Hilfert and König *Visualization in Engineering* (2016) 4:2
DOI 10.1186/s40327-015-0031-5

Visualization in Engineering
a SpringerOpen Journal

RESEARCH Open Access

Low-cost virtual reality environment for engineering and construction

Thomas Hilfert* and Markus König

Abstract

Background: Presenting significant building or engineering 3D-models is a crucial part of the planning, construction and maintenance phases in terms of collaboration and understanding. Especially in complex or large-scale models, immersion is one of the major key factors for being able to intuitively perceive all aspects of the scene. A fully immersive system needs to give the user a large field-of-view with reduced latency for lifelike impression. Technologies such as VRwalls and shutter glasses can deliver high refresh rates, yet fail to give a large field-of-view. Head-mounted-devices for virtual reality fill this gap. Head tracking mechanisms translate movements of the user's head into virtual camera movements and enable a natural way of examining models. Unlike a stereoscopic representation with projectors, point-of-view tracking can be achieved separately for each individual user. Hardware costs for such systems were very high in the past, but have dropped due to virtual reality systems now gaining traction in the mainstream gaming community.

Methods: In this paper we present a way to build a low-cost, highly immersive virtual reality environment for engineering and construction applications. Furthermore, we present a method to simplify and partly automate the process of reusing digital building models, which are already used in construction, to create virtual scenes, instead of having to do parallel content creation for visualization. Using the Oculus Rift head-mounted display and the Leap Motion hand-tracking device, we show the possibilities of naturally interacting within a virtual space in different use cases. The software, based on the popular game engine Unreal Engine 4, will be used as a basis for further research and development.

Results: Building Information Modeling data can be imported to UE4 with our presented plugin. Using an automated database for mapping materials to the geometry simplifies the process of importing Building Information Modeling entities. The refresh rate of the system stays within acceptable margins needed for virtual reality applications using head-mounted devices.

Conclusions: Head-mounted devices present a great potential for the Architecture, Engineering and Construction industry, as a person can experience realistic first-person situations without having to care about injuries. Automated processes for the simplification of content creation, leveraging existing models, and the usage of visual programming languages enable even nonprogrammers to create scenarios to their needs.

Keywords: Virtual Reality, Visualization, Construction, Engineering, Head-mounted devices, Building Information Modeling

* Correspondence: thomas.hilfert@rub.de
Computing in Engineering, Ruhr-Universität Bochum, Universitätsstraße 150, Bochum 44801, Germany

Background

Head mounted devices (HMD) for Virtual Reality (VR) are currently experiencing a renaissance. In the past, these systems were only accessible for large companies at high costs (starting at several thousand Euros) and using specialized systems. Even then, the user experience was under par. Devices were lacking the refresh rate needed to smoothly translate head movements in the virtual world and had inadequate resolution and a low field-of-view (FOV) for realistic impressions. Sufficient refresh rates of displays and updates in the virtual world are key factors to consider when using VR, as the user will be otherwise likely prone to motion sickness on longer exposure.

In contrast to traditional surface based stereoscopic methods, such as 3D shutter glass monitors or beamers, HMDs enable individual rendering of the user's perspective. Movements of the head and / or body are being translated into movements of the virtual avatar. The camera location and rotation will then be modified to the matching position. These techniques are similar to the field of Augmented Reality (AR), omitting the blending with the real world. AR will become more important in the future, but lacks the fully free designed environment without the boundaries of reality VR has.

While a designer or engineer has learned to use his imagination during the design phase to visualize the final product, this is difficult for outsiders. Being able to present 3D models and scenes to a wide range of audience is therefore beneficial in engineering and construction. We chose to use a game engine as basis of our research, as they are trimmed to maximum performance at a high level of detail. Also, in contrast to using a pure graphics engine, sound playback, physics and networking abilities are already present. Having game developers targeting the engine for plugin development is also a plus, as ongoing feature extensions and bug fixes will be available.

When creating 3D building models in construction, additional resources have to be assigned for the conversion into a realistic visual representation or the buildings have to be recreated from scratch for the different 3D formats for visualization, which are most likely incompatible among each other. If additional iterations of those models are created by the engineers, time consuming and costly effort has to be spent on adapting the newest changes into the VR scenes. Also, the visual representation of the entity in 3D space has to be defined after each import, so that realistic materials can be used in the visualization. Minimizing the complexity and automating this migration process into the VR presentations would be a great benefit for the construction industry.

In this paper we present a solution for building a VR environment, consisting of different hardware and software components to achieve a deep level of immersion to users while minimizing the effort of creating 3D scenes for VR from building models. We show applications for the environment via three different, common use cases. Additionally, all of the components mentioned are cheap to buy, enabling even non-professionals access to such simulations. Every part can be used independently, but play well together on maximizing the immersion experience and interaction.

Related research

In Sampaio and Martins (2014) the application of VR to visualize the construction of a bridge using two different construction methods is described. The target of this implementation was to give students a deeper understanding about how bridges are built, as they can't have the same level of insight on a real construction site due to safety reasons. They target mainly the creation and deployment of 3D models over traditional teaching practice (verbal description / pictures / diagrams) and come to the conclusion that the interaction is one main benefit of using VR. However, they don't elaborate on proper input methods for natural interaction with the system and the high level of immersion a HMD would give.

Grabowski and Jankowski (2015) are testing VR training for coal miners using different HMD hardware setups in conjunction with joystick and VR glove input methods. They found out that the test subjects preferred high immersive VR, but the FOV was negligible. The reason may have been, the hardware used for the 110° FOV is known to be prone to ghosting (artefacts from previously rendered frames still partly visible, causing motion sickness) and this may be diminishing the positive effects of having a high FOV. Nevertheless, the study showed that the use of a vision based system for detecting natural hand movements is better than wireless 3D joysticks and that the result of the training "[…] is maintained in the long term." (Grabowski and Jankowski, 2015, p. 321).

