing detail are created offline for high-resolution scene objects and selectively rendered at runtime. As soon as the object falls below or exceeds a certain distance threshold from the viewer, the system switches to a more or less detailed version. Alternatively, instead of the distance, the projected object size in screen space can be used as an indicator for the LOD level to be selected.

High-detail objects may be simplified in many ways. For example, versions with reduced polygon count are just as conceivable as versions with low-resolution textures or quality-reduced lighting. Provided that the switching times and quality levels are correctly selected, the exchange of the levels can be unnoticeable in practice. Especially for objects with “infinite” detail, such as terrain data, the LOD method makes a decisive contribution for maintaining interactive frame rates. In general, scenes with many complex objects benefit most from the use of the LOD technique.

An obvious disadvantage of the LOD technique is the extra memory requirement, because in addition to the original model, several other, less detailed models must also be stored. However, since the low-detail models contain less information anyway, these costs are usually not a big concern in practice. A bigger problem is usually the generation of the LOD levels. The automated generation of visually appealing simplified versions of a high-resolution polygon mesh is a non-trivial problem. There are algorithms that can reduce the polygon count of given meshes. However, such algorithms usually require checking of results and manual corrections to achieve appealing results.

In practice, therefore, the detail levels are often modeled by hand, which, however, significantly increases the effort and costs involved in their creation. Parametric models such as free-form surfaces allow the automatic creation of versions in different resolutions. However, non-parametric, mesh-based modeling tools are much

more widespread and also more intuitive to use. A comprehensive overview of LOD techniques is given in Luebke et al. (2003).

7.3.2 *Hardware-Related Strategies*

There are good reasons to hide the complexities of modern (graphics) hardware from the application developer. Suitable abstraction levels enable the developer to write programs that can be executed on different devices with similar efficiency. Nonetheless, a certain knowledge of special hardware features can provide starting points for performance improvements of the application.

The following strategies for *real-time rendering* of virtual worlds show ways to minimize memory consumption, utilize hardware processing units and optimize the usage of hardware caches.

Object Size

Current graphics hardware is capable of displaying several hundred million triangles per second. This processing speed is achieved because the problem of image rendering can be solved mostly independently for each pixel and because the highly parallel graphics hardware is optimized for this task. Modern GPUs contain dozens of stream processors where each stream processor in turn consists of many shading units. For example, an Nvidia Geforce RTX 3080 has 68 stream processors with 128 shading units each, for a total of 8,704 shading units. While all shading units execute in parallel, shading units within the same stream processor perform the same operations on different parts of the input data, e.g., projecting vertices to *NDC* (*Normalized Device Coordinates*). It is the task of the graphics driver (or the hardware) to partition the input data, e.g., polygon meshes, into groups and assign them to the available stream processors. To make a very simplified example: assume a GPU with four stream processors with 32 shading units each. Now a scene object consisting of 100 vertices is to be transformed. For this purpose, four subtasks must be created, which are then assigned to the four available stream processors. Say three stream processors are tasked to transform 32 vertices each, and the last one the remaining four vertices. As all threads of a stream processor run the same code, 28 of them are masked so as not to provide invalid results. That is, $28/128 \approx 22\%$ of computational resources are wasted! The problem also occurs in the following situation: 100 cubes of a scene are to be drawn. Since the cubes consist of only eight vertices each and each cube has to be assigned to a stream processor of its own (each cube is transformed/projected differently), the utilization of the hardware's processing resources is very unfavorable. From this it can be concluded that scene objects should be modeled with sufficient detail if graphics processors are to benefit from their parallel computing hardware. Another conclusion is that simple scene objects should be combined to larger objects so that they can be passed as a whole to the

GPU. This gives the graphics driver (or the GPU) the opportunity to allocate available execution and shading units in a resource-efficient way.

Indexing

Often, the geometry data of scene objects are available as unsorted triangle meshes. This data representation is often the output of modeling tools and is also known as *triangle soup* or *polygon soup*. These terms highlight that the polygons of the mesh are completely unstructured and have no explicit relation to each other. Metaphorically, the triangles “float” at arbitrary places in the soup. The actual data structure is just a vector (array, list) of vertices. A sequence of three vertices defines a triangle. However, triangles (and vertices) that are close to each other in the mesh are not necessarily close to each other in the data vector. Another consequence is that the vertices of a triangle mesh are typically contained several times in the data vector (once for each triangle they belong to). The memory requirement for such triangle soups is actually about three times the size of a memory-optimized variant (see Sect. 7.3.2 “Stripping”). Furthermore, the disadvantageous fact that a vertex may be contained in multiple copies in the data vector also means that it must be processed by the graphics pipeline multiple times (transformation, lighting, projection, etc.). Without additional measures a previously calculated result of vertex processing cannot be reused.

To avoid these inefficiencies, an indexing scheme can be introduced (see also Sect. 3.3.1: *indexed face set* or *indexed mesh*). The vertex coordinates (usually three floating point values with 4–8 bytes each per vertex) are stored in one data vector. A second data vector, the *index vector*, defines which vertices combine to a triangle. Each sequence of three indices (integer with 2–4 bytes per value) defines a triangle. While the index vector requires extra memory space, this is more than compensated by the absence of multiple copies of a vertex in the vertex vector. Overall, the memory requirements of a polygon mesh can be significantly reduced. Fig. 7.15 (left) illustrates the indexed mesh data structure.

Software systems for graphical data processing sometimes use not only one index vector for all vertex data but separate index vectors for vertex coordinates, normals and other attributes (e.g., colors). This can be useful if a vertex is to use different attributes depending on the triangle from which it is referenced. However, this multiple indexing is not supported by typical graphics hardware. If 3D objects have been modeled in such a representation, they must be re-sorted to a single index data structure before they are passed to the hardware.

Caching

Indexing alone does not solve the problem of reusing already computed vertex processing results when the vertex is part of more than one triangle. As the index vector presupposes no particular order of triangles, in particular, geometrically adjacent

triangles may occur at totally different positions in the index vector. To put it in slightly different words, a vertex shared by two triangles may occur at very different positions in the index vector. With *caching* it is possible to reuse *recently* computed vertex data. For this, it is necessary that a second occurrence of the vertex is close to its first one in the index vector. If the distance is too large, the vertex data in the GPU must be completely recalculated. The sort order of the index vector thus becomes relevant. A desirable property is a high *locality* of the index vector, i.e., spatially adjacent triangles are also in each other's neighborhood in the index vector (see also Fig. 7.15: the geometric positions of the vertices are not reflected in the index vector, i.e., low locality).

The typical model of a computer – the von Neumann architecture – provides that data and instructions use the same memory. From the programmer's point of view, the flow of a program is therefore strictly sequential. Problematic, however, is the data transfer between memory and the CPU, the so-called *von Neumann bottleneck*. Nowadays it takes much more time to transport the data to the CPU than it takes the CPU to actually process this data. Without further mitigations, a modern CPU could never be used to full capacity.

Caches were introduced to compensate for this memory latency. Caches are fast intermediate memories. Often, they store data in the form of an associative array. Such caches are also used on the graphics hardware to avoid or minimize memory latencies. An important limitation of these caches is their storage capacity. To keep access times to these caches as low as possible, they are physically placed near the processing units. But especially there, chip area is an expensive commodity. Therefore, the capacities (compared to RAM/VRAM) are usually very small and only a few entries can be kept in the cache. Exact data about GPUs is difficult to access but the capacities are typically in the low megabyte range for level-2 caches

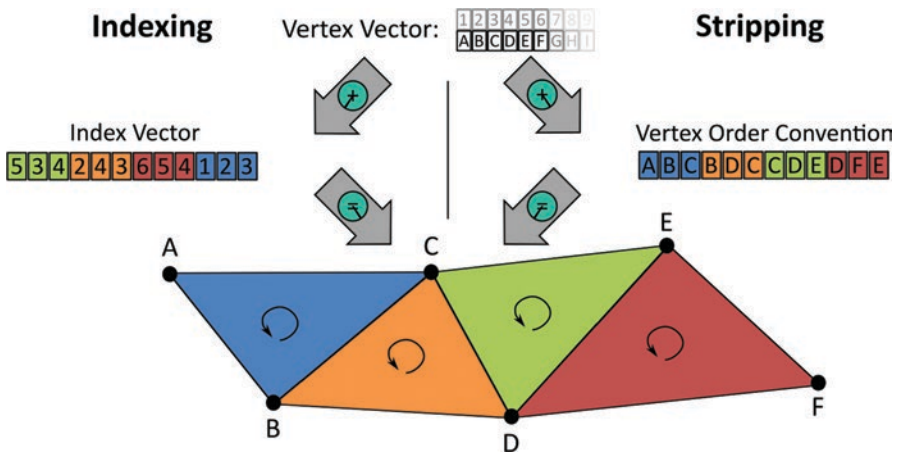


Fig. 7.15 Triangle mesh representation with indexing and stripping. Left: Vertex and index vectors define a triangle mesh. Right: Vertex vector and a convention on vertex ordering yield the triangle mesh

and in the kilobyte range for level-1 caches. Since the cache size is usually much smaller than the GPU RAM, not all data can be cached. A strategy must be implemented that defines the assignment of cache entries to memory entries. Often a memory entry cannot be placed at any position in the cache (full associativity), but several memory entries/regions are mapped to the same cache entry (set associativity). To make a practical example, this means: if a vertex is needed to project a triangle A, it must first be transferred from the slow GPU RAM to the cache. If another triangle B accesses this same vertex immediately afterwards, it is highly probable that the vertex data is still available in the fast cache. However, if calculations are made in the meantime that require other data, these will replace the vertex data in the cache. Then, for the projection of triangle B, the vertex must be reloaded from the GPU RAM.

Since cache properties are generally very hardware-specific, it is hardly possible to define generally applicable procedures. One consequence for real-time rendering of virtual worlds, though, is that the index vector for a triangle mesh should be sorted in such a way that it fulfills the locality property well. Furthermore, the program code (including shader code) should also take into account the properties of the available caches and, if possible, access memory sequentially (instead of randomized access patterns).

If the cache size is known, the optimization can be done very well (Hoppe 1999). However, as Bogomjakov and Gotsman (2002) have shown, good results are possible even if the cache size is unknown. A concise discussion with sample code can be found in Forsyth (2006).

Stripping (Triangle and Quadrilateral Strips)

One way to convert polygon data into a cache-optimized form is *stripping*. Stripping, i.e., the transformation of a polygon mesh into triangle strips or quadrilateral strips, was already introduced in Sect. 3.3.1. In the context of rendering efficiency, their second advantage, besides the cache-optimized form, is that they explicitly describe which vertices form a triangle (or quadrilateral). Thus, duplicate vertices or vertex indices are avoided, making triangle and quadrilateral strips also a very memory-efficient representation of polygon meshes.

The vertices of a data vector are interpreted according to a fixed convention. Assume a vector with four vertices A, B, C and D. These data can be interpreted, for example, in such a way that (ABC) and (BCD) each represent a triangle. The problem with this interpretation, however, is that the orientation differs between the two triangles, since by convention the clockwise direction determines the normal direction. In the interpretation presented here, the normals point to different sides (i.e., one triangle is front facing, the other one back facing). A better interpretation is therefore to specify the second triangle via the vertex sequence (BDC). Fig. 7.15 (right) shows the stripping for a triangle mesh and also shows the orientation of the triangles. If vertex data are used as triangle strips, the geometric positions of the

associated triangles are also automatically reflected in the data vector. With respect to caching, the data are thus available in a favorable form.

By means of stripping, n triangles can be specified with only $n+2$ vertices. As compared to indexing, which requires both a data vector and an index vector, stripping is more favorable from a memory consumption perspective. Compared to polygon soups (see Sect. 7.3.2 “Indexing”), stripping even reduces memory consumption by almost two thirds.

Strips offer a compact geometry presentation (no duplicate vertices, no index vector) and can positively influence the reuse of data on the GPU side. However, it is a non-trivial problem (NP-hard computational complexity) to find an optimal strip representation for an object. Instead, approximative *greedy algorithms* are usually used for strip generation that do not yield optimal but still very good results in very short times. Since, in general, an object cannot be represented by a single strip, either multiple strips or strips with degenerated triangles (i.e., triangles that degenerate into points or lines) must be used. This also results in reduced memory and display efficiency. To counteract this, modern 3D APIs offer “restart” interfaces (e.g., *glPrimitiveRestartIndex* for OpenGL). Instead of transmitting degenerated triangles, this interface can be used to tell the GPU that the strip interpretation should be restarted from a given index.

In the literature, there are several articles and papers on the subject of calculating the strips (e.g., Evans et al. 1996; Reuter et al. 2005). Furthermore, programs are available that generate strips from polygon meshes (e.g., NVTriStrip (NVidia 2004) or Stripe (Evans 1998)). While polygon soups are easy to handle but memory-consuming, strips are at the other end of the scale: they are memory efficient but much more difficult to handle and create.

Minimizing State Changes

As the saying goes, time is money. For this reason, a contract painter will be inclined to finish pictures in the shortest possible time. Since he needs different brushes and colors for the paintings, he will try to change the drawing equipment or the paint color as rarely as possible. After all, for every change of brush, the old brush has to be cleaned and stowed away. The graphics hardware is not unlike the painter in this respect – although an even more accurate metaphor would be a large group of painters who must all use brushes of the same kind with the same paint color at a time.

As discussed earlier, a GPU is composed of many parallel processing units. These execute the same instructions at different points of the input data where common *state* information specifies how, e.g., with which texture a vertex or fragment is to be processed. When drawing a given object it is therefore important to make only those state changes that are actually necessary.

Also, it is advisable to organize the order of object transfer to the graphics hardware in such a way that as few state changes as possible have to be made for an image to be drawn. If many objects are to be drawn, where some use one material (textures, colors, shaders), others a second material, and even others are a third

material, etc., the objects could, e.g., be sorted by material before transferring them to the GPU.

Furthermore, changes to the graphics pipeline configuration (e.g., changing the shader program) can lead to time-consuming operations in the driver or hardware.

Virtual worlds are usually not designed according to the above principles. Which sort order (e.g., by material or shader program) is useful depends strongly on the specific virtual world and cannot be prescribed in a generally valid way. While this task cannot be performed by the graphics driver or the graphics hardware, software systems for virtual worlds can be helpful tools.

7.3.3 *Software Systems for Virtual Worlds*

The previous sections described a number of methods that can help to increase the rendering speed of a virtual world. Ideally, these methods would be part of the graphics driver or hardware and any application could achieve optimal performance. However, this is not the case.

The graphics driver (and the APIs provided, e.g., Direct3D, OpenGL, Vulkan) provides a thin abstraction layer between the actual hardware and the application program. It mainly serves as a unified interface to the hardware of different manufacturers and contains no application-specific optimizations. These are left to the application developer, who has the freedom and responsibility to flexibly make design choices that suit the needs of the specific application.

Furthermore, the graphics driver does not have information about the entire scene (but only of individual objects), so that certain optimizations (e.g., view volume culling) cannot be implemented in a meaningful way. To support the developers of VR software, who cannot be expected to completely implement all algorithms and procedures, software systems exist which take over this task and thus support and accelerate application development. A widespread principle is the scene graph.

Scene Graph Systems

The general concept of a scene graph was introduced in Sect. 3.2. This section focusses on processing aspects of scene graphs that are useful for the real-time capability of a VR system.

The basic idea of scene graphs is to represent the entire virtual world, including some metadata, in a hierarchical graph, either a *tree* or a *directed acyclic graph (DAG)*. At runtime, the scene graph software then traverses this graph and performs operations on individual nodes or subgraphs. In many cases, the hierarchy is traversed *top-down* and *depth-first*. Examples for these operations are intersection testing during a user interaction, updating the position of dynamic objects, calculation of bounding volumes for both leaves and inner nodes and visibility testing on the basis of these bounding volumes.

During a single time step, a scene graph is typically traversed several times. In this context, one often speaks of different phases:

- APP: Application phase (change structure and states of the graph)
- CULL: View volume culling
- DRAW: Rendering on the GPU

A trivial implementation of a scene graph sends all contained nodes to the graphics hardware, even those representing objects not seen by the camera. However, as the entire scene and its hierarchy are contained in the graph, the scene graph system can easily calculate *bounding volumes* and *bounding volume hierarchies*. Based on these data and the view volume specified by a special camera node, the scene graph system can determine during the CULL phase which objects are within the field of view. LOD calculations are also easily performed. However, before the objects within the field of view are sent to the graphics hardware to be finally rendered during the DRAW phase, they are usually sorted in such a way as to minimize changes of the graphics state.

This APP-CULL-DRAW model became popular through Iris Performer and its successor OpenGL Performer (Rohlf and Helman 1994). The model is particularly interesting because it provides a good basis for parallelization of scene graph processing. This enables scene graph systems to benefit from modern multi-core processors and thus to process more complex scenes in real time.

Scene graph systems can significantly accelerate the development of complex VR applications. They offer a wide range of tools for scene generation, animation, user interaction and various optimizations (e.g., cache optimization of vertex data, merging of static structures). They abstract the complexity of these methods and provide VR developers with accessible interfaces that enable them to achieve their goals quickly. Many scene graph systems also support special effects that are not completely performed by the graphics hardware (e.g., shadow calculations).

The price for these benefits is often a somewhat limited flexibility. Adding new algorithms to a complex system, such as a scene graph system, can be much costlier than implementing them from scratch. It is therefore not surprising that, for example, scientific visualization or virtual communication applications often implement customized solutions without using a scene graph system.

Game Engines

Game engines are development and runtime environments for computer games. In the field of real-time 3D computer games, game engines often combine high visual quality with comfortable development tools. Besides target platforms such as desktop PCs, game consoles and smartphones, many game engines also support the development of VR/AR applications.

Modern game engines are complex software systems consisting of various sub-systems, such as a rendering engine, physics engine and audio system. In addition, game engines offer support for animation, multiplayer play modes, game AI and

user interaction. For level design, i.e., the modeling of virtual worlds, some game engines provide their own development environments, which are usually strongly customized to the respective functionalities of the game engine. Virtual worlds are often modeled based on scene graphs. Typically, special modeling systems are also provided for the creation of vegetation, terrain and particle systems as well as for the animation of virtual humans (see Chap. 3). Most game engines also offer scripting support for programming the game logic.

Chapter 10 illustrates the authoring process in game engines as well as their configuration for VR/AR applications using Unity and the Unreal Engine as examples.

7.4 Summary and Questions

Real-time capability is of crucial importance for believable VR/AR experiences. In combination with head-tracking, a *latency* of at most 50 ms is recommended for HMD-based systems (Brooks 1999; Ellis 2009). Higher latencies are more tolerable for projection-based VR systems. Latencies occur in all subsystems of VR/AR systems. In addition, latencies of data transport between the subsystems must be considered to minimize the overall latency (end-to-end latency) of a VR/AR system. In this chapter, methods for measuring the latency of tracking systems as well as end-to-end latency were presented. Furthermore, typical latencies for different hardware components of VR/AR systems were discussed, including different types of tracking systems and network components. The latencies of other VR/AR subsystems, such as world simulation and rendering are more dependent on the specific application. A generic task during world simulation is collision detection. For this purpose, a number of methods exist that allow efficient collision detection even in large environments with a high number of objects. The scene graphs commonly used in VR systems support efficient rendering in a variety of ways, e.g., different culling methods, level of detail techniques, and memory-effective and cache-friendly data structures for polygonal models, as well as optimization of the rendering order of the 3D objects in the virtual world.

Check your understanding of the chapter by answering the following questions:

- Why is low end-to-end latency so important for VR/AR systems?
- Where do the latencies of VR/AR systems come from?
- Sketch a concrete VR application and discuss the relevance of different kinds of latency on this example!
- How can latencies be measured or estimated?
- What are the typical requirements for bounding volumes? What consequences result from these requirements?
- What is a separating axis and how can one be found for two OBBs?
- Explain the Sweep & Prune procedure using a self-drawn sketch. Explain the advantages and disadvantages of the procedure!

- Scene graphs can be organized according to different criteria. In a logical or semantic structure, objects could be grouped according to their type, e.g., by having one common group node for all cars, another common group node for all houses etc. In a spatial structure, on the other hand, objects that are close to each other would be grouped together. What type of grouping is more efficient for view volume culling? Also explain hierarchical view volume culling!
- In scene graphs, bounding volumes such as cuboids or spheres are automatically generated for all inner nodes. How can this be exploited with the different variants of culling (view volume culling, occlusion culling, small feature culling)?

Recommended Reading

- Jerald JJ (2010) Scene-motion- and latency-perception thresholds for head-mounted displays. *Dissertation*, UNC, Chapel Hill, <http://www.cs.unc.edu/techreports/10-013.pdf>. Accessed August 11, 2020 – *Jerald's doctoral thesis deals intensively with the topic of visual latencies in virtual reality and contains an extensive collection of literature on the subject.*
- Ericson C (2004) *Real-time collision detection*. CRC Press – *The book provides a comprehensive and in-depth overview of collision detection methods.*
- Akenine-Möller T, Haines E, Hoffman N, Pesce A (2018) *Real-time rendering*, 4th edn. CRC Press – *Textbook on advanced topics in computer graphics, providing a comprehensive overview of techniques for real-time rendering of 3D worlds.*

References

- Abrash M (2012) Latency – the sine qua non of AR and VR. <http://blogs.valvesoftware.com/abrash/latency-the-sine-qua-non-of-ar-and-vr/>. Archived at <https://perma.cc/J29Q-KEQ8>. Accessed 6 Feb 2021
- Adelstein BD, Johnston ER, Ellis SR (1996) Dynamic response of electromagnetic spatial displacement trackers. *Presence* 5(3):302–318
- Akenine-Möller T, Haines E, Hoffman N, Pesce A, Iwanicki M, Hillaire S (2018) *Real-time rendering*, 4th edn. Taylor & Francis
- Assarsson U, Möller T (2000) Optimized view frustum culling algorithms for bounding boxes. *J Gr Tool* 5(1):9–22
- Baraff D (1992) *Dynamic simulation of non-penetrating rigid bodies*. Dissertation, Cornell University
- Bauer F, Cheadle SW, Parton A, Muller HJ, Usher M (2009) Gamma flicker triggers attentional selection without awareness. *Proc Natl Acad Sci* 106(5):1666–1671
- Bogomjakov A, Gotsman C (2002) Universal rendering sequences for transparent vertex caching of progressive meshes. *Comp Gr Forum* 21(2):137–149
- Brooks FP (1999) What's real about virtual reality? *IEEE Comp Gr Appl* 19(6):16–27
- Cameron S (1997) Enhancing GJK: computing minimum and penetration distances between convex polyhedral. *Proceedings of International Conference on Robotics and Automation*, pp 3112–3117

- Carmack J (2013) Latency mitigation strategies. #AltDevBlog. Internet Archive: <https://web.archive.org/web/20140719085135/http://www.altdev.co/2013/02/22/latency-mitigation-strategies/>. Accessed 6 Feb 2021
- Catto E (2020) Box2d – a 2D physics engine for games. <http://box2d.org/>. Accessed 6 Feb 2021
- Dong Y, Peng C (2019) Screen partitioning load balancing for parallel rendering on a multi-GPU multi-display workstation. Eurographics Symposium on Parallel Graphics and Visualization, Eurographics Association
- Eger Passos D, Jung B (2020) Measuring the accuracy of inside-out tracking in XR devices using a high-precision robotic arm. HCI International 2020 – Posters. HCI International 2020, 22nd International Conference on Human-Computer Interaction, Proceedings, Part I, pp 19–26
- Ellis SR (1994) What are virtual environments? IEEE Comp Gr Appl 14(1):17–22
- Ellis SR (2009) Latency and user performance in virtual environments and augmented reality. Distributed Simulation and Real Time Applications, DS-RT 09, p. 69
- Ericson C (2005) Real-time collision detection. Morgan Kaufmann, San Francisco
- Evans F (1998) Stripe. <http://www.cs.sunysb.edu/~stripe/>. Accessed 6 Feb 2021
- Evans F, Skiena S, Varshney A (1996). Optimizing triangle strips for fast rendering. In: Proceedings of Visualization '96, IEEE, pp 319–326
- Forsyth T (2006) Linear-speed vertex cache optimisation. https://tomforsyth1000.github.io/papers/fast_vert_cache_opt.html. Accessed 6 Feb 2021
- Gilbert EG, Johnson DW, Keerthi SS (1988) A fast procedure for computing the distance between complex objects in three-dimensional space. J Robot Autom 4(2):193–203
- Gregory J (2009) Game engine architecture. A K Peters, Natick
- He D, Liu F, Pape D, Dawe G, Sandin D (2000) Video-based measurement of system latency. In: Fourth international immersive projection technology workshop (IPT2000)
- Hoppe H (1999) Optimization of mesh locality for transparent vertex caching. In: Proceedings of 26th Annual Conference on Computer Graphics and Interactive Techniques, pp 269–276
- Hübner T, Zhang Y, Pajarola R (2007) Single pass multi view rendering. IADIS Int J Comp Sci Infor Syst 2(2):122–140
- Jerald J, Whitton M, Brooks FP (2012) Scene-motion thresholds during head yaw for immersive virtual environments. ACM Trans Appl Percept 9(1):1–23
- Lengyel E (2002) *Mathematics for 3D game programming and computer graphics*, 2nd edn. Charles River Media, Rockland
- Liang J, Shaw C, Green M (1991) On temporal-spatial realism in the virtual reality environment. In: Proceedings of UIST, pp 19–25
- Lin MC, Canny JF (1991) A fast algorithm for incremental distance calculation. Proc IEEE Int Conf Robot Autom 2:1008–1014
- Luebke DP, Reddy M, Cohen J, Varshney A, Watson B, Huebner R (2003) Level of detail for 3D graphics. Morgan Kaufmann, San Francisco
- Meehan M, Razzaque S, Whitton MC, Brooks FP (2003) Effect of latency on presence in stressful virtual environments. In: Proceedings of IEEE Virtual Reality, pp 141–148
- Mine M (1993) Characterization of end-to-end delays in head-mounted display systems. Technical Report 93–001, University of North Carolina at Chapel Hill
- NVidia (2004) NvTriStrip library. <https://github.com/turbulenz/NvTriStrip>. Accessed 6 Feb 2021
- Reuter P, Behr J, Alexa M (2005) An improved adjacency data structure for fast triangle stripping. J Gr GPU Game Tool 10(2):41–50
- Rohlf J, Helman J (1994) Iris performer: a high performance multiprocessing toolkit for real-time 3D graphics. In: Proceedings of 21st annual conference on computer graphics and interactive techniques. ACM, pp 381–394
- Sathe R, Lake A (2006) Rigid body collision detection on the GPU. In: ACM SIGGRAPH 2006 research posters. ACM, New York
- Skogstad SA, Nymoen K, Høvin M (2011) Comparing inertial and optical MoCap technologies for synthesis control. In: Proceedings of the 8th Sound and Music Computing Conference

- Steed A (2008) A simple method for estimating the latency of interactive, real-time graphics simulations. Proc VRST:123–129
- Swindells C, Dill JC, Booth KS (2000) System lag tests for augmented and virtual environments. Proc UIST 00:161–170
- Weller R (2012) New geometric data structures for collision detection. Dissertation, Universität Bremen. <http://nbn-resolving.de/urn:nbn:de:gbv:46-00102857-18>. Accessed 11 Aug 2020
- Welzl E (1991) Smallest enclosing disks (balls and ellipsoids). In: Results and new trends in computer science. Springer, Berlin/Heidelberg, pp 359–370
- Xavier PG (1997) Fast swept-volume distance for robust collision detection. Proc IEEE Int Conf Robot Autom 2:1162–1169
- You S, Neumann U (2001) Fusion of vision and gyro tracking for robust augmented reality registration. Proceedings of IEEE Virtual Reality, pp 71–78

Chapter 8

Augmented Reality



Wolfgang Broll

Abstract This chapter covers specific topics of Augmented Reality (AR). After an introduction to the basic components and a review of the different types of AR, the following sections explain the individual components in more detail, as far as they were not already part of previous chapters. This includes in particular the different manifestations of registration, since these are of central importance for an AR experience. Furthermore, special AR techniques and interaction types are introduced before discussing individual application areas of AR. Then, Diminished Reality (DR), the opposite of AR, is discussed, namely the removal of real content. Finally, Mediated Reality, which allows for altering reality in any form, including the combination of AR and DR, will be discussed.

8.1 Introduction

The following overview provides a quick introduction to the most important aspects of augmented reality.

8.1.1 Getting Started

In accordance with the definition already given in Chap. 1 (see Sect. 1.3), *Augmented Reality (AR)* can generally be understood as the enrichment of reality by artificial virtual content. In doing so, a fusion of reality and virtuality occurs (see also

Dedicated website for additional material: vr-ar-book.org

W. Broll (✉)

Department of Computer Science and Automation/Department of Economic Sciences and Media, Ilmenau University of Technology, Ilmenau, Germany
e-mail: wolfgang.broll@tu-ilmenau.de

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022

R. Doerner et al. (eds.), *Virtual and Augmented Reality (VR/AR)*,
https://doi.org/10.1007/978-3-030-79062-2_8



Fig. 8.1 Fusing a real environment (left) with a virtual object (right) to achieve Augmented Reality (center). (Single images: © Tobias Schwandt, TU Ilmenau 2018. All rights reserved)

Milgram et al. 1995). Figure 8.1 shows an example of a real scene and its augmentation by a virtual object.

It is crucial that this augmentation does not happen statically and at once, as in the above illustration, but continuously and adapted to the current point of view of the respective viewer. In simplified terms and neglecting individual alternatives, AR can be realized through the following five steps:

1. Video capturing
2. Tracking
3. Registration
4. Visualization
5. Output

The individual steps and components are only briefly explained here to give the reader an initial idea. As far as they were not already considered in previous chapters, they will be discussed in detail in the following sections.

Video Capturing

In the first step, a video image or, more precisely, a *video stream* of the observer's surroundings is usually recorded. The purpose is to capture the reality, i.e., the real environment of the user (see also Fig. 8.1, left). This can be done using any kind of camera (webcam, smartphone camera, television camera, etc.). It is important that the camera has been calibrated accordingly; see also Szeliski (2011). Later we will introduce other types of AR for which a camera image of the environment is not necessary (see Sect. 8.1.2).

Tracking

Tracking is generally understood to be the calculation (or more correctly the estimation) of the position and/or pose/orientation (see Chap. 4). In the case of AR, it is necessary to capture the observer's point of view continuously and as accurately as possible. However, when reality is represented by the video image just captured, the position and orientation of the camera used are usually estimated instead. Nowadays, pose estimation can usually be obtained quite reliably using hybrid 3-DOF pose sensors (consisting of inertial sensors, gyro sensors and magnetometers; see also Sects. 4.2.2 and 4.2.3). Such sensors are now built into all current smartphones and tablets, but can also be available as separate input devices. In contrast to pose



Fig. 8.2 Simple registration using a fiducial marker

estimation, sufficiently exact position estimation is mostly difficult. In outdoor applications, GPS or one of its alternatives typically provides the basis in the context of AR, while for indoor applications, computer vision-based approaches are generally applied. The latter have the additional advantage that they can estimate the position as well as the orientation and may also additionally be used outdoors. Tracking thus results in a transformation from the user or camera coordinate system into the coordinate system of the real environment. Tracking in general is introduced in detail in [Sect. 4.2](#).

Registration

Registration (more precisely geometric registration) refers to the anchoring or correct fitting of the artificial virtual content into reality. On the basis of the position and orientation estimation from tracking, the coordinate system of the individual virtual content and the observed reality are put in relation to each other in such a way that the virtual content appears firmly located (registered) in reality. This leads to the situation that an artificial object not moving in the virtual world has an apparently fixed place in reality, independent of a changing point of view of the observer (or the camera). A simple registration scheme is shown in [Fig. 8.2](#). Geometric as well as photometric registration (the adaptation of the appearance of the virtual content to the illumination conditions of the environment) is presented in detail in [Sect. 8.3](#).

Visualization

Based on the transformation resulting from the geometric registration and the respective camera perspective, the virtual content is rendered. Thereby, the virtual content is superimposed on the recorded video image in the correct perspective (see [Fig. 8.2](#)). For seamless superimposition, many other aspects, such as resolution, sharpness, color range and contrast ratio of the virtual image, may have to be



Fig. 8.3 Perspective superimposition of an image used for tracking by a 3D object. (Image source: Jan Herling, TU Ilmenau)

adjusted. As an alternative to the video image overlay, an optical overlay of the observer's view can also be performed directly – see Sect. 8.1.2 for details. Other aspects that are important include mutual superimposition of the real and virtual content (c. Sect. 8.4.2) and the mutual influence between the virtual content and the real environment, also known as photometric registration (c. Sect. 8.2.2). If some of these aspects are not sufficiently taken into account, the virtual content can very quickly appear detached from reality despite correct geometric registration (see Fig. 8.3).

Output

Finally, the superimposed video images (or the augmented video stream) are shown on a display to which the camera is usually attached. This can be a handheld device such as a smartphone (see Fig. 8.4), a tablet or AR glasses. In principle, the output can also be on a separate monitor or via projection. However, in this case, the impression of a seamless augmentation of reality is only partially created for the observer. AR glasses are discussed in detail in Sect. 5.3, while other more specific AR output techniques are presented in Sect. 8.4.

8.1.2 AR – An Overview

In addition to the examples of AR presented in the first subsection, many other types exist. However, all types of AR have in common that they are based on a perspective-correct projection of the virtual content into the user's environment or onto the previously recorded video image. The point of view and the direction of view between the real and virtual environment must always be consistent. Furthermore, the virtual field of view must correspond to the actual field of view of the respective



Fig. 8.4 Output of an augmented video stream on a smartphone (here from the viewpoint of a second observer). (Image source: Jan Herling, TU Ilmenau)

display. Finally, the scaling of the virtual content must be adapted to the real environment.

Ideally, the perspective of the captured image and the perspective of the user (viewing the augmented image) should match as well, so that the user actually gets the impression that the real environment is really altered. Then, the user virtually looks through the display at the reality behind it (even if, depending on the characteristics of the AR, only a video image of the reality may be visible on the display). In this case we speak of the Magic Lens metaphor (see also Fig. 8.5 and Brown and Hua (2006)). In practice, this is only achieved when using AR glasses, as handheld devices typically do not support stereo vision (c. also Sect. 8.3.1).

The individual types of AR are explained below and then compared in terms of their limitations and capabilities.

Video See-Through AR

So-called video see-through AR (VST-AR), also known as video pass-through AR, is very similar to the approach described in the introduction. Therefore, first the real environment is captured by a video camera. Then the video image is superimposed with virtual content in the correct perspective and displayed on an output device afterwards (see Fig. 8.6).

To achieve the aforementioned Magic Lens effect, it is crucial that the viewpoint, viewing direction and field of view of the video camera and the output (i.e., the virtual camera) match. Otherwise, the viewer will experience a decoupling between their real environment and the augmented environment observed (see also Sect. 8.4).



Fig. 8.5 Example of a magic lens effect



Fig. 8.6 Perspectively correct superimposition of a camera image of the real environment by virtual content using video see-through AR – here on a smartphone

Optical See-Through AR

In contrast to the AR technology described above, optical see-through AR (OST-AR) does not require video capturing of the real environment. Instead, the real environment is always perceived directly by the observer. For this purpose, virtual content is optically superimposed on reality by the output device. This requires an output device with a semi-transparent display, so that both the reality behind it and the

added virtual content can be perceived simultaneously. To ensure that the perspective of the real environment and the virtual extension match, the point of view of the observer in relation to the display must be known. Generally, it is necessary to use a separate display for each eye. If both eyes look at the same display, it must be ensured that the areas observed are separated, so that the perspective for each eye can be adjusted correctly. In monoscopic displays, which are viewed with both eyes simultaneously, the perspective is at best correct for just one eye.

Projection-Based AR

Projection-based AR is characterized by projecting virtual content onto objects in the real environment (see Fig. 8.7). When projecting on arbitrary surfaces, this typically does not allow for the creation of new spatial structures and thus is typically limited to the manipulation of surface properties (like color or texture) and the display of additional information on the objects' surface (explanations, highlights, symbols, etc.). On suitable surfaces or with suitable channel separation techniques such as shutter glasses, front or back projections may be used to create objects before or behind the canvas similar to VR (see Sect. 5.4.4). This, however, also restricts virtual objects to the field of view covered by such surfaces (e.g., a dashboard in a driving simulator (Weidner and Broll 2019)).

Projection-based AR is a variant of *Spatial AR (SAR)* (see Bimber and Raskar 2005), in which the augmentation is not achieved using a display in an HMD or handheld device, but “in space”. However, in general, *Spatial AR* setups can also be based on video see-through or optical see-through AR.



Fig. 8.7 Example of projection-based AR (virtual door, virtual color design of the wall). (© Oliver Bimber 2005. All rights reserved)

Comparing the Individual Types of AR

Basically, the AR types described above differ in the extent to which they can expand or change reality. Using optical overlay techniques, dark virtual content and light backgrounds are particularly problematic. Tables 8.1, 8.2 and 8.3 provide an overview of the individual display capabilities and limitations.

In contrast to projection-based AR, both the optical see-through technique and the video see-through technique allow the display of virtual 3D objects at arbitrary positions within the space covered by the field of view (see Table 8.3, right column). Nevertheless, the perception of the surrounding reality and the virtual objects differs considerably between the two techniques, so that the respective other technique may be more suitable in a particular situation, depending on the application scenario.

When using OST AR as well as projection-based AR, dark virtual objects may appear completely transparent, as the overlay is purely optical, i.e., it is achieved by adding light (see Fig. 8.8). This means in particular that no shadows of virtual objects can be added (see Table 8.3, left column). This considerably limits the possibilities of photometric registration (see Sect. 8.3). Thus, the suitability of this technique depends very much on the respective real environment. Further, when illuminating the virtual scene and selecting the material properties, it must be taken into account that objects with a (too) low light intensity appear transparent. Such limitations may be overcome by applying an additional occlusion layer in OST glasses, which so far has only been demonstrated in research prototypes (see e.g., Hamasaki and Itoh 2019).

Table 8.1 Visibility of bright virtual content on different backgrounds depending on the AR type

	On bright background	On dark background
Optical see-through	Partially visible, high transparency	Good visibility, low transparency
Video see-through	Good visibility	Good visibility
Projection	Partially visible	Good visibility

Table 8.2 Visibility of dark virtual content on different backgrounds depending on AR type

	On light background	On dark background
Optical see-through	Not visible, almost complete transparency	Partially visible, high transparency
Video see-through	Good visibility	Good visibility
Projection	Not visible	Partially visible

Table 8.3 Display of virtual shadows and virtual objects in space depending on the AR type

	Virtual shadows	Universal object location
Optical see-through	Not possible	Possible
Video see-through	Possible	Possible
Projection	Not possible	Not possible/limited (on surfaces allowing for stereo projections)

Fig. 8.8 With the OST AR techniques, dark virtual objects sometimes appear transparent (here the less illuminated lower part of the red sphere)



Fig. 8.9 Typical perception when using the optical see-through technique (left) compared to the video see-through technique (right)

In general, it can be said that when using optical see-through technology, reality is perceived directly, i.e., without a limitation of resolution, but is instead perceived significantly darkened (see Fig. 8.9, left). Thereby, the virtual objects always appear partially transparent, i.e., as described above. Depending on the brightness of the virtual object and the real background, the latter may be clearly visible through the former.

VST AR, in contrast, allows the real background to be displayed with the same optical quality and brightness as the virtual content (see Fig. 8.9, right). However, this leads to a reduced resolution for the representation of the real environment compared to reality and the optical see-through technique. The reason for this is the limited resolution of the camera and the display used. For a coherent overall impression, the camera resolution should not be lower than that of the display. Otherwise, virtual objects may stand out sharply against the background (see e.g., Fig. 8.18). To achieve a coherent overall impression, it may therefore be useful to reduce the resolution of the virtual image to that of the camera.

8.2 Registration

As already introduced in Sect. 8.1, registration in the context of AR refers to the correct fitting of virtual content into the real environment. On the one hand, this has to be made perspective-correct (this is called *geometric registration*), but on the other hand it should also be correct with respect to appearance, i.e., in particular lighting. The latter case is also called *photometric registration*.

8.2.1 Geometric Registration

Tracking (see Sect. 4.3) provides the basis for geometric registration. Using the estimated transformation \mathbf{T}_{mc} between the viewpoint of the camera (in the case of video see-through augmentation) or that of the observer respectively (in the case of optical see-through augmentation), and the tracked object, the latter is displayed in the current field of view with correct position and orientation. In other words: geometric registration implies that a virtual object appears to be in the same place in reality even if the camera perspective is changed, i.e., if it is not an animated virtual object, it does not move in relation to the real environment (see Fig. 8.10). This is achieved by compensating for each change of the real camera position and orientation by a corresponding transformation of the virtual camera pose, and thus the displayed perspective of the virtual object is subsequently correct again with respect to the real pose.

The quality of the tracking used is crucial for the visual quality of the registration. However, the tracking update rate and the latency of the tracking may even have a larger influence on the visual appearance of the AR application.

Ideally, the tracking update rate exactly matches the frame update rate of the visual output (i.e., typically at least 60 fps). If the tracking rate is too low, the virtual objects seem to move with the head for some time (neglecting possible latency – see below) when the camera or head is moved, and then jump back to their correct position in the real world (see also Fig. 8.11).

With video see-through AR, the effect can be mitigated by adjusting the frame rate to the tracking rate. The effect of the virtual objects jerking or jumping disappears, but the abrupt image changes during strong camera movements can then be perceived as equally disturbing by the viewer. In addition, a discrepancy between the perceived motion (vestibular perception) and the optical perception (see also Chap. 2) is created. With optical see-through AR this possibility does not exist, because the observer perceives the surrounding and thus every faulty registration at any time.

