Evolution of VR Software and Hardware for Explosion and Fire Safety Assessment and Training

Chad Jarvis, Veronika Solteszova, Dag Magne Ulvang, Djurre Siccama, and Daniel Patel

Abstract Building and maintaining a custom-made Virtual Reality (VR) system is expensive and time consuming. The recent availability of affordable and capable head mounted displays (HMD) and graphics cards along with powerful 3D game engines has created new opportunities for implementing VR systems. In this chapter we describe the historical development of software and hardware for VR systems and applications from the 90s to today, as well as the advantages, disadvantages and challenges that the recent developments have introduced. The development is described through the journey of porting a VR system from a custom made game engine displayed on a backprojected screen to using an inexpensive off-the-shelf system with the Unity game engine displayed in an HMD.

1 Introduction

Several VR systems used in the areas of oil and gas exploration and medical visualization had been developed by Christian Michelsen Research (CMR), which is now

Chad Jarvis, Veronika Solteszova: Contribution described here was done when working at NORCE Research.

C. Jarvis (✉)
GraphCore, Oslo, Norway
e-mail: chadj@graphcore.ai

V. Solteszova
Equinor, Bergen, Norway
e-mail: ves@equinor.com

D. M. Ulvang · D. Siccama
Gexcon AS, Bergen, Norway
e-mail: dagu@gexcon.com

D. Siccama
e-mail: Djurre.Siccama@gexcon.com

D. Patel
Rapid Geology AS and Western Norway University of Applied Sciences (HVL), Bergen, Norway
e-mail: daniel.patel@hvl.no

D. Patel (ed.), *Interactive Data Processing and 3D Visualization of the Solid Earth*,
https://doi.org/10.1007/978-3-030-90716-7_8

part of Norwegian Research Centre (NORCE). One of these systems was for risk assessment, risk communication and safety training in fire and explosion scenarios. The VR hardware consisted of a large screen display and shuttered glasses for stereo viewing, including positional and rotational tracking. The software was an in-house developed custom 3D engine. Custom 3D engines are time consuming to develop and maintain, and the VR hardware we used took much space and was costly. However, the recent availability of affordable but advanced VR HMDs, combined with powerful 3D game engines such as Unity [1] and Unreal Engine [2], has created a new opportunity in VR technology.

To test the potential of this new technology we ported the functionality of our custom VR system into the Unity game engine and used the Oculus Rift DK2 [3] for rendering and tracking. In this chapter we will discuss our previous experience with VR technology and then compare and contrast it with the new technology.

2 Background

HMDs have been used extensively for military training purposes [4] and are entering the public domain for medical training [5]. Recently, companies are increasingly embracing VR for training their personnel in fields such as safety and security, sales, engineering, hospitality, HR, leadership and education [6].

In 2004, CMR started development of a VR solution called VRSafety [7, 8], funded by Equinor and Norsk Hydro. VRSafety was developed for evacuation and safety training and for discussing and reviewing structural changes in existing or planned industrial environments. Computational Fluid Dynamics (CFD) simulations calculated in external software were imported and visualized to simulate fire, gas leak and explosion scenarios inside the models. To visualize the simulation results, we implemented volume rendering and isosurface rendering. To discuss mitigation steps, we made it possible to do basic editing of the geometry model. In effect, we had created a digital twin with respect to visual appearance, and visualization of fire, gas and explosion scenarios that could be used for learning and training on e.g. hazard identification and to find evacuation routes. As spatial understanding and the sense of presence is important, particularly in training scenarios, we implemented support for 3D immersion using VR. In addition to training, experts and non-experts could use the solution to communicate hazard and risk topics, e.g., by showing how much larger an explosion would be if a certain opening would be closed off with a wall.

VRSafety had implemented a real time connection to the FLACS (Flame Acceleration Simulator) CFD simulator [9], this enabled defining explosion, gas leak and fire scenarios in VR and immediately starting a FLACS CFD simulation allowing real-time visualization of the calculated results in the VR environment. However, real-time visualization of the simulation output was not practically useful due to long computation times. It took in the order of seconds to simulate a single timestep, which could be between a fraction of a second to a few seconds long depending on the type of simulation. So in most cases the scenarios to be discussed were computed in advance before running the VRSafety application.

2.1 Evolution of VR Hardware and Software

Before starting the development of VRSafety, CMR had already implemented another 3D engine for supporting VR applications in oil and gas exploration and production, called SHIVR. This 3D engine, which is described in the chapter titled "When Visualization and Virtual Reality made a Paradigm Shift in Oil and Gas", was developed for UNIX running on an SGI ONYX visualization system. The graphics rendering of the 3D engine was built on top of the low-level OpenGL library [10], and it supported immersive visualization in CAVE setups [11] through the use of the (discontinued) CAVELib library. CAVElib handled the graphics display on each wall of the CAVE, including computation of correct angles and projections. For tracking the user, hardware technology (discontinued) was used which consisted of wired electromagnetic sensors (Ascension Flock of Birds [12]). The engine had collaborative capabilities where several instances of the software could run at different sites, enabling geographically dispersed users to meet and collaborate within the virtual environment.