Rüppel and Schatz (2011) describe the creation of a serious game environment for evacuation simulation in case of a fire. They harness Building Information Modeling (BIM) data as a base for their building-modelling concept and describe the advantages of having material data included to simulate the structural damage. The VR-lab concept described uses ideally most human senses (visual, tactile, auditory, olfactory) and enables interaction with the scene. However, their proposal references expensive components for the environment, by means of VR-lab facilities. Using the low-cost equipment described in this paper may benefit creating an easier deployable and affordable setup with only minor drawbacks (missing olfactory or tactile senses), but with more immersive visual representations. Surround headsets and / or binaural

recordings can replace the audio installation described in their paper.

Merchant et al. (2014) explore different publications on virtual reality-based learning with regard to the outcomes in education. During the assessment they found out there are many examples of VR being beneficial for learning and "virtual reality-based instruction is an effective means of enhance learning outcomes" (Merchant et al., 2014, p. 37). As stated, one reason for not having widespread VR in education is financial feasibility. Also, VR environments based on desktop 3D computers do not provide fully immersive experiences, but enhance the learners' engagement (Merchant et al., 2014, p. 30). We conclude that having a low-cost VR environment at disposal, which works on a normal desktop level, may be favourable for the Architecture, Engineering and Construction (AEC) industry, as well as for education outside of this field. There may be some applications where regular serious games on a computer monitor may be better to use, as mostly everyone can use a mouse or keyboard to operate traditional user interfaces (e.g., entering numeric values). However, operations, such as pushing buttons or levers, may manifest more naturally in a students' memory if real world movements are usable and being tracked into VR.

Roupé et al. (2014) support the aforementioned statement, in saying that "body movement can enhance navigation performance and experience" (Roupé et al., 2014, p. 43), but the ability of tracking parts of the body usually needs expensive equipment. The authors describe a system using an Xbox Kinect sensor for navigating planned urban environments. By using body postures (leaning back and forth, turning the shoulders) participants are able to navigate through a VR-model of a city. They were able to perceive distances and scales inside the environment better when using body movements. As translating postures to VR navigation is a first step, an optimal solution would be to map walking movements directly with dedicated hardware, if possible.

Edwards et al. (2015) show the feasibility of using a game engine to include end-users in the BIM design process. They use an Autodesk Revit plugin for communicating with the Unity game engine and enable collaboration over a networked connection. They also extract BIM information via a local webserver started in Revit, but this is only a short-lived instance and is disabled, as soon as a dialog is closed. A central server for administrating projects and revisions would be beneficial to integrate the already existing design process. Furthermore, the implementation is missing the VR integration we aim to achieve.

All studies show that there is demand for VR in many fields. It may have not been considered testing the actual deployment in some due to financial concerns of such a system. Providing a low-cost flexible environment which delivers safe testing and natural interactions is desirable. Especially in fields where hazards limit trial and error testing, VR can gain a foothold.

Concept

The goal of our approach is to unify an environment for different use cases in engineering and construction. As BIM is being introduced throughout the construction industry, support for Industry Foundation Classes (IFC) is beneficial. The IFC enable data exchange between a wide range of software applications and design tools. IFC models consist of optional geometry information with attached metadata, such as materials used and product structure. By retaining this metadata information within the virtual environment (VE), interaction with the elements enables more profound experiences. If a user wants to check which material a certain wall is composed of, this is seamlessly possible without breaking immersion.

A VE consists of a visual output element and one or multiple input elements to enable interaction with the system. Creating natural input methods are a hurdle at building such a system. They are either camera-based rigs with multiple cameras using visual detection algorithms, handheld devices or specialized depth sensors. Tracking the hands in relation to the virtual body position seems to be the most promising way to manipulate objects in the virtual world, as the learning curve for new users is modest and tracking reliability is not limited by hidden body parts due to an unfavourable camera position.

Ideally, off-the-shelf equipment should be usable for the whole system, as this will lower the total cost and increase deployment options.

Hardware setup

Several different display technologies are possible to immerse users into a virtual world. In contrast to beamer supported solutions, namely CAVE (Cave Automatic Virtual Environment) or similar shutter / polarized 3D walls, HMDs allow for full first person immersion. Therefore, we will only further elaborate these kinds of display devices. For our application, the currently most widespread, low cost single device, the Oculus Rift, will be used.

Oculus Rift

Recent innovations in display technology enable us nowadays to have low cost HMDs by using display panels intended for mobile phones. The most prominent device is the Rift from Oculus, commonly referred to as "Oculus Rift" (Oculus, 2015). It uses a single mobile phone display, derived from the Samsung Galaxy Note 3 device (1920x1080 pixels, overclocked at a 75 Hz refresh rate). While older HMDs incorporate independent

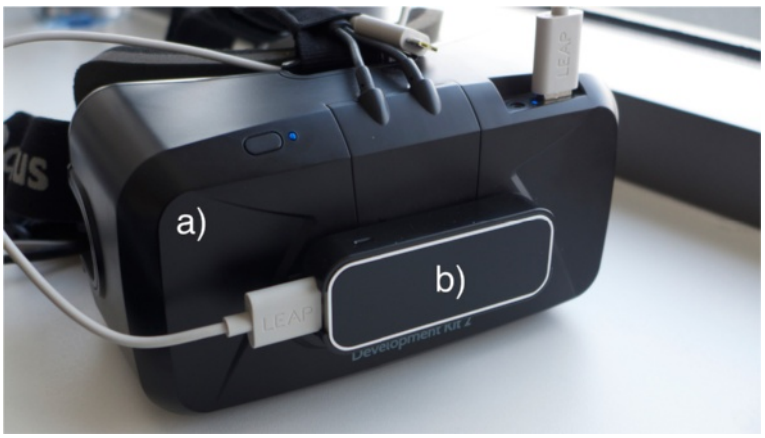

Fig. 1 Oculus Rift (**a**) with mounted Leap Motion (**b**)

displays, the picture displayed on a Rift screen is divided into two different parts and then separated for each eye by using lenses, providing an effective resolution of 960x1080 pixels per eye with 100° nominal field-of-view. An internal gyroscope, an accelerometer and a magnetometer are polled at 1000 Hz for detecting head movements and rotations in all three dimensions of space. Connection to the host system is realized with a regular HDMI connector and USB ports.