Latency (see also Sect. 7.1) is another major problem regarding correct geometric registration. While the symptoms here are very similar to those of a too low tracking update rate and the latter also affects the latency, the actual problem is of a different nature. In tracking, latency is the delay between the moment of movement



Fig. 8.10 Left image: Correct geometric registration of the virtual trash can. Image top right: Virtual object is displayed at the same position as in the left image, but it is geometrically not registered with the surrounding reality. Image bottom right: Based on the tracking data, the correct perspective of the virtual object is displayed from the current viewpoint and the current viewing direction of the camera; the virtual object is geometrically correctly registered with the surrounding reality

(of the camera and/or the tracked object) and the moment when the resulting transformation of the virtual objects can actually be observed. Neglecting the delays caused by the tracking rate (see above) and the actual rendering, here the time between measuring or estimating the position and orientation and applying it to the object transformation remains. The longer this time span is, the more noticeable is

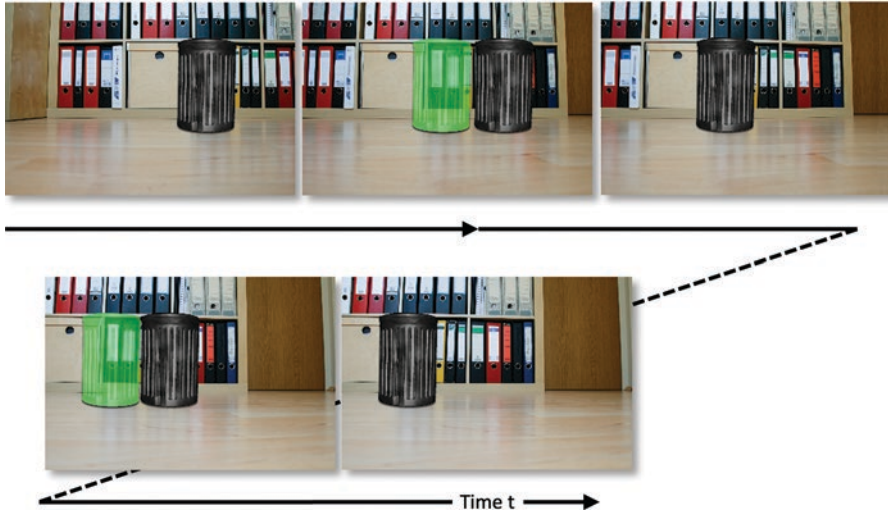


Fig. 8.11 Incorrect geometric registration due to a too low tracking rate: the camera moves from left to right; due to missing tracking updates, the virtual object (the black bin) first moves along with the camera (second and fourth images) and then suddenly jumps to the correct position (third and fifth images) when new tracking data becomes available (actual correct positions shown in green)

the resulting effect. The causes of high latency can be manifold: mostly it is due to relatively complex tracking techniques requiring a rather long time for calculation. In particular, feature-based approaches should be mentioned here. But also, other causes like long signal runtimes can result in high latency. Similar to a too low tracking rate, a virtual object will first move with the corresponding movement of the camera or the head of the observer. However, the movement does not jump or jerk, but the object remains (in the case of uniform movements) more or less at a fixed position in relation to reality, but has an offset to the correct position as long as the movement continues. Not until the movement of the camera or the head stops again is the virtual object registered correctly again (see Fig. 8.12).

In contrast to too low tracking rates, the problem of too high tracking latency, at least for video see-through AR, can be mitigated in most cases without serious degradation of the user experience. For this purpose, it is necessary to measure the resulting latency, and the camera images must be buffered over the corresponding period of time. If the tracking data is available, the transformed virtual objects are now combined with the camera image at the time of their capturing (see Fig. 8.13). Thus, the latency no longer exists between the virtual and real content of the considered image, but for the entire image. However, as long as this latency does not become too high (see Sect. 7.1.2), this will not be noticed by the observer and thus has no disturbing effect. Here, too, a corresponding correction is basically not possible with OST-AR, since the surrounding is perceived immediately – without any delay. The only alternative here is estimating the tracking data to be expected. In the

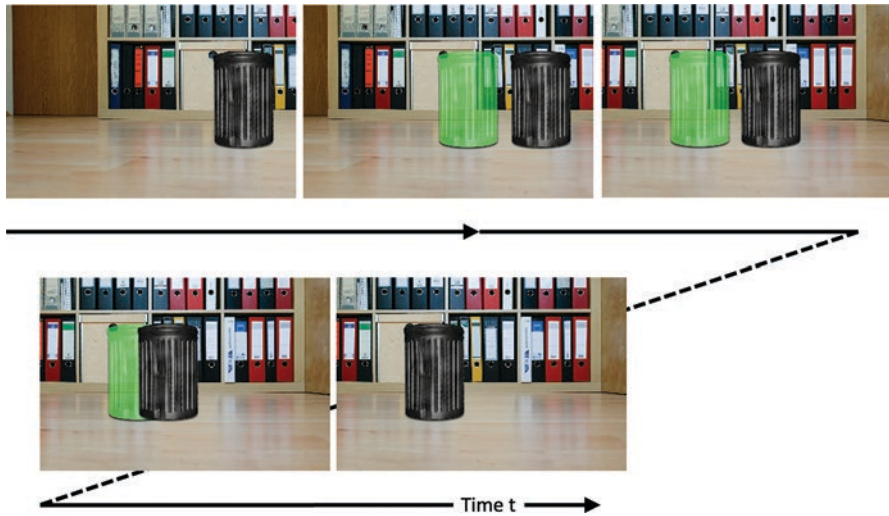


Fig. 8.12 Incorrect geometric registration due to high tracking latency: the camera moves from left to right; the virtual object first moves with the camera for a short time and then mostly freezes at a wrong position; only after stopping the movement of the camera does the virtual object move to its correct position (correct positions in green)

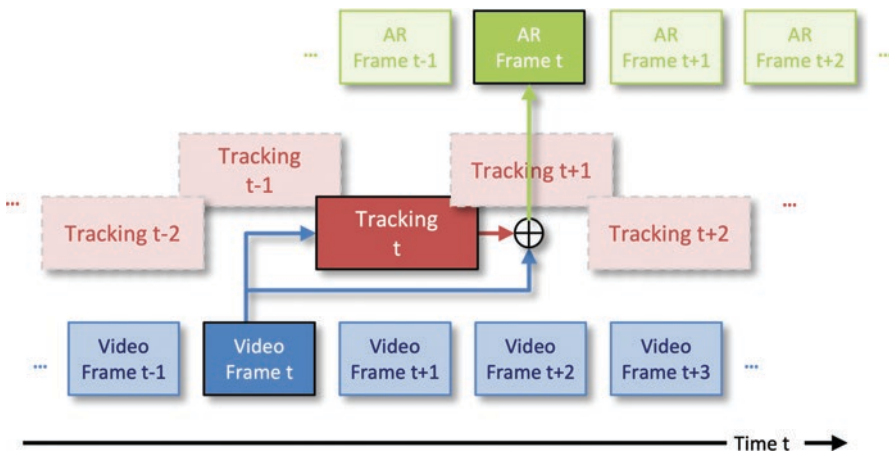


Fig. 8.13 Reduction of latency-related effects through caching of camera images and parallelization

past, Kalman filters were used for this purpose. Current approaches also partly use neural networks in this context.

A temporary incorrect geometric registration, whether caused by too low tracking rates or too high latency, destroys the illusion of a seamless integration of the virtual content into reality for the observer, which means that the corresponding AR application is only of limited use.

8.2.2 Photometric Registration

In contrast to geometric registration, which is a basic requirement for the use of AR, even today the photometric registration of virtual objects in the AR context is mostly performed only very rudimentarily, if at all. A prerequisite for a successful photometric registration, i.e., a correct adjustment of a virtual objects' appearance to its real environment, is – analogous to tracking for geometric registration – the acquisition or estimation of the respective data.

Generally, various methods can be used to capture the real lighting conditions. One option is the use of so-called light probes. In most cases, spheres are placed in the scene (Debevec 1998). Depending on the color and shininess of the spheres, different information about the lighting of the environment can be acquired. For diffuse lighting, corresponding virtual light sources are calculated and added to the virtual scene based on the highlights or bright parts of the image that are reflected there. For glossy reflections a mirroring sphere is used. While this approach can be used to adjust the appearance of the virtual objects well to the real environment, it is also fundamentally limited. On the one hand, it is often not possible or desirable to introduce corresponding light probes in the environment to be augmented, and on the other hand, the influence of the virtual objects and their virtual illumination on the illumination of the surrounding reality is neglected. An example of this is shown in Fig. 8.14.

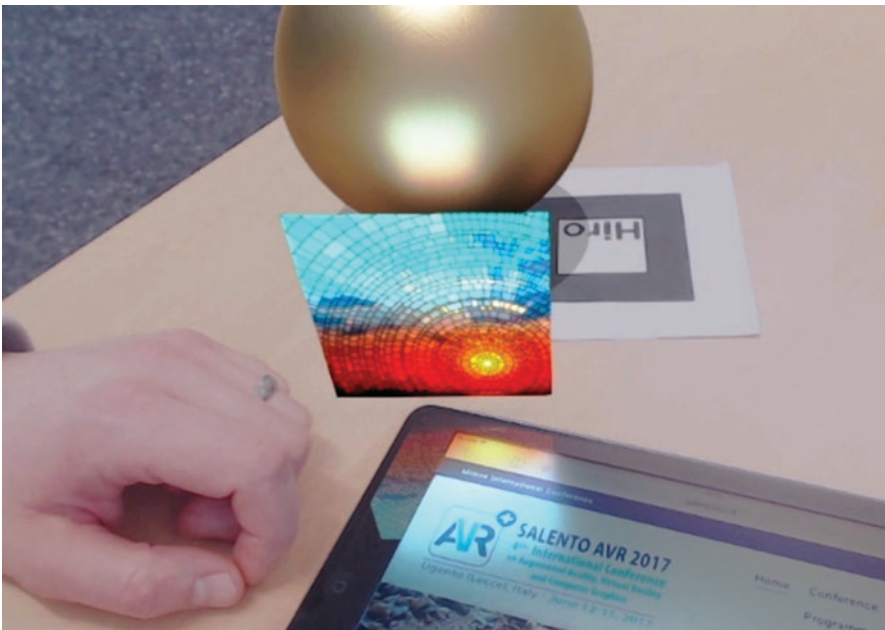


Fig. 8.14 Illumination of virtual content influencing the real environment (virtual reflection on tablet computer). (© Tobias Schwandt, TU Ilmenau 2018. All rights reserved)



Fig. 8.15 Comparison of an AR scene without and with photometric registration: in the right image, (real) light is reflected by the red sheet onto the virtual object; furthermore, light from the virtual object is reflected onto the background. (Picture source: Philipp Lensing, TU Ilmenau)

A complete adaption of the illumination of real objects can only be done using video see-through AR. A prominent example is the shadow cast by a virtual object onto the real environment. When using optical see-through AR, changes are restricted to those adding light (in Fig. 8.14 the reflection of the virtual image on the tablet would be possible, but not the shadow of the sphere on the desk). For correct photometric registration, an augmentation of parts of the reality is crucial (see Fig. 8.15). Therefore, this can already be considered a simple form of Mediated Reality (as it is limited to lighting) (see Sect. 8.7.2). An incomplete or incorrect photometric registration can very quickly destroy the illusion of seamless integration of reality and virtuality. Conversely, correct or at least plausible photometric registration can dramatically increase the perceived credibility of an AR scene for the viewer.

Current AR frameworks (e.g., Apple's ARKit or Google's ARCore) allow, to a limited extent, easy analysis and simulation of ambient light using cameras integrated into smartphones and tablets. One of the approaches used here is to estimate the direction of directional light sources based on face recognition and the brightness distribution in these faces (see Knorr and Kurz 2014). In combination with camera images from different directions, the local lighting situation can be estimated and simulated using so-called *spherical harmonics* coefficients (Kautz et al. 2002).

While a simple approximation of a virtual shadow for a virtual object lying on a flat surface (such as a table top) can still be done easily, correct shadow casting on arbitrary geometries requires precise knowledge of the topology of the real environment. While this knowledge may be available in individual cases, depending on the AR application (e.g., for projection-based AR or phantom objects; see also Sect. 8.3 or Sect. 8.4.2), such information is often not readily available. The same applies to



Fig. 8.16 Reflective surfaces of virtual objects. (© TU Ilmenau 2018. All rights reserved)

reflections between real and virtual objects. To be able to reproduce them close to reality, at least basic information about the surfaces or normals of the real environment of the virtual object must be available. This information can either be derived from SLAM (Simultaneous Localization and Mapbuilding) methods (see Sect. 4.3.4), from the processing of depth camera images (see Lensing and Broll 2012), or more recently from LiDAR sensors now available in some tablets and smartphones. For example, ARKit uses Light Probes (see above) to realize glossy reflections. Instead of spheres, the physical objects used are the planar surfaces (or their illumination) detected by the framework. Unrecognized areas are initially black and are filled with color information using a neural network. Similar approaches can be found in Schwandt and Broll (2016) and Schwandt et al. (2018); see Fig. 8.16).

8.3 Visual Output

The visual output of the augmented content can be achieved using various devices. In Sect. 5.2.2, the use of head-mounted displays for AR has already been introduced. When using the video see-through technique (see Sect. 8.1.2), VR glasses are used, which capture reality via integrated or external cameras. In contrast, special AR glasses are used in order to provide a direct view on reality using the optical see-through technique (see Sect. 8.1.2) for augmentation. While AR glasses allow an immediate augmentation of the user's visual field and thus represent the most immersive form of AR, most current AR applications use handheld display devices (smartphones and tablets).

Besides visual output devices, AR usually only adds audio output devices. In mobile AR systems this is usually limited to stereo headphones, whereas stationary AR systems can use all kinds of audio output devices (see also Chaps. 2 and 3).

8.3.1 *Handheld Devices*

Due to the availability of corresponding AR frameworks for handheld devices (ARCore for Android and ARKit for iOS), handheld devices (tablet computers and smartphones) are currently the most important and most frequently used output devices for AR. They are equipped with a rear camera, which is used to capture the environment and for optical tracking, and mostly with sensors to detect the pose (see Sects. 4.2.2 and 4.2.3). Analogously to video see-through displays (see below), augmentation is usually performed correctly in terms of perspective for the position and orientation of the camera, but not for the actual viewing point of the observer.

The problem is mainly caused by the fact that the field of view of the camera used for video capturing is typically fixed in relation to the display, whereas the viewer's field of view depends on the respective point of view and the viewing direction in relation to the display (see Fig. 8.17).

As a result, the Magic Lens effect described above is only achieved to a limited extent (see Fig. 8.18). Due to this and the very small proportion of such displays in the viewer's field of view, the immersion is significantly lower.

8.3.2 *Projection-Based Output*

For realizing projection-based AR, one or more projectors illuminate surfaces of the environment in such a way that the perception of real objects changes (see Fig. 8.19). Due to this restriction to existing surfaces, no free positioning of the virtual contents in space is possible. For the correct projection of the virtual contents onto the real surfaces, the position and orientation of the individual projector in relation to the projection surface must be known. This can be achieved, for example, by determining the position and orientation of the projector. Additionally, a model of the objects to be projected onto must be available. If these are movable, they must also be tracked. If the additional content is now projected onto the virtual models (without having them illuminated otherwise) and the resulting rendered image is output by the projector, it is geometrically correctly registered (assuming the projector is correctly calibrated with respect to field of view, distortion, etc.). However, often the corresponding models of the environment are not available and/or tracking of the projector is not possible. In this case, other methods must be used to determine the depth information of the individual projection surfaces. For this purpose, approaches applying *structured light* (e.g., patterns like stripes or grids (see also Scharstein and Szeliski 2003)) can be used, as they are sometimes used in depth cameras. Due to

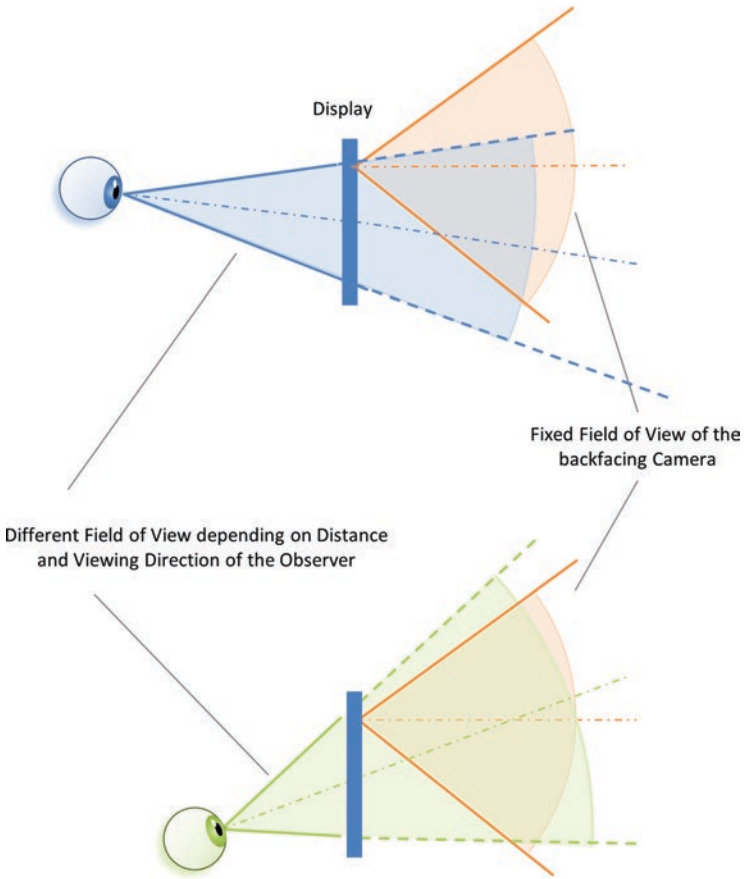


Fig. 8.17 Different fields of view of viewer and camera in handheld AR

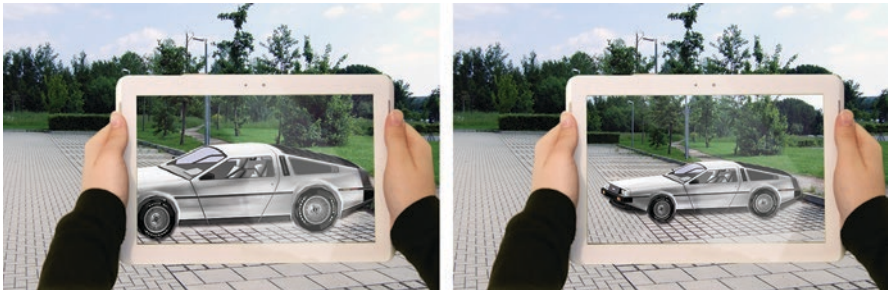


Fig. 8.18 Left: Matching perspective between reality and augmented image (Magic Lens effect). Right: Camera image and reality are perceived with a deviating perspective

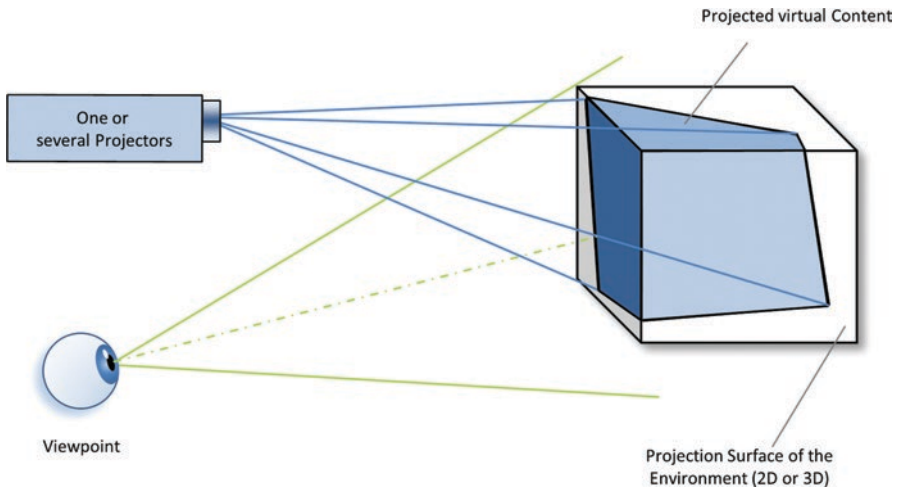


Fig. 8.19 Schematic structure of a system for projection-based AR

the projection by the projector, approaches with structured light only require the attachment of a camera to the projector. In this case, as with the attachment of other sensor technology, one often speaks of so-called *smart projectors*.

In addition to the geometric registration, photometric calibration (not to be confused with photometric registration; see Sect. 8.2.2) plays an important role in projection-based AR. Since projection surfaces are usually not ideal white diffuse surfaces, but rather the physical properties of the surface (structure, reflective properties, color, etc.) and the environment (brightness, shadows, highlights, etc.) influence the projected image, the resulting variations must be compensated by an appropriate photometric calibration. Of course, this is only possible within certain limits determined by the capabilities of the projector, the properties of the surfaces and the content to be projected. For a detailed discussion of the necessary calibration steps, please refer to the book by Bimber and Raskar (2005).

A special variant is rear projection on a screen or projection panel that is part of a real object. An example is a 3D dashboard in a driving simulator. Essentially, this corresponds to a 3D projection as used for VR (see Chap. 5). In contrast to VR, however, the projection surface itself is part of reality and its shape is modified by the projection.

8.3.3 Further Types of Spatial AR

Other forms of spatial AR (SAR) often use glass plates or foils as mirrors instead of projections onto surfaces to be augmented. Here, the user looks through the glass at the object to be augmented. However, since the glass is not perpendicular to the viewing direction, it acts as a mirror on a correspondingly placed display (see

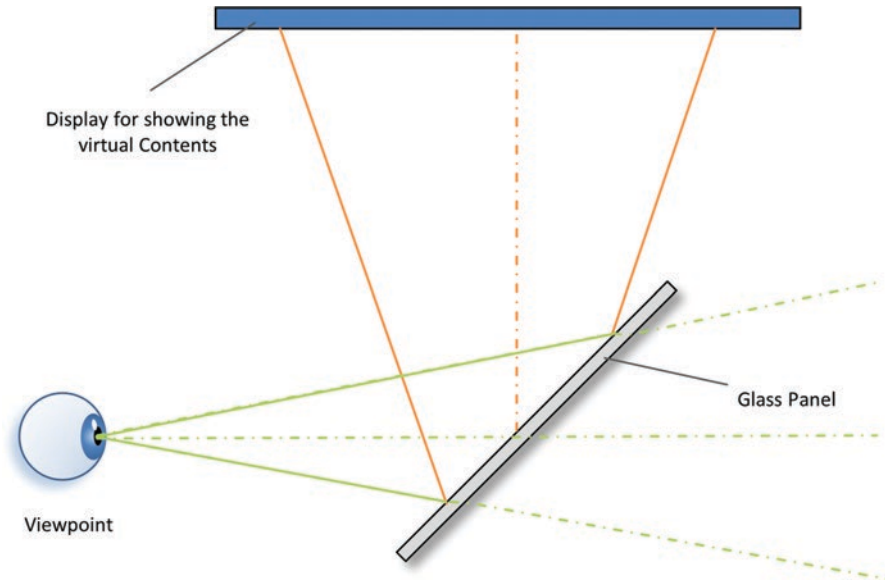


Fig. 8.20 Spatial AR using a transparent, reflective surface

Fig. 8.20). This results in a kind of optical see-through AR. For correct geometrical registration, the head position must be tracked as well. However, due to the fixed spatial arrangement of the components, the range of movement of the user is strongly limited. Therefore, only those applications where the user will usually only change her head position to a small extent are useful.

8.3.4 AR Mirrors

AR mirrors have in particular become popular to simulate the fitting of clothes. Here the viewer sees himself and his surroundings in a mirror. The mirror image is enriched by virtual content. Looking at the mirror image, the viewer gets the impression that the virtual content is part of the real environment. AR mirrors can be realized via the video see-through approach as well as by optical see-through (see Fig. 8.21). For VST, a camera is attached to a display that represents the mirror, capturing the environment or the user. This image is then mirrored, superimposed with virtual content and then displayed (Mottura et al. 2007; Vera et al. 2011). For OST approaches, a real, semi-transparent mirror is used (Fujinami et al. 2005; Li and Fu 2012). The environment is perceived in the same manner as with a conventional mirror, while the virtual content is represented by a display located behind the mirror. Only for the OST approach, are the real, mirrored objects automatically perceived stereoscopically. For correct localization of the virtual contents (in the case of VST also of the real contents) an additional technique for stereoscopic viewing is required (see also Sect. 5.4.4).

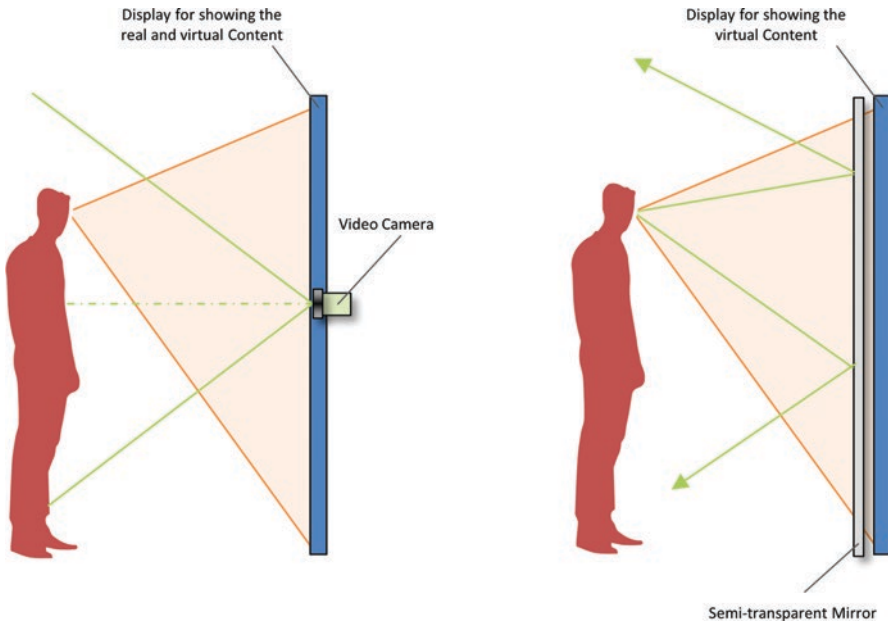


Fig. 8.21 VST vs. OST AR mirrors

8.4 Special AR Techniques

In this section, a number of techniques found in AR applications will be discussed.

8.4.1 Head-Up Content

Head-up content, sometimes called a dashboard, refers to content that is displayed regardless of the position and orientation of the viewing direction. Typical examples are status displays or environment maps. Widely used in 3D games played from the first-person perspective, these techniques are sometimes also found in virtual reality applications. Here the position and orientation of the content are always unchanged in relation to the display. Often such content is just 2D objects, but 3D objects are also used, having a corresponding spatial position in front of the observer's view-point, although this is rarely the case. An example of 3D content would be (again in the tradition of games) the representation of the (apparently in the hand) weapon held by the user, known from first-person shooters.

8.4.2 Occlusions and Phantom Objects

As soon as real objects are closer to the observer than the virtual objects behind them, the perception and behavior of the virtual object no longer match. The reason for this is that the virtual content is always visible due to the optics of the HMD (in the case of optical see-through displays) or the superimposition of the video image (in the case of video see-through AR). For the observer this results in a conflict (which of the two objects is actually closer) that cannot be resolved. Hence, it immediately destroys the impression of the correct location of the virtual object in reality (see also depth cues in Chap. 2). To prevent this, the real objects, with respect to the image areas covered by them, which actually should occlude the virtual objects lying further away (*occluders*), need to be identified. This then allows for proper masking and hence removal of those areas of the virtual objects that should not be visible. The actual recognition and, if necessary, localization of the covering real objects can be accomplished in different ways. The most common case is occlusion by the hands of the user, because they are typically closer to the point of view than most virtual objects. Cameras mounted on a handheld device or on HMDs can be used to identify the corresponding image parts (e.g., based on color segmentation). The virtual contents can be masked at the appropriate places so that they are apparently occluded by the hands. In the case of other real objects that could potentially occlude virtual content, we distinguish between static and moving objects. While the position of static objects can be determined in advance, the position and orientation of moving objects may have to be tracked. In both cases, the objects must be available as virtual objects and must be integrated into the virtual scene at the corresponding position (i.e., correctly located, if necessary based on corresponding tracking data). Since these virtual objects should not be rendered, but only serve to correctly occlude other virtual content, they are called *phantom objects*. For correct visualization, phantom objects are rendered as black, unlit objects in the case of optical see-through AR. In places where such a phantom object is closer to the viewer than another virtual object, the content of the frame buffer is replaced by a completely black pixel. Since black pixels appear transparent in optical see-through AR, the viewer ultimately sees only the real object here (see Fig. 8.22).

In the case of video see-through AR, the procedure does not work in this way, because the black object surfaces would stand out from the video background. In this case, the phantom objects must therefore be rendered in a separate pass before all other objects. However, only the *depth buffer* must be modified accordingly. Therefore, virtual objects behind those areas do not affect the frame buffer in further render passes and the underlying video image thus remains visible. If there is no possibility of inserting the occluding real objects as 3D models or to detect their correct position and location, or if the objects are not rigid objects, such as a person who is sometimes partly in front and partly behind the virtual objects, phantom objects cannot be used. The only way to achieve correct masking in this case is to acquire or calculate the depth information for the field of view. In principle, this can be done with two cameras, but it is easier to use *depth cameras (RGBD cameras)*.



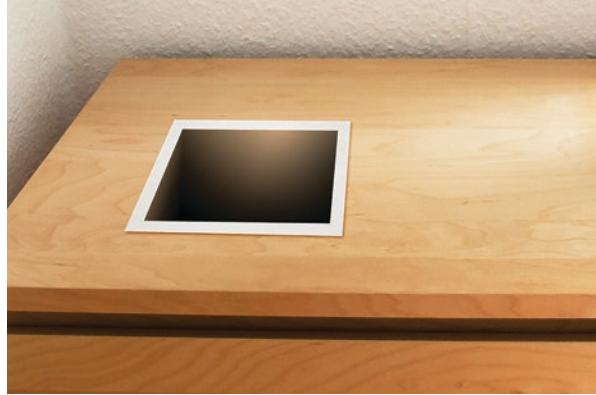
Fig. 8.22 Phantom objects enable correct mutual occlusion between real and virtual objects. Without a phantom object, the virtual object seems to float in front of the real objects, whereas with correct masking by the phantom object it seems to be behind the real objects

After transformation to the perceived image segment, the image pixels can be set directly according to the procedures described above to guarantee correct occlusion. Due to the currently still quite low resolution and quality of most depth cameras, determination of the boundaries usually does not achieve the same precision as when using phantom objects.

8.4.3 *Crossfading Markers*

Due to their straightforward applicability, markers are still used for some AR applications, despite the alternatives that are meanwhile available. However, due to their clear distinguishability from the rest of the environment, which is important for tracking, they often appear to the viewer as particularly disturbing foreign objects. While the virtual content based on the position and location of the markers is usually superimposed on the image above them, the markers themselves often remain clearly visible in the background. A simple and effective way to remove disturbing markers from the displayed scene (as long as they are not completely covered by virtual objects anyway) is to superimpose them with a simple flat *camouflage object*. This can visually match the surrounding background. However, since this background is often not known in advance, the option of a neutral object for covering should always be considered as well, as it is perceived as less disturbing in almost all cases. However, there are also AR applications that specifically cover markers with virtual objects, which in turn look exactly like these markers. This can be used, for example, to give the impression that the real marker can be removed, deformed, or otherwise modified, for example to reveal the view of an underlying (virtual) hole (see Sect. 8.4.4). This often leads to surprising reactions on the part of the viewer, since he or she (especially with video see-through AR) often cannot initially recognize the difference between the real and virtual markers.

Fig. 8.23 By superimposing a marker on the table surface with a virtual object that has a deepening extending through the tabletop, the viewer gets the impression of an actual hole



8.4.4 *Virtual Holes*

It is often overlooked that Augmented Reality can also be used to apparently take away parts of the real environment by adding virtual content. For example, it is very easy to model a virtual hole that extends through the floor or a tabletop (see Fig. 8.23), or to make an object such as a cube appear hollow. For a correct representation the real parts around the virtual hole must be modeled as phantom objects. Since the observer can then (apparently) look more or less into the deepening, depending on the angle of view, the effect is usually much more amazing than a mere augmentation of reality. Although parts of reality are apparently removed, this is usually not referred to as Diminished Reality (see Sect. 8.7.1), since the virtual three-dimensional hole overlays reality, i.e., augments it, and does not allow a view of the floor beneath the table top.

8.4.5 *X-Ray Vision*

AR can also be used to look through, or at least into, solid objects, such as walls, as if looking through them with *X-ray vision*. Typical applications are the visualization of pipes and cables in walls (see Fig. 8.24) or under the pavement of streets. Other applications are in road traffic (viewing through the truck in front, through the forest to the road behind the bend, through the building at the street corner, etc.) and in the military application context (visualization of own and foreign units in buildings, in the forest, under water, etc.). As far as virtual content inside or behind real structures is just visualized without removing the real content, this is not considered Diminished Reality (c. also Sect. 8.7.1). However, the transition here is almost seamless.

In this context, occlusion is another problem. As already discussed in the context of phantom objects, the fusion of reality and virtuality is rendered effectively impossible due to incorrect depth perception (see also Chap. 2) and the resulting conflict. There are various approaches to resolve this conflict or at least to significantly



Fig. 8.24 Pipes and cables behind a cover and in the wall are made visible using AR for X-ray vision. (© Leif Oppermann, Fraunhofer FIT 2018. All rights reserved)

reduce its effects on perception. A simple option is the use of virtual holes or trenches (see Sect. 8.4.4) with corresponding phantom objects for the surrounding (occluding) surfaces. This way, the objects inside or underneath an object are only visible within the (virtual) hole, so they remain correctly registered in relation to the real surface. Another possibility is to reproduce the real surfaces by visible but partially transparent virtual objects. In this case, these objects do not serve as phantom objects but suggest a partially transparent surface to the observer. A similar effect can also be achieved using spatial AR by projecting hidden (e.g., inner or underlying) structures onto existing surfaces (see Fig. 8.25). If a part of reality is apparently taken away (e.g., by a previously taken picture), the perceived effect is quite comparable to that of Diminished Reality (see also Sect. 8.7.1). A sharp distinction from the latter is then sometimes difficult, although Diminished Reality would require a real-time generation of the view (in the simplest case e.g., by a camera).

8.5 Special AR Interaction Techniques

In principle, most interaction techniques from VR may also be used in AR environments (see Chap. 6). However, it should be noted that the aspect of user interaction in AR applications is still clearly underdeveloped, with many AR applications still focusing on visualization aspects.

Fig. 8.25 Use of SAR to represent hidden parts of reality



8.5.1 *Interaction by Navigation*

With AR, the user navigates through his movement in reality, i.e., an unintentional decoupling between the real movement and the virtual movement, which often occurs in VR environments, is not possible here. However, since AR content is inevitably closely related to reality, the interaction with virtual content is in turn mostly bound to its physical proximity. This means that an interaction is only possible when the user is at a certain location, partly additionally restricted by the user having to look in a certain direction so that the objects are within her field of view. In outdoor AR applications especially, an explicit selection of virtual objects is often omitted and only a simple user action (for example, a keystroke or voice command) is used instead. In some cases, even this is waived, i.e., an interaction occurs when the observer simply approaches the virtual object.

8.5.2 Gaze-Based Interaction

In AR applications, the selection of objects or menu items is different and often more difficult than in VR environments, where corresponding input devices are usually available for hand-operated control. Eye-tracking, i.e., the detection of the point currently focused on by the viewer, represents a promising approach for realizing gaze-based interaction, but sufficiently precise mechanisms require either the integration of appropriate sensor technology into the AR glasses (as, e.g., used by the HoloLens 2 or Magic Leap One) or the simultaneous use of the backward pointing camera of smartphones or tablets (selfie camera) as well as an appropriate calibration (see Sect. 4.5). The touch controls available on handheld devices can alleviate this problem, but at the price of massively obscuring the already quite limited field of view.

A simpler but robust form of this selection mechanism without the requirements of eye tracking can be achieved by using the orientation of the head (for AR glasses) or the camera (for handheld devices) instead of the actual line of sight, since it has to be tracked anyway. By aligning the orientation so that the object to be selected comes into the center of the field of view (often supported by a corresponding visual marker, e.g., a crosshair), simple and fast selection is possible, which only needs to be supplemented by a trigger action. To avoid additional input mechanisms (like language or a button press), a *dwelt time* is often used here. When the selection sticks to an object for a certain time, the corresponding action is triggered. A drawback in this context is that even for experienced users this prevents faster operation.

8.5.3 Tangible User Interfaces

Tangible User Interfaces (TUI) (Ullmer and Ishii 2000; Azuma 1997), or *Tangibles* for short, are a tangible form of user interface. Here, real objects in the user's environment are linked with virtual objects in such a way that the state of the real object (placeholder object or *proxy*) is mapped to the state or a property of the virtual object. In the context of AR user interfaces, one can distinguish between a direct and an indirect form of use. In the direct form, the physical properties of a real object correspond directly to those of a virtual object. This is rather the rule in AR environments, since this is already the case when a virtual object is displayed on top of a marker and this marker can be moved by the user (see Fig. 8.26). Here, for example, the position and orientation of the real object and its virtual counterpart correspond. However, the approach is not limited to markers, but can be extended more generally to any objects whose properties can be captured and transferred to the corresponding properties of a virtual object.

In the indirect form of a TUI, however, the physical properties of the real object are mapped to the other attributes of one or more virtual objects. A simple example here would be a (real) cube whose position affects the color or size of a virtual



Fig. 8.26 Tangible User Interfaces: a real proxy object is used to interact with a virtual object

object. Ultimately, the interaction techniques possible in the context of AR are only limited by the possibilities to capture the physical properties of real objects. Further examples are a real pen, through which virtual writing is applied, or a real spray can for virtual graffiti as well as an orange as a real representative of a virtual ball.

8.6 AR Applications

AR applications are manifold and, due to their often mobile or at least nomadic usage (see also Sect. 5.2) and their use on a large number of widely used mobile devices, they now clearly exceed those of VR. The following compilation of AR application areas can therefore only represent a selection giving an impression of the variety of possible applications.

Training and Maintenance

The training of workers in the installation of wiring harnesses in aircraft at Boeing was the first known use of AR in a commercial environment. In the area of training

and maintenance, AR provides assistance by displaying appropriate hints, directions, etc., for the execution of work steps until they have been sufficiently learned. Furthermore, AR can also be used in the area of maintenance. This is particularly useful in cases where systems are extremely complex and many different variants exist, so that an individual cannot be sufficiently trained for all cases that occur. This occurs, for example, with cars and airplanes, as well as large machines and industrial plants. AR can support necessary work processes by visualizing the work steps and, if applicable, the required tools and spare parts (see also Chap. 9).

Television Broadcasting

One of the best known fields for the application of AR, which at the same time is the least associated with it, is the overlay of auxiliary information, especially in sports broadcasts. It is now state of the art that virtual help lines are drawn into television pictures in the correct perspective for sports such as soccer, American football and ski jumping, so that the viewer sees distances, offside positions or world records directly in the context of the current situation.

Military Applications

AR has been used for many years in the helmets of fighter pilots. Graphics are typically restricted to line graphics. However, in particular for mobile units, AR offers new possibilities to combine information based on the knowledge of other units with reconnaissance data (from satellites, drones and airplanes) as well as terrain information into the visual field, depending on the individual viewing position and direction. Although recent developments can only be speculated about due to military confidentiality, the lack of full daylight capability of optical see-through displays is probably one of the main obstacles to widespread use. It should be noted that Smartglasses are often used in the military context. However, as information displayed there is not really geometrically registered to the environment in 3D, this should not be considered as AR.

Teaching, Education and Museums

In the field of teaching and learning, AR opens up completely new possibilities for teaching complex contexts. Physical as well as macroscopic or microscopic experiments, which are otherwise often only taught through literature and video material, can be experienced interactively with AR. This increases understanding sustainably. Similarly, AR can be used in science centers and technical museums to explain effects directly at the exhibit instead of separating exhibit and explanation.

Architecture and Urban Planning

While in the field of architecture and urban planning real models and sophisticatedly rendered films still dominate in large projects, the use of AR allows you to get a picture of future buildings or urban development changes on site, taking into account the real environment.

Medicine

In the medical field, AR is particularly suitable for supporting surgical procedures, especially in the minimally invasive area. By combining different measurement data

(camera images, X-rays, previous model data from nuclear spin tomography, etc.), information that is otherwise only available separately can be displayed in parallel and in the correct perspective in the surgeon's field of vision. However, so far, AR has been used mainly in the field of education and training.

Information, Navigation and Tourism

With the spread of powerful smartphones and tablets, AR-enabled handheld devices are available to a wide range of users in virtually any location. This makes it possible to display general information, navigation instructions or descriptions of tourist attractions directly on top of the current video image. Often, however, the content here is limited to text, images and graphic symbols, where precise registration is not required. A prominent example is Google Maps' "Live View" mode.

Archaeology and History

AR here allows us to virtually complete buildings and objects that are only partially preserved and thus show the viewer the former state in context. Another possibility is the addition of further buildings or other objects or persons important for the historical context of a scenario. The augmentation here does not have to be limited to visual impressions, but usually conveys these impressions more easily if further senses are also addressed. It is sometimes the case that buildings and places still exist today, but their appearance has changed over time. Here, AR can be used to superimpose the former appearance on the present one.