Interfacing OpenGL directly is a time-consuming way of programming computer graphics applications, as almost every high-level functionality has to be implemented from scratch. Therefore we based the 3D engine for the VRSafety application on a higher graphics abstraction level provided by the OpenGL Performer scene graph API by SGI (now discontinued). On top of OpenGL Performer we implemented our own navigation system, event system, and XML meta-language for controlling behavior in the virtual environment [13]. As SGI started experiencing financial difficulties, the support on the hardware and libraries we were using became uncertain. Therefore, we switched from SGI's OpenGL Performer scene graph to OpenScenegraph [14] and ported our 3D engine from UNIX to Microsoft Windows. Due to the rapid development of the game industry, we were now able to replace the expensive SGI Onyx OS and hardware systems with a high-end PC and commodity graphics cards costing a fraction of the price, without losing performance. In addition, we no longer needed a separate server room for the large SGI Onyx rack setup. At the same time, we exchanged CaveLib by VRjuggler [15] which had no license fees associated with it. We were also able to get rid of wired tracking by replacing the Flock Of Birds tracking system with the IO Tracker [16] tracking system which was using infrared cameras, emitters and reflective markers.

VRSafety was designed for running in a CAVE environment, but during development we were running the application in a downscaled immersive VR setup consisting of a $4.6\,m \times 1.6\,m$ back-projected screen, powered by two partly overlapping Barco Galaxy NW-7 projectors. The overlap was smoothed using edge blending, resulting in a resolution of 2048×1600 120 Hz. Stereo was achieved using active stereo shutter glasses from Nvidia that were synchronized with the display through an infrared signal. Positional and rotational tracking of the glasses and of the pointing device was achieved with the IO Tracker system using reflective markers. The markers were tracked by infrared emitting and receiving cameras positioned around the bevel of the display, see Fig. 1.

Fig. 1 Immersive display environment consisting of a back projected screen with infrared tracking cameras around the screen bevel

2.2 Experiences with VRSafety

The new VRSafety setup was considerably improved, but still consisted of expensive hardware that required a custom-made visualization room, with an additional projector room located behind the screen. In addition, an initial manual calibration process had to be performed by trained personell, both for the projectors and the tracking system. The setup was ideal for simultaneously immersing multiple people in the virtual environment, and for facilitating collaborative discussion sessions between experts from different disciplines. In our particular setup, we only had the front wall projected, resulting in a lower degree of immersion than in a multiwall CAVE. We used this setup as it could also be used as an ordinary large screen for meetings, and it was more affordable than a CAVE setup. Our immersive work sessions did usually not produce motion sickness among the participants. Compared to the currently available rendering engines and frameworks, requests for added functionality was time consuming to implement due to our in-house developed framework. The framework also naturally did not benefit from bug fixes and feature updates from external actors.

3 Head Mounted Displays and Game Engines

3.1 Evolution of HMDs

Because of the recent interest in VR from companies such as Facebook, HTC, Samsung and Sony, affordable HMDs have now become commercially available. These

devices offer similar features, such as six degrees of freedom and a wide field of view (FOV). Sony's device is geared toward the PlayStation game console whereas the others are run by connected PCs or inbuilt mobile devices. Now that head mounted displays are commercially available and there are several companies competing in the marked, the hardware is rapidly improving. HMDs available in the 90s suffered from a very narrow field of view. Some had angular tracking, but to also have positional tracking, external sensors had to be attached such as the wired electromagnetic Flock of Birds sensor. Since the first prototypes of HMDs, the field of view has increased dramatically, and 6 degrees of freedom tracking is supported both on the HMD and on the hand-controllers. Tracking has evolved from using several external cameras for detecting the position and orientation of a device, called outside-in tracking, to using cameras embedded in the device, called inside-out tracking. The latter solution requires less hardware setup and calibration, and does not limit the movement to a confined area within the cameras' view. Most HMDs require cables, but cableless HMDs are appearing. One example is the Oculus Quest device which is based on a stand-alone Android mobile device with inside-out tracking. Such systems allow the user to move freely around in the physical environment. The disadvantage with using mobile computation of VR is that it is less performant than an HMD connected with cables to a powerful computer. It is possible to send images wirelessly from a powerful computer to an HMD, but this adds some latency between a user's action and an updated rendering. If latency can be reduced to become unnoticeable, one will be able to achieve high quality rendering on lightweight wireless HMDs which could improve the user experience dramatically. Another recent development is eye tracking integrated into the HMD which makes it possible to track what the user is looking at. This feature is already available in the VIVE Pro Eye device [17]. Eye tracking can open up for more intuitive interaction with the VR world, and for higher framerates or better-quality renderings by focusing the rendering computation to the area that the user is looking at. This is called foveated rendering [18].

3.2 The Oculus Rift HMD

We used the Oculus Rift DK2 HMD, as this was the only commercially available HMD at the onset of the project. The lenses used in the Oculus Rift creates two distortions: pincushion distortion and chromatic aberration. These distortions are corrected for by convoluting the image with a barrel distortion and distorting the red, blue, and green components of the image to cancel out the chromatic aberration. Both the 3D position and angles of the DK2 is tracked. The position is measured by a stationary infrared camera observing at 60 FPS infrared (IR) emissions from an array of IR-LEDS on the DK2. The angles are measured with an accelerator in the DK2.

3.3 Unity and Unreal Engine 3D Software

To find out which 3D engine we should port VRSafety to, we evaluated the Unreal and Unity game engines because of their state-of-the-art features and liberal end user licenses. The Crytek CryEngine also offers state-of-the-art features but is only allowed to be used for game development [19]. The engine cannot be used for scientific simulations and serious games as the end user license agreement states: "Under this Agreement the following will not be considered Games: military projects; gambling; simulation (technical, scientific, other); science; architecture; pornography; Serious Games".

Unity and the Unreal Engine are multi-platform game engines. They provide advanced functionality that was lacking in our previous system such as a powerful scene editor for adding geometry, interactive landscape shaping, (dynamic) foliage and visual environmental and weather effects. Additionally, all objects can be scripted for adding behavior to them.