The current status of the Rift is the Development Kit 2 (Fig. 1, a), which first introduced positional tracking by employing infrared LEDs. An additional camera, mounted in front of the user, films these blinking emitters enabling the detection of positional changes of the head and body at 60 Hz. Also, ghosting effects have been reduced due to a technique called "low persistence," setting the screen to black (pixels off) between two rendered frames.

Oculus announced their first consumer headset release for the first quarter of 2016. The display refresh rate will be boosted to 90 Hz and the total resolution for both eyes is aimed at 2160x1200 pixels. Oculus is using its already proven camera tracking system (constellation tracking) for additional input devices, which can be held in both hands. They will enable precise manipulation with buttons and gestures.

HTC Re Vive

Another important hardware manufacturer stepped recently into the area of head-mounted displays. HTC, known mainly from the smartphone market, is developing their Re Vive headset (short name "HTC Vive") in cooperation with a major games distribution company. The Vive uses a different technique for tracking the users' position on room scale. Where the Oculus Consumer Version 1 (CV1) will use the visual constellation tracking system with optical LEDs (outside-in tracking), HTCs approach is inside-out tracking. The user has to fix two laser emitters in each corner of the room, which will move an arc of laser light in a sweeping pattern. The headset can then track the exact position in the room via multiple, spatial distributed photo diodes, based on the time of signal received at each diode. This enables the user to move freely in the room and to be tracked all the time in the VE, instead of having a seated-only experience. However, as no currently available wireless display technology has the bandwidth or latency to present VR to the user, one is still limited by the cables of the headset. Specialized wearable computers already exist to counter this caveat, but are not considered here, because they are specialized builds and / or expensive to build and maintain.

The Vive also supports tool tracking, sporting the same technology as the headset tracking, and enables the user to precisely grab and manipulate objects in the environment. The joysticks can also be visually presented in the VE, so that the user is able to see the orientation and position of them, as he is using the system. The room scale tracking and the precise sensors enable a highly interactive and immersive environment, but the locomotion problem itself is still present. If the user wants to move beyond his current room without breaking immersion, this is only possible by playing tricks on his mind, e.g. simulating a blink of his / her eyes and a relocation while having a blacked out screen.

Leap Motion

The Leap Motion controller (Leap Motion, 2015) is a hand-input device, which uses two cameras and infrared LEDs to capture stereoscopic images of the user's hands. With calibrated camera positions and proprietary software algorithms it is able to calculate the finger, hand and wrist positions. It is attachable to the Rift via a separately sold VR mount (Fig. 1, b).

Tracking performance is dependent on lighting and environment. False positive detections of hands are possible when the distance is comparable to other nearby objects, such as monitors or keyboards. With optimal tracking conditions it is possible to fluidly convert detected joint positions to a model in virtual space.

Natural tracking devices, such as the Leap Motion controller, are the most intuitive way to interact with a virtual environment, but often lack the needed precision for pinpoint movements. As Oculus and HTC are releasing their new consumer headsets with specialized tool tracking solutions, natural interaction seems more and more obsolete. But for entry-level users usability is much higher with hand movements and gestures, than with specialized tools and buttons. Also, the user is free to utilize his hands, as he may. The whole physical boundaries of his fingers and hands are interacting with the VE and allow for more freedom than only using a few buttons. However, immersion may break more easily when tracking fails.

Additional hardware inputs

Using the Leap Motion, only hand (and tool) tracking is possible. Tracking the users legs may be beneficial for further immersion if he is looking down in the VE. Also, error correction for misalignment of the body tracking points and the virtual world may be countered by using multiple input sensors.

For obtaining whole body positions, we've been working on integrating Microsoft's new Kinect 2.0 (Microsoft, 2015) sensor. This also allows for spatial tracking inside a medium range and lets the users move around. As the Rift is not wireless, HDMI- and USB-extenders are needed to allow for freedom of movement. Larger location changes are only supported by controller inputs (e.g., by an analogue stick of a game controller) or by gesture detections (e.g., flat palms up and fingers extended = move in this direction) in this approach.

Research and development using only Inertial Measurement Unit (IMU) sensors for relative input is also done. One promising product to feature these capabilities in conjunction with low latency is "Control VR" (Control VR 2015), which lately went out of pre-order phase. Tracking sensors on the hand, down to the finger level, the lower and upper arm and chest measure movements and rotation. The system uses a calibration gesture to reset all stored positions to zero, by holding the arms straight down to the sides of the body. Relative changes to the individual sensors' position can then be measured and translated as VR input. The sensor drift in this application should be held reasonably low, as errors in position detection multiply and prediction of current posture gets worse over time. Using an absolute tracking system, such as the Kinect / Vive Lighthouse / Oculus Constellation, could aid in minimizing sensor drift.