Games and Entertainment

Since the recent hype about Pokémon Go, AR games have become known to the general public. However, a real breakthrough in the distribution of AR games came with the availability of corresponding development frameworks from 2017 onwards. ARKit and ARCore made it possible for game developers to create AR games for the most important mobile platforms (iOS and Android) with relatively little effort. Meanwhile hundreds of AR games already exist in the respective online stores.

In-Car AR

Many people associate AR in cars with Head-up Displays (HUD) (see also [Sect. 9.3](#)). However, as these displays do not yet have the capability to provide individual images to each of the driver's eyes, they do not allow stereo vision. Thus, correct geometric registration of the virtual content is not possible, which can be critical, e.g., when visually highlighting lane boundaries. Stereoscopic HUDs are currently still under development and it will probably be a few more years before we see them in standard vehicles. They will enable augmentation of other cars, pedestrians or traffic signs, as well as the perspective-correct projection of navigation instructions onto the road. Currently, however, AR is already being used almost universally in conjunction with rear-view cameras, for example to display lanes depending on the steering angle.

8.7 Diminished and Mediated Reality

Of course, Diminished and Mediated Reality are not AR (see Fig. 1.6). Thus, why is there a section on them within the AR chapter of this book? As we have seen throughout this chapter, it can sometimes be difficult to tell the difference. For this reason, we will take a closer look at these concepts and show how they are related to AR with respect to how AR is part of them.

8.7.1 Diminished Reality

As already introduced in Chap. 1, *Diminished Reality (DR)* refers to the removal of parts of reality (see Fig. 8.27). For this purpose, an area – usually a specific object – is removed in real time from the view of the observer. This is done in such a manner that the compiled view provides a view of otherwise not visible content. Basically, one can distinguish between two different types of Diminished Reality: approaches that attempt to reconstruct the actual real background and approaches that merely create a plausible overall impression, i.e., showing some alternative content for the object removed.

Retrospective removal of persons from pictures has a long tradition – almost a century. Nowadays, context-sensitive or context-aware filling is a standard function in image editing software and therefore simplifies this process drastically in most cases. However, removing objects, buildings, backgrounds and people from videos is still relatively new. In motion picture productions especially, image areas have been and still are processed frame by frame to remove cameras, microphones,



Fig. 8.27 Example of diminished reality: the sink drain is removed from the live video stream in real time. (© TU Ilmenau 2018. All rights reserved)

holding ropes, etc. Automated removal of content from videos was shown by Wexler et al. (2007) and for more complex scenes by Granados et al. (2012). However, due to their computational time of up to several days in some cases, these approaches were not suitable for real-time representation and thereby Diminished Reality. The first approaches to real Diminished Reality were based on reconstructing the background of an obscuring object by further views, usually by additional cameras, using homographies (Zokai et al. 2003; Enomoto and Saito 2007). The areas covered by the objects to be removed are identified in other views and then transformed into the view to be diminished (see Fig. 8.28).

Another application area for DR was the removal of tracking markers (see Sect. 8.4.3). Since these markers are often perceived as disturbing for the observer, the image area covered by them was superimposed by a background texture (Siltanen 2006; Kawai et al. 2013). The approaches of Herling and Broll (2010, 2014) are among those that do not attempt to reconstruct the real background, but only a plausible one. In this approach, an object to be removed is first marked and then tracked in each frame to find its silhouette in the current image. This silhouette is then masked and filled using context-sensitive filling (see Fig. 8.29).

It is crucial that tracking and filling are done in real time and that coherence is ensured not only with respect to the surrounding image areas, but also with respect

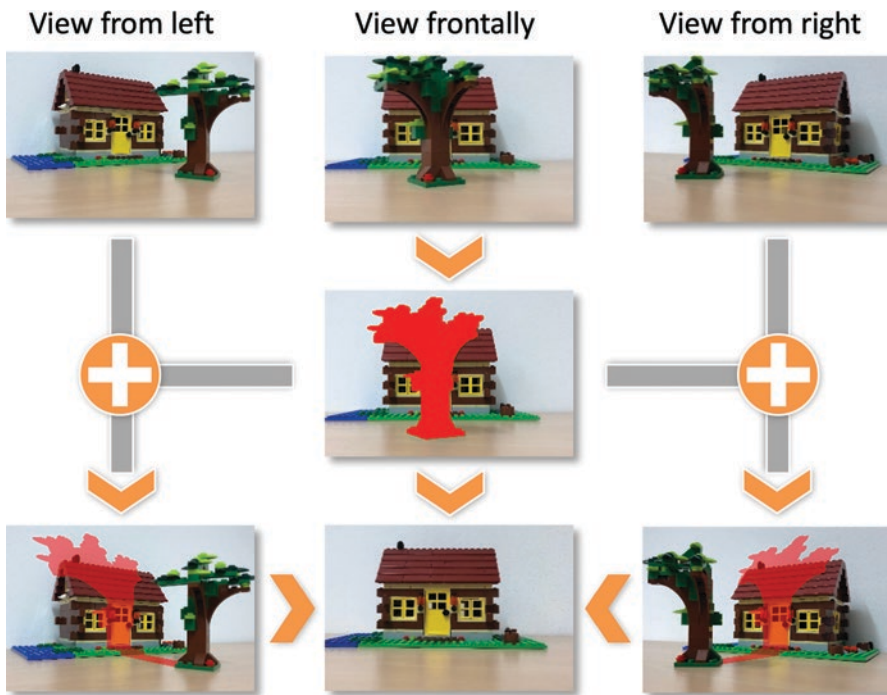


Fig. 8.28 Reconstruction of the real background using several camera views. (Individual images: © TU Ilmenau 2018. All rights reserved)

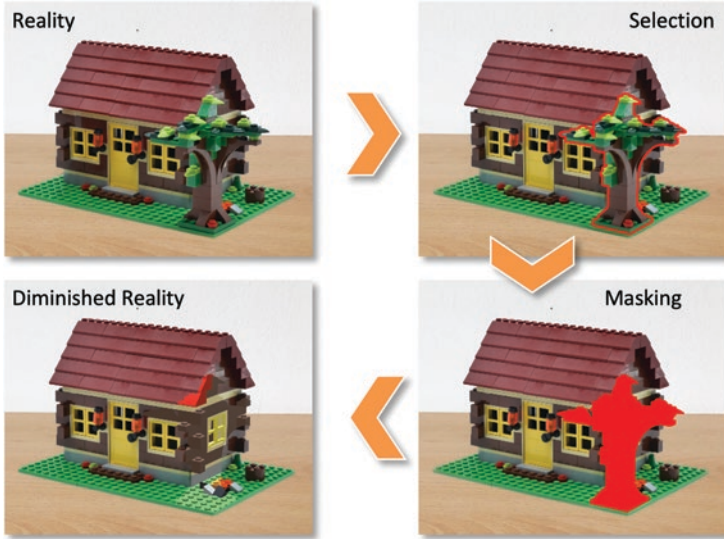


Fig. 8.29 Diminished reality by masking and context sensitive filling. (Individual images: © Tobias Schwandt, TU Ilmenau 2018. All rights reserved)



Fig. 8.30 Using information from the surrounding frame areas to fill the masked area. (© Jan Herling 2013. All rights reserved)

to the previous frame. This is possible by a randomized approach, which randomly selects image patches from the surrounding image areas (see Fig. 8.30) and optimizes those selections using a combination of several cost functions, which can be taken as a measure of incoherence. Due to the required real-time capability, typically not an optimal but only a sufficiently good solution is found. Since the calculation is based exclusively on 2D image data (current and previous frame), the approach for removing 3D structures is problematic. It may happen that a previously hidden real background becomes visible by the camera movement and is not coherent with the previously synthetically assembled content. Due to the

homography used here, the approach basically only works if the background is on a single plane. Kawai et al. (2016) extended the approach to several planes, making it possible to remove objects on walls or in corners of rooms, for example. However, this does not allow the removal of arbitrary 3D backgrounds either. Thus, more recent approaches (Kunert et al. 2019; Mori et al. 2020) are based on a 3D reconstruction of the real environment.

It can be assumed that deep learning approaches will increasingly be used for advanced forms of DR in the future (Kido et al. 2020). This will make it possible to create a plausible and coherent overall picture even without information available in the immediate vicinity of the object to be removed, especially when information is included that is not available in the original video stream.

8.7.2 *Mediated Reality*

As already introduced in Chap. 1, Mediated Reality involves altering the perception of reality in any form (Man 2001). This means in particular that both AR and Diminished Reality are partial aspects of Mediated Reality. However, Mediated Reality is in principle not limited to adding or removing content. Rather, it also enables the replacement of parts of reality. Finally, we can define a Mediated Reality continuum in analogy to Milgram's Reality-Virtuality Continuum. In contrast to the former, however, it has two dimensions: one dimension where reality is increasingly replaced by virtuality, and a second dimension where reality is increasingly removed (diminished) (see Fig. 8.31).

If, for example, you want to see what a suite would look like in your own living room, the furnishing apps already available using AR are often of little help here, as they merely project the selected new furniture additionally into the current environment. However, since the existing furniture usually must remain in the room (Who wants to clear it out for this purpose?), the resulting overall impression is often unsatisfactory. If, however, the existing furniture is first removed virtually using Diminished Reality and then the new furniture is inserted using AR in a second step, the overall impression corresponds much more to the users' expectations (see Fig. 8.32). The approach can be directly extended to other areas, such as new buildings in existing developments or the renewal of a machine in a factory.

Beyond that, Mediated Reality also allows us to change reality directly. The first examples of this have already been presented in the section on photometric registration (see Sect. 8.3), where parts of reality were changed in their illumination by virtual light sources, reflections from or on virtual objects, and shadows or caustics of virtual objects. However, changes can also affect the geometry of reality, which is modified in its spatial position or structure (see Figs. 8.31 and 8.33).

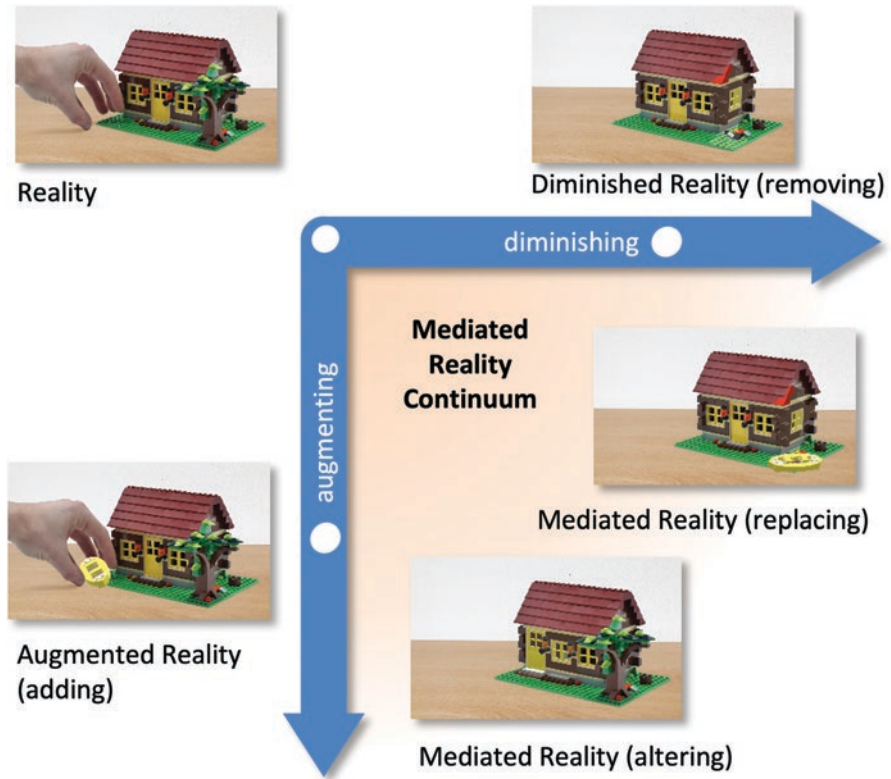


Fig. 8.31 The mediated reality continuum: reality may be augmented (AR) by adding virtual content as well as diminished (DR) by removing real content. In combination, this allows both the replacement of real content with virtual content as well as its modification (Mediated Reality). (Still images: © Tobias Schwandt, TU Ilmenau 2018. All rights reserved)



Fig. 8.32 Combining diminished reality and AR. (© Christian Kunert, TU Ilmenau 2018. All rights reserved)

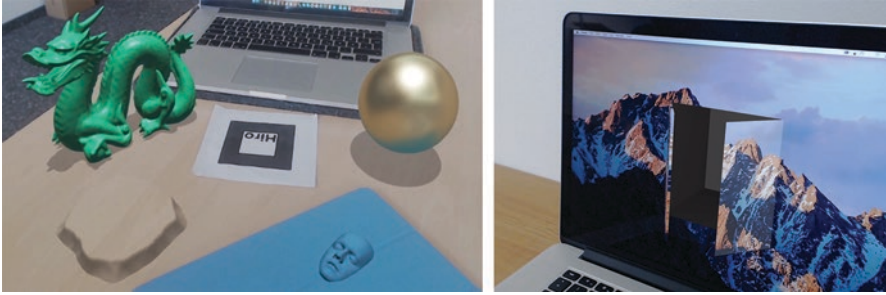


Fig. 8.33 Examples of changing real geometry using Mediated Reality. Left: virtual elevation of the tabletop and face on tablet. (© Tobias Schwandt, TU Ilmenau 2018. All rights reserved.) Right: virtual door in the monitor. (© Jan Herling 2017. All rights reserved)

8.8 Summary and Questions

Augmented Reality combines VR technologies with reality and thus enables users to seamlessly integrate virtual content into their natural (real) environment. Most AR applications are nowadays created on mobile handheld devices such as smartphones and tablets, as they already come with the necessary hardware (including appropriate sensor technology) and software. However, these are limited to video see-through AR. Optical see-through AR (based on HMDs) as well as spatial AR are further possibilities for augmenting the environment. The tracking methods used are in part similar to those used for VR, but here too the focus is very clearly on mobile use and therefore on a combination of sensors for position measurement and usually camera-based approaches. Crucial for the impression of seamless fusion between virtuality and reality is the correct registration of virtual content in the real environment. This must be done with respect to their position and orientation (geometric registration), but also with respect to correct illumination (photometric registration). While many VR interaction techniques can basically be used in AR applications, otherwise rather simple techniques (such as gaze-based selection) or techniques involving reality (such as tangibles) are used here, since the users must still (in parallel) continue to act in reality. In contrast to VR, AR can be used almost always and everywhere. On the one hand, this opens up a wide range of possibilities, but on the other hand it is also one of the biggest challenges, since AR systems have to work in very different environments.

With Diminished Reality, parts of reality can be specifically removed in real time. Although this functionality is not yet supported in commercial software, it is foreseeable that it will become available within the coming years, especially with regard to its use for Mediated Reality.

After working through the chapter, you can check your knowledge by answering the following questions. The questions are sorted by topic.

Magic Lens

- What is the Magic Lens metaphor and how does it relate to AR?
- What are the limitations of handheld AR and why?
- How could these limitations be mitigated or circumvented?

Registration

- What is the difference between tracking and registration?
- What is meant by geometric registration, and what by photometric registration?
- How are those realized?
- Which one is unidirectional and which one is bidirectional, and why?
- What are the effects on the user experience of an incorrect geometric or photometric registration?

Visualization

- Which AR techniques can be used to create shadows of virtual objects?
- Should I use an OST or VST display in bright sunlight outside?
- Does this assessment change if the view of the (real) environment can be critical to safety (construction site, road traffic, etc.)?
- What aspects need to be considered for seamless visual integration of virtual objects into the real environment?

Tangible User Interfaces

- What do you understand by Tangible User Interfaces?
- Give an example of their usage, applying both direct and indirect interaction techniques.
- Are Tangible User Interfaces also suitable to be used in VR? Why or why not?

Occlusion Handling

- Why do you need phantom objects for AR?
- Why do they have to be realized differently depending on the type of augmentation?
- What is the consequence of missing or faulty phantom objects?
- What is the relationship between phantom objects and virtual holes?

Diminished Reality/Mediated Reality

- What are the two fundamental approaches to Diminished Reality? Why do they only make sense for certain scenarios?
- An application allows you to remove the facade of a real building and thus to have a look inside. Is this an AR or a Mediated Reality application?
- You would like to create a “Good Weather App” based on Mediated Reality, in which the sky is always blue and sunny during the day and full of stars at night. How would you proceed?

Recommended Reading

- Bimber O, Raskar R (2005) Spatial augmented reality: merging real and virtual worlds: a modern approach to augmented reality. AK Peters. The book gives a comprehensive overview about Spatial AR
- Furt B (2011) Handbook of augmented reality. Springer, New York. A collection of articles on various topics of AR, covering both the technical aspects and the application side. It is advisable to check beforehand which articles are of interest to the reader and to purchase only these
- Szeliski R (2011) Computer vision: algorithms and applications. Springer. A must for all who are concerned with camera-based procedures, be it camera calibration or camera-based tracking
- Schmalstieg D, Höllerer T (2016) Augmented reality: principles and practice. Addison Wesley. The book gives a comprehensive overview of augmented reality, including a detailed description of computer vision techniques for AR tracking

References

- Azuma R (1997) A survey of augmented reality. *Presence Teleop Virt* 6(4):355–385
- Bimber O, Raskar R (2005) Spatial augmented reality: merging real and virtual worlds: a modern approach to augmented reality. AK Peters
- Brown LD, Hua H (2006) Magic lenses for augmented virtual environments. *IEEE Comput Graph Appl* 26(4):64–73
- Debevec P (1998) Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In: *Proceedings of the 25th annual conference on computer graphics and interactive techniques*, pp 189–198. <https://doi.org/10.1145/280814.280864>
- Enomoto A, Saito H (2007) Diminished reality using multiple handheld cameras. In: *ACCV'07 workshop on multi-dimensional and multi-view image processing*, vol 7, Tokyo, pp 130–135
- Fujinami K, Kawsar F, Nakajima T (2005) AwareMirror: a personalized display using a mirror. In: *International conference on pervasive computing*. Springer, Berlin/Heidelberg, pp 315–332
- Granados M, Tompkin J, Kim K, Grau O, Katuz J, Theobalt C (2012) How not to be seen – inpainting dynamic objects in crowded scenes. *Eur Secur* 31:219–228
- Hamasaki T, Itoh Y (2019, May) Varifocal occlusion for optical see-through head-mounted displays using a slide occlusion mask. *IEEE Trans Visual Comput Graph* 25(5):1961–1969. <https://doi.org/10.1109/TVCG.2019.2899249>
- Herling J, Broll W (2010) Advanced self-contained object removal for realizing real-time diminished reality in unconstrained environments. In: *Proceedings of IEEE ISMAR 2010*, pp 207–212
- Herling J, Broll W (2014) High-quality real-time video inpainting with PixMix. *IEEE Trans Vis Comput Graph* 20(6):866–879
- Kautz J, Sloan P-P, Snyder J (2002) Fast, arbitrary BRDF shading for low-frequency lighting using spherical harmonics. *13 EG workshop on rendering*, Eurographics
- Kawai N, Yamasaki M, Sato T, Yokoya N (2013) Diminished reality for AR marker hiding based on image inpainting with reflection of luminance changes. *ITE Trans Media Technol Appl* 1(4):343–353

- Kawai N, Sato T, Yokoya N (2016) Diminished reality based on image inpainting considering background geometry. *Trans Vis Computer Graphics* 22(3):1236–1247
- Kido D, Fukuda T, Yabuki N (2020) Diminished reality system with real-time object detection using deep learning for onsite landscape simulation during redevelopment. *Environ Model Softw* 131:104759
- Knorr SB, Kurz D (2014) Real-time illumination estimation from faces for coherent rendering. In: *Proceedings of IEEE ISMAR 2014*, pp 113–122
- Kunert C, Schwandt T, Broll W (2019) An efficient diminished reality approach using real-time surface reconstruction. In: *Proceedings of CYBERWORLDS 2019*, pp 9–16
- Lensing P, Broll W (2012) Instant indirect illumination for dynamic mixed reality scenes. In: *Proceedings of ISMAR 2012*, pp 109–118
- Li WHA, Fu H (2012) Augmented reflection of reality. In: *ACM SIGGRAPH 2012 Emerging Technologies*, Article no. 3
- Man S (2001) Mediated reality. *Linux Journal*, Article no. 5 1999(59)
- Milgram P, Takemura H, Utsumi A, Kishino F (1995) Augmented reality: a class of displays on the reality-virtuality continuum. In: *Photonics for industrial applications*. International Society for Optics and Photonics, pp 282–292
- Mori S, Erat O, Broll W, Saito H, Schmalstieg D, Kalkofen D (2020) InpaintFusion: incremental RGB-D inpainting for 3D scenes. *IEEE Trans Vis Comput Graph* 26(10):2994–3007
- Mottura S, Greci L, Travaini E, Viganò G, Sacco M (2007) MagicMirror & Foot-glove: a new system for the customized shoe try-on. In: *The future of product development*. Springer, Cham, pp 441–450
- Scharstein D, Szeliski R (2003, June) High-accuracy stereo depth maps using structured light. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2003)*, vol 1, Madison, WI, pp 195–202
- Schwandt T, Broll W (2016) A single camera image based approach for glossy reflections in mixed reality applications. In: *Proceedings of IEEE ISMAR 2016*, pp 37–43
- Schwandt T, Kunert C, Broll W (2018) Glossy reflections for mixed reality environments. In: *Proceedings of CYBERWORLDS 2018*, pp 138–143
- Siltanen S (2006) Texture generation over the marker area. In: *Proceedings of IEEE ISMAR 2006*, pp 253–254
- Szeliski R (2011) *Computer vision: algorithms and applications*. Springer
- Ullmer B, Ishii H (2000) Emerging frameworks for tangible user interfaces. *IBM Syst J* 39(3–4):915–931
- Vera L, Gimeno J, Coma I, Fernández M (2011) Augmented mirror: interactive augmented reality system based on Kinect. In: *Proceedings of IFIP human-computer interaction*. Springer, Berlin/Heidelberg, pp 483–486
- Weidner F, Broll W (2019) Exploring large stereoscopic 3D dashboards for future automotive user interfaces. In: *Proceedings of AHFE*. Springer, Cham, pp 502–513
- Wexler Y, Schechtman E, Irani M (2007) Space-time completion of video. *IEEE Trans Pattern Anal Mach Intell* 29(3):463–476
- Zokai S, Esteve J, Genc Y, Navab N (2003) Multiview paraperspective projection model for diminished reality. In: *Proceedings of IEEE/ACM ISMAR 2003*, pp 217–226

Chapter 9

VR/AR Case Studies



Ralf Doerner, Alexander Tesch, Axel Hildebrand, Stephan Leenders, Tobias Tropper, Wilhelm Wilke, Christian Winkler, Julian Hillig, Alec Pestov, James A. Walsh, Bruce H. Thomas, Gerhard Kimenkowski, Stephen Walton, Torsten W. Kuhlen, Geert Matthys, Holger Regenbrecht, Chris Heinrich, Xiumin Shang, Marcelo Kallmann, Benjamin Lok, Francisco A. Jimenez, Cheryl Wilson, Marc Erich Latoschik, Carolin Wienrich, Silke Grafe, Mario Botsch, and Jonny Collins

Abstract This chapter is a collection of selected VR/AR case studies from academia and industry.

9.1 Introduction and Overview

Ralf Doerner

For the conception of applications in VR and AR, a large design space exists with an unmanageable number of conceivable realization alternatives. The large number of available input and output devices alone, which are themselves available in different variants and which can be combined in different ways, makes a systematic analysis and evaluation of all implementation alternatives difficult. This is especially true since a sufficient theoretical foundation for such an analysis is not available today. Therefore, case studies in the sense of best practices provide a good orientation. VR/AR designers often take existing successful case studies as a starting point for the initial conception. In case studies, one can see how different technologies interact and how interaction techniques can be selected and adapted for the technical circumstances in a meaningful way. Case studies are an important source of experience. Since most VR and AR applications today are “one-offs” for a specific VR/AR setup and a specific application goal, one cannot consult any standards, but one can try to benefit from the experiences of previous successful applications.

Dedicated website for additional material: vr-ar-book.org

R. Doerner (✉)
RheinMain University of Applied Sciences, Wiesbaden, Germany
e-mail: ralf.doerner@hs-rm.de

This chapter contains a selected collection of case studies. On the one hand, they illustrate the basic principles of VR and AR taught in the other chapters and show examples of how virtual worlds have actually been realized or reality has been enhanced with virtual content. On the other hand, they provide an insight into how case studies can serve as a basis or inspiration for the development of future applications with VR/AR. Each case study is self-contained. Since the context in which case studies were created is also of interest, each case study not only mentions the authors directly, but also the organization or company in which the case study was created.

The first case study shows that the use of VR in certain applications, such as the construction of automobiles in the automotive industry, is already very well established. VR and AR are therefore not only something that researchers in academia are dealing with in the prototype stage but something that is being used in a commercial environment. In the assessment of *Technology Readiness Levels* (TRLs), as defined in the ISO 16290:2013 standard (ISO 2013), the maturity of VR/AR technologies today comprises all stages from basic technology research to system test, launch and operations. The next three case studies provide further examples of successful commercial use of VR/AR in different application domains, such as entertainment/infotainment, life sciences and diagnostics, as well as civil engineering. The case studies illustrate the added value of VR/AR. These include cost savings, for example, when physical models in the design process are at least partially replaced by virtual models that can be created more cheaply, or when costly excavation damages during construction work are avoided. But the examples also illustrate other benefits, such as the improvement of human-machine interaction in Case Study 9.3 or the realization of telepresence and computer-supported collaboration in Case Study 9.4, as well as completely new possibilities of visualization.

While Case Study 9.5 uses mobile devices such as smart phones or tablets, the next two case studies (9.6 and 9.7) show examples of large installations that use specially equipped rooms. In contrast to Case Study 9.5, Case Study 9.6 does not visualize construction data such as blueprints in reality, but utilizing *Spatial Augmented Reality* in a permanently installed, dedicated hardware setup, which can display construction data flexibly and at life size in their spatial context. Case Study 9.7 shows how a CAVE, a sophisticated hardware infrastructure, can be used to convincingly present a virtual world. This VR hardware is located in an academic environment and also highlights the added value of VR for scientific applications.

Case Study 9.8 is an example of the use of VR/AR in the field of medicine and health. It shows that AR can also be used for treatment, in this case for therapy of people who have suffered a stroke. It also shows how ideas and approaches for the use of VR/AR are developed in the academic environment. Case Study 9.9 shows how a transition from the academic environment to commercial exploitation can be accomplished.

The next three Case Studies, 9.9 through 9.11, demonstrate the value of integrating not only objects but also virtual characters into VR/AR. These virtual characters can be used either to graphically represent users as avatars in the VR/AR environment, or to populate the world with virtual people, e.g., in the form of virtual agents capable of acting autonomously. All three case studies also demonstrate the potential that VR/AR offers for teaching and training. For example, Case Study 9.9

illustrates basic research on collaborative virtual trainers. Case Study 9.10 shows how virtual patients already serve as established support in medical education. Case Study 9.11 is an example of how embodied social XR also supports social skills training. Furthermore, this case study also shows how avatars can be created and what effects avatars may have on the users they represent. This feedback effect can also be used for therapeutic purposes, for example. Case Study 9.12 is another example of how VR can be used for rehabilitation and training. This case study also shows that diverse user groups can benefit from VR/AR. In this case, VR opens up new possibilities for training that rely on playful effects, which can extend to *serious games* (Doerner et al. 2016) that are realized in VR/AR.

All in all, the 11 case studies show the wide range of possible applications of VR and AR technologies and the associated objectives, which can range from training to visualization, therapy, design, construction and entertainment.

9.2 Using Virtual Reality for Design Processes in the Automotive Industry

Alexander Tesch, Volkswagen AG

The design process of a car consists of various consecutive steps where several qualities such as aesthetics and feasibility are reviewed. For this purpose, physical mock-ups are manufactured on a 1:1 scale and presented at specific milestones. In Fig. 9.1 (left), an example of a partial physical model is shown. However, the production of these mock-ups is time-consuming, as it can take weeks until the whole prototype is ready for presentation. As a consequence, prototypes do not represent the current state of the car development project. Moreover, they often lack several components, as the manufacturing costs for a fully detailed mock-up would be too high. The overall cost of one prototype varies with the desired quality and can take up to several hundred thousand US dollars.



Fig. 9.1 A physical mock-up serving as a real model of the front part of a car in reality (left) and a virtual counterpart (right). (©Volkswagen AG. All rights reserved)

To mitigate the problems associated with physical prototypes, today's design process employs Virtual Reality (VR). VR can support the decision-making already in the early phases of the development. Using powerwalls, CAVEs and VR Head Mounted Displays (HMD), car components are visualized and reviewed in different variants, leading to a reduction in the number of physical mock-ups needed. With VR being part of the daily work in a variety of different fields, such as ergonomics or lighting design, VR has become a valued technology and can be considered a standard technology throughout the development phase of a car.

For certain design reviews, a highly immersive virtual environment is required to guarantee that an executive is enabled to make a valid decision based on VR visualization. Common examples of such reviews are ergonomics and visibility checks in a car's interior. Besides photo-realistic rendering techniques, adjustable *seating bucks* are used to achieve a high degree of immersion in both examples. Figure 9.2 shows an example of such a seating buck. These seating bucks are physical car seats in combination with VR HMDs. They provide the user with the feeling of being fully surrounded by the interior with a natural view out of the windows. For ergonomic checks, the alignment of the virtual seating position with the physical seat is key for creating a highly immersive experience, as the executives are typically experts in the field of interior design and consequently highly sensitive to any positional discrepancies. They are capable of noticing offsets and height differences of only a few millimeters between the real seating position and the virtual seating position. These discrepancies can result in a significant reduction in the feeling of presence, which in turn could make it impossible to continue with a meaningful



Fig. 9.2 A seating buck as shown in this picture is used to provide haptic feedback for an immersive experience in VR. (©Volkswagen AG. All rights reserved)

evaluation. Thus, a precise alignment of the virtual and physical world is crucial for a valid evaluation result. Visibility checks deal with questions such as: To what extent do the C-pillars affect the driver's visibility? Does the front vent glass restrict the driver's visibility on pedestrians? Does the car's shape limit visibility through the side- or rear-view mirrors? While the first two questions might be answered by varying the car's geometry or exchanging certain components directly in VR, virtual mirrors require a correct simulation of how light rays behave.

For the evaluation of the car's surfaces on the exterior and interior, the demands for a realistic depiction are particularly high. Here, a virtual presentation on a powerwall represents the standard tool as it allows for an agile demonstration during which numerous variants can be presented instantaneously under the direct control of a presenter. Figure 9.3 shows an example of a presentation room equipped with a powerwall. The VR system allows the presenter to dynamically change the virtual environment as well as the materials on the interior's surfaces. As a result, the visualization of different variants of a component not only shows the changes made to the geometry but also emphasizes the impact on the impression of the whole car. As the model used in the VR visualization is automatically derived from the most recent version of the construction data, it is guaranteed that the presented component is always up to date. Furthermore, a rendering cluster enables the rendering of virtual models on a powerwall with global illumination. Therefore, the powerwall can offer a high-quality presentation of the car. With advances in graphics hardware and rendering algorithms, the need for physical prototypes throughout the design process might be further reduced. On the other hand, a powerwall also enables the



Fig. 9.3 A prototype is visualized on a powerwall. (©Volkswagen AG. All rights reserved)

user to investigate the model from perspectives that no customer is likely to take. This in return can lead to inappropriate decisions and high costs.

A VR presentation using an HMD offers the possibility to confine the user to natural viewing perspectives. This enforces an examination of the car similar to how a potential buyer of this car would look at it. With the focus on surface evaluation, an expert study has shown that a surface analysis with a VR HMD can achieve almost equivalent results to the use of physical mock-ups (Tesch and Doerner 2020). Furthermore, a VR presentation on an HMD enables the user to experience the models with natural dimensions while also being able to interactively change not only the model itself but also the virtual scene. For instance, a virtual parking lot can be provided as a context, exhibiting a variety of different car models for comparison purposes. Another example is the provision of a virtual studio with sophisticated lighting controls that facilitates the design evaluation of exterior surfaces.

There are several challenges when using virtual reality with an HMD. One major problem is the occurrence of cybersickness in a variety of different scenarios, such as in driving simulations. Another drawback of a VR presentation with an HMD is that experts have low confidence in the validity of the appearance of virtual objects. One reason for this is imperfections in VR presentations. For instance, displays in an HMD still exhibit the screen-door effect, i.e., the pixel grid can be perceived. Even though there are technologies to achieve an almost realistic look of the virtual data such as raytracing, a real-time presentation with sufficient quality on an HMD is currently not feasible.

In summary, there are multiple use cases where VR has the potential to reduce costs and to accelerate the development phase of a car. Among these use cases are visibility checks and surface evaluations. More examples of VR applications can be found in Berg and Vance (2017). VR opens up new ways of interaction between multiple users as well as between the user and the virtual data. For example, while two experts or executives can only discuss a physical prototype from different perspectives (while sitting next to each other), a virtual environment enables them to be in the same position, allowing them to have a similar view. Nonetheless, physical models remain the most trusted basis for final decisions and are still indispensable. Even though the use of VR has proven to be successful and has already led to a reduction in the need for physical mock-ups, there is room for improvement (e.g., in the area of usability of VR tools or the area of dedicated VR authoring processes). The potential of VR for design processes in the automotive industry has not been fully exploited yet. Besides VR, also Augmented Reality (AR) is set out to be used frequently in a variety of different areas, such as for constructing and designing vehicles. For instance, AR offers the possibility to enrich simple physical mock-ups by superimposing virtual data on top of these prototypes (Zimmermann 2008). Dummy components on these models can be replaced with the most recent construction data when viewing a live video image augmented with AR methods on a tablet. Thus, AR also facilitates new ways to reduce the level of detail worked into the physical mock-up and thereby further reduces manufacturing cost. By using the camera of a tablet and a precise tracking algorithm that merges a virtual model with its physical counterpart, the image of the physical exterior can be virtually projected

on a physical mock-up. Thus, a simple physical object can be augmented with additional virtual details in the correct position. Examples of such details are gaps, different car paints or car components such as different headlights.

9.3 AR/VR Revolutionizes Your In-Car Experience

Axel Hildebrand, Stephan Leenders, Tobias Tropper, Wilhelm Wilke, and Christian Winkler, Daimler Protics GmbH

Although VR and AR have become commodity technologies with high awareness even in the non-tech community, employing AR/VR in a car is still a challenging objective. In doing so, several conditions need to be considered to ensure a seamless and well-received in-vehicle AR/VR experience.

The advantages are obvious and numerous use-cases exist, e.g., getting the right information at the right place via AR without the need to take the driver's eyes off the road, or gaining a new quality of in-vehicle entertainment leveraging VR. However, aligning the almost non-predictable vehicle motion and vehicle space to an augmented or virtual environment requires a careful and well-defined transition to achieve consistent storytelling. In addition, specifically regarding AR, a precise localization of the vehicle is of particular importance.

This case study describes our journey, lasting more than 15 years, from prototyping and research up to the MBUX 2.0 Augmented Reality Head-Up Display available within the new Mercedes S-Class launched at the end of 2020 (see Fig. 9.4). Furthermore, we present how Head-Mounted Displays (HMDs) can become part of an immersive in-car gaming or entertainment solution.

Given the recent advances in navigation and driver assistance systems, there are three major questions that arise during the path to fully autonomous driving:



Fig. 9.4 Different AR features (left to right): Distronic (adaptive cruise control), lane departure warning, assisted lane change, route guidance, destination. (©Daimler Protics GmbH. All rights reserved)

1. How can the driver oversee, understand and leverage the growing number of increasingly powerful assistance systems of a car?
2. How can the driver gain trust and lie back while the car takes over part of their job?
3. How can an entirely new in-car experience be created when there is no human driver?

AR can play a key role in all these questions and it is happening already. From a technical point of view there are multiple ways in which AR can be used within a car. These include showing an augmented video on a screen, AR glasses or projecting the virtual content directly on the windshield. The latter can be achieved by using a recent head-up display with a comparatively large field of view (e.g., $10^\circ \times 5^\circ$).

After the introduction of the video-based MBUX Augmented Reality for Navigation in 2018, Mercedes-Benz introduced a novel AR HUD starting with the S-Class presented in 2020. It features both contact analog visualization of navigation as well as assistance systems to address the first two questions in particular. So why are there no other AR HUDs available yet? Apart from the hardware, implementing such a system is much more challenging than it might seem in the first place.

Pose Estimation and Sensor Fusion For the navigation use case especially, it is crucial to know exactly the position and orientation of the car. Every little bump on the street needs to be considered, as the contact analogy would suffer otherwise. A precise and high-resolution pose is important for the quality of AR in general, but it is even more crucial for the HUD, as it acts like a magnifier that makes changes of orientation of well below 0.1° obvious. This is aggravated by the fact that most of the sensor data comes with a different frequency, latency, reliability, coordinate system and resolution.

Projection The content of the head-up display must appear in a way that fits exactly to the reality in front of the car. To achieve this, every piece of hardware involved needs to be calibrated. Additionally, the head position of the driver (head tracking) as well as the windshield distortion need to be taken into account (warping).

Latency A head-up display does not forgive visual inaccuracies. As you directly see the reality behind the windshield, the requirement is no less than to have zero latency on the screen as well. As this cannot be achieved, the goal is at least to reduce the latency as much as possible and then use proper prediction to make up for the rest.

User Interface Although the size of HUDs has become bigger and bigger over time it is still limited to a rather small area just in front of the car. Thus, it easily happens that relevant information leaves the field of view. So, the challenge for the UI is to get the most out of AR, but at the same time not to overload the screen and to handle information outside the screen.

Autonomous driving is going to release huge amounts of free time in the car. But already (rear seat) passengers are looking for distraction during long-haul trips. While traditional activities such as reading, working, playing games or watching movies on smartphones and in-car displays will have their place, car movements easily induce motion-sickness during these activities. In addition, unfavorable viewing angles for handheld devices (eyes below road level) and the confined space of the car can further limit perceived comfort and enjoyment for a significant amount of people during longer periods of transit.

Head-Mounted Displays (HMDs) that allow full immersion into a virtual or (in the future) augmented reality promise to alleviate many of these problems. They can present any content at eye level (AR/VR) and, considering VR, include the car motion to counteract motion sickness and place the user in completely new environments to escape the confines of the car. In addition, the interior of a car, with all its sensors/actuators and the computable driving forces that act on human bodies, represents an instrumented environment with unprecedented opportunities for new types of entertainment and gaming pertaining especially, but not only, to HMDs. Potential use-cases for VR headsets include:

- gaming (e.g., a space shooter where the story and route depend upon the real navigation route)
- entertainment (e.g., VR roller coaster or scenic ride through a fictitious or historic landscape (Haeling et al. 2018))
- working (e.g., virtually larger office inside the confined car space)
- recreation scenarios (e.g., sailing over a calm sea while listening to relaxing music)

However, enabling the use of HMDs inside cars is very different compared to living rooms in multiple aspects. First, the space for body movement is much more confined. To one side of the seat, in particular, there is usually almost no space available. Thus, the virtual interfaces must be intelligently adapted so that users do not accidentally collide with the car interior. Yet more importantly, the driving forces work on the body and induce motion sickness if the visual perception cues contradict those that the body senses through the vestibular system. That means that for any shown VR content, some visual representation of movement of the real car and the surrounding environment is beneficial for the user.

Other remedies against motion sickness are the visualization of landmarks, rest frames in the real world (e.g., a part of the car that appears in the virtual environment to hold on to) and subtle information cues about the expected acceleration or deceleration of the car. These showed that they are even able to reduce motion sickness instead of increasing it (Carter et al. 2018). For AR, digital content has to match the real-world environment precisely, by analogy with the head-up-display. Because motion sickness, once experienced, can take hours for symptoms to resolve, solving this problem is a key enabler for prolonged use of both AR and VR HMDs inside cars.

However, all these visualizations require precise alignment of the virtual and physical worlds (see Fig. 9.5). The previous section has already elaborated the

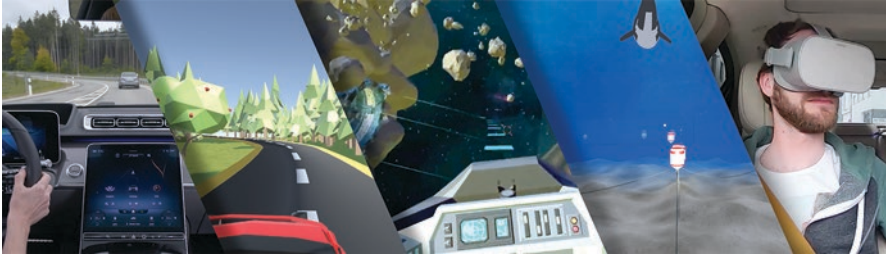


Fig. 9.5 Different types of dynamic virtual entertainment matched to the route ahead (left to right): original view – dynamic procedural environment as basis for, e.g., virtual cinema – space shooter game – edutainment – all viewed inside a (mobile) VR headset. (©Daimler Protics GmbH. All rights reserved)

challenges of calculating a precise car pose in relation to the real world. In combination with HMDs, the precise alignment is further impeded, since conventional tracking methods for HMDs will not work out of the box in a vehicle. This is because the physical forces measured by the HMD's Inertial Measurement Unit (IMU) will reflect the car and head motion combined (e.g., accelerating will have your virtual head turn slightly down), while other tracking sensor data (e.g., optical tracking) may still only provide evidence for the head motion. This creates observation conflicts during the sensor fusion for the final HMD pose. This sensor fusion, however, is required, as the drift of today's IMUs is much too high for them to deliver sufficient tracking quality while driving on their own.