Both game engines are reasonably priced with scalable fees according to either game revenue or development licenses used. Unreal has a 5% royalty fee for revenues above 1 million USD per product [20]. If the product is distributed through the Oculus store however, the royalty free revenue limit is raised to 5 million USD [21]. In 2019 Unreal removed the restriction regarding using the engine for gambling-related activities, for military activities with live combat, in nuclear facilities, or in critical aircraft software.

Unity has three licenses. With the personal license, products created with Unity can be used, distributed and sold without fees by entities earning less than 100,000 USD per year. Entities earning less than 200,000 USD can use the Plus licence for 35 USD per month or 299 USD per year. Entities earning more than 200,000 USD must use the professional license which will cost 150 USD per month per developer or 1800 USD per year [22]. In 2019 Unity removed the restriction regarding using the engine for gambling-related activities. The licenses described here are from 2021 and may change from year to year.

4 Porting from VRSafety to a Modern Head Mounted Display and Game Engine

In the following text we will refer to the new platform which we port VRSafety to, as VRFlacs. The porting consisted of two main tasks, transferring assets such as geometry models to a game engine, and implementing/porting the software functionality found in VRSafety.

4.1 Porting of Geometry Models

The geometry constituting the industrial model was originally in a CAD format but was provided to us as an OpenScenegraph Binary file (IVE) of 668 Mb. The geometry consists approximately of 5 million triangles

Both game engines support the FilmBox format (FBX) [23], while Unity additionally supports Collada (DAE) [24]. Therefore we focused on converting the geometry to FBX. We identified two geometry software packages which advertised the ability to convert from OpenScenegraph exportable file formats to FBX. Sketchup Pro [25] could import DAE and export FBX, but the software crashed during import. Blender [26] could transform both Autodesks [27] 3ds Max format (3DS) and DAE format to FBX, but Unreal failed to load the FBX file created both from 3DS and from DAE. It appears that Unreal's FBX importing was not suitable for large CAD files. Although, we were pleased with the Unreal Engine editor and impressed with the visual realism of the Unreal Engine, due to the failure to import our model, we decided to continue with the Unity engine. We did not investigate the many third party importer plugins for Unreal, nor the latest version of the engine, which might have solved this problem.

Unity successfully imported the FBX file, however, the original smooth shading was turned into flat shading. Loading the DAE format that had been exported from OpenScenegraph maintained the smooth shading, however all textures had been lost during conversion, but it was relatively easy to reassign the textures thanks to the Unity editor allowing for editing geometry. In addition, many surfaces flickered due to overlap with other surfaces with different colors (Z-fighting). This issue was also quickly resolved in the Unity editor by deleting one of the overlapping surfaces. Figure 2 shows an overhead view of the model imported into Unity after being updated in the Unity editor.

4.1.1 Adding Terrain, Sea and Sky

Compared to VRSafety which lacks a graphical editor, Unity supports many advanced features that can be created easily within the integrated graphical scene editor. Unity also has an assets store where one can purchase textures, models and visual effects. Our original model used in VRSafety was positioned inside a large sky-textured box that rendered the sky. It also included geometry representing the surrounding terrain. As Unity supports several sky-models and allows for interactive terrain sculpting and the adding of foliage, we replaced the original sky and terrain with Unity-made models. In addition, we added animated trees and an animated sea into the model. Using the interactive editor in Unity, we were able to create surroundings that more accurately represented the real surroundings of the model, see Fig. 3 bottom.

Unity supports several extensions for improved rendering quality, such as advanced shading models with shadow casting, and dynamic content such as animated environmental effects in water, trees and leaves. Adding too many of these features quickly degraded the framerate to below comfortable levels for VR. Rendering our scene in

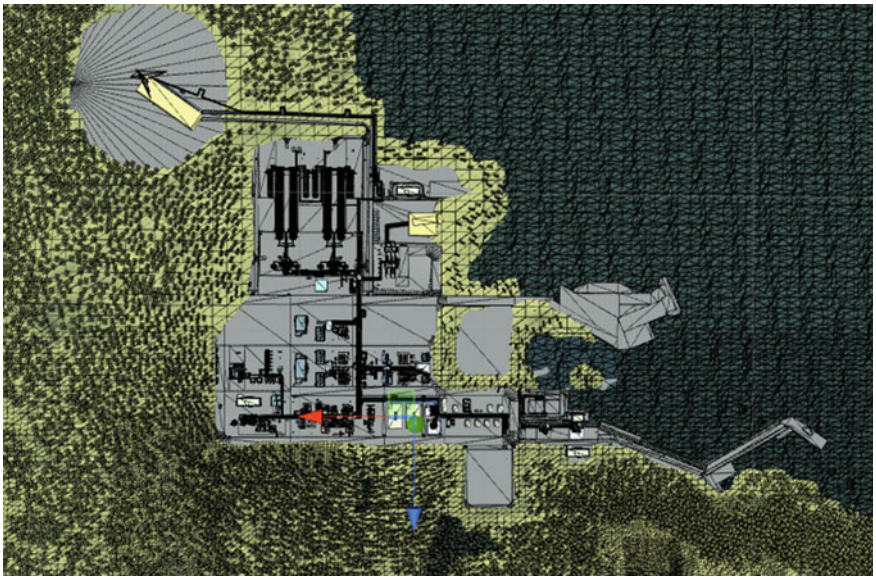

Fig. 2 Top view of the industrial model in the Unity editor. The sea and the surrounding terrain with foliage were created in the Unity editor

Oculus Rift with the basic Phong shading model as used in our previous VRSafety system, and having disabled animation of water and trees, resulted in interactive framerates at just over 70 FPS on an Nvidia GTX580 graphics card from 2011. This is an outdated graphics card, but provides at least a reference number for the framerate.