As an alternative to giving the user movement ability in a limited real world space, recent "treadmill" systems for consumers have been presented. They work by fixing the user on a stationary point and reducing friction on the feet. One widely known product using this technique is the Virtuix Omni (Virtuix, 2015). Special shoes limit sideways slipping, but enable forward sliding movements, while sensors track the movements and convert them to regular game controller signals. This allows for a high compatibility with most 3D-engines and platforms.

Software engine

While the Oculus Rift runtime can be attached to many graphics engine implementations, we decided to use an existing game engine at our disposal. Game engines, used to power computer games, are targeted at the highest performance (measured in frames per second) while providing the most possible realistic user experience, in contrast to scientific visualization. Rift support is enabled in the Unity game engine (Unity Technologies, 2015) and the Unreal Engine 4 (Epic Games, 2015) through a plugin-based architecture. Unity is more mature, but the Unreal Engine allows for more detailed graphics and is completely open-source for subscribers. It also features a complete network stack for communication between servers and clients, usually intended for multiplayer gaming. This enables collaboration between different collocated or remote users.

The UE4 comes included with an editor for creating and managing scenes (levels), materials, characters and animations, to name a few. The editor and engine are modifiable by plugins, written in C++, so that a developer is able to extend the functionality or graphical user interface, if needed. Support for hot-plugging code changes is also included. As long as the code changes are not too complex for the running environment, Dynamic-Link Libraries (DLL) files will be compiled from the integrated development environment (e.g. Visual Studio) and reloaded at editor runtime.

Programming in UE4 can also be done via Blueprints, which are connectable event driven structured logical building blocks (cf. Fig. 2). Every Blueprint is a logical sequence of commands, which may read and write variables. Subfunctions may be defined by the user in Blueprints and will be callable as a custom node. Multiple start events are defined inside of UE4 for beginning the sequence, but those can be extended by custom implementations. This enables modification of the application behaviour to be carried out even by non-C++ programmers. Blueprints support context sensitive assists, suggesting several alternatives when drawing a connection from one component to another. Entities inside a scene

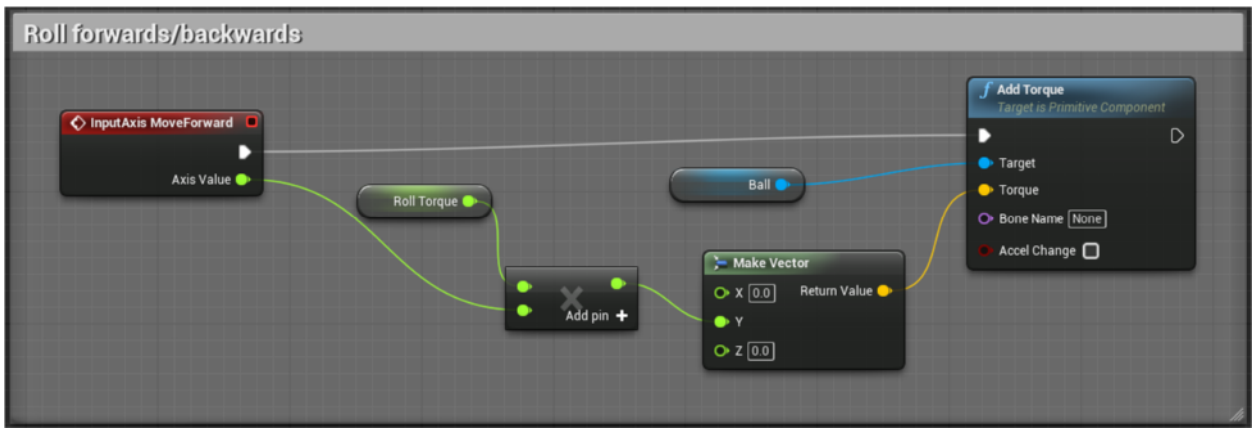

Fig. 2 Example of blueprint logic modelling in UE4

can be referenced inside the Blueprint graph to change their attributes and they generate events that a programmer can subscribe to. Even custom C++ code is accessible from the Blueprints, which enables a developer to rapidly create custom classes for his application and then extend the functionality and logic by using Blueprints. This also gives the advantage of not having to recompile the code and does enable visual debugging in real-time. One possible example application for using Blueprints is to map different user inputs from the keyboard or mouse events to actions inside UE4. Upon having the input event called, custom Blueprint code can be called, e.g. calculating a ray hit-detection from the users' point of view and getting the first UE4 actor that is underneath the cursor / crosshair for further manipulation (grabbing, moving, deletion). Figure 2 shows the game logic to add a torque impulse to a ball the player controls as an example. First the input axis value is multiplied with a variable "Roll torque" and fed into a "Make Vector" function (green lines). Then the resulting vector is applied to the "Add Torque" function, so that the result is added to the movement of the ball in the