Regarding storytelling, the biggest challenge is to create thrilling experiences on-the-fly for dynamic routes at dynamic paces (e.g., sudden stops at red lights or traffic jams) which can both change at any time during the course of the experience. To this end, the car can provide a lot of interesting information, like the current route, traffic situation along the route, immediate traffic surrounding and possible alternative routes the driver could take. This data can help game designers redesign their game story and content positions according to dynamic properties, which could be part of an SDK offered by car manufacturers.

Finally, the crash safety of HMDs is a topic that must be researched and solved for wide adoption. Overall, while technical challenges such as fusing precise car and HMD poses at low latency are coming into reach (e.g., as demonstrated by Haeling et al. (2018), questions regarding social acceptability (Is it okay to use VR HMDs over the whole duration of a family trip?), safety concerns (e.g. crash safety and driver distraction), and business plans (What are users willing to pay for these experiences?) increasingly gain importance, for which McGill et al. (2020) provide a good overview. Both software in general and user interfaces in particular will gain even more importance to deal with the increasing complexity of driving situations, especially considering the trend towards autonomous driving. In addition, technical innovations such as waveguides or holographic displays, as well as more natural and holistic user-interaction will lead towards a seamless human–machine interface. For

gaming and entertainment especially, in the short term the usage of HMDs has already enabled new applications.

9.4 VR-Based Service Training in the Life Sciences and Diagnostics Industry

Julian Hillig, *realworld one GmbH & Co. KG*

The adoption of virtual (VR) and augmented reality (AR) technologies has expanded rapidly across various enterprise sectors, and the technology is now experiencing accelerated integration within the life sciences and the analytical and diagnostics, pharmaceutical, chemical, and processing industries. While VR and AR are considered breakthrough technologies and are expected to substitute for computers and smartphones in the coming decades, the global crisis of COVID-19 has forced many companies to accelerate the digitalization of their workforce and operations, leading to increasing general adoption of the technology. VR represents the ideal technology for companies to continue conducting training courses and holding group events without any risk to health and safety. In addition to being an important contribution to a company's overall digitalization and global sustainability strategy – as drastic reductions in air travel for companies will result in a significant decrease in their carbon emissions – the cost-saving potential from VR-based training is around US\$450,000–650,000 per product per year.

Along with the continuous growth of businesses in the life sciences and diagnostics industry, the global headcount of their Field Service Engineers (FSEs) is consistently increasing. Additionally, as product lines have become increasingly complex, the knowledge and skill requirements for FSEs have also grown. Previously, FSEs conducted only basic maintenance and repair tasks, but now they also provide expanded services. Because of these factors, the post-service and support teams are reaching their full training capacities, resulting in extended waiting periods for newly hired FSEs to conduct their on-site training. By implementing VR into the training process, businesses can significantly increase their global training capacities and will be able to meet both their short- and long-term training needs. VR provides the global FSEs with highly realistic and fully interactive training scenarios created by service specialists that can be accessed at any time from any location.

While there will be pre-recorded training content prepared for FSEs, any internal specialist within a company can easily connect to the VR environment and join the FSEs at any time to answer specific questions or even test an FSE, without the need for anyone to leave the office or home. Also, FSEs can always revisit any training material on their own to refresh their knowledge of particular aspects of the instruments and equipment.

The VR-based software platform developed by *realworld one* now serves as a standard for applications in training, sales, marketing and service (see Fig. 9.6).



Fig. 9.6 Example view within a multiuser VR training session. (© realworld one. All rights reserved)

This software platform has been specifically designed for the life sciences and analytical and diagnostics industries and includes the following functionalities:

- The CAN functionality enables users to create and preserve their own VR content and share it with others across the globe. People can record and save interactive training sessions, product explanations, events, meetings, and more within their VR environments.
- The multiuser-based software allows users from all over the world to meet and collaborate in real time, as well as interact directly with products in virtual environments (see Fig. 9.7).
- Users can upload 3D and CAD data, PDFs, PowerPoint presentations, images, videos, and notes from their desktop into VR to share them with colleagues and business partners.
- The virtual desktop function lets people use their personal computers in VR. One can browse the web, view files, answer emails or work with BI (business intelligence), CRM (customer relationship management) or ERP (enterprise resource planning) systems on a giant virtual screen.
- The avatar configurator gives users the option to select and configure their own avatars for a personalized virtual experience.
- *realworld one* provides multipurpose rooms, including user, conference, training and showrooms, as well as an auditorium hall for larger events. Users can host and invite people to join at any time.
- The *realworld one* software is designed to be used with the latest virtual and mixed reality head-mounted display devices from various manufacturers.



Fig. 9.7 Example interaction in VR training. (© realworld one. All rights reserved)

- The non-VR mode enables users to connect to virtual environments without requiring a VR headset.

The implementation of VR solutions into the service training process provides companies with the following performance enhancements:

- Consistency: Access to highly consistent information throughout the entire organization, while providing a coherent format for product user education.
- Efficiency: Significant gains for global service and support teams through a VR-based strategy that provides easier and quicker access to service experts, while simultaneously reducing the resources required from these experts.
- Time: Significant reduction in the time required to train personnel on instruments. This shortens the length of the certification process for staff, as there is no waiting period to participate in training sessions.
- Capacity: The capacity to conduct training on certain instruments is determined by training staff, facilities and hardware availability. By implementing VR-based training with virtual instruments, these dependencies can be significantly reduced.
- Cost savings: As the number of training participants has considerably increased over the years, the costs incurred by companies for hosting trainees, by providing travel, accommodation and food expenses, have risen markedly. In addition, the depreciation and maintenance of instruments required for training also represent a significant cost that can be saved by moving them into a virtual environment.
- Flexibility: The ability to receive/conduct training can be made as flexible as is necessary, using a VR-based approach, as content is available at any time and experts can quickly connect into the various environments for one-on-one sessions.

The typical rollout plan for VR software implementation at *realworld one* is usually a 3- to 6-months process, requiring extensive consultation with the client's technical team to bring the full spectrum of training features into a virtual environment. The initial phase calls for the complete 3D rendering of all technical equipment involved in the training. After the client provides feedback on the VR prototype module, the final version will be completed for international distribution.

9.5 Utilizing Augmented Reality for Visualizing Infrastructure

Alec Pestov, vGIS Inc

Municipalities and utility companies maintain vast networks of underground and aboveground infrastructure. This infrastructure is difficult to access – many assets such as pipes, cables, valves, etc., are buried underground – and often complex, as multiple utility types reside densely near each other. The combination of complexity and inaccessibility leads to the high cost of any infrastructure-related initiative. Additionally, utility workers' inability to see buried assets directly occasionally leads to excavation damages, which are estimated at US\$6 billion annually for North America alone.

The traditional approach to locating utility assets relies on using printed and digital maps in conjunction with specialized equipment such as electromagnetic locator devices. The locator then paints the horizontal location of the asset on the ground, produces a sketch and compiles a report. The sketch and report are then provided to the excavator. Often, locations are independently validated by another person through a quality assurance process. The location work process is complicated, relies on records that can – at times – be inaccurate or incomplete, involves personnel with varying degrees of experience and is an important component of the damage prevention and workplace safety programs of the construction industry.

In the AEC (architecture, engineering and construction) industry, unseen infrastructure can cause design errors or construction problems in the field. At the design phase, it can be costly to redesign an already developed plan. If issues come up during construction, they can be extraordinarily costly, leading to long delays and project cost overruns. Furthermore, it can be difficult for engineers to analyze blueprints to understand 3D spatial relationships with regard to construction projects. As a result, it takes longer to work on designs, and those designs are more likely to have errors, which could lead to delays, rework and cost overruns.

Emerging technologies such as Mixed Reality (MR) and especially Augmented Reality (AR) have great potential to positively influence the fieldwork (see Fig. 9.8). Using AR tools, field workers and engineers can see an unobstructed physical world in front of them, as well as virtual representations of lines, pipes and proposed structures that can be perceived similar to holograms (see Fig. 9.9). By interacting with



Fig. 9.8 Using smart devices with a location sensor to visualize underground infrastructure such as buried pipes and cables for construction: (a) with a tablet and (b) with a smart phone. (© vGIS Inc. All rights reserved)



Fig. 9.9 Screenshots from the vGIS application showing AR scenes with additional annotations such as distances measured. (© vGIS Inc. All rights reserved)

virtual ‘digital twins’, the user should be able to perform the job faster, more easily, more safely and more accurately.

vGIS is an AR/MR application designed by vGIS Inc. for high-accuracy field services operations (vGIS 2021). The app either uses the HoloLens, a holographic headset by Microsoft equipped with cameras, audio, various sensors or traditional smartphones and tablets to display underground pipes and other assets as holograms. While wearing the HoloLens or using the smart device, workers see an unobstructed physical world in front of them as well as carefully placed virtual imagery of proposed buildings and bridges, lines of wastewater pipes underground and reality capture displays. The virtual representations are color-coded and projected to scale at job sites, while advanced positioning algorithms designed by vGIS Inc. maintain its real-time-created virtual imagery world – positioned at the correct physical location – with up to 1 cm accuracy (see Fig. 9.10).

The vGIS platform combines client-provided BIM (building information modeling), GIS (geographic information system), Reality Capture and other types of spatial data with third-party information from multiple sources to create visuals to

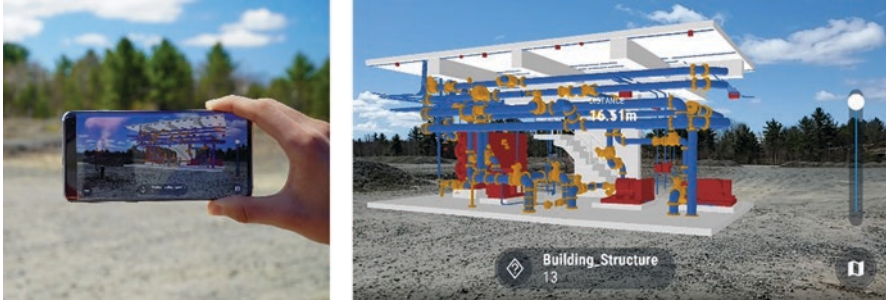


Fig. 9.10 Visualizing BIM data in AR. (© vGIS Inc. All rights reserved)

power purpose-built applications. The information is converted into unified 3D visuals in real time to display on the end user's devices (see Fig. 9.10).

The broad range of devices covered by vGIS allows AR users to deploy tools that work better in specific environments. Phones and tablets offer a unique combination of accessibility and convenience. They are familiar, easy to use and always on, which enables apps to run within a few seconds or less after unlocking the phone. Depending on the model and screen size, they are fast and offer excellent visuals, even in bright light. On top of this, they already run numerous apps that comprise a standard toolkit of any enterprise. It is not surprising that approximately 90% of vGIS app deployments are on mobile devices.

HoloLens and other dedicated AR devices are the best tools for complex or busy visualizations. These include visualizations of sophisticated BIM models, structures, multi-layered utility corridors, subsurface utilities of a busy downtown street, intertwining fibre-optic cables, etc. HoloLens delivers depth perception, which helps the user understand complex 3D objects almost instantly. The superiority of the stereoscopic 3D visuals exclusive to HoloLens and similar devices warrant deploying at least a few of these units to support advanced construction and engineering jobs, critical utility maintenance tasks (e.g., field crew supervisors), utility location validators, public works and similar roles where speed, deeper understanding and accuracy are important.

The hands-free environment is another type of deployment where HoloLens shines. If the user needs to remain hands-free to perform his or her job, paper records and tablet/phone-based tools will not suffice. HoloLens provides a rich and interactive user experience for displaying manuals, guides and collaboration tools while keeping the user's hands free to do the job.

vGIS helps field technicians close service tickets more quickly by reducing the time required to locate assets. Depending on the complexity of location and availability of utilities data, the system can save up to several hours on a single locate job. A study conducted by vGIS clients found that utility locators could reduce the time required to complete jobs by 50%. At the same time, QA validation time was reduced by 66–85%. This translated to cumulative savings of 12–20 h per locator per month.

Additionally, vGIS helps avoid costly repairs and line breaks. A line strike means that work comes to a halt until repairs are made. Many of those problems occur because the aboveground markings are inaccurate or incomplete. A simple two-hour markup may easily turn into a \$23,000 dig up and repair. vGIS helps reduce the number of such strikes.

The impact in the AEC space is yet to be measured. However, early deployments conducted by several multinational corporations have demonstrated tangible improvements in infrastructure-related projects, such as light rail construction and road work.

9.6 Enhancing the Spatial Design Process with CADwalk

James A. Walsh and Bruce H. Thomas, University of South Australia
Gerhard Kimenkowski and Stephen Walton, CADwalk Global Group Pty Ltd

Building design remains a uniquely challenging problem, involving a variety of stakeholders (novices to experts), waterfall development and high costs. In addition to these problems, given the huge physical size of the buildings being created and the fact that they must be scaled down for planning, design becomes increasingly abstract and complex in nature. This is especially difficult for clients who are not architects themselves, but instead are stakeholders who will have to utilize the end product. Fundamentally, clients require some way to bring abstract CAD plans into the real world for collaborative validation and optimization of the proposed project. Design experts can visualize the designs as final built constructs, but this is a complicated process for clients on plans given in a non-1:1 scale, with many layers of complexity (electrical, heating and cooling, etc.). Ideally, clients would be able to see the life-size end result as early in the design phase as possible, and throughout the entire process.

Projection mapping as a research topic enables the real world to be augmented and enhanced. More specifically, *Spatial Augmented Reality (SAR)* allows large-scale collaboration with a blend of physical and virtual experiences. Given the unique affordances of SAR as a display and interaction medium, the question arose: how could we leverage the unique affordances of SAR for visualizing and editing large-scale, life-size building designs (Thomas et al. 2011)? In exploring this problem, a joint project between the University of South Australia and Jumbo Vision International (now CADwalk Digital) was established to explore how SAR could be employed. The end result of the research project and subsequent commercialization is CADwalk Lifesize, a large-scale, projection-based, collaborative building design tool that allows end-users (novices and experts alike) to explore their plans in real time and at life size.

Built using the Unreal Engine, CADwalk utilizes multiple floor-facing projectors working in concert with a wall-facing projector in large warehouse-style spaces (see Fig. 9.11). The floor-facing 2D projectors display life-size blueprints and CAD

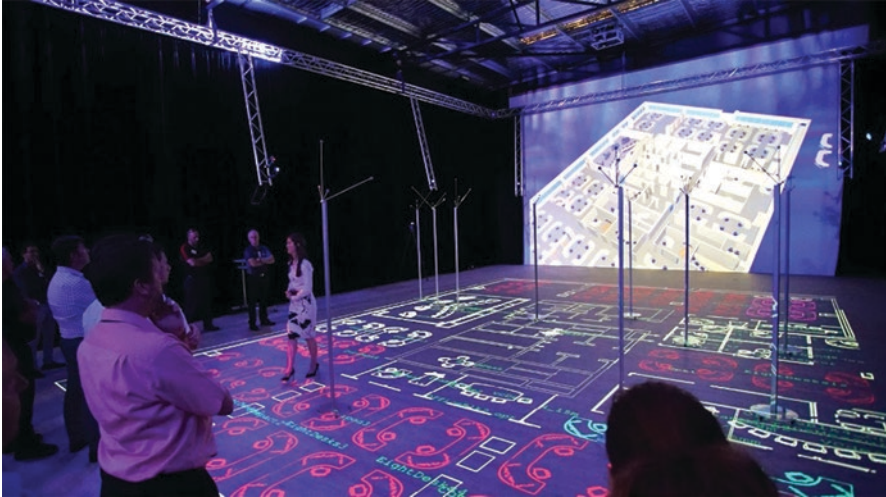


Fig. 9.11 CADwalk session showing blueprints on the floor and perspective correct 3D rendering of the scene on the end wall. Trees are visible in the scene as the thin vertical stands. (© 2020 CADwalk Global Group Pty Ltd. All rights reserved)

designs of buildings, enabling end-users to physically walk through their new spaces before they have been built. Using the wall-facing projector, a 3D view is projected, showing the 3D textured real-time rendering of the current plans, allowing users to see both 1:1 blueprints on the floor and the rendered 3D floor view of the space simultaneously. A roaming Surface tablet is used as a control screen for the session facilitator.

To collaboratively edit the plans a novel interaction device is used: a “tree”, which consists of an aluminum pole approximately 2 m high, on a wheeled base, with retro-reflective balls attached to the top. Using an optical-tracking system present throughout the whole space, users can wheel the trees onto content, rapidly spin the tree one way, and then back, to have the tree “pick up” the content underneath, which is then fixed to the tree to be moved and rotated around the scene. The user then rapidly spins the tree back-and-forth again to uncouple the projected content from the tree, leaving it in its new location. The trees act as a shared, mobile method for directly interacting with projected content, along with other functions, such as a digital tape measure showing the distance between multiple trees. For plans larger than the physical space available, blueprints can be panned and scaled as desired, including moving between floors in multi-floor structures.

Additional functions, such as adding/removing models, is performed with the aid of a user at a desktop placed to the side of the main space. Newer versions of CADwalk seek to leverage tablet input, and employ head-mounted displays (i.e., Microsoft HoloLens) to let users visualize and interact with the full 3D CAD model rendered above the projected blueprints. A miniature version of CADwalk (CADwalk Mini) also allows users without access to a purpose-built installation to

still leverage the collaborative and direct interaction offered by CADwalk, albeit at a much smaller, non-1:1 scale. A Virtual Reality (VR) view is also offered, allowing the current scene to be immediately viewed by users in a VR headset. While VR obviously also allows users to view the plans in 1:1 scale, the lack of natural collaboration and the spatial perception issues present in VR (Henry and Furness 1993) impact its effectiveness when needing to ensure accurate representation of structural plans to end users. Multiple CADwalk installations can be networked together, enabling remote collaboration at real-world scale.

A CADwalk session starts with ingesting the CAD models from the designers/architects. Given the plethora of data formats in use, data must first be prepped for import to the system in a compatible format. As CAD models are increasingly complex, data preparation may involve polygon reduction, among other tools, to create a scene that can be rendered effectively by the system. This process is done offline, before the session begins.

When the session commences, a CADwalk staff member (facilitator) is present to facilitate the session and system, enabling the stakeholder's users present to focus on their discussions around the space, not on the system itself. Users are able to freely roam the space and use the multiple trees to modify the layout of the environment, measuring, moving and rotating items in the scene, such as doors, walls, furniture or other fixtures. Scenarios can be saved, and actions can be undone/redone and recalled for final decision-making from all stakeholders.

Project stakeholders can then commence their own interaction regarding points of concern, either previously identified or new factors identified from being able to view the plans at scale in CADwalk. These include not just cosmetic changes, but legal requirements (safe distances, minimum clearances, etc.), clash detection (e.g., does the air-conditioning duct interfere with the placement of other elements?) and domain-specific investigations.

Given the wide application domains across which spatial design occurs, e.g., manufacturing/industrial, domestic housing, aerospace, defense and city planning, the applicability of CADwalk for improving the current design process has been demonstrated for its fast, efficient and cost-saving properties. CADwalk Lifesize Studios are used from kitchen and bathroom design validation and optimizing "dream home layouts", to highly specialized mission-critical control centers. The European Space Agency (ESA) utilized CADwalk to understand current workflows and spaces for their current and future space exploration missions, and subsequent validation and optimization of new workstations for their highly specialized operators. This will be the blueprint for all future ESA facilities globally.

While largely ignored for consumer AR, the use of projection in SAR provides unique affordances for commercial and industrial applications, where requirements such as having a fixed setup are not a restriction to adoption. In representing structural plans at life size, CADwalk allows novice and expert end users to collaboratively explore plans on an equal footing. Whereas novice users looking at traditional blueprints or CAD plans may only be able to visualize and understand a subset of the overall plans, including spatial relationships, the intuitive representation of

those plans in CADwalk means structural plans are now accessible to all stakeholders, for both viewing and modification.

9.7 The aixCAVE at RWTH Aachen University

Torsten W. Kuhlen, RWTH Aachen University
Geert Matthys, Barco

At a large technical university like RWTH Aachen, there is enormous potential to use VR as a tool in research. In contrast to applications from the entertainment sector, many scientific application scenarios – for example a 3D analysis of result data from simulated flows – not only depend on a high degree of immersion, but also on the high resolution and excellent image quality of the display. In addition, the visual analysis of scientific data is often carried out and discussed in smaller teams. For these reasons, but also for simple ergonomic aspects (comfort, cybersickness), many technical and scientific VR applications cannot just be implemented on the basis of head-mounted displays. To this day, in VR Labs of universities and research institutions it is therefore desirable to install immersive large-screen rear projection systems (CAVEs) to adequately support the scientists (Kuhlen and Hentschel 2014). Due to the high investment costs, such systems are used at larger universities such as Aachen, Cologne, Munich or Stuttgart, often operated by the computing centers as a central infrastructure accessible to all scientists at the university.

At RWTH Aachen University, the challenge was to establish a central VR infrastructure for the various schools of the university with their very different requirements for VR solutions. In cooperation between the RWTH IT Center and the Belgian company Barco, a concept was therefore developed and implemented as aixCAVE (Aachen Immersive eXperience CAVE), which, as a universal VR display, equally meets the requirements of full immersion and high-quality projection.

To achieve the highest possible degree of immersion, a configuration consisting of four vertical projection walls was chosen, completely surrounding the user. To enter and exit the system, an entire wall can be moved using an electric drive. This avoids door elements that interfere with immersion – when closed, no difference to the other projection walls is visible. However, extensive security measures had to be implemented so that no one could be locked in the CAVE in an emergency. Although a ceiling projection would have further contributed to the degree of immersion in the system, it was not used, as the complex audio and tracking integration planned for the CAVE in Aachen would not have been possible then. To nevertheless achieve largely complete immersion, the vertical screens are 3.3 m high.

The 5.25×5.25 m area, which is quite large compared to conventional CAVE installations, offers smaller teams of scientists enough space for collaborative analysis session, enables natural navigation (“physical walking”) within certain limits, and creates a realistic feeling of space in the virtual environment (“spatial

awareness”). Since the floor should not bend noticeably even with such a large base, 6.5 cm thick glass was used, on which thinner acrylic glass was placed as the actual display. This two-stage structure decouples the static requirements from the display requirements. Glass has better rigidity, while the acrylic glass has very similar properties to the sidewalls, which are also made of acrylic glass. For structural reasons, two glass elements lying next to each other had to be installed instead of a single glass plate. This inevitably creates a gap that, with a suitable mechanical design and skillful alignment of the projectors, turned out to be very narrow at 2 mm.

Figure 9.12 shows the basic structure of the solution with a total of 24 projectors. To achieve the required high image quality, projector and screen technologies were used that guarantee sufficiently high resolution, brightness, brightness uniformity and luminance. The final solution (see Fig. 9.13) is based on active stereo projection technology with 3-chip DLP projectors, each with a light output of 12,000 lumens and a WUXGA resolution (1920×1200 pixels). To meet the requirements for the resolution of the system as a whole, four of these projectors were used for each vertical side and eight for the floor, each in a 2×2 tiled display configuration with soft edge blending (see also Sect. 5.2)

Apart from the resolution and the brightness of the selected projectors, the properties of the rear projection screens are critical for the resulting image quality. These should provide a uniform brightness distribution without hotspots, so that users can walk from one corner to another within the CAVE without the image quality or

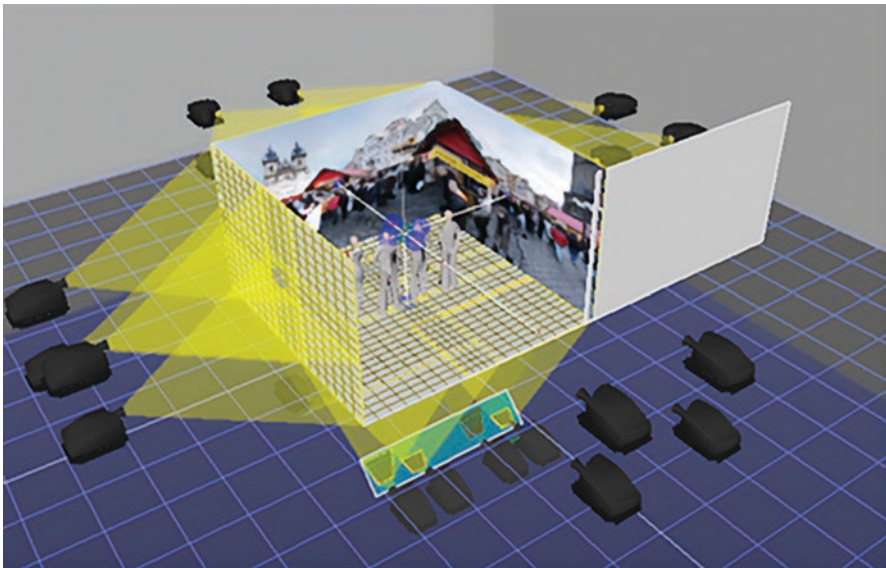


Fig. 9.12 Concept of the aixCAVE with 24 projectors. (© TW Kuhlen, G Matthys. All rights reserved)



Fig. 9.13 Complex installation of the glass plates for the floor rear projection of the aixCAVE. (© TW Kuhlen, G Matthys. All rights reserved)

perceived brightness suffering from the different perspectives. This requirement was achieved by using canvas materials with excellent diffuse properties (low peak and half gain, see also [Sect. 5.2.2](#)).

[Figure 5.2](#) shows the fully installed aixCAVE in operation. By combining a precise mechanical construction with high-quality projection technology, a CAVE system could be implemented that allows an intuitive visual analysis of high-resolution scientific data in three-dimensional space. Ergonomic factors such as high luminance and brightness uniformity, high contrast and excellent channel separation of the stereo projection, as well as small gaps between the individual screens have been consistently taken into account. As a result, the Aachen CAVE goes beyond a pure presentation system, providing a valuable tool that users from science and industry actually use in longer, intensive sessions for exploratory data analysis. In particular, the clear ergonomic advantages over HMDs, as well as the possibilities of a combined analysis of geometric and abstract data resulting from the high resolution, justify – at least at RWTH Aachen University – the very high installation and operating costs. Since its inauguration in 2013, the aixCAVE has proven to be a valuable tool in research projects in production technology, fluid mechanics, architecture, psychology and neurosciences. In addition, the CAVE is not only used as a tool for data analysis, but also as a tool for basic VR research by the computer scientists at RWTH to develop new navigation and interaction paradigms in virtual environments (Kuhlen [2020](#)).

9.8 Augmented Reflection Technology: Stroke Rehabilitation with XR

Holger Regenbrecht and Chris Heinrich, University of Otago, Dunedin, New Zealand

Millions of people experience a stroke and require rehabilitation therapy every year. Most stroke survivors are left with unilateral impairments, e.g., the inability to move one arm, and have to undergo a very long period of rehabilitation and training to regain motor function. The efficacy of this training depends on four intertwined factors: (1) the patient's (stroke survivor's) motivation, (2) the meaningfulness of the tasks in training, (3) the training intensity, and last but not least (4) the provision and effectiveness of stimuli for neuroplastic change. XR techniques, i.e., the full spectrum of computer mediation of reality between Virtual Reality and Augmented Reality, can play a major role here and we are presenting two systems based on the concept of *augmented reflection technology (ART)* we have developed and empirically and clinically tested.

With ART we are focusing on the factor of neuroplasticity, i.e., the brain's ability to lastingly change in response to environmental stimuli, while maintaining patient engagement with the other three factors for rehabilitation efficacy (motivation, meaningfulness, intensity). The neuroplastic effect is achieved by "fooling the brain" (Regenbrecht et al. 2011) about what it is "perceiving", e.g., by visually exaggerating movement capabilities of a limb or by mirroring over the healthy limb's movements to the impaired side (Regenbrecht et al. 2012; Hoermann et al. 2017). XR offers great possibilities here for (1) precisely directing what the patient controls and perceives, (2) suppressing potential disbelief, i.e., believing in the virtual magic of the technology and (3) keeping patients engaged with the rehabilitation process.

ART is based on the principle of decoupling what the patient is doing from what they are seeing. We sense and capture patients' limb movements (here upper limbs), feed this into an XR system and manipulate the perceivable output in a way that the (neurorehabilitation) effect can be achieved. Over the last decade we have built different versions of ART using tailored input, computing and output modalities.

ART4 (Fig. 9.14, left) comprises two closed boxes into which the patient puts their hands and lower arms. The boxes are closed with curtains, like with magician's boxes, so that the patient cannot see their actual hand movements. Both boxes are equipped with a particular form of diffuse lighting and cameras, which capture what is inside the boxes. The camera feeds are used to (1) foreground segment the hands and (2) track the hand movements. The segmented hands are put inside (in front of) a virtual environment, so that the user gets the impression to interact within that space. These segmented hands can then be selectively shown, hidden and/or mirrored at the therapist's discretion. We can also augment the users' perceived hand movement: for instance, an actual movement of say 10 mm will result in 30 mm movement as perceived by the patient on the screen. ART4 was designed for use in

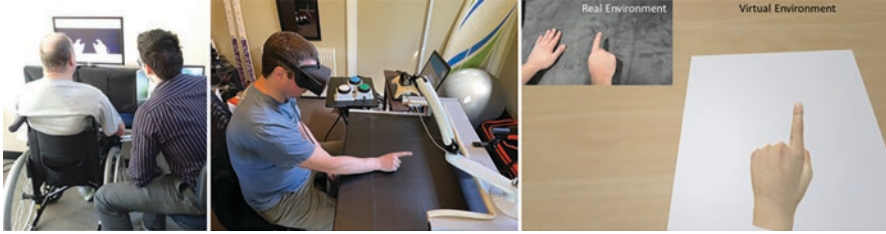


Fig. 9.14 Augmented Reflection Technology systems in action. Left: ART4 with “magician’s boxes” and operator; center: ART6 for home use without an operator; right: ART6 mirroring a stroke survivors’ right (unaffected) hand movement and presenting it to them in XR as their left (affected) hand carrying out the mirrored hand movements. (© H Regenbrecht, C Heinrich. All rights reserved)

clinical settings for the treatment of chronic stroke patients. However, XR features were implemented to allow for its use in other rehabilitation scenarios. These include hot/cold virtual environments for burn victims, enlarged or smaller hands for pain management and the ability to change the color appearance of hands (complex regional pain).

If we want to apply ART in users’ homes, then we have to (1) allow the system to be self-controlled and (2) be suitable for installation in people’s homes. ART6 (Fig. 9.14, center and right) utilizes a head-mounted display, a Leap motion controller, big arcade-style push buttons and foot pedals, individualized virtual hands and machine learning-based feedback mechanisms in conjunction with a tailored rehabilitation protocol (Heinrich et al. 2020). Our stroke application scenario has unique requirements in that our user has an impaired arm (no/limited movement), their unaffected arm is carrying out the mirrored hand movements (and thus cannot be used to control the system while in VR), and survivors can have low technical competency, which means the system has to be easy and intuitive to use. To account for these requirements, we developed an interface that consists of arcade style push buttons for the user to interact with the system outside of VR (start/stop system, switch between system modules). While in VR, the user can interact with the system by using two foot pedals (move on to next hand exercise or show a virtual training hand which demonstrates the hand exercise to the survivor in VR). Our XR hardware was chosen for survivors’ home use because it provides an inherent decoupling of the survivors’ view from their home (real) environment into our XR environment. For our stroke rehabilitation scenario, this serves three purposes. (1) Survivors are completely immersed in our virtual illusion and this can lead to a more convincing “fooling of the brain” because of the mixing of what is real (hand movements, real-world/virtual environment correspondence) and augmented (mirrored hand position and mirrored movement), which can help lead to that suppression of disbelief that is desired for neuroplastic effects to occur. (2) It allows for the mirrored virtual hand to be observed in the most spatially congruent and natural position for the survivor. (3) It disconnects the survivor from their home

environment, which often consists of various distracting stimuli, and allows them to focus their complete attention/gaze on their mirrored virtual hand and rehabilitation exercises.

Besides the neurorehabilitation effects of ART, both systems are valuable instruments for patient engagement. The “newness” of XR, the game elements of the training tasks, the control of the exercises, including the individually tailored pace, and the realism and meaningfulness of the experience lead to increased patient engagement.

To make ART more widely available – currently, our systems are used with patients and users in Dunedin, New Zealand (Dunedin Hospital) and Berlin, Germany (MEDIAN Klinik Kladow) – we are going to bring our systems to market in the near future. The improvements in the quality of XR technology in combination with increasing affordability of that technology will allow more and more users to benefit from our ART approach. While stroke rehabilitation is our main focus at the moment, ART can be used with other conditions, like traumatic brain injuries, (phantom limb) pain management and hand therapy, but also for education and training, entertainment and other related sectors.

9.9 Collaborative Virtual Trainers in VR Applications

Xiumin Shang and Marcelo Kallmann, University of California, Merced

We use the term “*virtual trainer*” to refer to a simulated human-like character that can collaborate with humans to complete a given task with the use of interactive verbal and/or non-verbal movements and behaviors. Virtual trainers collaborating with human users can be achieved in different ways. Here we discuss two important types of collaboration that are representative of indirect and direct types of interaction. We consider an indirect collaboration when the virtual trainer collaborates with the user only by providing verbal or non-verbal feedback as instructions, therefore helping the user to complete a given task but letting the user perform the task independently. In a direct collaboration, the virtual trainer will instead jointly complete the task with the user. Here we focus on the particular case of collaborative object manipulation where both the virtual trainer and the user need to manipulate a virtual object together in order to complete the given task. We summarize in this chapter our current work on implementing both indirect and direct types of collaborative virtual trainers. Both of our projects are being developed with the use of the Unity game engine.

To achieve effective interactions when assisting humans to perform tasks in a given scenario, a feedback strategy has to be identified and implemented. In general, feedback is a language or gesture signal given by the virtual trainer and which might change the user’s thinking or behavior to improve his/her learning or training

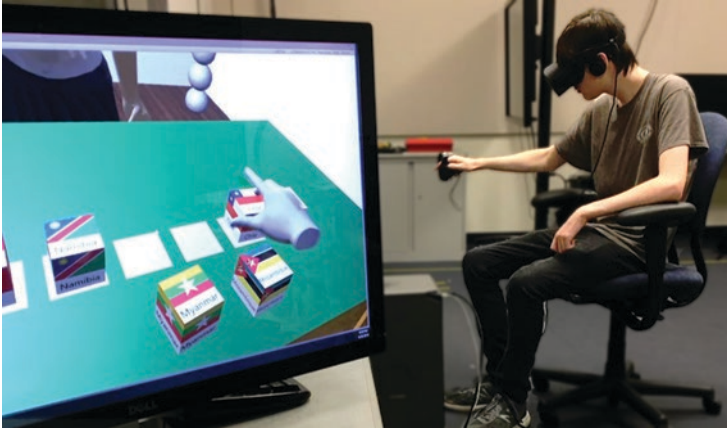


Fig. 9.15 In this VR training environment the virtual trainer provides feedback to assist the user to sort virtual cubes such that the represented countries appear in increasing area order. (© X Shang, M Kallmann. All rights reserved)

performance (Arif et al. 2017; Blair 2013). A feedback strategy will specify how feedback is provided, including types of feedback and several other parameters, such as frequency and adaptation. We have investigated two particular types of feedback strategies for virtual trainers assisting participants in a VR task, as illustrated in Fig. 9.15. Strategies based on Correctness Feedback (CF) and Suggestive Feedback (SF) were compared as possible feedback strategies used by the virtual trainer to help users to memorize relative areas of given countries.

A scenario was designed where the virtual trainer assists the user to sort cubes representing countries according to the area of the countries. The user needs to complete the sorting task with different levels of difficulty, which are implemented with an increasing number of countries to be sorted. Under this task scenario, CF is defined as providing correctness feedback by fully correcting human responses at each stage of the task, and SF is defined as providing suggestive feedback by only notifying if and how a response can be corrected. We have conducted a pilot user study with four participants and a formal user study with 14 participants to investigate the effects of the feedback strategies provided by the virtual trainer on the user's performance. Our final study results show that CF was more effective because it had higher user preference and shorter task completion time with equivalent performance outcomes. This study exemplifies the importance of implementing an appropriate feedback strategy for a given scenario and application. More details are available in our previous work (Shang et al. 2019).

Using virtual trainers to assist users during direct manipulation tasks, in either simulated environments or physical environments, requires the use of some specific approach for achieving adaptive motion control. While in some cases a

hard-coded solution involving a step-by-step procedure for the virtual trainer to follow may be possible, in such cases the virtual trainer will not be able to adapt and execute a similar but different task, or to address the same task in a different environment. To increase the adaptability of this type of collaborative virtual trainer, different machine learning methods can be applied. A common approach is to rely on imitation learning methods able to learn human behaviors using some type of action mapping and then to apply the learned knowledge to the robotic or virtual trainer for it to cooperate with human users on given tasks. Another popular approach is to apply reinforcement learning to improve a robotic or virtual trainer's sequential decision-making policy by interacting with the environment periodically.

Previous work (Yu et al. 2020) has demonstrated the effectiveness of using deep reinforcement learning (DRL) for virtual trainers or robotic agents, and for agent–human collaboration. We focus on applying the DRL methodology to a virtual trainer collaborating with a human user immersed in a VR environment. In our simulated environment we have designed a task involving two virtual trainers collaboratively moving a tray from a random position to a target position in a dynamic environment with an object on top of the tray. The goal is to reach the target location while avoiding collisions with obstacles and while keeping the tray balanced. Based on this design, we have trained an efficient initial policy in this virtual environment, as illustrated in Fig. 9.16.

The use of virtual trainers assisting humans in a variety of scenarios represents a promising application for VR technologies and the study of collaborative behaviors for virtual trainers is key for achieving effective virtual trainers. When properly implemented the discussed types of collaborative virtual trainers can significantly enhance the learning and training experiences of users by achieving interactions that can closely resemble intuitive human–human exchanges.



Fig. 9.16 Two virtual trainers move a tray collaboratively in the VR environment. (© X Shang, M Kallmann. All rights reserved)

9.10 Virtual Patients: A Case Study from Research to Real-World Impact

Benjamin Lok, Computer and Information Science and Engineering,
University of Florida
Francisco A. Jimenez and Cheryl Wilson, Elsevier

In this case study, we will explore the journey of virtual patient technology from research to a commercial system that is educating hundreds of thousands of healthcare students a year. Virtual patients are computer simulations of a real patient encounter. Virtual patients are used in the training of healthcare students, including nursing, physician, pharmacy and physical therapy. Virtual patients provide students with opportunities for practice, remediation, feedback and exposure to a wide range of conditions and symptoms. Virtual patients are diverse in their background, being able to present patient scenarios that involve various ages, genders, ethnicities, races and personalities. Virtual patients are used by educators to develop psychomotor, cognitive and social skills in learners. This case study will cover the research that was conducted by the Virtual Experiences Research Group at the University of Florida, lessons learned through commercialization of the research by Shadow Health® from Elsevier and implications for nursing education and virtual reality as their simulations are the most used virtual patient platform in the world.

Research began in the early 2000s into using virtual patients to improve healthcare students' conversational skills. Early systems experimented with a wide range of modalities including head-mounted displays, large projection displays and desktop monitors (Johnsen and Lok 2008). Research studies evaluated multiple input modalities, including enabling the user to speak to the virtual patient, type questions to the virtual patient and gesture to the virtual patient.

Dozens of user studies were conducted with healthcare students to explore the potential and limitations of virtual patients, including exploring the validity of virtual patients (Johnsen et al. 2007), learning empathy with virtual patients (Deladisma et al. 2007), the impact of different display types (Johnsen and Lok 2008), physical mannequin integration (Kotranza et al. 2008), reflection with virtual patient training (Raij and Lok 2008) and team training (Robb et al. 2014).

The resulting body of publications demonstrated the educational benefits of virtual patients, including developing clinical reasoning, empathy and communication skills. With the benefit and limitations identified through scientific study, the next stage was to identify how to help as many healthcare students as possible with a curriculum of virtual patients. Designing a curriculum of virtual patients would require resources that were beyond standard academic mechanisms of grants and collaborations.

The researchers worked with the University of Florida Office of Technology Licensing to identify pathways to commercialization. In 2011, a team of entrepreneurs and some of the core researchers founded Shadow Health.

Three important pivots occurred during the transition from a research platform to a commercial product: market identification, change in delivery mechanism, and adapting the virtual patient to curriculums. First, the nursing student market was

identified as the healthcare group that had the largest need for virtual patient training. There are over 400,000 nursing students in the United States and Canada alone. Second, an effective method for delivery of the virtual patients was identified. As head-mounted displays were not widely available at the time, standard laptop/desktop computers with both typed and speech recognition capabilities were used. Finally, the virtual patients moved from short 15-min scenarios used in the research studies to a series of virtual patient assignments that could be integrated throughout a course and provide over a dozen hours of educational content.

As of 2020, thousands of universities and colleges use virtual patients from Shadow Health® in their curriculum. Each year, over 100,000 nursing students use Shadow Health® products in their classes, reaching over 25% of nursing students in the United States and Canada. When they graduate, these nursing students will see approximately half of the US and Canada population, making the impact of the research into virtual patients a reality that is improving healthcare.

Each Shadow Health® product has a set of Digital Clinical Experiences™ (DCE). The DCE is the virtual patient encounter (see Fig. 9.17). Each DCE simulation starts with a pre-brief with a virtual preceptor that introduces the scenario, provides goals and instructions, and delineates what is expected from the learner in terms of performance. Next, the learner conducts a patient interview and physical assessment with the virtual patient, engages in therapeutic and non-judgmental communication, documents findings, and applies clinical reasoning skills to develop nursing diagnoses, care plans or interventions relevant to the scenario (e.g., administer medications, write a prescription or conduct a mental status exam). Upon completion of the patient exam in each DCE simulation, the learner is presented either with self-reflection prompts or a structured debrief where they can revisit actions and decisions taken throughout the simulation as well as reflecting on how they could improve in future patient interactions.