4.2 Implementing Software Functionality

Adding functionality such as navigation and collision detection in Unity was simply a matter of importing an asset and clicking a check box. Other functionality available in VRSafety, such as moving objects around in the scene, has not been implemented in Unity, but by inspecting the game engine design and scripting functionality we believe this would be easy to implement. VRSafety supported a two-way interactive connection with the FLACS CFD fire, leak and explosion simulator. However, this was not practically useful due to slow simulation times, therefore it was not ported to VRFlacs. For visualizing the simulation output, VRSafety had implemented an iso-surface renderer for showing boundary surfaces for a user defined value and attribute, and a volume renderer for showing all values of an attribute mapped with a color table. In VRFlacs, we implemented only a volume renderer since it could be used to render isosurfaces also.

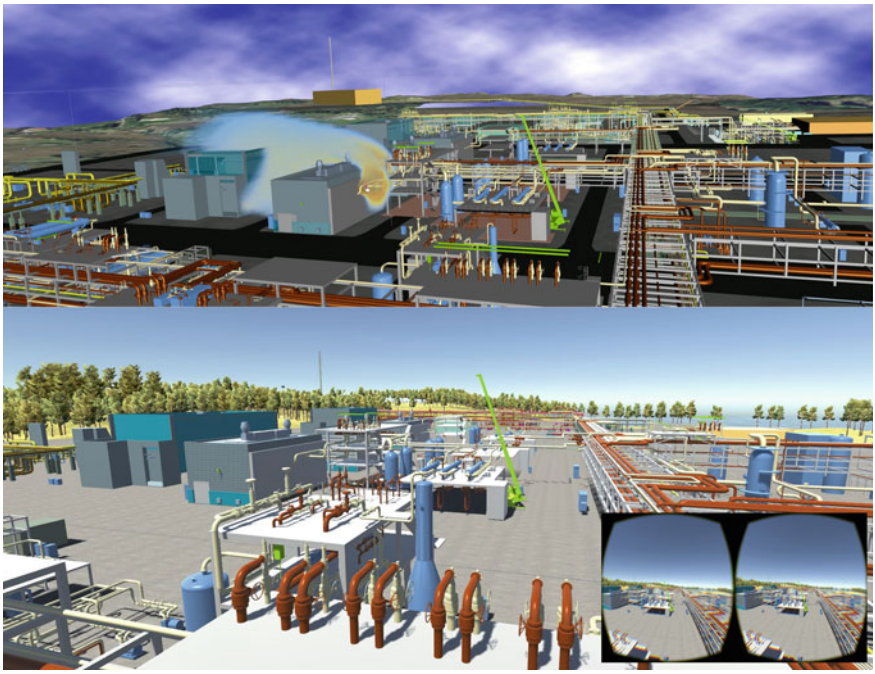

Fig. 3 Top: Rendering from VRSafety showing geometry and volume rendering of a gas leak. Bottom: Rendering from VRFlacs using Unity showing geometry with trees, shadows and water in the distance. Bottom Right inset: Stereoscopic rendering generated for the DK2

4.2.1 Volume Rendering

The CFD simulator used for computing gas dispersion and gas explosions generates a 3D volumetric dataset for each attribute and timestep. Volume rendering is a suitable method for visualization of such datasets. As Unity does not have built-in volume rendering capabilities, we implemented this ourselves. There are parameters that must be set before starting a simulation. The timestep size and spatial resolution of the simulation grid is manually set to achieve sufficient accuracy and to capture relevant features. Simulations such as explosions that start and end in a fraction of a second require smaller timesteps than e.g. a slow burning oil fire. The attributes to be output could be temperature, pressure, gas composition or a multitude of other attributes that are calculated by the simulator. As the volumetric data output consumes much memory, a decision to only store e.g. every 2nd or 4th timestep to the VR environment can be taken.

To show the data quantitatively, a slice through the volume at a user-defined position can be displayed. The user chooses which scalar simulation value to display, for instance temperature, and chooses a colormap for translating the scalar value to a color. This is useful for reviewing the simulator output.

For giving a qualitative impression of a fire scenario, the fire is rendered in a realistic manner. To achieve this, the simulator must calculate temperature and soot particles per volume unit (i.e. per voxel). The renderer then maps temperature to a wavelength spectrum using Planck's law of black-body radiation [28], and soot is mapped to opacity. The wavelength spectrum is further mapped to an RGB color value using the CIE 1931 color spaces [29] which define quantitative links between a wavelength spectrum and physiologically perceived colors in human vision. The RGB color represents chromaticity only (Planckian locus). To add lightness to the color according to how much energy is radiated from the point, the color is modified using the power of the spectrum. An example is shown in Fig. 5, bottom.

Gas leaks are mostly invisible, so in order to visualize them in VRSafety and in VRFlacs in Unity, we used a non-physically based mapping from gas density to color, which gave the leak a cloud-like appearance. Such a rendering from VRSafety is shown in Fig. 3, top. We can also show the extent of the gas leak by volume rendering a single isosurface for a user defined gas density, for instance for lethal or combustible levels of gas, giving the impression of a growing (semitransparent) bubble around the critical area.