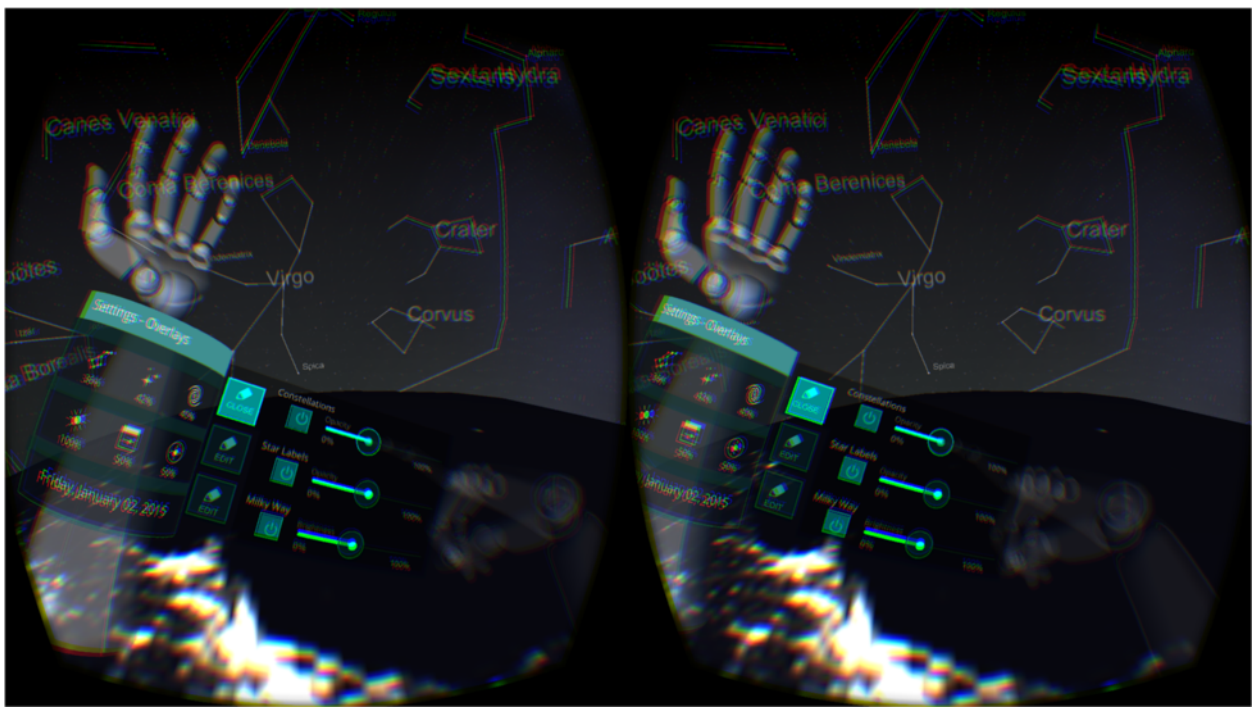

Fig. 3 Example of a menu inside a VE

map. The white line shows the sequential order of execution, where the "InputAxis MoveForward" event triggers the "Add Torque" function.

As the normal input methods are not in view to the user when wearing the Rift, menus should be integrated into the VE. Unlike traditional menu placement, which is overlaying the 3D view, menus in VE need to be placed in the rendered scene itself. A recent release from Leap Motion covers this possibility by attaching a menu on the user's virtual arm.

Figure 3 shows the implementation inside a planetarium demo application. Users are able to change different settings of the software while keeping them immersed. Rotation of the wrist changes the displayed submenus and interaction with the second hand changes the displayed settings. This is done by hit testing the individual fingers with the plane on which the menu resides. UE4 already has interaction handlers for menus, which can be used in conjunction with a 3D-menu. This allows for context based manipulation, such as selecting an element and changing its ID, name, material or colour in the scene. It is also possible to display the user's current position inside a model or to use this to teleport him to different places inside. Storing special positions and viewports for later presentation is also a feasible application, as cumbersome navigation through large-scale models is simplified by having such a "positional bookmark".

The presented widget in Leap Motion (2014) can be reproduced in multiple software engines, using the input provided by the Leap Motion controller. Alternatively, mapping controls to objects on surfaces inside the VE is also possible. This should be done when triggered actions result in modification of the VE's objects, mimicking a real world control circuit, such as a lift control. UE4 allows also for very realistic rendering, supporting up-to-date graphics.

Software architecture

Due to UE4's flexible plugin architecture, it is possible to connect it to various different systems. When working in teams, having a central server which holds all the BIM data is beneficial. Therefore, we've decided to connect to the OpenSource BIMServer (Beetz et al., 2010). Figure 4 shows all the important components of the system.

First, an engineer or architect creates BIM models via the regular design and iteration process. The resulting file can be imported to the BIMServer, which will then initially parse the geometry and store it alongside the BIM model data. The project manager, or a dedicated engineer, may manage different revisions in the development process of a building and update the data