After submitting their attempt to their instructor for review, the learner is automatically scored on their clinical reasoning. Shadow Health's team of instructional

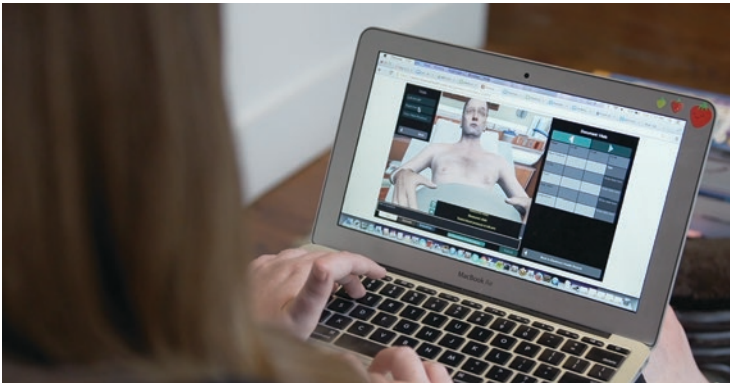


Fig. 9.17 Learning scenario with a virtual patient. (© B Lok, FA Jimenez, C Wilson. All rights reserved)

designers, psychometricians, nurse educators and computer scientists have collaborated with educators to develop the scoring for each DCE simulation. This development process includes rigorous discovery, design, construction, pilot testing and psychometric evaluation of each instrument so that it is aligned to the learning objectives and target learner population of each DCE simulation.

Shadow Health® DCE is addressing evolving nursing education needs. The landscape of nursing education allows for increased innovation and technological advancement in education programs. Students growing up as digital natives embrace the utilization of technology in their training programs. Faculty of nursing have recognized the impact that the technology has on the learning potential of their students.

With the development of technology delivering virtual patients, faculty time can be devoted to translating the virtual patient experience into clinically relevant applications instead of developing, implementing, debriefing and evaluation of the simulation experience. Faculty can also be assured their students are participating in a standardized experience. Integration of virtual patient experiences has allowed faculty to see how their students develop communication skills and clinical reasoning throughout a course.

Virtual patients are one of a growing number of virtual reality technologies that are transitioning from research to commercial product that is impacting our daily lives. So the next time you interact with a nurse or physician, you will know that your healthcare provider has likely practiced and improved their interpersonal skills using a virtual human.

9.11 Embodied Social XR for Teaching, Learning and Therapy

Marc Erich Latoschik, Carolin Wienrich, and Silke Grafe, University of Würzburg, Germany

Mario Botsch, TU Dortmund University

The Breaking Bad Behaviors (BBB) system utilizes the power of embodied social VR to teach and test classroom management skills with student teachers (Latoschik et al. 2016). The system simulates individual and group behavior through a parameterized AI-based model. The model includes typical patterns of student behaviors and their dynamic development from real classroom situations. Users can then slip in a teacher's role in front of a simulated class and experience different, even critical situations in a realistic way. BBB lets them try out and reflect on suitable response strategies alone or in groups and acquire important media skills during the process. Figure 9.18 shows snapshots from real-life use, which has been implemented for several years at the Julius-Maximilians-Universität Würzburg in the teacher training program. Initial empirical findings show significant advantages compared to the previous gold standard.

In principle, Virtual Reality has the power to release us from the need to physically meet at the same places and times and thus significantly increase the potential

for participation. Virtual agents can realize group experiences for individuals at any time. At the same time, the virtual worlds can include and support the ever-increasing volume of digital data, multimedia content, and information required by almost every aspect of collaborative knowledge work, specifically in the domain of learning and teaching.

The project ViLeArn explores teaching and learning with avatars and agents in an immersive social VR (Latoschik et al. 2019). ViLeArn preserves the diversity of embodied interpersonal communication for digital teaching. For example, a heterogeneous group of avatars that are not homogeneously represented (see Fig. 9.19, left) does provoke some eeriness but also increases the perceived possibility of



Fig. 9.18 Virtual training of classroom management skills in the 2018 FraMediale – Award-winning Breaking Bad Behaviors project. Left: a user within a virtual class of AI-simulated virtual agents. Right: a student teacher discusses her classroom management experiences with fellow students, showing her first-person view. (© ME Latoschik et al. All rights reserved)

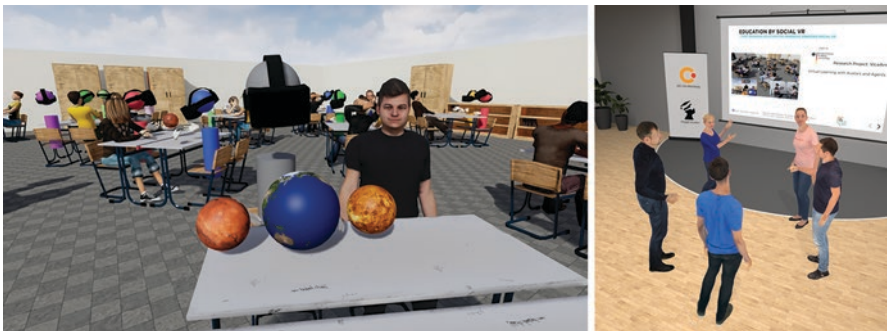


Fig. 9.19 Left: A virtual classroom with a different embodiment of participants during work in small groups. Right: Discussion in front of an interactive screen. The personalized photorealistic avatars maintain important non-verbal communication cues while providing a shared spatial reference system for communication. (© ME Latoschik et al. All rights reserved)

interaction. In this context, an immersive realistic personalized embodiment increases body ownership, presence and emotional response (Waltemate et al. 2018). Moreover, non-verbal communication signals such as gestures, facial expressions or gaze and eye contact are important mediators of, for example, our intentions (Roth et al. 2018). These are important factors, especially for the intended collaborative learning progress.

The work on ViLeArn has contributed, among other things, to the first non-commercial German social VR platform that supports a wide range of avatar embodiments up to photorealistic avatars. The system provides access to multimedia and text-based teaching/learning content: it supports a markdown-to-HTML5 processing pipeline and integrates personal and shared virtual large-screen interactive HTML5 panels. It also supports necessary functions for text input and sketch creation. The platform is largely independent of big IT service providers and also takes into account important data protection and privacy issues.

In general, avatars are our digital replicas in virtual worlds. The acceptance of virtual bodies as our own is called the *Virtual Body Ownership (VBO) illusion*. The VBO illusion is significantly determined by three different factors. These are (a) the perception and acceptance of the virtual body as our own body and thus as the source of sensory input (body ownership), (b) the perception of control over the virtual body and thus control over actions taken in the environment (agency), and (c) the change in the perceived body schema evoked by the stimulation (change). Figure 9.20 illustrates these three factors. A VBO illusion, in turn, is one of the central initiators and promoters of the *Proteus Effect* (Yee et al. 2009). The Proteus effect describes a change in behavior induced in the user/wearer of the avatar solely by the appearance of the virtual body and the properties associated with this body by the user/wearer.

The plasticity of one's own body schema opens up far-reaching possibilities for therapies, e.g., in the treatment of chronic pain, or in eating disorders such as obesity and anorexia, which, in indicated cases, also correlate with a disturbance of the body schema. The goals of the project ViTraS (Virtual Reality Therapy through Stimulation of Modulated Body Perception) are the development of the necessary avatar technologies and the design of appropriate therapy concepts. ViTraS utilizes

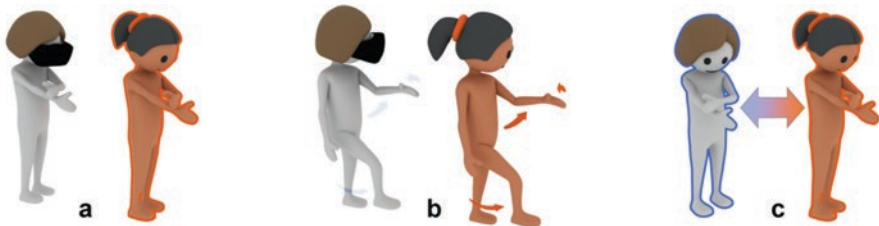


Fig. 9.20 Illustration of the three identified embodiment factors (from left to right): body ownership (a), agency (b) and change (c). The user appears in gray, the avatar in orange. Illustration after Roth and Latoschik (2020). (© ME Latoschik et al. All rights reserved)

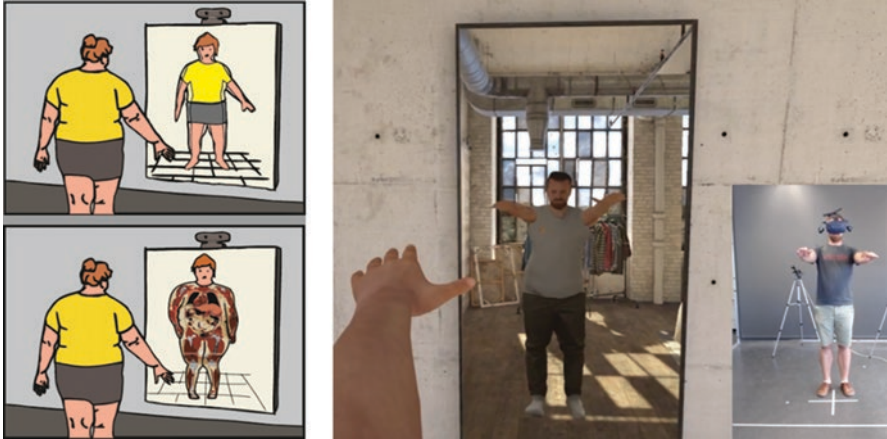


Fig. 9.21 Mirror confrontation with the digital self. Left: Illustrating the consequences of obesity by looking into one's virtual body. Right: A user testing a mirror therapy with a modified (made fatter) avatar. The overlay shows the user from outside the VR surrounded by a camera-based tracking system. (© ME Latoschik et al. All rights reserved)

the plasticity of one's own body schema for therapeutical interventions to help patients that suffer from obesity. The project explores different approaches from the wide spectrum and design space of XR-based therapies, including interactive sketch systems, social VR group therapies, or mirror expositions as shown in Fig. 9.21.

The application scenario of the ViTraS project combines new methods for virtual embodiment, self-(mis-)perception, and faithful avatar reconstruction and its manipulation using digital XR-based interventions. Among other things, the developed solutions increase participation, as they also support distributed therapies for the rampant worldwide health problem of eating disorders, especially obesity, which has far-reaching negative individual as well as overall social and economic consequences. The project strongly demonstrates the great potential of embodiment, especially embodied XR with photorealistic avatars.

The avatars for XR-assisted therapy are created via an optimized photogrammetry-based approach (Wenninger et al. 2020). The method combines 3D reconstruction of geometry and textures with an automated rigging process. As a result, personalized fully animated photorealistic virtual replicas of a user's body are created within a few minutes (see Fig. 9.22). These avatars can then instantly be used with common XR platforms (e.g., Unity 3D). Therapeutically, they can be used to realistically modify and simulate body proportions at the push of a button (or change of a slider). The avatars in Fig. 9.19 were created by the same process. Personalization and photorealism are important to increase the efficacy of XR exposures and the therapeutic interventions. The accompanying user-studies identified personalization and photorealism as strong promoters, especially of the VBO illusion and other important XR factors like presence, acceptance or emotional response (Waltmate et al. 2018).



Fig. 9.22 Photogrammetry system at the Chair of HCI at the University of Würzburg with about 100 SLR cameras for photorealistic 3D reconstruction of user avatars. Left: The multi-camera system that was initially used. Center: A user during the 3D scan process. Right: The result of the reconstructed avatar in a virtual scene. Figure adapted from Latoschik et al. (2019). (© ME Latoschik et al. All rights reserved)

9.12 Virtual Reality for Teaching Literacy to Prisoners

Holger Regenbrecht and Jonny Collins, University of Otago, Dunedin, New Zealand

Numeracy and literacy skills are very low in corrections facilities around the world – New Zealand not being an exception. A large proportion of prisoners are illiterate to a degree that their reading skills do not allow them to participate in normal social life, e.g., being able to comprehend job advertisements or to write a job application. Hence, when released from prison they often cannot reintegrate successfully into society and the chances are that they will end up in criminal activity again. This negative cycle can be broken by, for example, giving prisoners better opportunities to learn how to read and write.

While in prison, prisoners' motivation to learn is usually much lower than with average people outside prison – for many, complex reasons. Classes in literacy are offered within the prison, but in rather traditional classroom settings, i.e., front of class teaching using standard literacy teaching methods. For some prisoners, those settings have positive effects, but many drop out of classes or do not fully engage in learning. The question arises: How to motivate and engage prisoners in literacy learning? Immersive Virtual Reality (VR) might be one promising vehicle for this – at least it is new and potentially exciting for a number of prisoners; for many it is probably their first encounter with such technology.

The Methodist Mission South, a provider of learning services to our local corrections facility, approached us at the Otago University Human-Computer Interaction

(HCI) lab about developing a VR system that can be used for literacy training with prisoners. This task is not without challenges (McLauchlan and Farley 2019): Which technology can be used within a prison? Which virtual environment is exciting and motivating enough to carry the literacy learning task? How to test and evaluate solutions and how to bring them sustainably into the prison environment? We addressed all of those challenges and developed a prototypical system, the “Virtual Mechanic”, which was tested in a lab and in the prison environment, and handed over to a commercial partner for product development and market introduction (Collins et al. 2020).

For inherent reasons, corrections facilities are closed off from the rest of the societal environment. Being allowed to bring a VR system comprising a head-mounted display, a high-end computer and plenty of needed wiring and peripherals requires a huge amount of willingness, motivation and constructive cooperation from corrections facilities staff. The primary concerns of staff include outside communications potential, access to unmediated content and any other type of unauthorized behavior that could be facilitated by the technology. Prisoners are highly creative when it comes to exploiting the materials around them for their own purposes; therefore, what comes in and out of the facility is highly regulated.

Because our main focus was on how to motivate and engage prisoners in literacy learning, we tried to develop a virtual environment which aligns with the existing interests of prisoners. We learned that a common interest amongst prisoners is automotive engineering and cars in general. We selected this common interest as our context, and we built an environment that simulates a car workshop. We took 360° panoramic photos of an existing car workshop, stitched them together, and used this as a background (Fig. 9.23, left) including ambient workshop noise. We explored other environments as a context for learning; however, the remaining most common interests extracted from prisoners were not ethically viable.

Throughout the stages of development, an Oculus Rift HMD was used as the visual medium. During the prototyping stage, we opted for an Xbox controller combined with gaze-based selection to allow users to interact with the environment. In



Fig. 9.23 Virtual environment with panoramic environment (left), virtual brake system broken apart showing (1) syllabic version of a word as a voice reads it aloud for the user (middle), and (2) an active task in which a user attempts to complete rhyming words (right). Tasks are embedded in the context of the environment. (© H Regenbrecht, J Collins. All rights reserved)

this way they could explore the different virtual components and activities that were available. In the later commercial development iteration, the Oculus Touch controllers were used to enable interactions with the virtual world. Compared to the prior gaze-based approach combined with an Xbox controller, Oculus touch controllers lead to a more embodied experience, as users' real hand movements are mapped directly into the environment for interaction. This is a more intuitive form of interaction and can therefore lead to higher levels of engagement.

The actual task we chose was to disassemble and assemble the brakes of a virtual car. Therefore, we introduced a virtual car model with detailed parts modeled for the front disc brake which have been animated in a way to step-by-step reveal the inner structure of the brake. This task was then used as the medium to deliver literacy skills training by giving (interactive) instructions with words. The verbal instructions have been given in three different ways: displayed as words next to the parts of the virtual brake, decomposed into syllables, and read aloud by a computer-generated voice (Fig. 9.23, middle). In addition, we also developed some word rhyming exercises as part of the instructions in a multiple-choice, quiz-like style (Fig. 9.23 right).

Due to the prototypical nature of the application and therefore the lack of actual content during prisoner exposures to date, tangible learning gains have been difficult to evaluate empirically. However, we have gained some insights from our sessions. For instance, trust emerged as an issue with some prisoners, as wearing an HMD meant impeding their view of the real-world environment, which was shared with a small number of other prisoners. Issues arose regarding exposure times, as some prisoners' attention spans and patience levels are more volatile. We also found that a self-directed lesson approach is desirable, as outside intervention reduces a user's momentum and presence in/engagement with the system. The project is currently in the hands of the commercial sector, where it is in continued development. Hopefully, more robust evaluations of the application's educational impact will be conducted soon and can eventually lead to wider dissemination.

The entire process of research and development of this prototype application has been a very enlightening exercise for us. Everyone involved saw this as a clear step forward, especially the prisoners themselves. Virtual Reality carries a lot of potential for delivering training in those kinds of challenging environments. Despite the current lack of content and inability to robustly measure learning outcomes, collectively we could show that implementing VR-based, contextual learning applications in a prison can be done. The idea to "piggy-back" a less engaging task, here literacy training, on a more exciting and motivating task, here immersive VR car maintenance, seems to work well. Whether this approach will lead to transferrable results for the prisoners when leaving prison is still to be shown. VR has the potential to make a real difference here.

References

References for Sect. 9.1

- Doerner R, Göbel S, Effelsberg W, Wiemeyer J (2016) Serious games – foundations, concepts and practice. Springer, Cham
- International Organization for Standardization ISO (2013) Space systems – definition of the Technology Readiness Levels (TRLs) and their criteria of assessment. <https://www.iso.org/standard/56064.html>. Accessed on 2 Feb 2021

References for Sect. 9.2

- Berg LP, Vance JM (2017) Industry use of virtual reality in product design and manufacturing: a survey. *Virtual Reality* 21(1):1–17
- Tesch A, Doerner R (2020) Expert performance in the examination of interior surfaces in an automobile: virtual reality vs. reality. In: Proceedings of ACM MultiMedia '20, Seattle, WA
- Zimmermann P (2008) Virtual reality aided design. A survey of the use of VR in automotive industry. In: Product engineering. Springer, Cham, pp 277–296

References for Sect. 9.3

- Carter L, Paroz AWL, Potter LE (2018) Observations and opportunities for deploying virtual reality for passenger boats. In: Extended abstracts of the 2018 CHI conference on human factors in computing systems (CHI EA '18), paper LBW118. ACM, New York, pp 1–6
- Haeling J, Winkler C, Leenders S et al (2018) In-car 6-DoF mixed reality for rear-seat and co-driver entertainment. In: Proceedings of IEEE conference on virtual reality and 3D User Interfaces, pp 757–758
- McGill M, Williamson J, Ng A et al (2020) Challenges in passenger use of mixed reality headsets in cars and other transportation. *Virtual Reality* 24:583–603

References for Sect. 9.5

- vGIS (2021) Homepage of vGIS Inc. <http://www.vgis.io>. Accessed on Jan 2021

References for Sect. 9.6

- Henry D, Furness T (1993) Spatial perception in virtual environments: evaluating an architectural application. In: Proceedings of IEEE virtual reality annual international symposium, pp 33–40
- Thomas BH et al (2011) Spatial augmented reality support for design of complex physical environments. In: IEEE international conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), pp 588–593

References for Sect. 9.7

- Kuhlen TW (2020) aixCAVE at RWTH Aachen University. <https://www.vr.rwth-aachen.de/page/10/>. Accessed on 1 Nov 2020
- Kuhlen TW, Hentschel B (2014) Quo Vadis CAVE? *Computer Graphics Appl* 34(5):14–21

References for Sect. 9.8

- Heinrich C, Cook M, Langlotz T, Regenbrecht H (2020) My hands? Importance of personalised virtual hands in a neurorehabilitation scenario. *Virtual Reality*. <https://doi.org/10.1007/s10055-020-00456-4>
- Hoermann S, Ferreira dos Santos L, Morkisch N, Jettkowski K, Sillis M, Devan H, Kanagasabai PS, Schmidt H, Krüger J, Dohle C, Regenbrecht H, Hale L, Cutfield NJ (2017) Computerised mirror therapy with augmented reflection technology for early stroke rehabilitation: clinical feasibility and integration as an adjunct therapy. *Disability Rehabilitation* 39(15):1503–1514
- Regenbrecht H, Franz EA, McGregor G, Dixon BG, Hoermann S (2011) Beyond the looking glass: fooling the brain with the augmented mirror box. *Presence Teleop Virt* 20(6):559–576
- Regenbrecht H, Hoermann S, McGregor G, Dixon B, Franz E, Ott C, Hale L, Schubert T, Hoermann J (2012) Visual manipulations for motor rehabilitation. *Comput Graph* 36(7):819–834

References for Sect. 9.9

- Arif AS, Sylla C, Mazalek A (2017) Effects of different types of correctness feedback on children's performance with a mobile math app. In: *Proceedings of IEEE international conference on systems, man, and cybernetics (SMC)*, pp 2844–2849
- Blair KP (2013) Learning in critter corral: evaluating three kinds of feedback in a preschool math app. In: *Proceedings of 12th international conference on interaction design and children*, pp 372–375
- Shang X, Kallmann M, Arif AS (2019) Effects of correctness and suggestive feedback on learning with an autonomous virtual trainer. In: *Proceedings of 24th international conference on intelligent user interfaces, companion*, pp 93–94
- Yu T, Huang J, Chang Q (2020) Mastering the working sequence in human-robot collaborative assembly based on reinforcement learning. *IEEE Access* 8:163868–163877

References for Sect. 9.10

- Deladisma A, Cohen M, Stevens A, Wagner P, Lok B, Bernard T, Oxendine C, Schumacher L, Johnsen K, Dickerson R, Raij A, Wells R, Duerson M, Harper J, Lind D (2007) Do medical students respond empathetically to a virtual patient? *Am J Surg* 193(6):756–760
- Johnsen K, Lok B (2008) An evaluation of immersive displays for virtual human experiences. In: *Proceedings of IEEE virtual reality 2008*, pp 133–136
- Johnsen K, Raij A, Stevens A, Lind D, Lok B (2007) The validity of a virtual human experience for interpersonal skills education. In: *Proceedings of SIGCHI conference on human factors in computing systems*. ACM Press, New York, pp 1049–1058

- Kotranza A, Deladisma A, Lind D, Pugh C, Lok B (2008) Virtual Human + Tangible Interface = Mixed Reality Human. An initial exploration with a virtual breast exam patient. In: Proceedings of IEEE virtual reality, pp 99–106
- Raij A, Lok B (2008) IPSVIZ: an after-action review tool for human-virtual human experiences. In: Proceedings of IEEE virtual reality, pp 91–98
- Robb A, White C, Cordar A, Wendling A, Lampotang S, Lok B (2014) A qualitative evaluation of behavior during conflict with an authoritative virtual human. In: Intelligent virtual agents. Springer, Cham, pp 397–409

References for Sect. 9.11

- Latoschik ME, Lugin JL, Habel M, Roth D, Seufert C, Grafe S (2016) Breaking bad behavior: immersive training of class room management. In: Proceedings of ACM conference on Virtual Reality Software and Technology VRST, pp 317–318
- Latoschik ME, Kern F, Stauffert JP, Bartl A, Botsch M, Lugin JL (2019) Not alone here?! Scalability and user experience of embodied ambient crowds in distributed social virtual reality. *IEEE Trans Vis Comput Graph* 25(5):2134–2144
- Roth D, Kullmann P, Bente G, Gall D, Latoschik ME (2018) Effects of hybrid and synthetic social gaze in avatar-mediated interactions. In: Proceedings of IEEE International Symposium on Mixed and Augmented Reality (ISMAR), pp 103–108
- Roth D, Latoschik ME (2020) Construction of the virtual embodiment questionnaire (VEQ). *IEEE Trans Vis Comput Graph* 26(12):3546–3556
- Waltemate T, Gall D, Roth D, Botsch M, Latoschik ME (2018) The impact of avatar personalization and immersion on virtual body ownership, presence, and emotional response. *IEEE Trans Vis Comput Graph* 24(4):1643–1652
- Wenninger S, Achenbach J, Bartl A, Latoschik ME, Botsch M (2020) Realistic virtual humans from smartphone videos. Proceedings of ACM Symposium on Virtual Reality Software and Technology (VRST). <https://doi.org/10.1145/3385956.3418940>
- Yee N, Bailenson JN, Ducheneaut N (2009) The Proteus effect: implications of transformed digital self-representation on online and offline behavior. *Commun Res* 36(2):285–312

References for Sect. 9.12

- Collins J, Langlotz T, Regenbrecht H (2020) Virtual reality in education: a case study on exploring immersive learning for prisoners. In: Adjunct Proceedings of International Symposium on Mixed and Augmented Reality (ISMAR '20)
- McLauchlan J, Farley H (2019) Fast cars and fast learning: using virtual reality to learn literacy and numeracy in prison. *J Virtual Worlds Res* 12(3)

Chapter 10

Authoring of VR/AR Applications



**Wolfgang Broll, Florian Weidner, Tobias Schwandt, Kai Weber,
and Ralf Doerner**

Abstract This chapter deals with the authoring of VR and AR applications. The focus here is on the use of authoring tools in the form of software development kits (SDKs) or game engines. First, the actual authoring process will be briefly discussed before selected authoring tools for VR and AR are reviewed. Subsequently, the authoring process and the use of the tools will be illustrated through typical case studies. The other chapters of this book deal with the fundamentals and methodologies of VR and AR. These are generally applicable over a longer period. In contrast to this, this chapter looks at some very specific authoring tools and the authoring process based on them, which can inevitably only represent a snapshot in time. Features, releases and availability of these tools can change at short notice, so that individual sections may no longer be up to date when this book is in press. To take this aspect into account, the case studies listed here are stored in an online repository, where they are regularly updated to reflect the latest versions of the authoring tools and runtime environments.

10.1 Supporting Authors

The authors of a VR/AR application are confronted with a wide range of different tasks, which together require many individual skills. These include, for example, programming skills, knowledge of real-time computer graphics and image processing, human–machine interface design skills and usability know-how. They also often require knowledge of the generation of VR/AR assets (e.g., 3D models, textures and sounds), special algorithms and methods (like collision detection in

Dedicated website for additional material: vr-ar-book.org

W. Broll (✉)

Department of Computer Science and Automation/Department of Economic Sciences
and Media, Ilmenau University of Technology, Ilmenau, Germany
e-mail: wolfgang.broll@tu-ilmenau.de

© The Author(s), under exclusive license to Springer Nature
Switzerland AG 2022

R. Doerner et al. (eds.), *Virtual and Augmented Reality (VR/AR)*,
https://doi.org/10.1007/978-3-030-79062-2_10

371

virtual worlds or 3D registration in AR), and properties as well as software-engineering aspects, e.g., for connecting special VR/AR input and output devices. On top of that, knowledge and skills regarding the application itself are indispensable: in the case of a VR application for training minimally invasive surgery, for example, this requires knowledge of medicine as well as didactics. Hardly a single author will possess all these skills. Therefore, it is crucial to enable and foster interaction between different authors, but also to support each author individually in her tasks. Adequate support for authors is not only the key to an efficient and high-quality realization of VR/AR applications, but also allows particular people to participate actively in the creation process in the first place. Thus, providing good support not only enables the necessary experts to be involved, but additionally helps to reduce costs. In many cases, this is what makes the use of VR/AR applications technically and economically feasible.

Common support comes in the form of programming libraries together with programming tools (Software Development Kit, SDK) or programming interfaces (Application Programming Interface, API) to software packages and systems. For example, ARToolKit (Kato and Billinghurst 1999), which has been available as open source software since 2001, gave a boost to the use of AR at that time, as application developers could simply rely on an existing implementation of an essential foundation of AR, namely the realization of stable (in this case marker-based) tracking or working camera calibration, without having to deal with the corresponding concepts and algorithms themselves beforehand.

Another form of support is the use of software tools supporting various authoring tasks. Section 10.2 presents examples of such tools. But tools that are not specific to VR/AR are also used, e.g., image editing programs like *Photoshop* or *The Gimp* for creating textures and (utilizing special plugins) normal maps, or 3D modeling and animation tools like *Blender* or *3ds Max*. Typically, multiple tools are used (in parallel or sequentially) to address different aspects of VR/AR application development but also to serve different authoring groups. One speaks of a *tool chain* when one tool exports data that is then imported by another tool for further processing. Such exchange of data can be a source of problems if there is no common usable or open data format, or if the import and export processes involve a loss of information. For example, a 3D object such as an automobile may be designed in a CAD tool, but the exported CAD data is not directly usable for a VR tool and must be preprocessed (e.g., by reducing complexity). For this purpose, corresponding *conversion tools* (often also in the form of plug-ins) are available for existing CAD tools.

Authoring processes need to be carefully planned so that authors can collaborate efficiently and without any frictional losses using tools, APIs and SDKs. Development environments that integrate a variety of support functions into a single tool are popular. This often implicitly prescribes the authoring process, at least in part. Increasingly, *game engines*, development and corresponding runtime environments originally intended for the creation of computer games, are also being used for VR/AR applications. The processes for creating a 3D game world and a virtual world are fundamentally similar in many respects. In addition, modern game engines offer mechanisms such as plug-ins or APIs to support the special needs of VR/AR

applications, such as the use of certain VR controllers or the rendering of pre-distorted stereo images for HMDs (see Sect. 5.2.3).

Unfortunately, there is neither a universal authoring process nor a single tool that would be sufficient on its own to support the creation of arbitrary VR/AR applications. Rather, authoring processes and their support must be individually assessed and determined for each VR/AR application. Criteria for the selection include the existing skills of the authors, functionality offered, quality, performance, maturity, licensing costs, licensing model applied, quality of documentation and tutorials, and the availability of an active and responsive user community.

10.2 Foundations of Authoring Software

In this section, we will present two examples of popular game engines used for the development of VR applications, i.e., Unity (2021) and Unreal Engine (2021). Furthermore, we will present two frameworks used for developing AR applications: ARKit (2021) and ARCore (2021). As they are largely limited to AR-specific aspects, they are usually used in conjunction with game engines. While many game engines, including the two presented here, support a variety of different platforms (PC/Mac, consoles, mobile devices, web browsers), the two AR frameworks are currently exclusive to one of the major mobile platforms (ARCore: Android, ARKit: iOS).

Two of the most popular game engines are currently Unreal Engine (UE) and Unity. These game engines offer similar core functionality. Both support level design, realistic rendering, multiplayer applications, artificial intelligence, user interfaces, physics simulations, global illumination, animations and more. Also, both platforms offer distribution platforms for assets and applications (Unity: asset store; Unreal Engine: marketplace). After installing and starting one of these engines, both allow for the setup of a project. Such a new project can be based on a template or it can be a new and empty project. If we open such a project, both applications show a similar layout. It contains an area showing details and letting us change preferences, one area that lists all objects in our current scene, and an area that shows us all of the game assets (3D models, sounds, animations, textures, etc.) included in this project, which usually looks like the Windows Explorer (see Fig. 10.1).

However, if we have a closer look at both engines, we will notice some differences. For example, while both engines allow us to view their source code and by that, understand how they work, only Unreal Engine also allows us to modify the source code and change engine functionality (as of March 2021).

Both applications allow us to create and design levels by dragging objects into our scene following the what-you-see-is-what-you-get principle (WYSIWYG). However, if we need more complex functionality, there are again differences between them: Unity supports common programming languages like Javascript and C# to add functionality. Unreal Engine uses Blueprints and C++. Blueprints rely on a visual programming approach. Both C++ and Blueprints can be combined and are



Fig. 10.1 Unreal Engine (top) and Unity (bottom) user interfaces of the development environment

almost always interchangeable. However, the usage of C++ is in complex cases more straightforward and less troublesome.

In the past, support for novel AR and VR hardware was often included in Unity before Unreal Engine supported them. Meanwhile, both engines support almost all common hardware and VR headsets, AR headsets, and ARKit and ARCore. Currently, VR devices based on OpenVR (2021) are slightly better supported by Unreal Engine, as not all hardware manufacturers already provide support for a recent redesign of Unity’s plugin system. However, except for some extra effort, hardware support is basically the same in UE and Unity.

When developing with Unity, the community and forums are bigger compared to Unreal Engine. This might be especially important for people who are new to these engines, as a quick search on the internet can easily solve many problems. While the community of Unreal Engine is growing, the engine is not known for being beginner-friendly as it is quite complex and overwhelming in the beginning.

Finally, while free (except for royalties for sold applications), Unity also offers some Plus and Pro subscription plans. Unreal Engine is free to use until you sell your work (royalties).

In the end, selecting an engine for a new project depends on many factors (supported hardware, community, programming languages, cost, access to source code, etc.). Also, a very important aspect is the prior knowledge and skills of the team members working on the VR/AR application. Table 10.1 summarizes these aspects.

The following two subsections each explain the use of these engines for the authoring process.

10.2.1 Unity

Unity is a popular game engine that supports a variety of VR/AR hardware. This includes almost all VR HMDs as well as current AR HMDs like the Microsoft HoloLens 2 or the Magic Leap 1. Development for these devices is fully supported in Unity. Further, we can develop utilizing AR frameworks like ARKit and ARCore. In addition to that, Unity is known for being beginner-friendly and easy to use.

How It Works

Unity applies the entity-component model. Here, every object in a scene (or in the game) is an entity and has a relation to one or more other entities. In Unity, entities are also called *GameObjects*, *Prefabs* or *Scripts*. *Scripts* contain source code in C# or Javascript and allow for adding functionality and behaviors to an object. A

Table 10.1 High-level comparison of the game engines Unity and Unreal Engine regarding the development of AR and VR applications

	Unreal Engine	Unity
Community	☹	☺
Price	☺	☹
Beginner friendly	☹	☺
Source Code Access	Yes (modifiable)	Yes (read-only)
Programming languages	C++, Blueprint	C#, Bolt
VR Support	☺	☹
AR HMD Support	☺	☺
Mobile AR Support	☺	☺

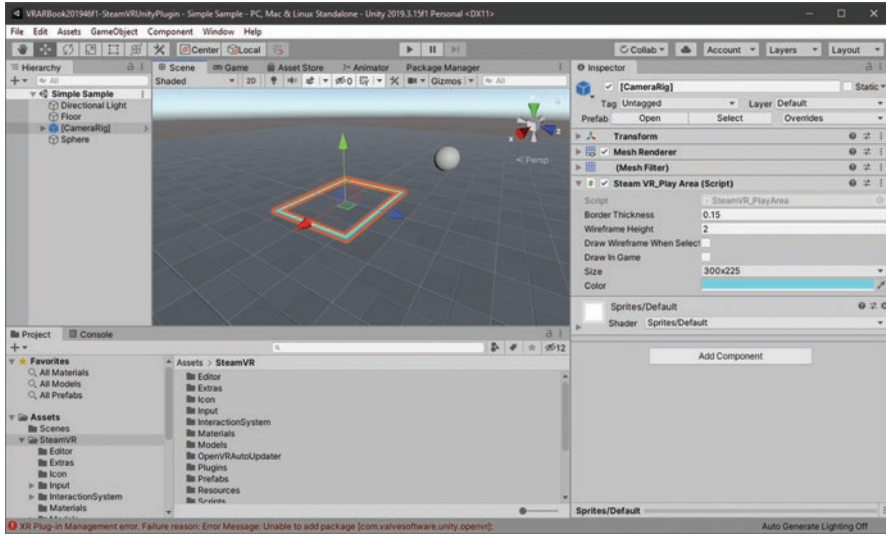


Fig. 10.2 Full view of the Unity editor. The *Inspector* shows a *GameObject* “CameraRig” with an attached *Script* called “Steam VR_Play Area”. The *Hierarchy* window shows the hierarchical organization of the *GameObjects* of a scene. The *Scene* view shows a 3D preview of the current level and the *Project* view all files of the project

GameObject is a collection of one or more assets like 3D models, sounds, textures and more. A *Prefab* is a special type of *GameObject*. It encapsulates several *GameObjects* and in that way eases reusability and sharing of content. Both *Prefabs* and *GameObjects* are hierarchically structured. For *Scripts*, developers can use typical concepts known from programming, like inheritance. *Scripts* can be added to *GameObjects*, so that we can add animations, behaviors, other *GameObjects*, artificial intelligence or other effects. Together, all *GameObjects* comprise our scene. Figure 10.2 shows the Unity development user interface for a sample scene, with hierarchy window, scene view, inspector window and the project explorer.

A scene in Unity represents a level in our application. Further, a scene in Unity is represented by a scene graph. The scene graph is a hierarchical data structure that contains all the *GameObjects* of a single scene. In the scene view in the center of the Unity application, we can see and also arrange all elements of our scene, and design the level according to our liking or external requirements. To modify not only the position and rotation but also other properties of a *GameObject*, the inspector on the right-hand side provides us with a variety of settings: among others, settings of attached scripts, behavior settings, positional attributes and lighting options. To get a preview of our scene, we can switch to the *Game* view. Here, we see the game in a pre-final version – it looks like the exported version. The project explorer contains all the scripts, assets, and prefabs that we have added to our project. It also allows us to manage them. The console lists all errors and warnings that arise during development and is a helpful tool when fixing errors.

VR/AR Development with Unity

Creating a VR application with Unity requires, as usual, that the drivers and supporting software of the VR headset like the Oculus software or SteamVR are installed on the system. These programs manage the communication between Unity (and any other VR application) and the HMD, tracking system and controllers. Without them, development for a VR headset is not possible. Assuming the necessary software is installed, we can open a Unity project. Depending on our VR HMD, we most likely need either a dedicated plugin from Oculus or from Valve. The latter uses OpenVR and can be used for most SteamVR-based headsets. While the Oculus plugin is already available within Unity, the OpenVR plugin needs to be installed or downloaded from Github (SteamVR Unity Plugin 2021). Similarly, development for VR HMDs that are based on Windows Mixed Reality require the installation of the Windows Mixed Reality Toolkit (MRTK Release 2021). If you do not depend on external plugins, you can directly enable your device for Unity in the *Edit* → *Project Settings* as shown in Fig. 10.3. The Unity documentation supports developers when setting up new projects or upgrading older projects to VR (Unity XR 2021).

All plugins – Oculus, StreamVR and Windows Mixed Reality – can be used as a starting point but can also be integrated into a project at a later stage in development. In addition to the core functionality, they offer sample scenes, *prefabs* and

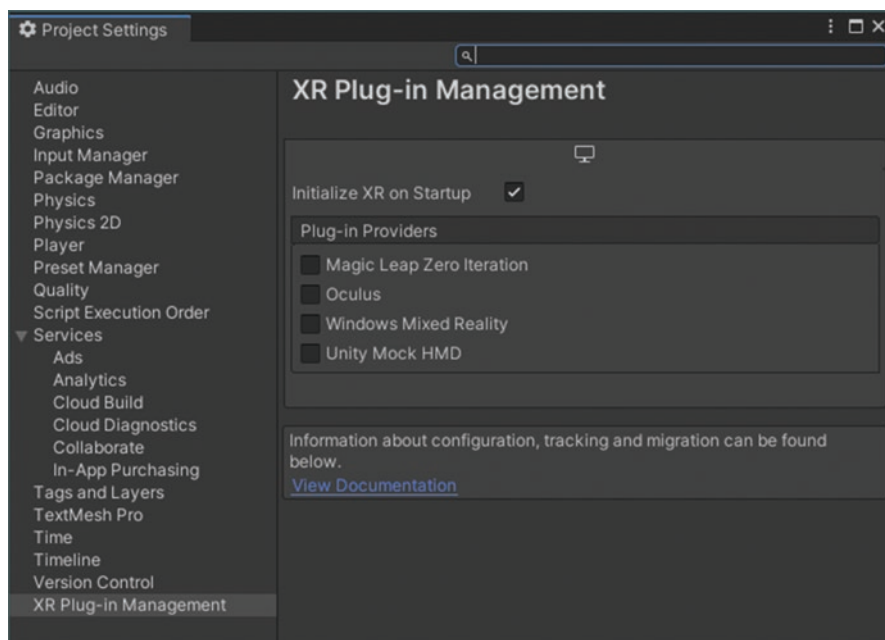


Fig. 10.3 Project settings showing the supported XR frameworks of Unity 2020.1.16f1. Only Oculus devices are directly supported without the need for additional external tools. Windows Mixed Reality requires an additional plugin

scripts that support developers by offering code examples for, e.g., basic navigation and interaction techniques.

When developing for AR, the respective plugins for devices like Microsoft's HoloLens, the Magic Leap 1 or recent smartphones using ARKit and ARCore need to be downloaded and installed. After the installation procedure for these third-party tools has been completed, development for AR applications becomes similar to that of VR and traditional desktop applications using the scene view and the game view. It is noteworthy that Unity also offers MARS (2021) a WYSIWYG editor for MR and AR. This editor promises to streamline the development of AR applications by integrating sensor data from devices into the development process.

Summary

In summary, Unity is a well-suited tool for developing AR/VR applications. All assets are organized in a scene graph and represented by different types of *GameObjects*. Behaviors can be added using scripts written in C# or Javascript. Such *GameObjects* and *Scripts* can be grouped to *Prefabs* to foster reusability and interchangeability. Several device manufacturers also provide prefabs for their HMDs to ease development for VR and AR devices. Among others, Unity supports the popular VR headsets from Oculus, HTC, Valve and HP. It further supports AR devices like the Microsoft HoloLens 2 and Magic Leap 1. AR development for smartphones is also supported via the integration of ARKit and ARCore. For AR development, Unity also offers a rich authoring tool called MARS.

10.2.2 Unreal Engine

Unreal Engine (UE) is the successor to the Unreal Development Kit. Generally, it offers similar features as Unity. The current version as used in this chapter is UE4 (as of March 2021). However, a tech demo of UE5 was presented in 2020. The development of applications with UE is quite comfortable. While the development of 2D or 2.5D games is also supported, the entire engine has been designed and optimized for the realization of 3D first-person applications.