In Unity, we implemented slice-based volume rendering [30]. This represents a volume as a set of semitransparent slices stacked behind each other, always facing the viewer, see Fig. 4 for a 2D schematic view. Each slice is a textured rectangle represented in Unity as a Quad GameObject. A higher number of slices improves the accuracy of the rendering at the cost of reduced framerate. Parts of slices that are behind existing scene geometry are automatically hidden due to depth buffering, and this ensures that the volume rendering is integrated with the geometry rendering. Because of this, only opaque geometry is supported inside the volume rendering.

To increase quality without decreasing the framerate, we use a smaller distance between slices when they are closer to the viewer instead of having slices evenly spaced. This works since data closer to the viewer is more visible which affects the resulting rendering more than data further away. We used an initial distance of 0.5 voxels for the first slice (Nyquist sampling rate) and increased this with 1% per additional slice. The user can speed up the volume rendering at the cost of reduced quality by increasing the initial and thereby the following slice distances. Each slice represents the optical properties of a slab around the slice. This is depicted in Fig. 4, where slices are shown with stippled vertical lines, and slabs with solid vertical lines. For simplicity, we use a constant distance between slices in the illustration. The blue rotated square represents the bounds of the volume data and the black circle represents opaque geometry inside the volume. The color and transparency of a given pixel on the texture of a slice depends on the data in the volume at the 3D position of the pixel, and how this value is mapped to color and opacity. Since the slice represents a slab of a specific thickness, this thickness will affect the transparency of the pixel. The thickness is affected by a slab's intersection with other geometry in the volume, and by its intersection with the volume bounds. In Fig. 4 is shown line intervals of different colors representing slab thicknesses. To improve rendering quality, the predefined slab thickness (sampling distance) is reduced to fit the front of the volume bounds (green), the back of the volume bounds (orange) or the front of geometry

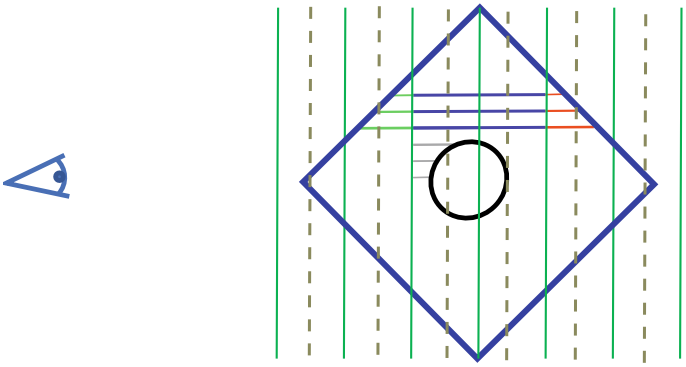

Fig. 4 The eye represents the viewing direction. Blue square shows volume bounds, black circle represents opaque geometry inside the volume rendering. Stippled vertical lines represent slices, green vertical lines represent slab borders. Horizontal lines represent different slice thicknesses due to intersection with volume bounds (green and red) or geometry (grey)

(gray). In blue is shown a few internal intervals where no adjustment is needed. There is also no adjustment behind the geometry since this area will not be visible as we only support opaque geometry. When not taking these distances into account, artifacts as shown in Fig. 5 top will be visible which are more pronounced when using larger slice distances. Figure 5 top shows jagged artifacts in square numbered 1) and 2) arising from not taking into account the distance to geometry and to front of volume bounding box respectively. Bottom figure shows the rendering when these distances are taken into account.

To calculate the distance to geometry, we accessed the depth buffer after rendering the scene geometry. The distance to the cuboid volume bounds was calculated analytically. We experienced that the programming interface for low level access to graphics features such as the depth buffer was not always direct and efficient in Unity. This resulted in a reduced framerate as compared to e.g. an OpenGL implementation.

4.2.2 Particle System Rendering

We faced several problems when visualizing the simulations using volume rendering. The graphics-intensive volume visualization reduced the framerate to below the recommended framerates for an optimal VR experience. In addition, simulations take up much space in memory which limits the length and size of the simulations that can be displayed. Finally, there is the problem of simulations that have long timesteps, e.g. the simulation of a long running fire. Since the fire lasts longer and is less dynamic than an explosion, the timestep of the simulator is set to one second to save memory. However, animating a fire which changes appearance only every second breaks the realism. In order to address the issues of memory usage and nonsmooth animation of fire, we looked at how fire and explosions are expressed in games with limited computing and memory consumption. One common technique in games to visualize