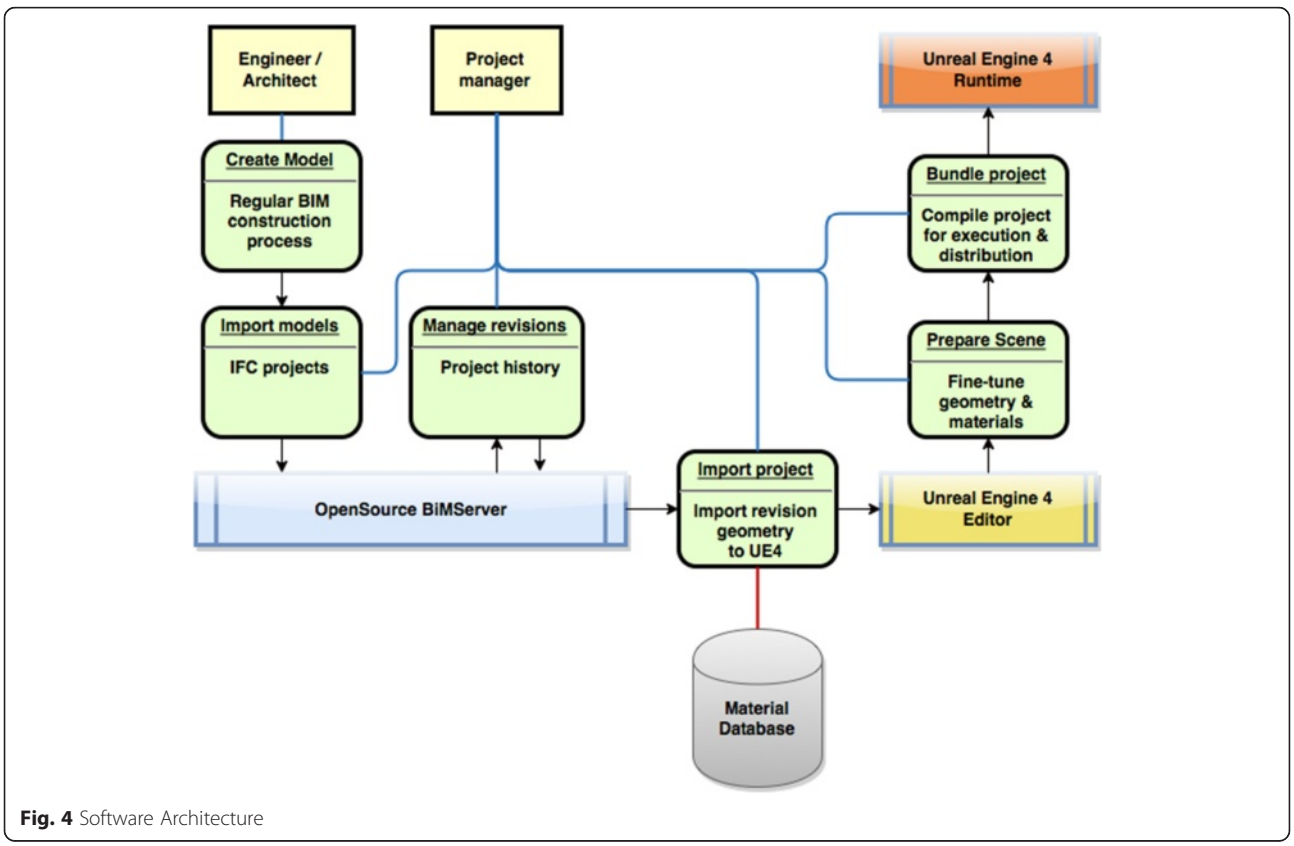

Fig. 4 Software Architecture

accordingly. The project may then be imported into the UE4 editor for further modification. The project can be bundled and distributed to different machines for testing purposes, as soon as the preparation is finished. Live previews of the resulting VE are also possible inside the UE4 editor to check for any major inconsistencies or problems.

As UE4 doesn't know about different BIM materials, e.g. concrete for walls or glass for windows, additional mapping is needed on the development side. UE4's standard starter pack materials are sufficient for fast prototyping, but further materials may be needed for realistic display of a building. Therefore, we propose to use a material database for mapping such information. The person in charge for the preparation of the scene may assign different materials anytime and even define new ones during the process.

The part of the UE4 code interfacing with the OpenSource BIMServer is exchangeable with different Application Programming Interface (API) calls to other software, so that even other server solutions, e.g. such as described in Das et al. (2015), are accessible, as long as they use HTTP calls. Even custom protocols are possible to implement, as the plugin structure for UE4 is dynamic enough to reference external libraries, if needed.

Regarding network connectivity between multiple clients, there are two different ways of implementing a multi-user VE. One possibility is to create the level beforehand, assign materials and optionally build scenarios. This allows for most creative flexibility, as all values can be fine-tuned before stepping in the VE. The major downside to this approach is that dynamic updating is somewhat limited. If the BIM data changes, the preparation process needs to be done again and also distributed to every client, as the actual model geometry is retained inside a UE4 level. An alternative way of distributing the data would be to fully automate the process of assigning materials and preparing the scene via configuration files and a common material database, which will work for the project. Clients can then dynamically parse the information given by the central BIMServer and every update since distribution of the client software will be honoured. A customizable avatar will represent other participants in the networked VE, so that social interaction regarding point-of-view is possible and virtual inspections are more realistic. Communication between clients can be realized via chatting by keyboard inputs or a dedicated voice server, which should be more immersive than using the keyboard. There are even technologies available for incorporating positional audio, so that the voice of the other participant seems to emerge from the direction his avatar is located at (Mumble, 2015).

If a problem is detected during the virtual inspection, users should be able to store this information. Therefore, we propose and plan to implement support for the BIM collaboration format (BCF / buildingSMART, 2015). Users can annotate specific elements of the building and add notes, either directly in the VE, or save a screenshot for later on and write a description of the problem later on their keyboard. This information should be recoverable for later user, so that users can jump to the specific viewport of the problem on a later revision and check for changes in the model. Support for BCF is included in BIMServer by the *Bimsie1BcfInterface* and is accessible via the JSON-API (JavaScript Object Notation). There is also a BCF Forum available as plugin for a Wordpress installation (opensourceBIM, 2015), which can load BCF information from a connected BIMServer and visualize the model alongside the separate issues, enabling even users without access to the VE to work on problems.

As of now, locomotion is an essential problem in the simulation of virtual worlds and we hope this will be solved in the near future. Even professional game companies are solely trying out different concepts at the moment. Therefore, we still resort to using a game controller / gamepad or keyboard and mouse for the locomotion. This should be easily adoptable for any user, which is familiar with first-person games and / or gaming consoles.

Application for architecture

UE4 uses global illumination (GI) for calculating static lighting in a level with an engine called Lightmass. Calculation results will be stored into a special texture channel during compilation time, taking load from the real time rending and resulting in very detailed light calculations for a scene.

Figure 5 shows a published engine example by Dereau (2015), which demonstrates photorealistic rendering by Lightmass usable for architectural design. We were able to run this demo fluidly on an NVidia GeForce GTX980 with no noticeable lags at all. In the past, such quality was only possible with pre-rendered videos or stills. If a client wanted to view the scene differently, another offline rendering run would have been needed to produce additional video files. Using UE4 to navigate the scene allows interactive viewing and presentation of architectural designs.