How It Works

UE provides two alternative ways to develop applications: traditional C++ programming and a visual programming approach called Blueprints. Blueprints are based on nodes and connections between those nodes. A node represents a function like move, get or set, or operations such as loops or if-statements. Each node has one or several pins. The pins are entry or exit points for connections. Developers can place nodes (squares with rounded corners and a header with different colors) in the

editor. Then, connections (white lines) are connected to the pins of nodes (either white or colored triangles at the nodes) to create the data and control flow. Adding nodes, and thus functionality, is supported by a context-sensitive auto-completion. As with other programming languages, developers can debug Blueprints using a debugger and can also copy and paste them between files. Figure 10.4 shows an example Blueprint from a VR example that sets the tracking origin depending on the connected HMD.

The main disadvantage of this variant of programming is that large Blueprints are prone to getting confusing and cluttered. Here, using advanced concepts like inheritance, interfaces and Blueprint libraries can help to declutter Blueprints. Further, it is important to know that developers can combine both methods, C++ and Blueprints. For beginners, Blueprints offer an easier entry point when working with UE. Unfortunately, many functions provided by UE as Blueprints are not similarly easy to use in C++. However, C++ provides more flexibility. It is noteworthy that internally, UE translates every Blueprint into C++ code. That means that using them does not result in performance loss. Microsoft Visual Studio is recommended when developing with C++.

Similarly, to Unity, UE follows the entity-component model. Instead of *GameObjects*, it uses *Actors*. Unity's scripts correspond to Blueprints, other *actors*, or C++ code files in UE. *Actors* are hierarchically organized in a scene graph and can be directly edited. *Assets* can be viewed, moved and otherwise organized using the file explorer of UE.

UE offers several preview modes for testing. Developers may test their game in the editor (*Play-in-Editor*), start it as a standalone game (*Play-as-Standalone-Game*) to simulate a build version of the game, or build and start a final version of the game via *Launch*. In addition to that, it offers the possibility to test the game using a simulated VR HMD without having access to a real one.

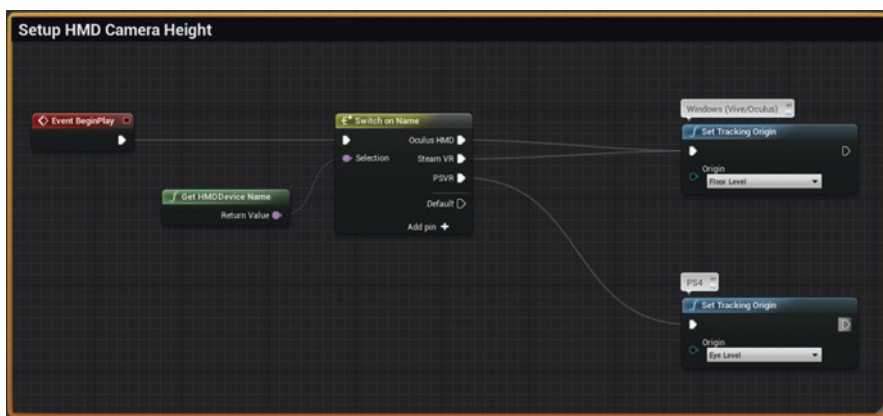


Fig. 10.4 Example of a Blueprint. This Blueprint sets the tracking origin of the application's HMD during start-up

Developing VR/AR Applications

After downloading the *Epic Games Launcher*, we can install the most recent version of Unreal Engine 4. If we intend to develop with C++, it is also possible to install debugging symbols. They ease the debugging process by providing more detailed error messages in case of an engine error. Next, we can open UE and create a new project. The project launcher already offers a template ready for VR applications (see Fig. 10.5). If possible, developers should base their applications on this template. If this is not possible – for example when upgrading from an old project – the template can still act as a useful reference. This template offers (in UE4.25) simple interaction and navigation methods that can be used by developers for their own applications.

Usually, no additional plugins need to be installed. Support for Oculus, HTC, Valve, Windows Mixed Reality (2021), HoloLens 2 and Magic Leap 1 is already integrated into UE. This also extends to the AR frameworks ARKit and ARCore. However, vendor-specific SDKs (e.g., SteamVR) still need to be installed.

If developers have no HMD at their disposal, they can enable a virtual HMD via *Edit* → *Plugins* → *Virtual Reality* → *SimpleHMD*. In this way, the application runs in VR preview mode and content is displayed in such a way that is like a real VR HMD. The key here is that the same rendering algorithms are used. That means that UE renders separate images for each of the eyes – however, without a dedicated HMD-specific barrel distortion (see Sect. 5.2.3). This simplifies level design and testing. Figure 10.6 illustrates this view.

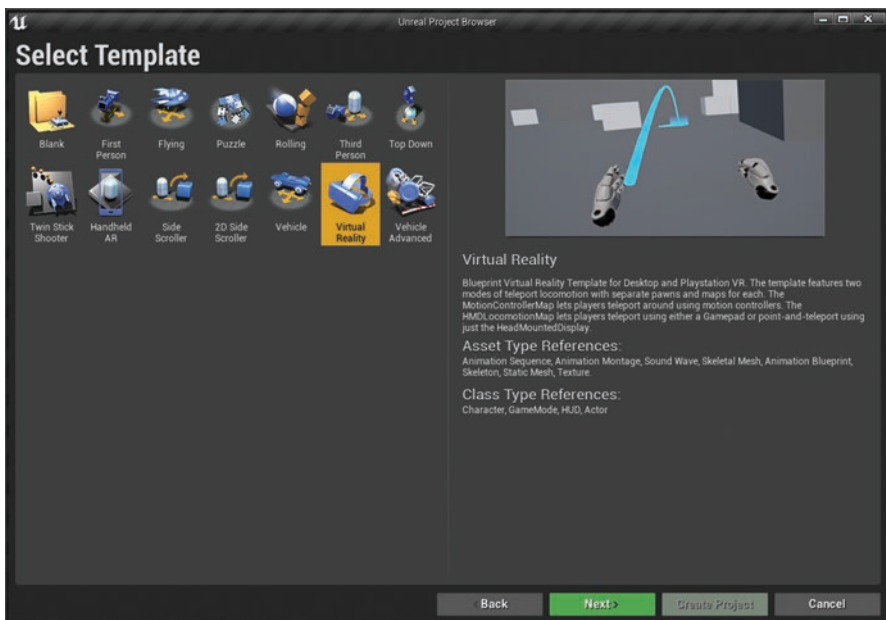


Fig. 10.5 Template browser of UE4 (the VR template is highlighted)

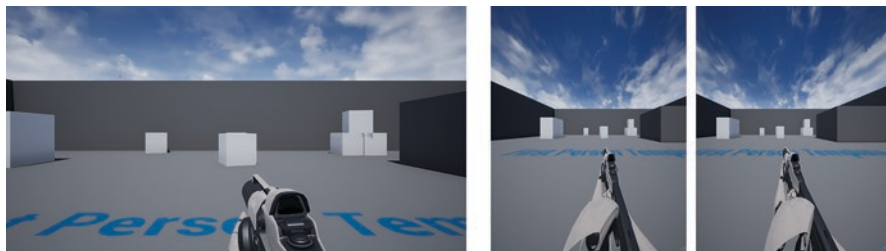


Fig. 10.6 The left image shows the default view of a game. The right image shows the preview using the *SimpleHMD* plugin

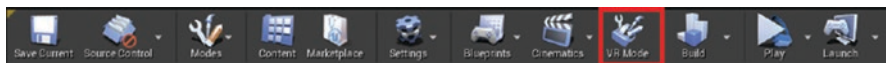


Fig. 10.7 *VR-Edit-Mode* of Unreal Engine

UE4 offers an immersive modelling/authoring mode. This *VR-Edit-Mode* allows designers and developers to create their level while wearing a VR HMD. In this mode, they can manage their actors, e.g., by adding, moving, rotating and scaling them or changing their properties. The advantage of this immersive authoring mode is that the scene can be viewed directly in VR. Thus, there is no need to constantly put the headset on and take it off. Also, it is easier to judge if the scales and distances of the VR scene are appropriate. This may result in more comprehensive level design and a faster development process. Figure 10.7 shows the entry point to this mode.

As mentioned above, UE natively provides support for many popular devices, including those from Oculus and HTC, as well as Windows Mixed Reality headsets. These plugins can be enabled or disabled on demand via the project settings. It also offers native integration for ARKit and ARCore as well as HoloLens 2 and Magic Leap 1. When activated, these plugins enable access to the device via C++ or Blueprints. In addition to the native plugins, further plugins may be added to UE.

Summary

Notably, the Unreal Engine has been designed for first person-games, i.e., games played from an egocentric perspective. It plays to these strengths in the development of AR and VR applications. Applications based on UE consist of *levels* and the *levels* contain *actors*. The *actors* are organized in a *scene graph*. UE natively supports many AR and VR devices, while also allowing the development for additional devices via its plugin system. The engine is free (at least until the lifetime gross revenue of the game exceeds US\$1,000,000), and the source code may be modified at one's own discretion.

10.2.3 *AR Frameworks: ARCore and ARKit*

Both Apple and Google have published their own frameworks in the field of augmented reality (AR), which can be used to develop mobile AR solutions. Both frameworks were specifically adapted to the hardware of recent smartphones. Since both ARCore (Google) and ARKit (Apple) became available at the end of 2017, more and more applications have been published that use AR technologies. This subsection is intended to provide an overview of the basic functionalities as well as the main differences between these two frameworks.

Availability

Both AR frameworks are available on devices with recent hardware and software. ARKit is available on all iPhone and iPad devices with iOS 11 or higher, including iPadOS (Apple 2021). For Android, these are smartphones with the Android 7.0 (Nougat) operating system or higher, as well as other selected devices (Google 2021). Recent data from 2020 shows that the availability of ARKit-enabled devices (1185 million) is noticeably higher than that of ARCore-enabled devices (633 million). Among active users of the frameworks, this difference is even more pronounced, with 950 million for ARKit and 122 million for ARCore (Makarov 2021).

The continued availability and distribution of corresponding hardware is a key factor here. However, only the latest hardware (currently, for example, LiDAR sensors on some Apple devices) offers the possibility of fully exploiting the functions available for AR. Due to the short life cycle of mobile hardware, it can be assumed that by the time this book is published, almost all active smartphones and tablets will be able to use one of the two AR frameworks.

Tracking and Mapping

One of the most important functionalities of an AR framework is tracking, which is based on a SLAM approach in both frameworks (see Sect. 4.3.4). Both frameworks can track the device (camera), planes, faces and 2D images. At the time of writing, ARKit additionally supports 3D object tracking and body tracking. However, the tracking of the camera in particular is crucial for sufficiently good visualization, and both frameworks deliver very good results in this area. This is mainly due to the utilization of the hardware installed in the mobile device. In addition to the camera, acceleration sensors, gyroscopes and partly LiDAR sensors are used. For mapping, both frameworks create a virtual map by detecting and storing features in space. When testing various applications, both can demonstrate robust mapping, although ARKit shows some advantage with respect to fast camera movements.

Reconstruction

In both frameworks the environment is defined by a set of feature points. For this purpose, features are detected in the environment and tracked over several frames, and their position in space is continuously improved. Based on these features it is possible to detect planes in space (plane detection). These planes provide a rough representation of the real environment. Figure 10.8 shows the reconstruction of a vertical plane based on feature points using ARCore. Both frameworks can create vertical as well as horizontal planes. By using LiDAR sensors, even the detection of complex (non-planar) geometries is possible.

Estimation of Environment Light

Both ARKit and ARCore provide a simple estimation of environment light to illuminate virtual objects of an AR scene correspondingly (cf. also photometric registration, Sect. 8.2.2). This estimation is based on the current environment, with each framework supporting different light estimation features (ambient light, specular highlights and reflections). Both support ambient light, which represents the overall diffuse lighting originating from arbitrary directions in the environment. This ambient light detection is provided by determining an intensity value between 0 and 1 as well as a color temperature. For a more sophisticated representation the frameworks offer an illumination detection for the complete HDR environment lighting, which also allows for plausible reflections. For ambient light, both frameworks apply (different) estimation approaches based on neural networks. ARKit uses reconstructed geometry like planes, renders them into a cubemap and completes the latter by a neural network. ARCore, in contrast, uses the current camera image only to

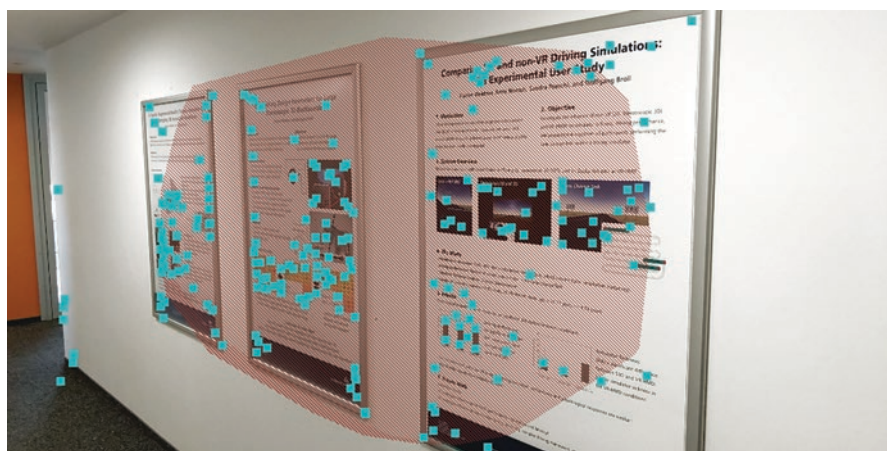


Fig. 10.8 Detection of a vertical plane (red) reconstructed from detected feature points (cyan). (© Tobias Schwandt, TU Ilmenau 2018. All rights reserved)



Fig. 10.9 Illustration of the illumination in ARCore with a 360° HDR ambient illumination in HDR (left) and simple intensity detection (right). Metallic objects in particular benefit from the 360° illumination reconstruction

determine a 360° HDR environment cubemap. In combination with the integrated detection of the main light source, more realistic shadow casts or even specular reflections are enabled, for example, Fig. 10.9 illustrates the difference between using a simple intensity value and an HDR illumination using ARCore.

Summary

While having their individual strengths and weaknesses, both frameworks provide very similar functionality, which will likely further converge in the future. Even supposedly decisive advantages, such as the support of LiDAR in ARKit, can be relativized with the next version.

However, Apple has a clear advantage in terms of market penetration. Any device with the latest iOS or iPadOS operating system can automatically display AR. In contrast, it can hardly be estimated to what extent current devices for Android support ARCore. Nevertheless, it can be assumed that more smartphones with AR support and further sensors will become available for Android. Thus, the question of the right framework remains a question of the target group, personal interests, the devices available and the type of application to be developed.

10.3 Examples of the Creation of VR/AR Applications

In this section, the creation of VR and AR applications will be illustrated using four practical case studies. The first example deals with a VR application for the presentation of CAD data with Unity using the Vive Cosmos. The second example creates an interactive VR application with the Unreal Engine using the Vive Cosmos again. The third example describes an AR application for Microsoft's HoloLens 2. The fourth and final example describes an ARCore application for Android.

Note that these examples can only provide a snapshot of what is currently available and possible. Obviously, a much wider range of game engines, toolkits,

frameworks, and devices exist, which cannot be covered here. We would, for example, have loved to include an example using a non-commercial game engine such as Godot (2021), or to show how easy AR development may be even for non-programmers using RealityKit (2021). We were also unable to include Oculus HMDs due to the limitations imposed by the manufacturer's service conditions. We will closely monitor ongoing developments and update the examples in future editions, but also more frequently in the online repository.

10.3.1 *Making of: Immersive VR Presentation of CAD Files with the Vive Cosmos in Unity*

In this section, we will set up a VR application in Unity. It will allow us to view a CAD model and offer simple interaction with it – we will be able to grab it with a handheld controller and release it again. For this, we assume that your headset is correctly set up and all necessary drivers like *SteamVR* or *Viveport* are installed.

We start with a Unity project, install and open *Unity Hub*, and then create a new project using the *3D template*. This case study is based on Unity version 2019.4.6f1. After loading Unity, we need to install the *SteamVR* Unity Plugin. We can download the package directly from within Unity via the *Asset Store* or at Valve Cooperation (2021). Such packages extend Unity with functionality and content. For this tutorial, we use the *Asset Store*. Search for “Steam VR” and import it to your project. This package contains all the necessary content for Unity to recognize your headset and to communicate with it. It will also work for other headsets that are based on *OpenVR*. In addition to that, it contains some sample content that we can use to realize our case study. Right after clicking *Import*, Unity downloads the package. Then we are offered a list of files for import. By default, all files are selected. Thus, select *Install all* here. One further needs to confirm the necessary settings by selecting *Accept all* in the upcoming dialog. If a project settings window pops up, we can safely close it.

Before we can fully utilize the plugin, we must create input mappings. To do this, we select *Window* → *Steam VR Input*. Unity then asks if we want to use the default input bindings, which we confirm with *Yes*.

Now, we already have a working VR setup. We can open the *Simple Sample* scene which is in *Assets* → *SteamVR* in the project window. The project window is in the lower left part of our application window. Figure 10.10 shows the scene view of this example. If we hit *Play* in our Unity application, *SteamVR* starts up and we can already see a scene in our headset. Also, head-tracking and controller tracking are already working, as shown in Fig. 10.11.

For our example, let us create a new scene. We can do this via *File* → *New Scene*. We name our new scene *MyExample*. In the beginning, it contains a *Directional Light* and *MainCamera*. First, we delete the *MainCamera*. We do not need this *GameObject* as the *SteamVR* plugin already provides us with a setup working with

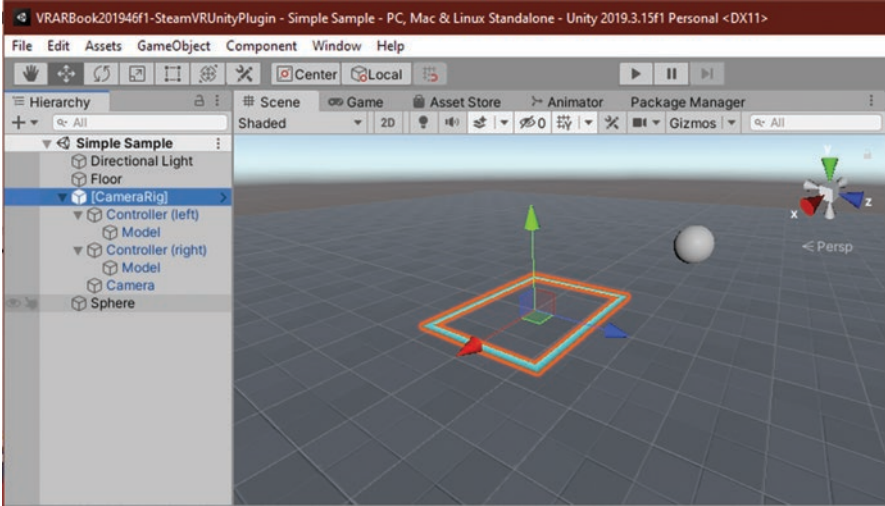


Fig. 10.10 Scene view and hierarchy of the example scene “Simple Sample” in Unity

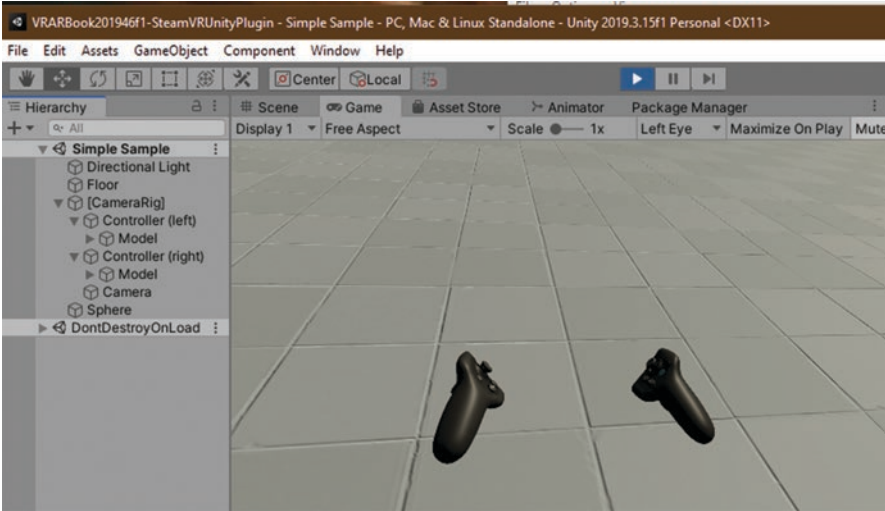


Fig. 10.11 Game view and hierarchy of the example scene “Simple Sample” in Unity after hitting play in Unity

the VR HMD. In the project explorer, we search for *Player* and add this *prefab* to our scene via drag and drop. Inspecting it more closely in the hierarchy panel, we can see that it not only contains a camera but also *GameObjects* for the controllers.

Next, we create a cube with size (0.1, 0.1, 0.1). This cube is going to be our object that we want to grab. To make it grabbable, the SteamVR plugin provides a

script called *Interactable*. In the *Inspector* of our cube, we click *AddComponent* and add this script to it. Next, two options are available: First, the SteamVR plugin provides a script in *Assets/SteamVR/InteractionSystem/Samples/Scripts/InteractableExample.cs*. We can open this script and delete everything that is related to text fields, as we do not need those for our example. Second, we can use the *SimpleGrab.cs* script, which is provided as online material to this book. We now add this script (either the modified *InteractableExample.cs* or the *SimpleGrab.cs*) to our cube. The final composition of our scene is illustrated in Fig. 10.12. Note the two scripts of the cube.

Now, if we start our application, we can grab the cube and move it around. When we release it, it returns to its initial position. In addition to that, the object gets highlighted if our controller (or hand) gets close to it. The grabbed object is highlighted in Fig. 10.13.

To really display a CAD model and interact with it, we first must load our CAD model and then add these scripts to it. However, Unity natively supports only *.fbx*, *.dae* (Collada), *.3ds*, *.dxf* and *.obj* files. For CAD models, it provides a Pixyz (2021) package to load various CAD formats. Having this plugin enabled (for example in the test version), we can add our CAD file as a *GameObject*, add the scripts, and grab and release it the same way we did with the cube. Alternatively, we can try to directly export our CAD model to a file format that is supported by Unity (e.g., *.obj*). Or, we can use a third-party tool like Blender (2021). Using Blender, you can import various formats and export them again to a format that is accepted by Unity.

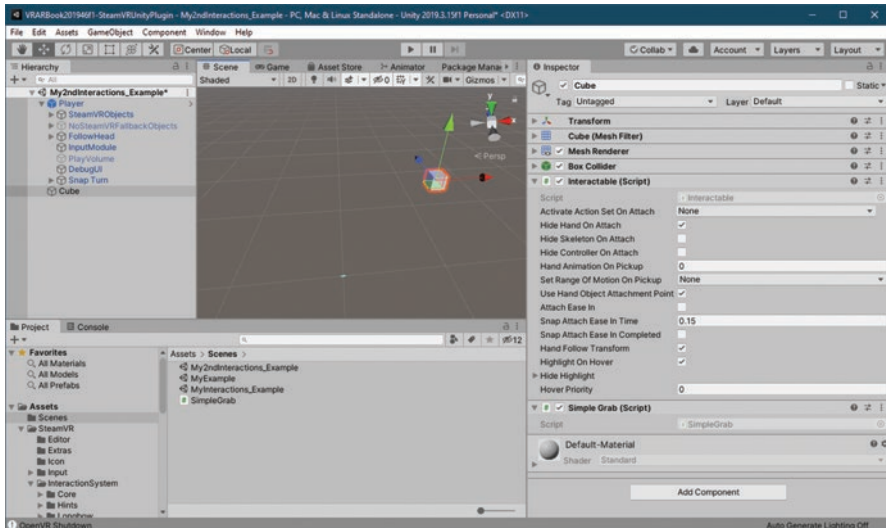


Fig. 10.12 Setup of our newly created scene *MyExample*, showing the player with controllers and a cube in the hierarchy

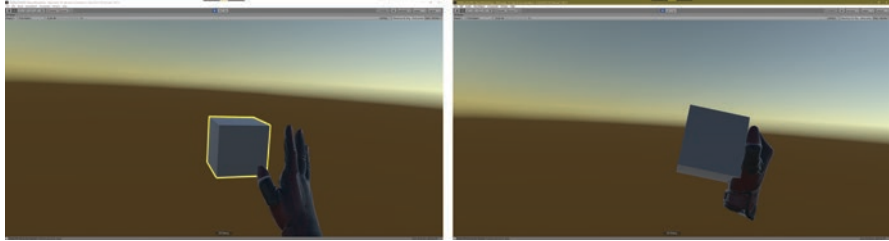


Fig. 10.13 Left: The virtual hand approaches the cube, which is then highlighted. Right: The virtual hand grabs the cube, which now follows the movement of the hand

10.3.2 Making of: Interaction in VR Using the Vive Cosmos and Unreal Engine

This case study will outline how to realize easy interaction techniques like grabbing objects in Virtual Reality using Unreal Engine. For this, we will use the *Vive Cosmos* and the accompanying motion controller. However, this example should work with all Virtual Reality HMDs that use StreamVR. We assume that the drivers and runtime software necessary for using the HTC Vive such as SteamVR and/or Viveport have already been installed on the computer. In this example, we will setup an Unreal Engine project, configure the virtual camera, create some objects and finally implement the grabbing mechanism.

First, we launch the *Epic Games Launcher* and create a new project for Unreal Engine. This case study is based on UE 4.25.4. When creating a new VR application, we recommend using an existing template as a starting point. Unreal Engine offers a template called *Virtual Reality* that has been designed especially for our purpose. It is located under the category *Games* (c.f. Fig. 10.5). In the next screen, we do not have to change anything. Using the template, our project has preconfigured settings and plugins that allow for easy development of VR applications.

After Unreal Engine has loaded the project, we can explore the *Content Browser* in the lower left part of the application window. We first create a new *level* with a right-click into the content browser. A *level* is the equivalent of Unity's scene. When we open this new map (a map in Unreal Engine is similar to a scene in Unity corresponding to a level in a game), we see only a black screen. That is because we don't have any objects in our scene – we can see that by looking at the *World Outliner* in the upper right part of the application window. Let's switch to the *Landscape* mode by clicking *Modes* → *Landscape Mode*. Here we can change, edit and create our level. For now, we just hit *Create*. After that, we switch back to the *Select Mode* via the *Modes* dropdown menu. In the *World Outliner*, we see that we now have a landscape. However, it is still dark. On the left side of the application window, we see the *Place Actors* panel. Here we can switch to *Lights* and drag a *Directional Light* into our scene. This acts as the sun. Now we should be able to see our checkerboard floor that we created earlier as part of the landscape. Because our project is based on the VR template, we can see some predefined objects in our

Content Browser. The *HMDMotionControllerPawn* in *Content* → *VirtualRealityBP* is the item we now drag into our scene. It provides us with the necessary means to use the VR HMD and the controllers in our project. If you select the *HMDMotionControllerPawn* in the *World Outliner*, you can also see a small camera window that shows a preview of the VR view. If we now select the *HMDMotionControllerPawn* in the *World Outliner*, we can see that the *Details* panel just below it shows some options for this game object. If we change the option *Auto Possess Player* from *Disabled* to *Player 0*, we already have our first working example! We can test it directly in VR by selecting the little arrow next to *Play* and then select *VR Preview*. Our app should now be visible in your VR HMD (see Fig. 10.14).

The hands already move, and the view is updated when we turn our head and move around in our play area. If we push the grab button on our controller, the hand changes as well.

The *HMDMotionControllerPawn* is a *Blueprint* (see also Sect. 10.2.2). It encapsulates functionality but can also have a 3D model attached. Almost all game objects in Unreal Engine are *blueprints*. A part of the *HMDMotionControllerPawn Blueprint* is shown in Fig. 10.15.

Nodes with a red header are events. Nodes with a blue header are functions. In this example, *Left Controller* is variable. In plain words, the snippet does the following: If we receive an input action called *GrabLeft*, execute the function *Grab Actor* on the Left Controller and when we release the button, execute *Release Actor*.

Next, we add objects that we can grab with our controllers. For this, Unreal Engine provides another *Blueprint*. This is called *BP_PickupCube* and is in *Content* → *VirtualRealityBP* → *Blueprints*. We can simply drag this cube in our scene. In the game, we can now use the controller to grab the cube and release it again. If we open

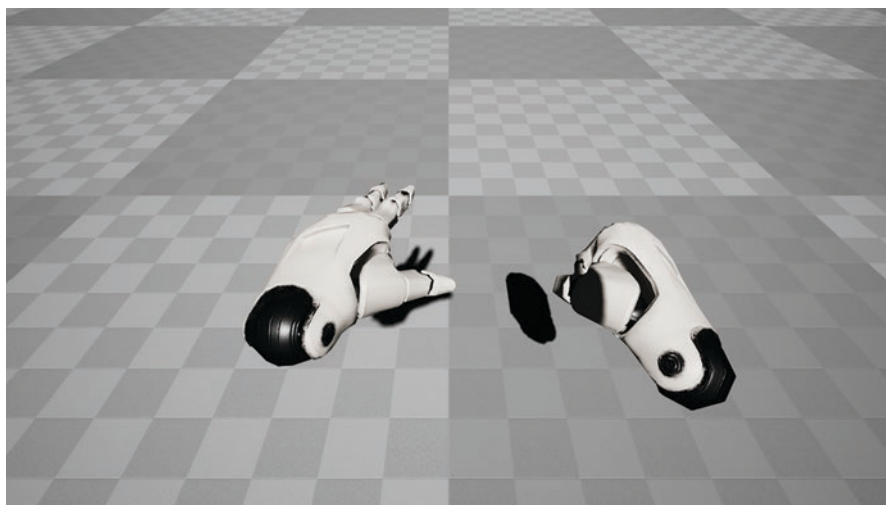


Fig. 10.14 Startup sequence to create a new Unreal Engine project based on the VR template

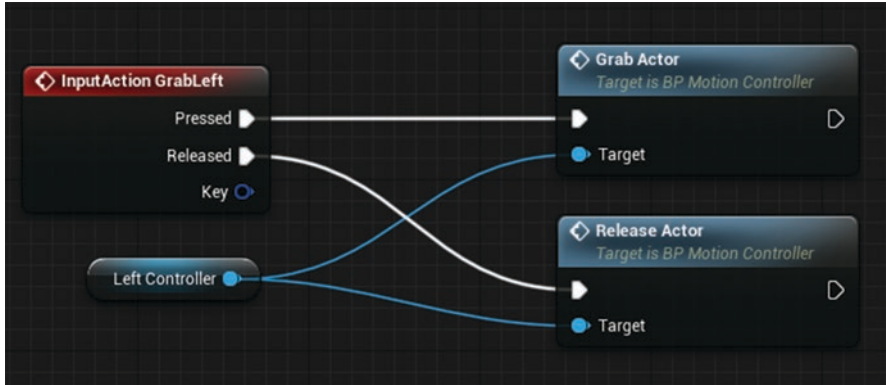


Fig. 10.15 When Unreal Engine receives a *GrabLeft* event, it calls *Grab Actor* or *Release Actor* via the respective *Controller*

this blueprint with a double-click, the *Blueprint editor* opens. We can see two events, *Event Pickup* and *Event Drop*. The former activates the physics calculation for our cube and attaches the cube to our hand. The latter reverses these changes (cf. Fig. 10.16).

If we want to make other 3D models grabbable, we simply copy these snippets into their *blueprint*. We also must make sure to add the interfaces *Drop* and *Pickup*. They allow for calling *Pickup* and *Drop* from other objects, such as motion controllers. We can do this via *Class Settings* → *Interfaces* → *Add*.

We can see the result of our demo in Fig. 10.17. The integrated system of Unreal Engine also allows us to test this project with other devices like the HTC Vive Pro or the Oculus Rift.

10.3.3 *Making of: An Application for the Microsoft HoloLens 2 with Unity*

This section explains the basic handling and functionality of the Microsoft *HoloLens* (HoloLens Documentation 2021) as well as an introduction to creating a simple HoloLens AR application with Unity.

The interaction with the HoloLens, the virtual world and the virtual objects (referred to as *holographic objects* by Microsoft) in it is realized by a combination of the viewing direction, gestures and voice commands. In the real world, it seems natural for us to look at things we want to interact with. The selection of objects for interaction within HoloLens applications is likewise represented by the focused *gaze* of the user. Since the HoloLens (1st Gen.) does not support eye tracking (cf. Sect. 4.5 Eye Tracking), the position and orientation of the head are primarily used for this purpose. The HoloLens 2, however, provides built-in eye tracking, here

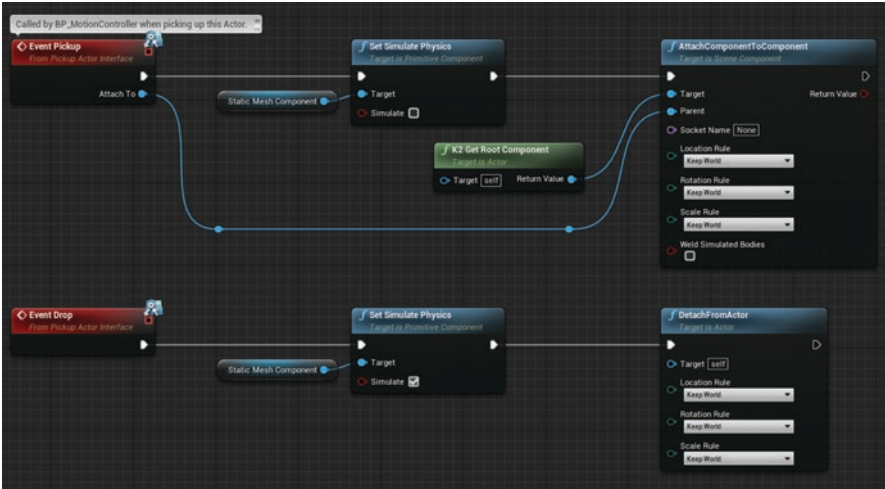


Fig. 10.16 When our 3D object receives a *Pickup* or *Drop* event, physics simulation is turned on/off and the 3D object is attached/detached to the controller (which means it follows the controller motion or not)

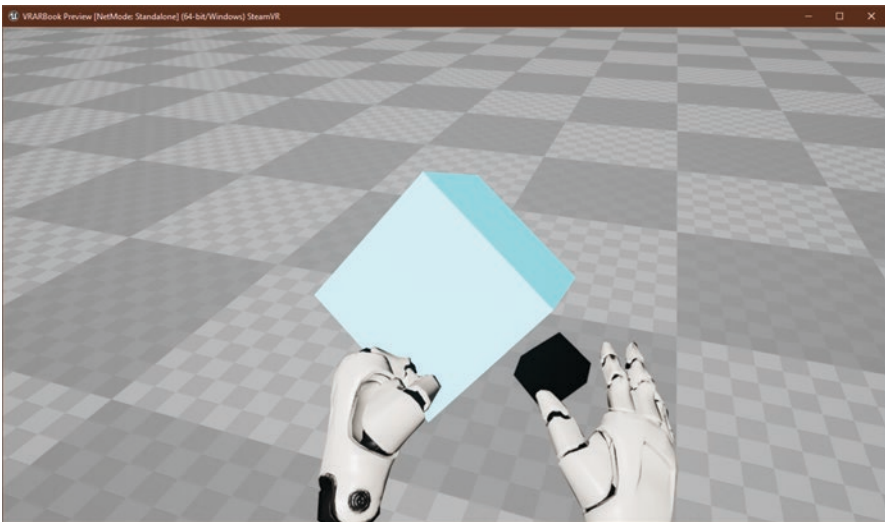


Fig. 10.17 Cube that can be grabbed and released using the motion controllers

called *eye-gaze* (Microsoft 2021a). It allows for using the actual viewing focus of the user in interactive AR applications.

The basis of the HoloLens gestures is the *air tap* gesture, which can be compared to a conventional mouse click. It is initialized by raising the index finger (called *ready position*) within the field of view of HoloLens' frontal camera. The target

object is then focused using the *gaze* or, in the case of the HoloLens 2, the *eye-gaze*. The index finger and thumb are brought together in a gripping movement and then moved back to their original position (cf. Fig. 10.18, right). If the fingers are not immediately returned to the starting position and held, it is called *air tap and hold*. This gesture, in combination with the subsequent movement of the hand, the *gaze*, or *eye-gaze*, offers the user different interaction, manipulation and navigation options, such as selecting and positioning virtual objects or scrolling through menu items (Microsoft 2021a).

The *bloom* gesture, which is reserved for the HoloLens (1st Gen.), represents the home functionality within the Windows operating system and always leads the user back to the start menu. This is done by bringing all the fingertips together and then quickly splaying the fingers (cf. Fig. 10.18, left). The HoloLens 2 uses a virtual Windows symbol, which appears next to the wrist when the hand is stretched out (*wrist button*) instead. Alternatively, the start menu can also be opened with just one hand, by focusing the *wrist button* with the eyes and performing an *air tap* gesture with the same hand. The HoloLens 2 also supports the execution of natural interactions through hand or finger tracking (cf. Sect. 4.4 Finger Tracking). This includes direct contact and interaction with virtual objects such as a button or menu options. With the *touch* gesture, which is characterized by stretching out the index finger with the hand closed, a floating cursor similar to a mouse pointer appears next to the fingertip. This is very suitable for direct interaction with virtual objects in the immediate vicinity of the user. The *hand ray* gesture is recommended for distant objects. This is characterized by an outstretched, open hand with the palm facing forward, where a laser pointer (*hand ray*) is projected from the palm. This gesture is used to target virtual objects at a distance, and in conjunction with the *air tap* gesture, represents another method for further interaction (Microsoft 2021a).

The voice commands of the Microsoft HoloLens are more flexible in their handling and can contain functionalities of different complexity depending on the implementation. The basic voice commands of the HoloLens include “*Select*”,

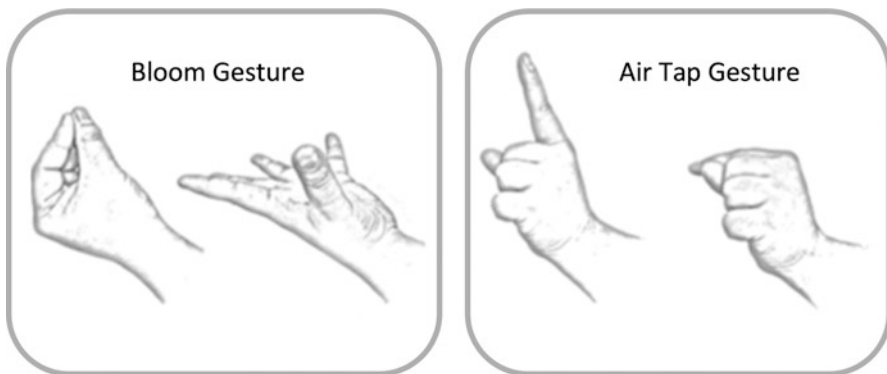


Fig. 10.18 Basic gestures for interacting with the HoloLens. The bloom gesture (left) and air tap gesture (right)

“Place”, “Face me”, “Enhance”, “Bigger” and “Smaller”. In addition, most of the menu items and options as well as other controls on the HoloLens can also be applied using voice commands. The voice command “Select” is, for example, used in conjunction with the HoloLens 2 to show the user the corresponding voice commands for menu items. With the HoloLens (1st Gen.) the existing voice commands are automatically displayed as a *voice dwell tooltip* (Microsoft 2021a).

The relationship between the real and virtual worlds (cf. Sect. 8.3 Registration) is implemented by Microsoft using what is known as *Spatial Mapping*. This provides the user with a mesh representation of the real environment, allowing virtual 3D objects to interact with real-world locations or objects (cf. Fig. 10.19). The primary objects used for *Spatial Mapping* are the *Spatial Surface Observer* and the *Spatial Surface*. The *Spatial Surface Observer* is responsible for the recording and management of the detected surrounding areas, whereas each *Spatial Surface* describes the virtual representation of a physical surface in the real world. With the help of the four integrated environment understanding cameras, the user’s surrounding is scanned for recognizable surface areas. The integrated depth cameras (time-of-flight cameras) work in two different operating modes (Ungureanu et al. 2020). The AHAT (Articulated HAnd Tracking) mode is used for a range of up to one meter with a rather high sampling rate of 45 fps. This is used for recognizing and tracking the user’s hands. The second operating mode, also called *Long Throw*, is used with a low-level sampling rate of 1–5 fps for the acquisition of depth information of the distant environment contributing to HoloLens’ SLAM approach (cf. Sect. 4.3.5.). The recognized surfaces, called *Spatial Surfaces*, are transferred into a metric, Cartesian, right-handed coordinate system as a reconstructed triangle mesh. This spatial assignment and the visualization of the recognized surfaces can be viewed directly in the *Microsoft Device Portal* (cf. Fig. 10.19).