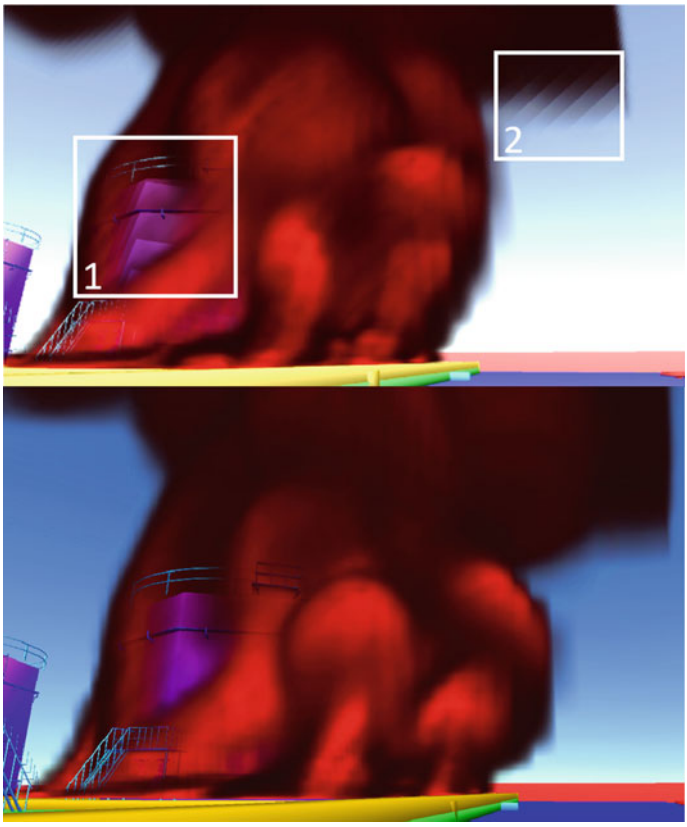

Fig. 5 Slice-based volume rendering shown together with geometry in Unity. Top: Jagged artifacts when not taking into account distance to geometry (1) and to volume bounding box (2). Colors have been exaggerated to better show the artifacts. Bottom: Volume rendering when taking these distances into account (The two images show data from two slightly different timesteps)

fires is to use particle systems [31] where each particle is a rising billboard having predefined fire animations playing on the surface of the billboard. A billboard is a 2D view-aligned textured surfaces with transparency masking. The animated billboard functionality is integrated in Unity's particle system module via the Texture sheet Animation module [32]. Several predefined fire particle systems can be directly downloaded through the Unity Asset Store, which made it easy for us to implement this functionality.

For a specific training scenario where accurate simulations have been performed, we simply positioned particle systems for fire at the positions of the fires, and set their parameters to approximatively match the simulation results. We also calculated and showed the accumulated radiation received by the player. This was calculated based on the volumetric output from the simulator. See Fig. 6 for renderings of fire and smoke using particle systems. To automatically get more accurate renderings

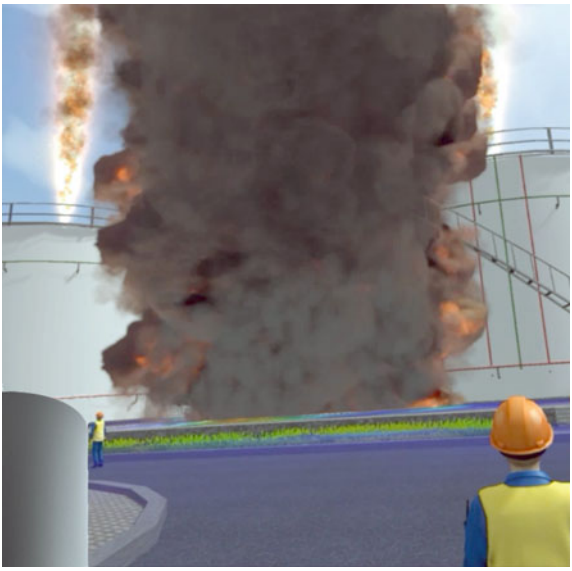

Fig. 6 Particle systems used to visualize fire and smoke. In the top left corner is visualized an ignited gas leak. Colors are exaggerated in the figure to better show the results

of fire using particle systems without manually setting parameters for the particle systems, we also experimented with limiting the particle movements based on the simulation data. This approach looked promising, but we did not have time to explore this sufficiently.

4.2.3 Support for Multiple Users

In VRSafety, local collaboration was possible simply because the participants were physically present in the same room and could see and talk to each other. In Unity, the application was running in an HMD, which is designed for one person only. To support multiple collaborators, the multiplayer functionality supported in Unity was used, where several instances of the application, each running on an individual computer, are synchronized. In this mode, the users can see the avatars of each other, see Fig. 7.

5 Modes of Work in VR

VRSafety supported a desktop mode where a single user would typically review and inspect simulation results on a desktop computer by looking at the volume rendering animations, using quantitative slice visualization at specific timesteps and using a probe to read out simulation values at specific positions in space. For discussing the simulation results with a group of people, VRSafety could be run in VR where

Fig. 7 Multiuser mode. Two users (bottom right) are immersed. Third-person view seen by each user is shown in bottom left and main image respectively

the same operations could be performed as in desktop mode. A different mode of operations was to run VRSafety in VR for the purpose of replaying a realistic recreation of a simulated situation and immersing a group of participants into the situation. This was useful for efficient communicating the consequences of leak, explosion and fire situations. In this mode, VRSafety needed an operator to guide the participants through the scenario.

In Unity, there was built-in support for navigation by walking and running, for animation of realistic characters and for creating game logic that triggers events based on a user's actions in the world. Together with the high immersion created by the HMDs, this made it easy to create a VR training mode where the user learns by being exposed to various situations in the virtual world which must be handled correctly. This mode of work enables safe and affordable training on dangerous situations. Figure 8 shows one such scenario that we created, where the user's actions affected how events unfolded. These scenarios supported multiple immersed users as shown in Fig. 7.

6 The Immersive Experience in VRFlacs Compared to VRSafety

Those that had used the VRSafety application in our back projected single wall setup with 3D glasses found the experience with the Oculus Rift far more immersive. Several people trying VRFlacs with an HMD found it so immersive that they expected to see their hands and arms when they moved them. Newer HMDs from Oculus (Rift S) now include two tracked hand controllers which makes this possible.