Running a scene like this with the Oculus Rift is possible, but it needs to be considered that the amount of render targets doubles in a worst case scenario. Each eye has to be rendered from it's own point of view, giving a moderate performance hit on slower machines. Networking support in the engine is also helpful again, if multiple users want to explore the space at the same time.

Table 1 shows the initial costs of each individual component. The costs for obtaining the starting package (Rift, Leap Motion and mount) are approximately around 420€ (without considering the computer system) as of August

Fig. 5 "Paris" Demo by Benoît Dereau

2015. Additionally, access to the UE4 source code is free of charge as of recent. If the Unreal Engine is used with this licensing model, then 5 % of gross revenue per product per year has to be paid to the creators of the UE4. However, this only applies to product sales after $3000 is exceeded per product per year. This price range is even affordable for hobbyists and home usage.

Implementation

For our development process, the Oculus Rift and Leap Motion are used as a base system to build upon to. UE4 is used as the central point for simulating VEs. Using the BIMServer as a central connection between clients, it is possible to load building data from multiple endpoints. Metadata about certain objects will be available to the user, such as the element ID of an IFC element. Additional information can be polled via the JSON-API. Extending the editor interface itself is easy, as source code examples are readily available. The BIMServer internally uses the ifcopenshell project (ifcopenshell, 2015) for parsing of IFC files.

Table 1 Costs of components as of August 2015 (w/o shipping)

Name	Category	Price
Rift DK2	HMD	350.00 $
Leap Motion	Hand detection	89.00 €
Leap Mount for Rift	Accessory	14.99 €
Microsoft Kinect v2	Body / hand tracking	(sensor + adapter) 199.98 €
Control VR	Body / hand tracking	(two arm package) 600.00 €
Virtuix Omni	Treadmill System	699.00 $
Unreal Engine 4	Game Engine	Free to use, royalties may apply

To query the BIMServer, we have created a plugin to connect to the JSON-API and download geometry information. Figure 6 shows the process as UML sequence diagram and the interaction with the API endpoints. First, the user has to login with his credentials of the BIMServer to get a session token. The plugin is then processing all projects that are readable to the user and stores their latest revision number. It is also possible to select different revisions of a model when querying. After the selection, we are finding the unique id of the BinaryGeometrySerializer plugin, as they vary between installations of the BIMServer. With the token, the revision number and serializer id, it is possible to download a binary representation of the project's geometry.

It is necessary for parsing the download to know more about the binary format, which was acquired by reading the BIMServer source code on serializing. Figure 7 shows a representation of the binary structure. The file starts with a padding of two bytes, followed by the String "BGS", which is short for BinaryGeometrySerializer. The next six float values describe the bounding box of the whole IFC project. After that, an entity count is given, which will be used to further iterate over the file. Everything in the red border of Fig. 7 is a separate IFC entity. Therefore, we need to evaluate the bordered file structure "Entity Count" times. Each IFC entity begins with the IFC class descriptor (e.g. "IfcWindow"), the entity ID (unique to the BIMServer) and the geometry type. Latter is 0 for triangle data and 1 for instances, which have no further geometry to parse. We then need to set the file pointer to the next offset being a multiple of four, because the server structures the data differently for the normal JSON-API and an additional WebSocket