The *Scene Understanding SDK*, which is new in the HoloLens 2, offers developers the option of using a static but very well-structured high-level representation of

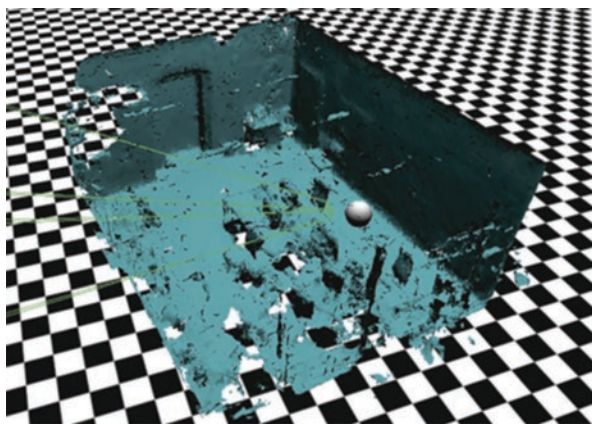


Fig. 10.19 Visualization of the Spatial Mapping in the Microsoft Device Portal of the HoloLens (1st Gen.)

the 3D environment. This is made possible by a combination of the less structured, but dynamic and very detailed *Spatial Mapping* and with the help of artificial intelligence approaches (Microsoft 2021a).

When creating a HoloLens application with Unity, it is recommended to use the Mixed Reality Toolkit (MRTK Documentation 2021) from Microsoft. This open-source framework contains a collection of components, scripts and tools allowing for cross-platform development of VR and AR applications. In addition, numerous MR scenes with various application examples are offered in additional packages, providing the user with a comprehensive overview of the individual components and their functionalities. It is therefore well suited for a quick introduction to the development of HoloLens applications and due to its integration in Unity reduces the overall implementation effort.

To start developing a HoloLens application in Unity, it is first necessary to ensure that all of the basic requirements are met. This includes the installation and configuration of the essential software, frameworks and SDKs as well as the settings for the used hardware or emulators. Among other things, this includes enabling developer mode on Windows and HoloLens, installing Visual Studio with Universal Windows Platform (UWP) development workload, using the current Windows 10 SDK and the activation and connection with the HoloLens *Device Portal*. All necessary requirements are listed and described in detail by Microsoft in the documentation of the MRTK (MRTK Documentation 2021).

Then, a new Unity project is created and the MRTK packages, *Foundation* and *Tools* are imported into the development environment via the menu option: *Assets* → *Import Package* → *Custom Package* and the project is set up for the development of MR applications (Note: if using the Unity Version 2019.4 and above, the *Unity Package Manager* can alternatively be used). For this purpose, the default settings of the MRTK, which are automatically suggested for configuration after the successful import, are accepted and applied. For realizing 3D audio, the MS HRTF (Head-Related Transfer Function; see Sect. 5.5 Audio Output Devices) *Spatializer* is selected as *Audio Spatializer*. Then further configurations must be made under the *Build Settings* (menu option: *File* → *Build Settings*). Here the *Platform* is set to UWP (Universal Windows Platform) and the *Target Device* is set to the Microsoft HoloLens. If Unity 2019 is used, it must be ensured that the *Architecture* option is set to either ARM or ARM64 – the latter is recommended. All other options can be left at the standard configuration or individually adjusted later. Under the *Player Settings* you should check whether the VR support is configured correctly (cf. Sect. 10.2.1 Unity). During development, it is advisable to deactivate the menu option: *Player Settings* → *Other Settings* → *Optimize Mesh Data*, as this option can drastically slow down the build process of the application. To ensure consistent real-time execution on the HoloLens, it is also recommended to configure the standard quality settings for UWP applications under the menu option: *Edit* → *Project Settings* → *Quality* to *Very Low* or *Fastest* (Microsoft 2021a; MRTK Documentation 2021).

After that, a new Unity scene is created (menu option: *File* → *New Scene*) and all essential components are automatically added via the menu option: *Mixed Reality Toolkit* → *Add to Scene and Configure*. The Mixed-Reality-Toolkit-Object created

in this way contains the *DeviceManager* and *SpatialMeshObserver* as well as the other different systems for configuring the Mixed Reality application. These include the systems for *Camera*, *Spatial Awareness*, *Diagnostics*, *Boundary* and the systems that are required for processing input and voice commands. In this step, the Unity *Main Camera* is also automatically provided with the necessary components, such as the *MixedRealityInputModule* or the *GazeProvider*, and assigned to the *MixedRealityPlayspace* within the scene graph. The resulting camera object has all the properties and components that are necessary to utilize the camera movement when using the HoloLens or to simulate it in the Unity editor.

The settings of the Mixed Reality Toolkit and its components can be fully configured within the Unity *Inspector* under their component entry of the same name. All settings of the MRTK and its sub-categories are managed in their individual profiles. Either the default profiles of the *Mixed Reality Core SDK* already contained in the MRTK can be used or individual profiles can be created. For example, to configure the visualization of the *Spatial Mapping*, it is advisable to choose one of the standard profiles such as the *DefaultMixedRealityToolkitConfigurationProfile* or, if a HoloLens 2 application is to be developed, the *DefaultHoloLens2ConfigurationProfile*. The selected profile is cloned using the menu option of the *Inspector* and is then available to the user as an individual profile. This process must be carried out individually for each category. It is then possible to configure all settings of the MRTK according to the individual requirements of the application to be developed and to save them as separate profiles for future use. The form of visualization and the associated material for displaying the *Spatial Mapping* can then be configured under the settings *Spatial Awareness* → *Spatial Mesh Observer* → *Display Settings*.

Since there are initially no objects in the scene for the user to interact with, new 3D objects are now created using the menu option: *GameObject* → *3D Object* and positioned in the scene (with the sub-item → *Sphere* e.g., a sphere). The position and size of a generated object and its relative location with respect to the camera may still have to be adapted. According to Microsoft, approximately 2 m is considered to be an optimal distance for using *gaze* and interacting with virtual objects. The 3D objects created are then assigned their respective materials. A wide selection of materials that are optimized for usage in AR applications can be found under *Assets* → *MRTK* → *SDK* → *StandardAssets* → *Materials*. When the user's *gaze* wanders over the 3D object that has been created, only the cursor is projected onto the object's *CollisionObject*. By adding the script *TapToPlace* (*Assets* → *MRTK* → *SDK* → *Features* → *Utilities* → *Solvers*), the 3D object can be selected using the *air tap* gesture and freely positioned in the room using the *gaze* or placed on any surface recognized by *Spatial Mapping*. Additional settings can be made in the *Inspector* under the *SolverHandler* script, which is automatically added. An example is the *Tracked Target Type* defining the reference point of the tracking used. It is possible to use the hand of the user or a hand ray instead of the head movement of the HoloLens. The *BoundingBox* script (*Assets* → *MRTK* → *SDK* → *Features* → *UX* → *Scripts* → *BoundingBox*) equips the 3D object with a visible bounding box (cf. Sect. 7.2.1). Thanks to the previously mentioned interaction options of the HoloLens, such bounding boxes allow the associated 3D object to be scaled and rotated at runtime. Another useful component is the *PointerHandler* script (*Assets*

→ *MRTK* → *SDK* → *Features* → *Input* → *Handlers*), which equips a 3D object with the function of reacting to individual *air taps*. This can be used to implement individual functionalities, such as a change to the material used while the 3D object is selected or moved. In addition, the *MRTK* package *Examples* offers a wide range of practical example scenes. These are helpful for getting to know the numerous components and their functionalities as well as being a starting point for your own projects. The *HandInteractionExample* is particularly suitable for testing the hand tracking functionality of the HoloLens 2, as it provides many ready-to-use interactive virtual objects (*MRTK Documentation* 2021).

To deploy the application on the HoloLens, the current scene must first be added to the *Build Settings*. After that, the Unity application and the *APPX* package must be built using the *Build Window*, which is included in the *MRTK Tools* package (menu option: *Mixed Reality Toolkit* → *Utilities* → *Build Window*). Alternatively, the deployment can also be done conventionally using Visual Studio (*MRTK Documentation* 2021). The HoloLens is then connected to the PC via USB to connect to the *Device Portal*. The *Device Portal* is a web server on the HoloLens, which can be reached via the browser at the IP address 127.0.0.1:10080. Alternatively, it is possible to access the *Device Portal* via a shared WiFi connection and the specific IP address. The *APPX* package can be installed under the menu entry: *System* → *Apps* of the *Device Portal*. It is important to consider all the necessary dependencies of the package. After the application has been successfully installed, it can be executed on the HoloLens. Another possibility to start the application directly on the hardware is realized by the *Holographic Remoting Player (HRP)*. First, the *HRP* application from the Microsoft Store is installed on the HoloLens and launched. Based on the information provided by the *HRP*, the HoloLens is then connected to the Unity editor using the menu option: *Windows* → *XR* → *Holographic Emulation*. The *Device Portal* enables the user to manage the configuration of the HoloLens and has many useful tools to analyze your own applications and support their development. For further information Microsoft provides detailed documentation (Microsoft 2021a).

10.3.4 Making of: Basics for the Development of a Native ARCore Application for Android

This case study describes the development of an AR application using *ARCore*, including an insight into more general aspects of creating AR applications. The underlying principles can also be applied to other frameworks.

Rather than using Java code and framework APIs, native development of an Android application allows us to develop parts of the application or even the entire project in C++. *Android Studio* provides easy access to many configuration tasks required to develop a native application. Additionally, the *Native Development Kit (NDK)* provided by Google is required. Nevertheless, it is necessary to write some

helper classes in Java to provide basic functionality. In this case the *Java Native Interface (JNI)* is used as the interface.

Before ARCore can be used, a session based on the current instance of the application is required. For this purpose, the function *ArSession_create* is called with information about the current Java environment, the current context, and the instance of the application. An *ARSession* describes and manages the current state of the system as well as the complete AR life cycle.

Using such a session, individual (camera) frames can be retrieved from a smartphone. A frame represents the current camera image and provides the necessary functions to determine *trackables* – objects that can be tracked with ARCore. To create a session, the desired screen size of the application and orientation of the device must be provided.

When the current camera image is provided by the session object, ARCore allows to store it as a texture directly in the graphics memory. For this purpose, a texture with the OpenGL-ES extension *GL_OES_EGL_image_external_essl3* is created and its native ID is passed to the session. The dimension of this texture does not depend on the settings of the session but is based on the resolution of the built-in camera. Usually, its aspect ratio is not identical to the aspect ratio of the screen or the application. Since the session has already been informed about the screen size, the UV coordinates of the camera image can be adjusted accordingly. These adjusted coordinates are also created by the AR session. This allows the camera image to be displayed at the desired size. In most cases, the camera image is displayed as full screen. For this purpose, the generated coordinates are used and passed to a shader for the visualization on the screen. The camera image can also be rendered in a separate texture (render target). This allows further use of the texture for special effects or for further analysis.

Essential parts of an ARCore application are so-called *trackables*. *Trackables* are objects in space that are recognized as geometries by the application and may be planes or points. These *trackables* are created, maintained and, if necessary, deleted by ARCore based on the session and the current camera image. It is important to consider the dynamics of these *trackables* accordingly. Anchor points can be set on *trackables*, which then give objects a position and orientation in space, i.e., allowing for a geometrical registration in world space (see Sect. 8.2.1).

The dimension and orientation of a *trackable* are determined by ARCore. However, the individual points of a plane can be extracted via the interface. Additionally, the transformation matrix for the center of the plane can be determined. Thus, all information is now available for further usage and rendering. The accompanying code example shows, for example, how planes can be visualized on the screen.

Besides a list of planes, points can also be used as *trackables*. Similar to a plane, points are objects in space. However, these points do not have their own transformation matrix, but define themselves as a single vertex (3D coordinate), having a position in the world. This makes visualization of these points easy, as each vertex must be transformed by the projection matrix only.



Fig. 10.20 Illustration of two differently illuminated scenes and the influence of the illumination situation on the material properties

As already explained, anchor points can be set on *trackables*, which can then be used to arrange virtual geometries inside the scene (see Fig. 10.8). To add an anchor to a trackable a ray is cast from the screen-space coordinates of a user tap gesture into 3D world space. The framework then provides a list of *trackables* hit by the ray. Now, an anchor point is created close to a trackable from this list. ARCore allows for requesting the transformation of an anchor point, which may then be used for the visualization of objects at that location.

In this example, the light intensity of ARCore is a single value calculated based on a complete frame (see Sect. 10.2.3). This is a floating-point number between 0 and 1. How this value is finally interpreted and used depends on the developer. A reasonable way to use this value is to consider it as the ratio of the illumination value related to the maximum illumination intensity. If a virtual light source is available, the author should design the lighting conditions for a bright room. In the case of a dark environment, the intensity of the light source can then be adjusted accordingly. This value can be used to attenuate the light intensity, for example by simple multiplication. Figure 10.20 shows the adjustment of the intensity in a dark room (left) and a bright room (right). It is clearly visible that in the first case the materials appear darker and specular reflections are less pronounced.

The information provided in this section represents the basis for creating a native ARCore application. However, the underlying principles can be transferred to other AR frameworks like ARKit without much effort.

10.4 Summary and Questions

In this chapter, the process of authoring VR and AR applications was first illustrated in general and then specifically using individual frameworks and case studies. Based on this chapter, the reader should have gained a rough idea of the authoring process using modern runtime environments and recent VR and AR hardware. Ideally, the reader has downloaded the examples, tried them out and developed them individually to get a good sense of the possibilities and limitations of the respective tools.

Check your understanding of the chapter using the following questions:

- You will be tasked with implementing a VR/AR training application for minimally invasive surgery using a common consumer VR HMD and an end effector display (cf. Sect. 5.6) as haptic input/output device. What authors are needed for this? Plan a suitable authoring process and select appropriate software to support the authors.
- What is a tool chain? At what point in the development process should you deal with it?
- You want to create an AR application with Microsoft's HoloLens. Which framework can you use for this?
- You want to create a mobile AR application for a wide variety of smartphones and tablets. Can you get by with a single code base for this? How can you keep your development effort as low as possible?
- You are to develop an AR application that realistically displays objects in both light and dark areas. How can you determine the illumination?
- When starting the application, an error occurs with a session. What is a session and what is it needed for?
- You are asked to develop an AR application dealing with complex geometry. Which framework and hardware are you going to use?

Recommended Reading

- Glomer J (2018) *Unity 2018 Augmented reality projects: Build four immersive and fun AR applications using ARKit, ARCore, and Vuforia*. Packt Publishing.
- Linowes J (2020) *Unity 2020 virtual reality projects: Learn VR development by building immersive applications and games with Unity 2019.4 and later versions*, 3rd edn. Packt Publishing.
- McCaffrey, M (2017) *Unreal Engine VR cookbook: Developing virtual reality with UE4*. Addison Wesley.
- Rabin S (2009) *Introduction to game development*, 2nd edn. Charles River Media, Boston – a reference book on computer games. Due to the manifold intersections of VR and computer games, the literature from the field of computer games is also relevant.
- Sewell B (2015) *Blueprints visual scripting for Unreal Engine (English Edition): Build professional 3D games with Unreal Engine 4's visual scripting system*. Packt Publishing.
- Vroegop D (2017). *Microsoft HoloLens developer's guide: A complete guide to HoloLens application development*. Packt Publishing.

References

- Apple (2021) Apple app store. <https://developer.apple.com/support/app-store/>. Accessed 16 Mar 2021
- Google (2021) Google developers. <https://developers.google.com/ar/discover/>. Accessed 16 Mar 2021
- HoloLens 2 (2021). <https://www.microsoft.com/en-us/hololens/>. Accessed 16 Mar 2021

- Magic Leap 1 (2021). <https://www.magicleap.com/en-us/magic-leap-1>. Accessed 16 Mar 2021
- Makarov A (2021) 9 augmented reality trends in 2021: the future is here. <https://mobidev.biz/blog/augmented-reality-future-trends-2018-2020>. Accessed 14 Jan 2021
- Kato H, Billinghurst M (1999) Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In: Proceedings of IWAR, p 99
- Ungureanu D et al (2020) HoloLens 2 research mode as a tool for computer vision research. arXiv preprint arXiv:2008.11239
- Unity (2021). <https://unity.com/>. Accessed 16 Mar 2021
- Unreal Engine (2021). <https://www.unrealengine.com/en-US/>. Accessed 16 Mar 2021

Software, Online Documentation and Tutorials

- ARCore (2021). <https://developers.google.com/ar/>. Accessed 16 Mar 2021
- ARKit (2021). <https://developer.apple.com/augmented-reality/arkit/>. Accessed 16 Mar 2021
- Blender (2021). <https://www.blendernation.com/>. Accessed 16 Mar 2021
- Godot game engine (2021). <https://godotengine.org>. Accessed 16 Mar 2021
- HoloLens Documentation (2021) Microsoft. <https://docs.microsoft.com/en-us/hololens>. Accessed 16 Mar 2021
- MARS (2021). <https://unity.com/products/unity-mars>. Accessed 16 Mar 2021
- MRTK Documentation (2021) Mixed Reality Toolkit Unity Documentation. <https://github.com/Microsoft/MixedRealityToolkit-Unity>. Accessed 16 Mar 2021
- MRTK Release (2021), <https://github.com/Microsoft/MixedRealityToolkit-Unity/releases/tag/v2.3.0>. Accessed 16 Mar 2021
- OpenVR (2021). <https://github.com/ValveSoftware/openvr>. Accessed 16 Mar 2021
- Pixyz (2021). <https://unity.com/products/pixyz>. Accessed 16 Mar 2021
- RealityKit (2021). <https://developer.apple.com/augmented-reality/realitykit/>. Accessed 16 Mar 2021
- Unity, download (2021). <https://unity3d.com/get-unity/download>. Accessed 16 Mar 2021
- Unity XR (2021). <https://docs.unity3d.com/2019.3/Documentation/Manual/XR.html>. Accessed 16 Mar 2021
- Unreal Engine, download (2021). <https://www.unrealengine.com/en-US/download>. Accessed 16 Mar 2021
- Unreal Engine XR (2021). <https://www.unrealengine.com/en-US/xr>. Accessed 16 Mar 2021
- Valve Cooperation (2021) SteamVR plugin 2.6.1. <https://assetstore.unity.com/packages/tools/integration/steamvr-plugin-32647>. Accessed 16 Mar 2021
- Windows Mixed Reality (2021). <https://www.microsoft.com/en-us/mixed-reality/windows-mixed-reality>. Accessed 16 Mar 2021

Chapter 11

Mathematical Foundations of VR/AR



Ralf Doerner

Abstract In Virtual Reality and Augmented Reality, mathematical methods offer fundamental principles to model three-dimensional space. This makes it possible to provide exact information and perform calculations, e.g., to determine distances or to describe the effects of transformations such as rotations or translations exactly. This chapter compiles the most important mathematical methods, especially from linear algebra, that are frequently used in VR and AR. For this purpose, the term *vector space* is defined and extended to a Euclidean space. Afterwards, some basics of analytic geometry are introduced, especially the mathematical description of lines and planes. Finally, changes of coordinate systems as well as affine transformations are discussed and their computation with matrices in homogeneous coordinates is explained.

11.1 Vector Spaces

In Virtual Reality, we are concerned with the real space that surrounds us. It is helpful to model this space with methods of mathematics, e.g., to be able to make exact, formal, mathematically provable statements or to perform computations. In VR, we use a *vector space*, a construct of linear algebra (a branch of mathematics), for this modeling.

Each vector space is formed over a *field* G . The elements of G are called *scalars* and we denote them by small Latin letters. Being a *field* in the sense of algebra means that G is a set with the two binary operations “+” (addition) and “ \cdot ” (multiplication), which combine two elements of G and as a result give an element of

Dedicated website for additional material: vr-ar-book.org

R. Doerner (✉)

Department of Design, Computer Science, Media, RheinMain University of Applied Sciences, Wiesbaden, Germany

e-mail: ralf.doerner@hs-rm.de

G . Moreover, there is an element 0 in G , called the *additive identity*, and an element 1 in G , called the *multiplicative identity*. Finally, the elements of G satisfy the following field axioms. For any scalar a, b, c, d (with $d \neq 0$):

$$a + (b + c) = (a + b) + c \quad (\text{associativity of addition})$$

$$a + b = b + a \quad (\text{commutativity of addition})$$

$$0 + a = a \quad (\text{commutativity of addition})$$

For each $a \in G$ there exists a $-a \in G$ with $-a + a = 0$ (additive inverses)

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \quad (\text{associativity of multiplication})$$

$$a \cdot b = b \cdot a \quad (\text{commutativity of multiplication})$$

$$1 \cdot d = d \quad (\text{multiplicative identity})$$

For each $d \in G \setminus \{0\}$ there exists a $d^{-1} \in G$ with $d^{-1} \cdot d = 1$ (multiplicative inverses)

$$a \cdot (b + c) = a \cdot b + a \cdot c \quad (\text{distributivity})$$

The set of *real numbers* \mathbb{R} , which comprises the set of natural numbers (e.g., 1, 2, 3, ...), integers, rational numbers and irrational numbers (e.g., π), fulfills the field axioms and is usually chosen in VR.

The set of elements of a vector space V over a field G is called *vectors*. We denote them by Latin letters, over which an arrow is placed. Two operations are defined on vectors. First, *vector addition* takes two vectors and assigns them a third vector. We write this operation as “+” (not to be confused with addition in scalars). The vector addition adheres to the associativity of addition and the commutativity of addition. There exists also an identity element of addition, the *zero vector* $\vec{0}$. For each vector \vec{u} there exists an additive inverse $-\vec{u}$ in V . Secondly, *scalar multiplication* takes a scalar and a vector and assigns them a vector. we write it as “ \cdot ”. Scalar multiplication adheres to distributivity:

$$\forall a, b \in G, \forall \vec{u}, \vec{v} \in V : a \cdot (\vec{u} + \vec{v}) = a \cdot \vec{u} + a \cdot \vec{v} \text{ and } (a + b) \cdot \vec{u} = a \cdot \vec{u} + b \cdot \vec{u}$$

An example of a set V that fulfills these properties of a vector space is the set of 3-tuples over the real numbers, i.e., the set of all lists of real numbers of length 3. We call this set \mathbb{R}^3 . The 3-tuple $(5, -2, 3)$, for example, is an element from the set \mathbb{R}^3 . In the following, we will not write the elements of \mathbb{R}^3 as a list next to each other but on top of each other:

$$\vec{u} = \begin{pmatrix} 5 \\ -2 \\ 3 \end{pmatrix}$$

To specify the set \mathbb{R}^3 completely as a vector space, we still have to specify the two operations “+” and “·” of the vector space. We do this by defining these operations based on the addition and multiplication of the real numbers (i.e., the field over which \mathbb{R}^3 was formed).

$$a \in \mathbb{R}, \vec{u}, \vec{v} \in \mathbb{R}^3 :$$

$$\vec{u} + \vec{v} := \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} + \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} u_1 + v_1 \\ u_2 + v_2 \\ u_3 + v_3 \end{pmatrix} \quad \text{and} \quad a \cdot \vec{u} := a \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} a \cdot u_1 \\ a \cdot u_2 \\ a \cdot u_3 \end{pmatrix}$$

In vector spaces, vector addition and scalar multiplication are generally used to define a *linear combination* of a number of n scalars and n vectors:

$$\vec{u} = a_1 \cdot \vec{u}_1 + a_2 \cdot \vec{u}_2 + \dots + a_n \cdot \vec{u}_n$$

If all n scalars must have the value 0 for the linear combination to yield the zero vector, the n vectors of the linear combination are called *linearly independent*. If one finds a maximum of d linearly independent vectors in a vector space V , then d is the *dimension* of the vector space V . In our example, the vector space \mathbb{R}^3 has dimension 3. By the way, it is not only the set of all 3-tuples that forms a vector space. If k is a natural number, then the set of all k -tuples of real numbers forms a vector space \mathbb{R}^k , which has dimension k .

If V is a vector space of dimension n and we find n linearly independent vectors, these vectors are called a *base* of V . We can then represent each vector of V by a linear combination of these base vectors. The n scalars that occur in this linear combination are called the components or *coordinates* of a vector.

11.2 Geometry and Vector Spaces

In geometry, *directed line segments* are called *geometric vectors*. You can visualize them with an arrow, having a length and a direction. The beginning of the geometric vector is called the *tail*, and the end of the geometric vector is called the *tip*. We define an addition operation of two geometric vectors as follows. We place the tail of the second vector at the tip of the first vector – the result of the addition is a geometric vector that then runs from the tail of the first vector to the tip of the second vector. We also define a *scalar multiplication*, where we choose the real numbers \mathbb{R} as scalars (see Fig. 11.1). If we multiply the scalar a by a geometric vector, we get

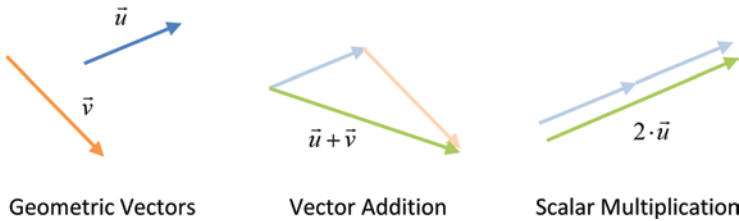


Fig. 11.1 Vector addition and scalar multiplication of geometric vectors

as a result a geometric vector with $a \times$ the length of the original geometric vector. If a is positive, the resulting vector points in the same direction; if not, the result vector points in the opposite direction. With these two operations the set of geometric vectors forms a vector space over \mathbb{R} .

Directed line segments are useful constructs when we want to model the space surrounding us. However, performing computations with them directly proves to be difficult. Therefore, we take a base from the space of geometric vectors – if we are in the three-dimensional space, it consists of three base vectors. We can represent each geometric vector as a linear combination of these three base vectors. The coordinates in this linear combination are three real numbers – which in turn we can understand as 3-tuples, i.e., an element of the vector space \mathbb{R}^3 .

We can proceed as follows. We assign a vector from \mathbb{R}^3 to each directed line segment, i.e., to each geometric vector, with the help of a base. In \mathbb{R}^3 we can calculate with vectors based on the addition and multiplication of real numbers. The result of the calculation is then transferred into the space of the geometric vectors by inserting the calculated result as a scalar into the linear combination of the base vectors. If, for example, we want to add two geometric vectors, then we assign two vectors from \mathbb{R}^3 , the “world of numbers”, to these two vectors from the “world of geometry”. In the “number world” we can calculate the result vector. We transfer this result vector back into the “world of geometry” and thus we have determined the geometric vector resulting from the addition by computation.

11.3 Points and Affine Spaces

However, the usefulness of our mathematical model is still limited: geometric vectors possess only *length* and *direction*, but no fixed *position* in space. This also means that we cannot model essential concepts from the real world, such as distances. Therefore, we introduce the term *point* in addition to scalar and vector. We write points with capital Latin letters. Points have no length and no direction, but a position. Let P and Q be two elements from the set of points. Then we define an operation “–”, called *point-point subtraction*, which connects two points and results in a vector:

$$P - Q = \vec{u} \Leftrightarrow P = \vec{u} + Q$$

With this we also define an addition between a point and a vector (called *point-vector addition*), where the result is a point. Thus, we can represent any point P in three-dimensional space as an addition of a point O (called the *origin*) and a linear combination of three linearly independent geometric vectors $\vec{u}, \vec{v}, \vec{w}$, the base vectors:

$$P = O + a \cdot \vec{u} + b \cdot \vec{v} + c \cdot \vec{w} = O + \vec{p}$$

We call these three base vectors, together with O , a *coordinate system* K . We call the 3-tuple (a, b, c) the coordinates of P with respect to K . Thus, every point in our “world of geometry” for a given K can be represented by an element from \mathbb{R}^3 , our “world of numbers”. So, we can “calculate” not only with vectors, but also with points, i.e., with fixed positions in our world. We call \vec{p} the *position vector* belonging to P .

A vector space that has been extended by a set of points and an operation, the point-point subtraction, is called an *affine space* in mathematics. Geometrically, we can interpret point-point subtraction like this: $P - Q$ is a vector that we get when we choose a directional path with starting point Q and final point P .

11.4 Euclidean Space

We add the concept of *distance* to our existing mathematical model of the space surrounding us. For this purpose, we introduce another operation, which we denote by “ \cdot ” and which takes two vectors and results in a scalar. We call this operation the *scalar product* (not to be confused with scalar multiplication, which takes a scalar and a vector and results in one vector – even if we write both operations with “ \cdot ”, we always know which operation is meant because of the types of the two operands). The scalar product must adhere to commutativity of multiplication and the following axioms for scalars a, b , vectors $\vec{u}, \vec{v}, \vec{w}$ and the null vector $\vec{0}$:

$$(a \cdot \vec{u} + b \cdot \vec{v}) \cdot \vec{w} = a \cdot \vec{u} \cdot \vec{w} + b \cdot \vec{v} \cdot \vec{w}$$

$$\vec{u} \cdot \vec{u} > 0 \text{ if } \vec{u} \neq \vec{0}$$

$$\vec{0} \cdot \vec{0} = 0$$

In our vector space \mathbb{R}^3 , we can define a scalar product as follows so that all the above conditions are fulfilled:

$$\vec{u} \cdot \vec{v} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} := u_1 \cdot v_1 + u_2 \cdot v_2 + u_3 \cdot v_3$$

In honor of the Ancient Greek mathematician Euclid of Alexandria, an affine space supplemented by the scalar product operation is called a *Euclidean point space*. Using the scalar product, we define the *amount* of a vector as follows:

$$|\vec{u}| = \sqrt{\vec{u} \cdot \vec{u}}$$

In our three-dimensional space, the amount of a vector is equal to its length. Thus, we can also determine the *distance* d between two points P and Q as

$$d = |P - Q| = \sqrt{(P - Q) \cdot (P - Q)}$$

The *angle* α enclosed by two vectors can be determined from the following equation:

$$\vec{u} \cdot \vec{v} = |\vec{u}| \cdot |\vec{v}| \cdot \cos \alpha$$

In the case $\alpha = 90^\circ$ (i.e., the two vectors are perpendicular to each other) the scalar product of the two vectors is 0. Two vectors whose scalar product is 0 are called *orthogonal*. If the two vectors also have length 1, they are called *orthonormal*. For the base in our space, we want to use orthonormal vectors in the following. A corresponding coordinate system (base vectors are perpendicular to each other and have length 1) is called a *Cartesian coordinate system*. In the case of \mathbb{R}^3 , we take the three unit vectors

$$\vec{e}_x = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \vec{e}_y = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad \vec{e}_z = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

in the given order and the point O as the origin point, whose position vector is the zero vector.

To be able to easily find a vector orthogonal to two vectors in \mathbb{R}^3 , we define an operator “ \times ”, which we call the *cross product* and which takes two vectors and results in one vector:

$$\vec{n} = \vec{u} \times \vec{v} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \times \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} := \begin{pmatrix} u_2 \cdot v_3 - u_3 \cdot v_2 \\ u_3 \cdot v_1 - u_1 \cdot v_3 \\ u_1 \cdot v_2 - u_2 \cdot v_1 \end{pmatrix} = -1 \cdot (\vec{v} \times \vec{u})$$

The resulting vector is called a *normal vector*. In this order, the vectors $\vec{u}, \vec{v}, \vec{n}$ form a *right-handed system*, i.e., if you take them as geometric vectors and place their tail on a common point, the vectors are oriented like the thumb, index finger and middle finger of the right hand. The vector product is not commutative. While one can generalize our definition of the scalar product from \mathbb{R}^3 to \mathbb{R}^n and thus obtain Euclidean point spaces of dimension n , the cross product is defined exclusively in \mathbb{R}^3 .

11.5 Analytical Geometry in \mathbb{R}^3

In \mathbb{R}^3 , our mathematical model of the space surrounding us, we can solve geometric problems by computation, e.g., finding an intersection of lines or determining the distance of a point to a plane. A *line* is the generalization of a directed line segment: it has no direction and has infinite length. A line is defined by two points. Mathematically we model a line g through points P and Q as a subset of \mathbb{R}^3 that includes all points X whose position vector \vec{x} satisfies the equation of the line, using the position vectors associated with P and Q :

$$g = \{ \vec{x} \in \mathbb{R}^3 \mid \exists t \in \mathbb{R}, \vec{x} = \vec{p} + t \cdot (\vec{q} - \vec{p}) \}$$

The scalar t is called the *parameter* and the equation above is also called the *vector equation* of a line. The vector that is multiplied by t is called the *directional vector* of the line g . Similarly, we can model a *plane* E as a subset of \mathbb{R}^3 . It is defined by three points P, Q, R and the equation of the plane contains two parameters and two directional vectors:

$$E = \{ \vec{x} \in \mathbb{R}^3 \mid \exists t, s \in \mathbb{R}, \vec{x} = \vec{p} + t \cdot (\vec{q} - \vec{p}) + s \cdot (\vec{r} - \vec{p}) \}$$

By means of the cross product, we can compute the normal vector \vec{n} from the directional vectors, which is perpendicular to E . For the distance d of a point X to a plane E we know the following equation in linear algebra, where the sign of the scalar product indicates on which side of E the point X is located:

$$d = \left| \frac{\vec{n}}{|\vec{n}|} \cdot (\vec{x} - \vec{p}) \right|$$

Thus, we can reformulate the condition that points X belong to the subset E . This is because all points X that have the distance 0 from E lie on the plane E . Thus, we obtain the *point-normal form* of a plane:

$$E = \{ \vec{x} \in \mathbb{R}^3 \mid \vec{n} \cdot (\vec{x} - \vec{p}) = 0 \}$$

With these definitions you can compute intersections between lines and between a line and a plane as well as intersections between planes. The first step is to equate the equations that define the set of points that form a line or a plane. Alternatively, substitution can sometimes be used. This results in either an equation to be solved or a linear system of equations, the solution of which can be computed by mathematical methods (for example, Gaussian elimination).

11.6 Matrices

In virtual reality, another mathematical construct is often used to compute transformations such as rotations or translations in three-dimensional space: the *matrix* (plural: *matrices*). A matrix is a table of n rows and m columns where each entry is a scalar. In the following, we will always assume that entries are real numbers. We find the scalar a_{ij} in row i and column j of the matrix. It is called the *entry* in place (i, j) . We write matrices with bold capital letters: $\mathbf{A} = [a_{ij}]$ and say \mathbf{A} is an $n \times m$ matrix. The matrix \mathbf{M} in our example has two rows and four columns, so it is a 2×4 matrix, and the entry $m_{1,3}$ has the value 5:

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 5 & 3 \\ 1 & 9 & 2 & 0 \end{bmatrix}$$

For matrices, we define three operations. First, the *scalar-matrix multiplication*, denoted by “ \cdot ”, which combines a scalar s and a $n \times m$ matrix $\mathbf{A} = [a_{ij}]$ to form an $n \times m$ matrix: $s \cdot \mathbf{A} = s \cdot [a_{ij}] := [s \cdot a_{ij}]$. This operation adheres to associativity. Secondly, *matrix-matrix addition*, denoted by “ $+$ ”, links two matrices \mathbf{A} and \mathbf{B} of the same size $n \times m$ to form a matrix of size $n \times m$: $\mathbf{A} + \mathbf{B} = [a_{ij}] + [b_{ij}] := [a_{ij} + b_{ij}]$. This operation adheres to associativity and commutativity. Third, *matrix-matrix multiplication*, denoted by “ \cdot ”, combines a matrix \mathbf{A} of size $n \times k$ and a matrix \mathbf{B} of size $k \times m$ to form a matrix of size $n \times m$:

$$\mathbf{A} \cdot \mathbf{B} := [c_{ij}] \quad \text{with} \quad c_{ij} = \sum_{l=1}^k a_{il} \cdot b_{lj}$$

This operation adheres to associativity. It should be emphasized that commutativity does not apply to matrix-matrix multiplication: $\mathbf{A} \cdot \mathbf{B}$ does not always equal $\mathbf{B} \cdot \mathbf{A}$.

If we swap the rows and columns in a matrix, we get the *transposed matrix*. The transposed matrix of matrix $\mathbf{M} = [a_{ij}]$ is $\mathbf{M}^T = [a_{ji}]$. The following applies: $(\mathbf{A} \cdot \mathbf{B})^T = \mathbf{B}^T \cdot \mathbf{A}^T$. A special case are matrices that have the same number of rows and columns. These are called *square matrices*. The square matrix \mathbf{I} for which the following applies

$$\mathbf{I} = [a_{ij}], \quad a_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

is called the *unit matrix*. The following applies: $\mathbf{A} \cdot \mathbf{I} = \mathbf{I} \cdot \mathbf{A} = \mathbf{A}$, where \mathbf{A} and \mathbf{I} are both $n \times n$ matrices. If a matrix \mathbf{A}^{-1} of the same size exists for an $n \times n$ matrix \mathbf{A} and the equation $\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{I}$ applies, then \mathbf{A}^{-1} is called the *inverse matrix* of \mathbf{A} . \mathbf{A} is then called *invertible*. The following applies: $(\mathbf{A} \cdot \mathbf{B})^{-1} = \mathbf{B}^{-1} \cdot \mathbf{A}^{-1}$. If the following applies to a matrix \mathbf{A} : $\mathbf{A}^{-1} = \mathbf{A}^T$, then \mathbf{A} is called *orthogonal*.

11.7 Affine Transformations

Assume that the point P has coordinates (x, y, z) with respect to a Cartesian coordinate system. If we translate P by t_x in the x -direction, by t_y in the y -direction and by t_z in the z -direction, we map point P to a new point P' . What are its coordinates? To calculate such *transformations*, we utilize matrices. We introduce a special notation for matrices that consist of only one column: we write them with small bold letters and call them *column matrices*. Now we want to represent the point P by the column matrix \mathbf{p} . We do this as follows:

$$\mathbf{p} = \begin{bmatrix} w \cdot x \\ w \cdot y \\ w \cdot z \\ w \end{bmatrix}, \text{ for any real number } w \text{ with } w \neq 0$$

We call $(w \cdot x, w \cdot y, w \cdot z, w)$ the *homogeneous coordinates* of P . In practice, for the sake of simplicity, usually $w=1$ is chosen. If one chooses $w=0$, one can represent a vector in a column matrix instead of a point by means of homogeneous coordinates:

$$\vec{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \equiv \mathbf{v} = \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

The translation from P to P' can be described by a matrix \mathbf{M} . The following simple equation applies:

$$\mathbf{p}' = \mathbf{M} \cdot \mathbf{p}$$

In our translation example, this equation looks like this:

$$P' = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} w \cdot x \\ w \cdot y \\ w \cdot z \\ w \end{bmatrix} = \begin{bmatrix} w \cdot (x + t_x) \\ w \cdot (y + t_y) \\ w \cdot (z + t_z) \\ w \end{bmatrix}$$

From the resulting column matrix \mathbf{p}' , we can obtain the coordinates of point P' after division by w : $(x + t_x, y + t_y, z + t_z)$. If instead of \mathbf{p} , which represents a point, we were to use the column matrix \mathbf{v} , which represents a vector, in the above equation, then \mathbf{v} would be mapped exactly back to \mathbf{v} . This is also what we expect: since a vector has no fixed position in space, it is not changed by a displacement. As we will see below, the transformation of a vector by a more complex transformation is slightly more complicated.

Let us take a closer look at the matrix \mathbf{M} that represents this translation. You can think of its four columns as column matrices. The first three columns represent vectors, because the value in the fourth row is zero. In fact, these are the base vectors of our three-dimensional space if we apply the translation to them. They do not change, because a translation does not change the length or the direction of a vector. The fourth column vector represents a point, because the value in the fourth row is not zero. This column vector represents the origin when the translation is applied to it. As a result of the translation, the origin $(0, 0, 0)$ is mapped to (t_x, t_y, t_z) . Therefore, this transformation can be seen as a change from one coordinate system of our three-dimensional space to another coordinate system. In fact, mathematicians have been able to show that each change of coordinate systems can be represented as a matrix \mathbf{M} . With 4×4 matrices \mathbf{M} , not only can translations be computed, but also other affine transformations that map one affine space into another. Besides translation, the following geometric transformations are also included: rotation, scaling, reflection and shearing. If you invert the matrix \mathbf{M} , you get the matrix \mathbf{M}^{-1} , which represents the inverse mapping of \mathbf{M} , i.e., it reverses the mapping represented by \mathbf{M} .

Let us assume that we perform n geometric transformations of the point P . We represent the transformation performed first by \mathbf{M}_1 , the second by \mathbf{M}_2 and so on, until finally the transformation performed last is represented by \mathbf{M}_n . This allows us to determine the coordinates of the point P' resulting from the back-to-back execution (*concatenation*) of these transformations as follows:

$$P' = (\mathbf{M}_n \cdot \dots \cdot \mathbf{M}_3 \cdot \mathbf{M}_2 \cdot \mathbf{M}_1) \cdot P$$

Note the order of the matrices and keep in mind that matrix multiplication is not commutative. If you perform the computation as indicated by the brackets, you only need to compute the product of all n matrices once, even if you transform hundreds of points with the same transformation. For a large number of points to be transformed, this results in a considerable saving of computing time. Matrix operations

for 4×4 matrices are implemented directly in hardware in graphics processors, which leads to another reduction in computing time.

Besides points, vectors can also be transformed by a matrix \mathbf{M} that describes an affine transformation. If we want to know where the vector $\bar{\mathbf{v}}$ is mapped to after the transformation described by \mathbf{M} , we represent the vector in the column matrix \mathbf{v} . We compute $\mathbf{v}' = (\mathbf{M}^{-1})^T \cdot \mathbf{v}$ and the first three rows of the column matrix \mathbf{v}' contain the coordinates of the transformed vector.

11.8 Determination of Transformation Matrices

To calculate geometric transformations or to perform a change between coordinate systems, we need a matrix \mathbf{M} that represents this transformation, as described in the last section. But how do we determine this matrix \mathbf{M} ? In principle there are two ways.