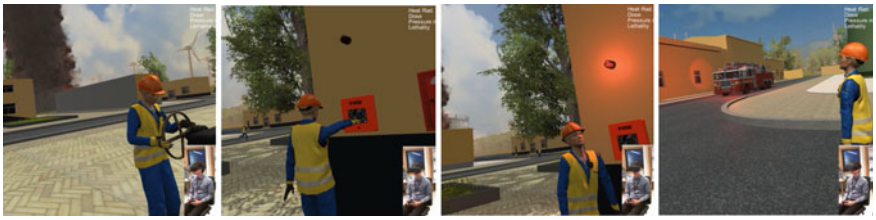

Fig. 8 A scenario playing out from left to right. A user detects a fire, localizes the valve and closes the gas supply to the fire. The user sets of fire alarm. Fire alarm is set of. The fire truck arrives

We did face some new challenges using an HMD for VR. Initially, a first-person perspective was used when being immersed. After an approximately 30 min of immersion, the main author experienced nausea. This lasted for over an hour. Reducing negative sideeffects from VR is an actively researched field [33]. Limiting the user's ability to perform changes in movement, and using HMD's with increased field of view or increased framerates are examples of techniques that help. In our case, we switched from a first-person to a third-person perspective which gave fewer viewpoint changes when navigating in VR and this reduced nausea.

A problem we faced with the complex terrain we added along with shadows and a dynamic ocean, is that it had a dramatic effect on the framerate of the application which was particularly noticeable during rotational head movements. Lag in rotational head movements was not a problem in the VRSafety solution since the rendering on the large-screen display did not have to change much when the user rotated his/her head. The low frame rate in our new VRFlacs framework not only damages the immersive experience but also contributes to motion sickness.

We believe that with a more modern GPU along with a better understanding of the Unity engine, we can achieve a sufficiently high framerate. For now, we have created two version of our demonstration: one with a complex terrain and ocean and one with a simple terrain and no ocean. In addition, we do not use volume rendering in immersive training scenarios. Instead we use the approximate particle system rendering.

Another challenge with the DK2 HMD compared to a large-screen setup is that the HMD blocks the view of the physical room one is situated in. Thus, collaborative sessions where several users see each other and e.g., point at parts of the model using hand gestures is not supported. Current HMDs come with tracked hand controllers, which will make it possible to show the participants as avatars with arms. Technology is developing that captures subtle body language and face mimic which is an important part of communication. Recently the VIVE Facial Tracker [34] has been released for the consumer market. It is is a device that works with HMDs and tracks the movement of the lower face including the mouth. This can substantially increase the ability to capture face mimic. There are now technological advances where the hand controllers might be replaced by sensors around the wrist that register the finger positions using cameras [35] or by reading the electric signals from muscles through the skin [36].

Table 1 Comparing the software aspects of the solutions. "+" is better than "(+)", which is better than "−"

	Custom Engine (VRSafety)	Commercial Game Engine
Importing large geometry	+	(+)
Fast implementation	-	+
Rendering quality	-	+
Adding "standard" functionality	-	+
Adding "nonstandard" functionality	+	-
Pricing scheme	Free	Per dev. licence / Per sale

Motion sickness and the reduced ability to communicate with collaborators in the same room are probably the two largest challenges using HMDs compared to using shared stereoscopic screens or CAVEs. We have summarized in Table 1 the differences between our previous system (VRSafety) that used a large stereoscopic screen with shuttered glass and the current one (VRFlacs) that used the Oculus Rift. In Table 1, we compare the software aspects of the two solutions. We use the term "standard" functionality for functionality that is common in games such as realistic environments, collision detection, animation and multiplayer support. Conversely, with "nonstandard" functionality we refer to features not common in games such as volume rendering, which we could not implement as efficiently as we wished due to lack of enough low-level graphics control in Unity. In sum, we spent a substantially less amount of time on implementation in Unity than with OpenSceneGraph in VRSafety. This is also thanks to Unity's integrated developer environment, good debugging abilities, and the interpreted C# language that does not require time consuming compilation. This is reflected by giving the category "Fast implementation" a + for Unity in the Table. VRSafety was developed in-house and did not have a purchase cost, while Unity has a cost per license. The license cost far outweighs the extra hours required for implementing a custom engine. The risk of using an externally provided game engine for a domain that it is not exactly designed for is that only after investing a certain (possibly large) amount of time, one may realize that a specific functionality is not possible to implement in a satisfactory manner. This is less likely to happen for a custom-made engine where one has more control.

Table 2 shows the differences between presenting VR through large screens and in CAVEs (Large-screen VR) compared to using HMDs (HMD VR). HMDs are more affordable than large-screen solutions. For fast rotational head movements in an HMD, a high framerate is required to not experience lag. However when the display is not attached to the head, this is not a problem. Perhaps partly related to the rotational framerate, motion sickness was less of a problem on large-screen VR in our experiences. When considering only the hardware, it is easier to collaborate with multiple people in the same room for Large-screen VR solutions than for HMDs as HMDs block the view of the room and the participants.

Table 2 Comparing the hardware aspects of the old and new solution

	Large-screen VR	HMD VR
Hardware price	high	low
"Rotational framerate"	high	medium/low
Motion sickness	low	medium/high
Collaboration support	high	medium/low

7 Discussion and Conclusions

VRSafety was developed as a tool for training and safety assessment of industrial environments. The costs for developing and maintaining a custom 3D graphics engine made it challenging to convince companies to pursue further investments. The recent availability of inexpensive HMDs with six-degrees of freedom tracking and a wide FOV, along with the availability of advanced 3D engines have rekindled our interest in VR as a viable platform for immersive training.

Our initial experience using the Oculus Rift DK2 device has been positive, as exploring the model of an industrial complex has never felt so immersive. Additionally, using Unity made it simple to reproduce many of the features we have in our custom 3D engine with much less effort, and the high visual quality provided by Unity generated a much more realistic visualization than we achieved with our 3D engine. The implemented volume rendering in Unity did not achieve high enough framerates for being used in a HMD in a training scenario. A faster but less accurate particle system rendering was instead used to give a qualitative impression of fire.

The major problem using the Oculus Rift DK2 was the motion sickness. This is a serious issue which needs to be carefully addressed. This may be as simple as avoiding low frame rates and lag to more restrictive solutions locking the user to a fixed inertial reference frame such as the cockpit of an automobile.

References

1. The Unity Engine. https://unity3d.com/unity. Accessed Mar 2021
2. The Unreal Engine. https://www.unrealengine.com/. Accessed Mar 2021
3. Oculus VR. https://www.oculus.com/. Accessed Mar 2021
4. A. Rizzo, A. Hartholt, M. Grimani, A. Leeds, M. Liewer, Virtual reality exposure therapy for combat-related posttraumatic stress disorder. IEEE Comput. **7 (2014)**
5. M. Bressler, A virtual reality training tool for upper limb prostheses. Masters thesis (2013)
6. Warp vr customer stories and use cases. www.warpvr.com/customer-stories. Accessed Mar 2021
7. J. O. Erdal, T. Langeland, D. Patel, I. Eliassen, FAVEum framework architecture for virtual environments applied to urban modelling, in *Next Generation 3D City Models* (2005), pp. 298–300

8. S. Hoiset, E. Glittum, Risk and safety training using virtual reality (VRSafety), in *SPE International Conference on Health, Safety, and Environment in Oil and Gas Exploration and Production* (2008)
9. FLACS Software, GEXCON. https://www.gexcon.com/products-services/FLACS-Software/22/products-services/flacs-software/. Accessed Mar 2021
10. The Industry's Foundation for High Performance Graphics. https://www.opengl.org/. Accessed Mar 2021
11. C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart, The CAVE: audio visual experience automatic virtual environment, in *Communication of the ACM 35* (1992)
12. Ascension Flock of Birds. https://est-kl.com/manufacturer/ascension/flock-of-birds.html. Accessed Mar 2021
13. D. Patel, I. Eliassen, T. Langeland, FAVE, a framework architecture for virtual environments, in *The 3rd CAVE-Programming Workshop* (Helsinki, Finland, 2003)
14. OpenSceneGraph. http://www.openscenegraph.org/. Accessed Mar 2021
15. A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, C. Cruz-Neira, Vr juggler: a virtual platform for virtual reality application development, in *Proceedings IEEE Virtual Reality*, vol. 2001 (2001), pp. 89–96
16. T. Pintaric, H. Kaufmann, Affordable infrared-optical pose-tracking for virtual and augmented reality, in *Proceedings of Trends and Issues in Tracking for Virtual Environments Workshop, IEEE VR* (2007)
17. Vive PRO Eye HMD. https://enterprise.vive.com/us/product/vive-pro-eye-office/. Accessed Mar 2021
18. A. Patney, M. Salvi, J. Kim, A. Kaplanyan, C. Wyman, N. Benty, D. Luebke, A. Lefohn, Towards foveated rendering for gaze-tracked virtual reality. ACM Trans. Graph. **35** (2016)
19. CryEngine. https://www.cryengine.com/ce-terms. Accessed Mar 2021
20. Unreal License. https://www.unrealengine.com/en-US/eula. Accessed Mar 2021
21. UnrealEULA. https://developer.oculus.com/documentation/unreal/unreal-oculus-license/. Accessed Mar 2021
22. Unity License. https://unity3d.com/legal/terms-of-service. Accessed Mar 2021
23. Fbx Format. http://help.autodesk.com/view/FBX/2017/ENU/. Accessed Mar 2021
24. Collada Format. https://www.khronos.org/collada/. Accessed Mar 2021
25. Sketchup. https://www.sketchup.com/products/sketchup-pro. Accessed Mar 2021
26. Blender Software. https://www.blender.org/. Accessed Mar 2021
27. 3DS Format. https://www.autodesk.no/products/3ds-max/overview. Accessed Mar 2021
28. R. Loudon, *The Quantum Theory of Light (third ed.)* (Cambridge University Press, 2000)
29. T. Smith, J. Guild, The C.I.E. colorimetric standards and their use. Trans. Opt Soc., **33** (1932)
30. J. E. Swan II, R. Yagel, Slice-based volume rendering. Technical Report OSU-ACCAD-1/93-TR1 (Ohio State University, Jan 1993)
31. W. T. Reeves, Particle systems—a technique for modeling a class of fuzzy objects. Trans. Graph
32. Unity Manual: Texture Sheet Animation module. https://docs.unity3d.com/Manual/PartSysTexSheetAnimModule.html. Accessed Mar 2021
33. T. Porcino, D. Trevisan, E. Clua, Minimizing cybersickness in head-mounted display systems: causes and strategies review, in *2020 22nd Symposium on Virtual and Augmented Reality (SVR)* (2020)
34. VIVE Facial Tracker. https://www.vive.com/eu/accessory/facial-tracker/. Accessed Mar 2021
35. F. Hu, P. He, S. Xu, Y. Li, C. Zhang, Fingertrak: continuous 3d hand pose tracking by deep learning hand silhouettes captured by miniature thermal cameras on wrist, in *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2020)
36. B. Mu, R. De Nardi, R. Newcombe, R. King, E. Gander, R. Wang, Armband for tracking hand motion using electrical impedance measurement. Facebook Technologies (2019)