The first alternative is to know formulas for these matrices for certain standard cases. The formula for translation has already been given in Sect. 11.7. For rotation by an angle α around the x -axis around the origin point, the following formula can be found for the matrix \mathbf{M} :

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Accordingly, one can also find formulas for transformation matrices for rotation around the y -axis, around the z -axis or around any other axis, for reflection, or for scaling in computer graphics textbooks. From these standard cases, more complex transformations can be computed by concatenation (see Sect. 11.7). For example, if you want to calculate a rotation of 30° around the x -axis around the center of rotation (1, 2, 3), you divide this transformation into three transformations for which a formula is known: first, you perform a translation by $(-1, -2, -3)$, which takes the center of rotation to the origin (because we only know the formula for rotations around the origin). Then you rotate 30° around the x -axis around the origin point and reverse the first translation performed with the inverse translation. The matrix for the entire transformation is obtained by multiplying the three matrices for the standard cases (note the order):

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos 30^\circ & -\sin 30^\circ & 0 \\ 0 & \sin 30^\circ & \cos 30^\circ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The second alternative to determine the matrix \mathbf{M} , which we need according to the formula $\mathbf{p}' = \mathbf{M} \cdot \mathbf{p}$ to compute a transformation or to change coordinate systems, is to construct \mathbf{M} directly:

- We start with our coordinate system K , which consists of three base vectors and the origin point. We also need to know the target coordinate system K' after the transformation, which results from the geometrical transformation of the three base vectors and the origin point of K . Let \mathbf{M} be the matrix that changes coordinates from coordinate system K to K' , i.e., \mathbf{M} computes the geometric transformation from K to K' .
- We represent the first base vector of K' as a column matrix of size 4 by entering its three coordinates with respect to K in the first three rows of the column matrix and a zero in the fourth row. Analogously, we obtain column matrices for the second and third base vector of K' . We represent the origin point of K' by entering its coordinates with respect to K in the first three rows of a column matrix of size 4 and a one in the fourth column. From these four column matrices, we form the matrix \mathbf{M}^{-1} of size 4×4 by writing them next to each other according to the above order. By inverting \mathbf{M}^{-1} we obtain the matrix \mathbf{M} that we are looking for.

If a point P has coordinates (x, y, z) with respect to the old coordinate system K , its new coordinates with respect to K' are calculated with the matrix \mathbf{M} as follows:

- We represent P as a column matrix \mathbf{p} with the homogeneous coordinates $(x, y, z, 1)$.
- We calculate the matrix product $\mathbf{p}' = \mathbf{M} \cdot \mathbf{p}$
- The values in the first three rows of \mathbf{p}' are the coordinates of P with respect to the new coordinate system K'

About the Authors

Dr. Steffi Beckhaus is an expert on VR/AR and HCI. Her main research interest is to understand human perception, the power of imagination and, ultimately, the influencing factors on our felt sense of “reality”. She works as a coach and consultant for women, entrepreneurs and people pursuing advanced academic careers in STEM. Here, her topics include potential development, creativity and innovation training, career counselling, leadership development and hypnosystemic coaching. She was professor for interactive media and virtual environments at the University of Hamburg from 2004 to 2011. The R&D in her lab included the chairIO, multisensory, immersive VR-systems including floor-haptics and olfactory elements, qualitative, emotional experience spaces, storytelling, and media art. In 2011/2012 she was guest professor at TU Darmstadt. She holds degrees in Physics (1991) and Computer Science (1993) and certifications in generative and systemic coaching and consulting (2014–2018). At the beginning of her career, she worked for 8 years in the private sector as a researcher and IT consultant. In 2002, she obtained her multiply awarded PhD on navigation in virtual environments, while being part of the Virtual Environments Group of IMK at GMD. In this book, she authored Sect. 6.8.

Mathias Buhr studied Engineering & Computing at the TU Bergakademie Freiberg and continued as a research assistant at the Institute for Computer Science. In addition to his teaching activities in the field of human–machine communication, multimedia and parallel computers, his focus was on methods for distributed and parallel rendering techniques for virtual environments. Since 2014 he has been working in the field of audio/video signal processing and real-time communication protocols for LogMeIn, Inc. Mathias Buhr is the author of Sect. 7.2 and co-author of Sect. 7.3.

Dr. Wolfgang Broll is a full professor at TU Ilmenau (Ilmenau University of Technology), heading the Virtual Worlds and Digital Games group. He received his Master's (Dipl.-Inf.) in Computer Science from TU Darmstadt (Darmstadt University of Technology) in 1993 and a PhD in Computer Science from Tübingen University in 1998. He was a lecturer at RWTH Aachen (Aachen University of Technology) from 2000 to 2009. From 1994 to summer 2012 he headed the VR and AR activities at Fraunhofer FIT in Sankt Augustin. He has been doing research in augmented reality, shared virtual environments, multi-user VR and 3D interfaces since 1993. In addition to his academic activities, he was the founder and CEO of fayteq GmbH, and later a member of the board of fayteq AG until it was sold to Facebook Inc. in 2017. He is a SIGGRAPH pioneer, and vice chair of the steering committee of the VR/AR chapter of Germany's computer society (GI). He is a member of the Steering Committee of the IEEE Symposium on Mixed and Augmented Reality (ISMAR), where he served as Program Chair in 2016 and 2017. He is currently concerned with AR-related technologies, including, but not limited to, collaborative augmented environments, Diminished and Mediated Reality, global illumination, natural interaction and the application of deep learning approaches to those. Prof. Broll contributed to this book as editor for Chaps. 5, 8 and 10 and as second editor for Chaps. 3 and 6. As an author, he contributed to Chaps. 1, 4, 5, 8 and 10.

Dr. Carolina Cruz-Neira, a member of the National Academy of Engineering, is a pioneer in the areas of Virtual Reality and interactive visualization, having created a variety of VR technologies that have become standard tools in industry, government and academia. She is known for being the inventor of the CAVE. Having dedicated a part of her career to the transfer of research results into daily use she spearheaded several open-source initiatives and VR-related commercialization efforts. She is also recognized for having founded and led very successful VR research centers, such as the Virtual Reality Applications Center at Iowa State University, the Louisiana Immersive Technologies Enterprise and the Emerging Analytics Center at the University of Arkansas at Little Rock. She has been named one of the top innovators in VR and one of the top three greatest women visionaries in VR. She is an IEEE Fellow and has been inducted as an ACM Computer Pioneer. She has received the IEEE Virtual Reality Technical Achievement Award and the Distinguished Career Award from the International Digital Media & Arts Society, among other national and international recognitions. Currently, Dr. Cruz-Neira is the Agere Chair in Computer Science at the University of Central Florida. As an author, she contributed to Chap. 5 (mainly Sect. 5.4) and Sect. 7.3.

Dr. Ralf Doerner has been professor of Computer Graphics and Virtual Reality in the Design, Computer Science, Media department of the RheinMain University of Applied Sciences in Wiesbaden, Germany since 2004. After obtaining his Diploma degree in Computer Science from the Technical University of Darmstadt with distinction, he worked for the Fraunhofer Society, first as a researcher at the Fraunhofer

Institute for Computer Graphics, and later as head of the Mixed Reality department and vice director of Fraunhofer AGC in Frankfurt. After receiving his PhD (Goethe-University in Frankfurt, summa cum laude) and working as a scholar in the United States (University of New Hampshire/NOAA) with a DAAD PostDoc grant, he became a professor at Harz University of Applied Sciences before becoming full professor in Wiesbaden. He received an honorary professorship from the University of Transylvania, is a member of ACM SIGGRAPH, whose Recognition of Service Award he received, and has been elected to the board of the GI-working group VR/AR. His research interests lie in the field of visualization (interactive information visualization, visual data analysis), VR and MR (especially in the field of authoring systems) and the use of Computer Graphics for e-Learning and entertainment. He has been responsible for numerous research projects and has published over 150 peer-reviewed articles. Ralf Doerner served as editor of this book, being responsible for Chaps. 1, 2, 6, 9, and 11. He served as secondary editor for Chap. 7 and contributed as author to Chaps. 1, 2, 6, 9, 10, and 11.

Dr. Christian Geiger has been Professor of Mixed Reality and Visualization at the Düsseldorf University of Applied Sciences since 2004. Before that, he was a professor of 3D graphics and animation at the Harz University of Applied Sciences in Wernigerode. He studied computer science at the University of Paderborn and received his doctorate there in 1998 with a thesis on the creation of interactive 3D animations. From 1997 to 2000 he was responsible for R&D projects in the field of 3D graphics, multimedia and VR/AR at Siemens AG in Paderborn. His research interests lie in the design and implementation of novel user interfaces, mixed reality applications and interactive visualization techniques. As an author he contributed to Chap. 6, especially Sects. 6.1, 6.4 and 6.5

Dr. Martin Göbel is a consultant at the Institute of Visual Computing at the Bonn-Rhein-Sieg University of Applied Sciences in Sankt Augustin, Germany. Before that, he was CEO of 3DAround GmbH and of flexilution GmbH. From 1996 he was a competence center director for Virtual Environments in the GMD in Birlinghoven, where he implemented the first CAVE in Europe. From 1987, he was a senior scientist in the Fraunhofer Institute for Computer Graphics. Martin Göbel studied Computer Science at the Technical University of Darmstadt where he received the diploma degree (Master in Science) in 1982 and his PhD (Dr.-Ing.) in 1990. He is author and editor of several books on Graphics Standards, Visualization and Virtual Reality and over 100 scientific publications. He established the Eurographics Workshops on Virtual Environments (EGVE) in 1993 and chaired them. In 2003 the ARVR-group of the German Computer Society (GI-ARVR) was founded by him. Göbel has been program cochair of the EUROGRAPHICS '95 and '98 conferences and the IEEE VR 2001, 2002 & 2004 conferences and General Chair of IEEE VR2005 and 2006 as well as the Honorary Chair of IEEE VR 2015, IEEE VR 2018 and the ACM VRCAI 2019.

Dr. Paul Grimm is professor for Software Design and Architecture of Extended Reality and 3D Game Engines at Darmstadt University of Applied Sciences (h_da) in Germany. He has been Professor of Computer Graphics at Fulda University of Applied Sciences from 2011 until 2021. Before, he was Professor for Computer Graphics at Erfurt University of Applied Sciences since 2004. After studying computer science and physics at Technical University of Darmstadt, he worked as a research assistant at the Fraunhofer Institute for Computer Graphics (Fraunhofer IGD) in Darmstadt and at the Fraunhofer Application Center for Computer Graphics (Fraunhofer AGC) in Frankfurt. From 1997 to 1998 he was a visiting scientist at the National Center for Supercomputing Applications (NCSA) in Urbana-Champaign, USA. From 2009 to 2010 he did a research semester at Daimler Protics GmbH in the Virtual Engineering & Consulting division. His research interests focus on simplifying the creation of Virtual and Augmented Reality, and he has been pursuing this for more than 20 years in various national and international projects. He is a member of ACM and the German society of computer science (Gesellschaft für Informatik, GI) and was spokesman for the GI interest group for Animation and Simulation as well as a member of the management committee of the GI interest group VR and AR. Prof. Dr. Paul Grimm contributed to this book as editor, especially as editor of Chap. 4. For Chap. 5, 8, 9 and 11 he acted as a second editor. For Chap. 10, he collected four examples. As an author, he contributed to Chap. 1, 4 and 5.

Dr. Rigo Herold is a research scientist for Augmented Reality systems based on data glasses at the Westsaxony University of Applied Sciences, Zwickau. He received a diploma degree and an MS degree in electrical engineering from the University of Applied Sciences, Dresden, Germany in 2006 and 2007. He received a doctoral degree in electrical engineering from University of Duisburg-Essen in 2011. From 2007 to 2013 he was a research assistant at Fraunhofer-Gesellschaft. In 2013 he was appointed to a professorship at the Westsaxony University of Applied Sciences, Zwickau. His research interests are in mobile AR systems based on data glasses, human–computer interaction, mobile computing and the customized design of data glasses. As an author, he contributed to Chaps. 4 and 5.

Johannes Hummel was a PhD student at the German Aerospace Center (DLR) in Braunschweig in the field of virtual assembly simulations in orbit. Before that, he studied computer science and electrical engineering at the Technical University of Munich (TUM) from 2003 to 2009 and graduated with a diploma. From 2005 to 2010 he was a freelance software developer responsible for projects in the field of user interfaces and data management in the automotive industry. His research interests are in the field of Virtual Reality, especially multimodal interaction techniques for virtual assembly simulation in space. He wrote Sect. 4.4.

Dr. Bernhard Jung has been the chair for Virtual Reality and Multimedia at the Institute of Computer Science, Freiberg University of Mining and Technology since 2005. He studied computer science and computer linguistics at the University of

Stuttgart, Germany, and the University of Missouri, Saint Louis. He received his doctorate in 1996 from the University of Bielefeld with a thesis in artificial intelligence as well as a Habilitation degree in 2002 for a thesis on intelligent virtual environments. From 2003 to 2005 he was full professor for Media Informatics at the University of Lübeck's International School of New Media. He is a member the German Society for Computer Science (GI) and the German Society for Cognitive Science (GK). Prof. Jung's research interests are in the fields of virtual & mixed reality, large data visualization, virtual prototyping with HPC workflows, human-computer interaction and advanced robotics. He served as one of the co-editors of this book, primary editor of Chaps. 3 and 7, secondary editor of Chaps. 1, 2 and 10 and co-author of Chaps. 1, 3 and 7.

Rolf Kruse has been teaching and researching in the field of digital media at the Applied Computer Science department at the Erfurt University of Applied Sciences, Germany, since 2012. At the beginning of the 1990s, parallel to his degree in architecture, he conducted research at the 1st Demonstration Center for Virtual Reality of the Fraunhofer Institute for Computer Graphics Darmstadt (IGD). This research was continued in 1994 at Art & Com in Berlin with a focus on urban planning and the interaction of lay people with digital spatial content. As founder of the Laboratory for Media Architectures in 1997 and Invirt GmbH in 2008, he created hybrid interactive installations for well-known companies and public clients. From 2002 to 2005 he headed Cybernarium GmbH, a spin-off of IGD, which developed XR applications for educational and entertainment purposes and ran exhibitions attracting a wide audience. His current research focuses on applications and technologies for immersive learning and user experience design for spatial computing. Rolf Kruse actively supported the editors and authors of this textbook in structuring the content and uniform graphical presentation.

Dr. Leif Oppermann is head of the Mixed and Augmented Reality Solutions group at Fraunhofer FIT in Sankt Augustin, Germany, which is a part of their Cooperation Systems research department. He is researching into applications of mobile Mixed Reality, web-based collaboration and ubiquitous computing for intelligence augmentation using a user-oriented cooperative design approach. Prior to joining FIT, he was a research fellow at the Mixed Reality Lab of the University of Nottingham, UK, under Steve Benford und Tom Rodden, where he worked on pervasive gaming projects and completed his PhD in 2009 with a thesis on "Facilitating the Development of Location-Based Experiences". Leif has a background in real-time graphics programming and finished his Mediainformatics studies in 2003 at the Hochschule Harz with a work on interacting with surfaces in AR using head-mounted displays and a pointing device with distinction. He continued to work as a research associate on AR projects with Christian Geiger and Ralf Doerner before moving to Nottingham in 2004. Dr. Oppermann joined FIT in 2009 and currently leads the German national project "IndustrieStadtspark" on mobile applications for 5G in an industrial campus setting. For this book he contributed to Chap. 6, especially to Sects. 6.1, 6.3, 6.4 and 6.5.

Dr. Volker Paelke has been professor of Human–Computer Interaction at the University of Applied Science in Bremen since 2015. In 2002 he completed his doctorate on the “Design of Interactive 3D Illustrations” at the University of Paderborn, working in C-LAB, a joint venture with Siemens AG. From 2002 to 2004 he worked as a postdoc in the special research cluster SFB 614 Self-Optimizing Systems, researching the use of VR in collaborative engineering applications. In 2004 he was appointed to the junior professorship for 3D geovisualization and augmented reality at the Leibniz University of Hanover. From 2010 to 2012 he worked as institute professor and head of the 3D visualization and modeling group at the Geomatics Institute in Barcelona. From 2013 to 2014 he deputized as the professor for user-friendly design of technical systems at the Ostwestfalen-Lippe University of Applied Sciences in Lemgo and set up the User Experience Design group at Fraunhofer IOSB-INA in Lemgo. His research interests lie in the user-centered design of visual-interactive applications, with a focus on 3D visualization, AR/MR techniques and natural user interfaces. Prof. Paelke contributed to Chap. 6, especially Sects. 6.3 and 6.6.

Dr. Thies Pfeiffer is Professor for Human-Computer Interaction and Head of the Mixed Reality Laboratory at the Department for Electrical Engineering and Information Technology, Faculty of Technology, University of Applied Sciences Emden/Leer, Germany. He received his PhD in Informatics in the Artificial Intelligence Group headed by Prof. Dr. Ipke Wachsmuth at Bielefeld University in 2010. From 2013 to 2019 he was Technical Director of the Virtual Reality Lab and the Immersive Media Lab at the Cluster of Excellence Cognitive Interaction Technology (CITEC) at Bielefeld University. In his research, he focuses on Mixed Reality technologies for assistance and training. As author he was happy to contribute to Sect. 7.1.

Dr. Dirk Reinert is an Associate Professor in the Department of Computer Science at the University of Central Florida. He has Masters and PhD degrees from the Technical University of Darmstadt, Germany. Before joining academia, he worked for more than 10 years at the Fraunhofer Institute for Computer Graphics, the largest research group in the world for computer graphics, leading a variety of industry and public research projects in virtual and augmented reality and directing the development of the OpenSG system. His research interests are focused on software systems for Virtual and Augmented Reality, different applications of interactive 3D graphics and immersive display systems of all kinds. As an academic, he has received several best paper awards, over \$15 million in funding and several patents and open source licenses. He is a member of IEEE and ACM. As an author, he contributed to Chap. 5 (mainly Sect. 5.4) and Sect. 7.3.

Dr. Tobias Schwandt is a research assistant at the Ilmenau University of Technology in the Virtual Worlds and Digital Games research group. In his dissertation, he was particularly concerned with the illumination of virtual content in AR, its influence on the real environment, the reconstruction of environmental light and

the manipulation of real geometry by virtual content. Prior to that, he obtained a Master of Science in Applied Computer Science at the Erfurt University of Applied Sciences in 2014. He has a strong background in computer graphics and visualization of virtual content. He also spent some time at Fraunhofer IDMT in the research area of exer-learning games as part of the project HOPSCOTCH. At TU Ilmenau, eXtended-Reality (XR) became his professional and personal focus. His results were presented and published, among others, by IEEE ISMAR, GRAPP, CW and Springer. He co-authored Sects. 10.2 and 10.3.

Dr. Frank Steinicke is full professor of Human–Computer Interaction at the Department of Informatics at the University of Hamburg, and head of the Human–Computer Interaction research group. His research interests focus on three-dimensional user interfaces for computer-generated environments, with a special focus on virtual and augmented reality, multi-sensory perception and human–computer interaction. He studied mathematics with a minor in computer science at the University of Münster and graduated in 2002. In 2006 he received his doctorate in computer graphics and visualization at the Institute for Computer Science at the University of Münster. He then worked as a visiting professor at the Department of Computer Science at the University of Minnesota in Duluth (USA) in 2009. In 2010, Frank Steinicke received the Venia Legendi for computer science from the University of Münster. Before he accepted the call to Hamburg in 2014, he worked as a W2 professor for media informatics in Würzburg between 2011 and 2014. As an author, he contributed Sects. 2.1, 2.2, 2.3, 2.4.5, 2.4.6 and 2.5.2.

Dr. Arnd Vitzthum currently heads the media informatics group at the University of Cooperative Education in Dresden and educates students in the fields of Computer Graphics and Virtual Reality. He studied Computer Science at the Dresden University of Technology. From 2003 to 2008, he worked as a research and teaching assistant at the University of Munich, where he wrote his doctoral thesis on Software Engineering of 3D-Applications. From 2008 to 2011, he was a scientific staff member at the TU Bergakademie Freiberg where he contributed to the VR-related project “Virtual Workers” and led the project “Roundtrip 3D”, which was based on the results of his dissertation. Both projects were funded by the Deutsche Forschungsgesellschaft (DFG). Dr. Arnd Vitzthum wrote Sects. 3.2, 3.3, 3.4, and 3.5.

Kai Weber is studying applied computer science as a masters student with a focus on media informatics at the University of Applied Science in Fulda. He works as a research associate in the field of data analysis and was previously involved in several research projects in the field of augmented reality. Furthermore, he also acted as a supporting teacher in the field of 3D modeling, animation and game programming. Before that, he worked as a freelance in the field of media production and as a system administrator. During his bachelor’s thesis, he dealt with the generation and analysis of synthetic training data for artificial neural networks in the field of context-based image segmentation. In addition, he contributes as a 3D artist and developer in a video game project. Kai Weber contributed Sect. 10.3.3.

Dr. Florian Weidner has been working as a research assistant in the Virtual Worlds and Digital Games Group at the Ilmenau University of Technology since 2016. As part of his dissertation, he worked on the development and impact of Spatial Augmented Reality for virtual dashboards in vehicles. From 2009 to 2015, he studied Media Informatics with a minor in Biomedical Engineering at the Dresden University of Technology, where he received his M.Sc. degree. His other research interests include Virtual Reality, Mixed Reality and Augmented Reality, as well as input and output devices for these technologies. His results were presented, among others, at the IEEE Conference on Virtual Reality and 3D User Interfaces and the ACM Conference on Automotive User Interfaces and Interactive Vehicular Applications. He contributed to this book as an author on Sects. [10.2](#) and [10.3](#).

Index

A

AABB tree, 262
acceleration, 114
accommodation, 41, 45
accommodation-convergence discrepancy, 56
accommodation distance, 165
accretion, 47
accuracy, 107
acoustic, 113
acoustic attenuation, 95
acoustic tracking, 113
action at a distance, 212
active stereo method, 190
affine space, 405
albedo, 83
ambient light, 94
ambient occlusion, 85
ambient reflection, 82
ambisonics, 96
analysis of variances (ANOVA), 233
animation, 89
APP-CULL-DRAW, 285
appearance, 81, 211
application programming interface (API), 372
AR applications, 318
arcball, 213
ARCore, 396
area lights, 94
AR glasses, 151
AR System, 29, 30
ARToolkit, 25, 372
asthenopia, 60
asymmetrical viewing volume, 63
ataxia, 60
atmospheric perspective, 46

attention, 64
attention map, 65
AttrakDiff, 230
audible output, 193
audio source, 95
auditory perception, 48
augmented reality (AR), 10, 18–20, 291
augmented reflection technology (ART), 353
augmented virtuality, 19
author, 371
authoring process, 372, 375
autostereoscopic methods, 191
avatar, 97, 216, 362
axis-aligned bounding box (AABB), 258–259

B

backface culling, 276
backfaces, 81
background sound, 95
base, 403
behavior, 75, 91–93, 98
behavior trees, 92
believability, 17
between-group design, 229
bias, 229
billboards, 88, 100
binary space partitioning tree
(BSP tree), 264–266
binaural hearing, 95
binocular depth cue, 45
Binocular HMD, 166
Blender, 387
blending, 184, 351
Blueprints, 378, 389

Bonferroni correction, 233
 boundary representation (b-rep), 80
 bounding sphere, 259
 bounding volume (BV), 80, 90, 257, 274, 276, 285
 bounding volume hierarchies (BVHs), 262, 285
 box-and-whisker diagram, 231
 box plot, 231
 break in presence, 17
 brightness, 118
 brightness uniformity, 154
 broad phase, 267
 building information modeling (BIM), 345
 bump mapping, 84

C

C#, 375
 C++, 378
 caching, 281
 calibration, 112
 camera-based tracking, 108
 “camera-in-hand” technique, 217
 camouflage object, 313
 Cartesian coordinate system, 406
 causation, 233
 cave automatic virtual environment (CAVE), 25, 350
 ChairIO, 221
 change blindness, 220
 channel separation, 185
 Chi-square test, 234
 circular polarization filter, 187
 clipping, 273
 Cochran’s Q test, 234
 coding, 230
 cognitive map, 214
 cognitive processor, 40
 Cohen’s kappa, 230
 collaborative virtual environment (CVE), 155
 Collada, 387
 collision detection, 255–271
 collision engine, 90
 collision response, 270
 color picking, 206
 color reproduction, 153
 color space, 153
 complete survey, 232
 concatenation, 410
 cones, 41
 constraints, 208
 context-aware filling, 321
 context-sensitive, 321

contingency coefficient, 234
 continuous collision detection, 271
 contrast, 117
 contrast ratio, 138
 controlled experiment, 233
 control techniques, 217
 convergence, 43, 45
 conversion tools, 372
 Cook-Torrance model, 83
 coordinates, 403
 coordinate system, 405
 Coriolis stimulation, 61
 correlation, 233
 corresponding points, 43
 coupling-in, 158
 coupling-out, 158
 crossed disparity, 43
 cross product, 406
 cross-reality, 20
 crosstalk, 187
 culling, 272
 curved screen, 176
 cybersickness, 60, 247
 cyclopean scale, 54

D

dashboard, 311
 data glasses, 24
 DataGlove, 24
 DaVinci-stereopsis, 44
 debriefing, 229
 decision trees, 92
 deep learning, 324
 deep neural network, 86
 deformable object, 91
 depersonalization, 236
 depth buffer, 275, 312
 depth cameras, 306, 307
 depth cameras (RGBD cameras), 312
 depth cues, 45
 depth of field, 56
 descriptor, 127
 design activities, 225
 diamond square algorithm, 101
 diffuse reflection, 82
 digital light projector (DLP), 162
 dimension, 403
 diminished reality (DR), 20, 21, 314, 315, 321
 diplopia, 53
 directed acyclic graphs (DAGs), 75, 284
 directed line segments, 403
 directional light, 93
 direct manipulation, 203

discrete-oriented polytopes
 (k-DOPs), 261–262
 disparity, 43, 45
 displacement mapping, 84
 display, 111
 display latency, 249, 252
 display surface, 44
 distance, 406
 distortion maps, 169
 DLP link, 191
 dome projection, 176, 179
 Doppler effect, 96
 double vision, 53, 54
 drift, 111
 dwell time, 317
 dynamic depth cue, 45
 dynamic range, 153

E

early Z rejection, 276
 edge collapse, 86
 elastic interface, 220
 electrical muscle stimulation, 197
 electromagnetic tracking system, 250
 elevation grid, 101
 ELSI/ELSA, 237
 Emmert's law, 57
 end effector, 144
 end effector displays, 144
 end-to-end latency, 249, 254
 energy conservation, 83
 entity, 375
 entity-component model, 375, 379
 equirectangular function, 169
 escapism, 236
 ethics, 237
 ethics guidelines, 237
 Euclidean point space, 406
 exit pupil, 164
 exoskeleton, 135
 exploration, 215
 eXtended reality (XR), 19
 eye-directed control, 216
 eye motion box, 164
 eye relief, 164
 eye-tracking, 252, 317

F

factor, 233
 feature-based tracking techniques, 127
 feature integration theory, 65

feature points, 383
 features, 382, 383
 field, 401
 field of view (FOV), 52, 76, 215
 finger tracking, 111
 finite state machines (FSM), 91
 first person-games, 381
 Fisher's test, 234
 Fitts' Law, 208
 fixation maps, 66
 fixed-directions hulls (FDH), 261
 flashlight technique, 210
 focus, 208
 force feedback, 195
 fovea, 41
 foveated rendering, 252
 fractal shape, 102
 fragment shader, 85
 frame cancellation, 55
 frame rate, 247
 frame-rate induced delay, 249
 frames per second (fps), 153
 Fresnel reflection, 84
 Friedman test, 234
 front luminance, 158
 front projection, 175
 frustum, 273
 functional decomposition, 227
 fusion, 44

G

gabor filter, 66
 gain factor, 183
 gains, 59
 game AI, 92, 98, 285
 game engines, 74, 83, 103, 285, 372
 GameObjects, 375
 gamut, 153
 gaze-based interaction, 317
 geographic information system
 (GIS), 345
 geometric field of view, 58
 geometric registration, 293, 300
 geometric vector, 403
 gestures, 205
 ghosting, 187
 GJK algorithm, 269
 global illumination models, 94
 glossy reflections, 306
 gl transmission format (gITF), 82
 go-go technique, 210
 graphical user interfaces (GUIs), 14

graphics processing unit (GPU), 86, 251, 256,
271–273, 275, 279, 282, 283, 285
grounded theory, 230
guided navigation, 222

H

haptic feedback, 256
haptic loop, 195
haptic perception, 49
haptics, 49
hard edge, 184
Hawthorne effect, 229
Head Related Transfer Function (HRTF), 194
headlight, 95
head-mounted displays (HMDs), 12, 13, 306
head-related transfer function (HRTF), 48
head-tracking, 12
head-up content, 311
head-up displays (HUD), 320
height field, 101
height in the field of view, 46
hermann grid, 3
heuristic evaluation, 229
hierarchical finite state machines, 92
hierarchical view volume culling, 274
holodeck, 12
holographic objects, 390
holographic optical elements (HOE), 159
HoloLens, 390
HOMER, 210
Homogeneity, 154
homogeneous coordinates, 409
homography, 322, 324
horizontal prototypes, 228
horopter, 43
HTC Vive, 388
human-centered design, 224, 225
human–computer interaction (HCI), 202
human information processing, 41
hybrid tracking system, 250
hybrid tracking techniques, 131

I

illumination model, 82
image blur, 45
image layer technique, 210
image sharpness, 154
immersion, 13, 51
inattention blindness, 64
indexed face set, 79, 280
indexed mesh, 79, 280
indexing, 280

indirect illumination, 94
indirect lighting, 94
inertial navigation system (INS), 115
inertial sensor, 114
inertial tracking, 114–115
inertial tracking system, 250
informed consent, 229
inhibition, 66
input devices, 107–146
inside-out, 113
inside-out tracking, 113
interaction by navigation, 316
Interactivity, 72
interference filter, 188
interpupillary distance (IPD), 166
interview, 230
intuitive user interface, 15
inverse kinematics, 98
inverse matrix, 409
involvement, 17
isometric interface, 220
isotonic interface, 220
ISO 9241-210, 225

J

java native interface (JNI), 397
Javascript, 375
joy of use, 202

K

Kalman filters, 131
k-d tree, 265, 266, 274
keyframe, 89
keyframe animation, 89
kinaesthesia, 49
known size, 46
Kruskal-Wallis test, 234

L

landmark knowledge, 214
Laser-based tracking, 115–116
latency, 245–248, 250–252, 300
latency determination, 250
latency requirements, 247
Latin square, 229
leaning-based interface, 220
lenticular lenses, 191
level of detail (LOD), 87, 278, 285
light attenuation, 93
light detection and ranging (LiDAR), 382, 383
lightmap, 94

lightmap baking, 94
 light probes, 94, 304
 light sources, 93
 Likert scale, 231
 line, 407
 linear combination, 403
 linearly independent, 403
 linear perspective, 46
 local coordinate system, 76
 local illumination models, 94
 local interaction techniques, 208
 logical input devices, 203
 L-shapes, 175
 luminance, 153
 luminous flux, 153
 luminous intensity, 153

M

magical 3D interaction, 203
 magic lens, 307
 magic lens metaphor, 295
 magic wand, 207
 magnetic field-based tracking, 114
 maneuvering, 216
 manipulation, 211
 Mann-Whitney-U-test, 234
 mapping, 382
 marker-based methods, 118–122
 marker-based optical tracking, 252
 marker-based tracking, 122
 markerless, 118
 material, 82, 283
 material systems, 82
 matrix, 408
 matrix-matrix addition, 408
 matrix-matrix multiplication, 408
 McNemar's test, 234
 mechanical tracking system, 254
 median, 231
 mediated reality, 20, 305
 mental model, 202
 metalness, 83
 metaphor, 15, 202
 micro-facets, 83
 Midas touch problem, 209
 midpoint displacement method, 101
 Minkowski difference, 269
 Minkowski sum, 269
 mixed reality (MR), 19
 mobile position tracking, 116
 mobile systems, 155
 mock-up, 227
 mode errors, 209

monocular depth cue, 45
 Monocular HMD, 166
 morality, 237
 motion capture, 98
 motion parallax, 47
 motion sickness, 60
 motion sickness assessment questionnaire (MSAQ), 62
 motion-to-photon latency, 249
 motor processor, 40
 multi-channel audio systems, 194
 multimodal, 13
 multisensory perception, 48
 multi-sided projection systems, 195

N

narrow phase, 267, 268
 natural 3D interaction, 203
 natural user interfaces, 213
 navigation, 214
 negative parallax, 44
 Newtonian physics, 100
 nimbus, 208
 nominal, 231
 non-player characters (NPCs), 97
 normalized device coordinates (NDC), 279
 normal mapping, 84
 normal vector, 407
 null hypothesis, 232
 number of degrees of freedom, 110, 208

O

OBB tree, 262
 obtrusiveness, 112, 155
 occluders, 312
 occlusion, 45
 occlusion culling, 275, 276
 occlusion query, 276
 octree, 266, 274
 off-axis method, 63
 omni light, 93
 one-sample *t*-test, 234
 one-tailed test, 233
 one-way ANOVA, 234
 opacity, 84
 openVR, 385
 optical flow, 50
 optical see-through AR (OST-AR), 296
 optical see-through displays (OST displays), 158
 optical tracking system, 250
 ordinal, 231

oriented bounding boxes (OBBs), 259–261
 origin, 405
 orthogonal, 406
 outdoor position tracking, 116–117
 outside-in, 118
 outside-in tracking, 146
 overuse, 238
 overview knowledge, 215

P

paired t-test, 234
 Panum's fusional area, 44
 paradoxical window, 55
 parallax, 44
 parallax barriers, 192
 parallax budget, 54
 particle system, 88, 99
 path planning, 222
 pathtracing, 94
 Pearson correlation, 234
 perceptual processor, 40
 perspective, 295
 PHANTom, 25
 phantom objects, 312
 Phong model, 82
 photogrammetry, 73
 photometric registration, 293, 304, 309
 physically based rendering (PBR), 82
 physics engine, 90
 physics simulation, 256, 270
 picking, 206
 pictorial depth cue, 45
 pilot, 229
 place illusion, 17
 plane, 407
 plausibility illusion, 17
 point & teleport method, 221
 pointing device, 206
 pointing gestures, 213
 point light, 93
 point-normal form, 407
 point-point subtraction, 404
 point-vector addition, 405
 polarization, 186
 polygon, 78
 polygon mesh, 79
 polygon soup, 280
 portal culling, 277
 pose estimation, 292
 position estimation, 293
 position vector, 405
 positive parallax, 44
 postural instability theory, 61

post-WIMP interface, 15
 preattentive processing, 65
 Prefabs, 375
 presence, 17, 51
 pre-test, 229
 priming, 235
 primitive instancing, 81
 prism-based glasses, 159
 probability value (p-value), 233
 procedural knowledge, 215
 procedural modeling, 73, 101–103
 programming interface, 372
 programming libraries, 372
 projection matrix, 274
 projection system, 151
 proprioception, 49
 props, 205
 Proteus effect, 362
 prototype, 228
 proxy, 317
 pupil distance, 53
 pupil forming HMDs, 164

Q

q% quantile, 231
 quadrilateral strips, 282
 quadtree, 266
 qualitative analysis, 230
 quasimodes, 209
 questionnaire, 230

R

radiosity, 94
 randomized approach, 323
 RANSAC, 127
 rational, 231
 ray-casting, 206, 210
 raytracing, 86, 94
 reality–virtuality continuum, 19
 real-time capability, 72, 246
 real-time rendering, 271
 rear projection, 174
 redirected free exploration with distractors (RFED), 219
 redirected walking (RDW), 67, 219
 refresh rate, 247, 256
 registration, 300
 regression analysis, 233
 relative size, 46
 remote interaction techniques, 208
 rendering latency, 249, 251
 repeated measures ANOVA, 234

- resolution, 110
- retina, 41
- retinal HMDs, 161
- RGBD, 128
- rigging, 97
- right-handed system, 407
- rigid bodies, 89
- robustness of linear perspective, 52
- rods, 41
- roughness, 83
- route knowledge, 215
- rubber hand illusion, 235

- S**
- saccades, 41
- saliency, 65
- saliency map, 65
- sample, 232
- scalar, 401
- scalar-matrix multiplication, 408
- scalar multiplication, 402, 403
- scalar product, 405
- scale invariant feature transform (SIFT), 128
- scenario-based design, 227
- scene, 75
- scene graph, 74–77, 376, 378, 395
- scene graph systems, 284–285
- Scripts, 375
- seamless display, 180
- search, 216
- seasickness, 60
- seating buck, 334
- selection, 205
- self-perception, 237
- semantic differential scale, 231
- sensitivity, 112
- sensory conflict theory, 61
- separating axis, 260
- separating axis theorem (SAT), 260
- serious games, 333
- shader, 83, 85, 86
- shadow cast, 305
- shadows, 46
- shape from shading, 46
- Shapiro-Wilks test, 232
- shutter glasses, 177
- significance level, 233
- simplification of polygon meshes, 74
- simulator sickness, 60
- simulator sickness questionnaire (SSQ), 62
- simultaneous localization and mapbuilding (SLAM), 306, 382
- single pass stereo, 251
- single-sided displays, 173
- size constancy, 46, 57
- skeleton-based animation, 97
- skinning, 97
- sky box, 96
- sky sphere, 96
- small feature culling, 277
- smart projectors, 309
- soft bodies, 89
- soft edge, 184
- software development kit (SDK), 372
- software tools, 372
- solid model, 80, 81
- solid particle systems, 100
- Sort & Sweep, 267
- sound, 95, 96
- space partitioning, 263
- spatial AR (SAR), 297, 309
- spatial audio sources, 95
- spatial augmented reality (SAR), 347
- spatial hashing, 264
- spatial mapping, 393
- Spearman correlation, 234
- specular reflection, 82
- speed constancy, 50
- speeded up robust features (SURF), 128
- speed teleporting method, 222
- sphere tree, 262
- spherical displays, 175
- spherical harmonics, 305
- spot light, 93
- standard deviation, 232
- state machines, 91, 98
- stationary systems, 151
- statistically significant, 233
- steamVR, 385
- stereo blindness, 44
- stereo display, 44
- stereopsis, 42
- stereoscopic rendering, 251
- stereoscopic window violation, 55
- stereotype, 235
- strafing, 216
- stripping, 282
- structured light, 307
- superimposition, 293
- supernatural user interfaces, 213
- surface model, 78
- suspension of disbelief, 7
- Sweep & Prune, 267, 268

T

tail, 403
 tangibles, 205, 317
 tangible user interfaces (TUI), 317
 task analysis, 227
 task load index, 230
 task maps, 65
 technology assessment (TA), 237
 technology readiness levels (TRLs), 332
 teleportation, 217
 temporal coherence, 268
 terrain modeling, 101
 test plan, 229
 tethered, 211
 texture, 84, 85
 texture baking, 87
 texture gradient, 46
 texturing, 84
 thinking aloud test, 230
 3D computer graphics, 13
 3D cursor, 207
 3D interaction, 12
 3D widgets, 204
 tiled displays, 151
 tiled projections, 179
 tiling, 101
 time, 107
 Time of Flight (TOF), 113
 tip, 403
 toe-in method, 63
 tool chain, 372
 tracking, 12, 292, 300, 382
 tracking latency, 248, 250, 252
 tracking systems, 250
 tracking techniques, 107
 tracking update rate, 300
 tracking using black and white markers, 122–126
 transcutaneous electrical nerve stimulation, 197
 transformation matrix, 76
 translucency, 83
 transparency, 83
 transport latency, 248, 251
 transposed matrix, 408
 traveling, 215
 treatment factor, 233
 triangle strips, 79, 282
 Tukey box plot, 231
 two-tailed test, 232

U

ultimate display, 12, 23
 ultrasound-based systems, 197
 UML use case diagram, 227
 uncrossed disparity, 43
 uniformity, 151
 Unity, 375, 385, 390
 unpaired t-test, 234
 unreal engine (UE), 378, 388
 update rate, 111
 urban canyon, 116
 usability, 202
 use case, 227
 user experience, 202
 user tests, 228–234

V

vection, 50, 62
 vector, 402
 vector addition, 402
 vector space, 401
 vergence-accommodation conflict, 56
 vergence-focus conflict, 56
 vertex shader, 85
 vertical parallax, 63
 vertical prototypes, 228
 vertigo, 247
 vestibular sense, 50
 vestibulo-ocular reflex, 248
 video pass-through AR, 295
 video see-through AR (VST-AR), 295
 video see-through displays (VST displays), 158
 video stream, 292
 view frustum culling, 273
 view volume, 256
 view volume culling, 256, 273, 274, 278, 285
 vignetting, 183
 virtual body ownership (VBO) illusion, 362
 virtual display, 161
 virtual environment, 6
 virtual eye separation, 53
 virtual field of view, 294
 virtual hand, 207, 213
 virtual humans, 97
 virtual prototyping, 97
 virtual reality (VR), 20
 virtual reality locomotion, 216
 virtual reality markup language (VRML), 74

virtual world, 6, 71
visibility testing, 272
visual field, 151
visual programming, 373
visual programming approach, 378
visual realism, 72
visual SLAM, 130
visual system, 2
Vive Cosmos, 388
voice commands, 205
volumetric displays, 44
von Neumann bottleneck, 281
voodoo dolls, 214
Vortex rings, 197
VR/AR application, 371
VR/AR assets, 371
VR glasses, 136
VR sickness, 60
VR system, 6, 26–28

W

walking in place, 218
wave field synthesis, 96
waveguide optics, 158

wavelength multiple, 188
wayfinding, 214
what-you-see-is-what-you-get principle
(WYSIWYG), 373
white flash, 191
whole-body illusion, 235
Wilcoxon rank sum test, 234
Wilcoxon signed rank test, 234
Windows, Icon, Menu, Pointer
(WIMP), 202
winner-takes-all approach, 66
wired clothing, 27
within-group design, 229
world-in-miniature (WIM), 210

X

X-ray vision, 314
X3D, 74

Z

Z-buffer, 275
zero vector, 402
Z pre-pass, 276