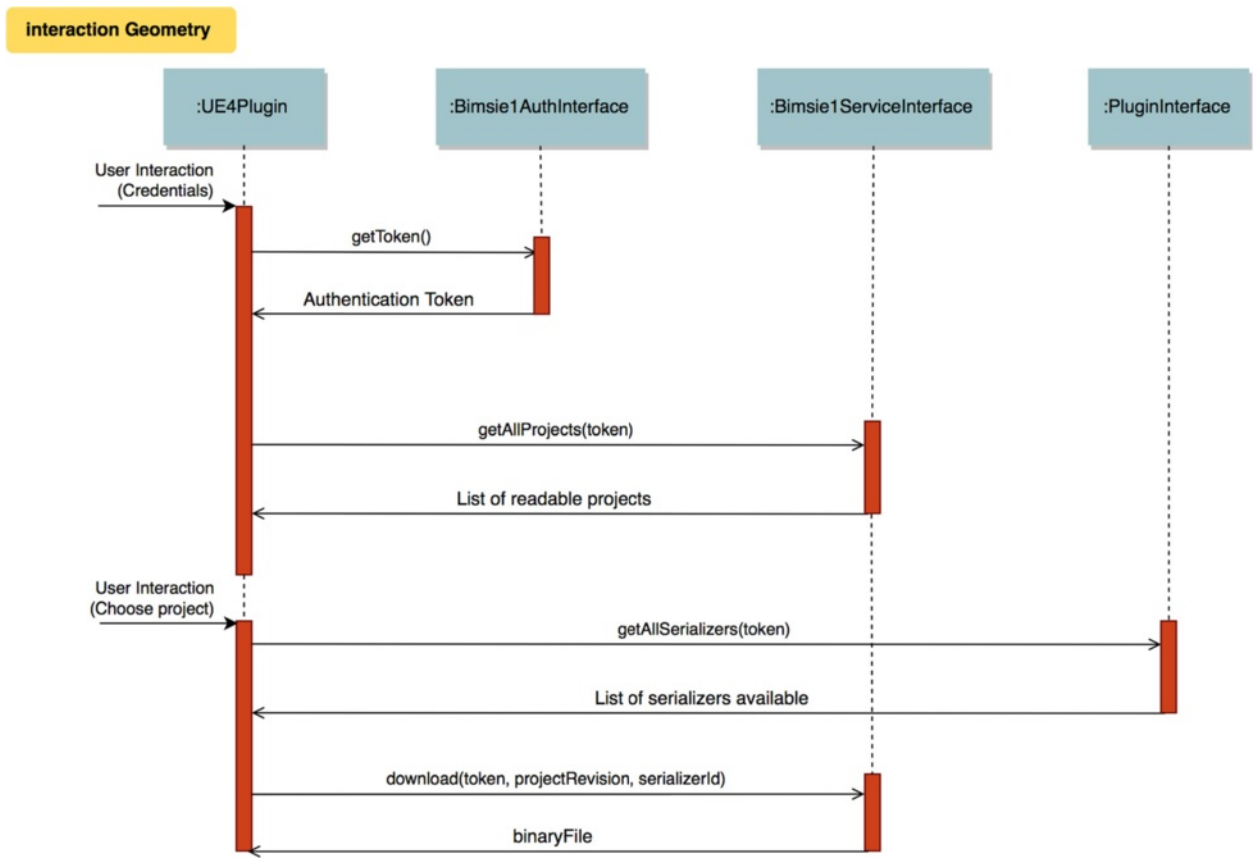

Fig. 6 UML diagram of the connection procedure

implementation. The transformation of the object in 3D space is done by using a 4x4 transformation matrix, with the upper left 3x3 submatrix defining the rotation. The following geometry data ID is referencing the unique id for the geometry on the server. The subsequent six float values define the bounding box of the IFC entity's geometry. Each triangular geometry is then defined by indices, vertices, normals and colors. A group of three indices define a triangle with the vertices' coordinates. Additional normals and colors help with visualization, although we are calculating the normals using UE4's calculation methods.

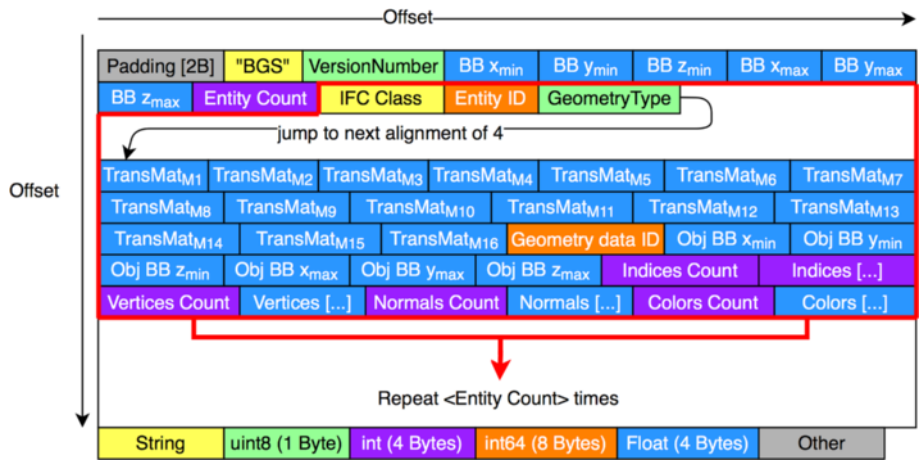

Fig. 7 Binary geometry serializer format (order left to right, then top to bottom)

After parsing the data structure contained inside the binary file, we are able to create actors in UE4 to display the geometry. We are using a class "IFCActor" that inherits from UE4's own AActor class and extends with additional attributes and functions. Usually, UE4 uses static meshes for presentation, but there is an option of having procedural geometry (i.e. created at runtime) with the "ProceduralMeshComponent" class. As no reference to static meshes can be stored, which is usually contained in UE4's "uasset" file format, the triangular data is retained inside the map we are importing into. This leads to larger map sizes, but simplifies asset management, as everything is stored at a single location.

To map the UE4 materials to the created actors, a rule based association is needed. We created a database, consisting of the project reference ID on the server, and the material to be assigned for specific IFC classes (Fig. 8). Windows, for example, may be set to a glass material, while walls will have a different one. The "mapping_set" table has a 1-to-n relation with the "mapping" table, defining the IFC entity class and the corresponding UE4 material. UE4 materials can be accessed using their path in the project directory and their unique name. If nothing is defined for the current project id, a template may be used as basic information. Templates may be composed of different mapping sets, which define parts of the whole set of IFC classes available. Furthermore, by using a template for a project and adding an additional mapping set, the user is able to leverage default settings and overriding them by specific settings. We are looking forward on continuing to simplify this workflow and make it easier for the user to define at runtime.

Using the Oculus Rift with UE4 is simple due to the engine having a plugin available. This will be loaded automatically, as soon as the user has a Rift connected. Visual corrections for chromatic abbreviation and distortion caused by the lenses and input mapping to a virtual camera works out of the box.

For communication with the Leap Motion, the internal plugin can be used or an external one can be downloaded. We found the event driven Leap Motion plugin (getnamo, 2015) to be simple to setup and reliable to use. Convenience content, such as already rigged character models, is available and is tuned to usage with the Rift. We had to modify the physical collision shapes in order to enable full ten-finger support for interactions. Unreal usually uses only a capsule shaped collision model on characters to check for interactions with enabled actors in the environment. Therefore, each finger needs a tuned capsule shape for the physics system (Fig. 9). UE4's socket system supports to attach objects to user-defined positions of the animated skeleton of an actor. After adding sockets to each finger, it is possible to attach a collision shape to them. Only rotational, scale and position offsets need to be fine tuned. Afterwards, the shapes' movement will be synchronized to the socket's parent bone, which is the individual finger we want to have collision be checked for. Specific collision channels allow to modify the interaction behaviour with the environment (static environmental objects, dynamic moving objects, other actors, etc.).

Grabbing objects in 3D space can't be achieved with regular collision modelling, as the object would bounce back and forth between fingers. Gestures or special finger combinations can be used to attach objects to certain slots

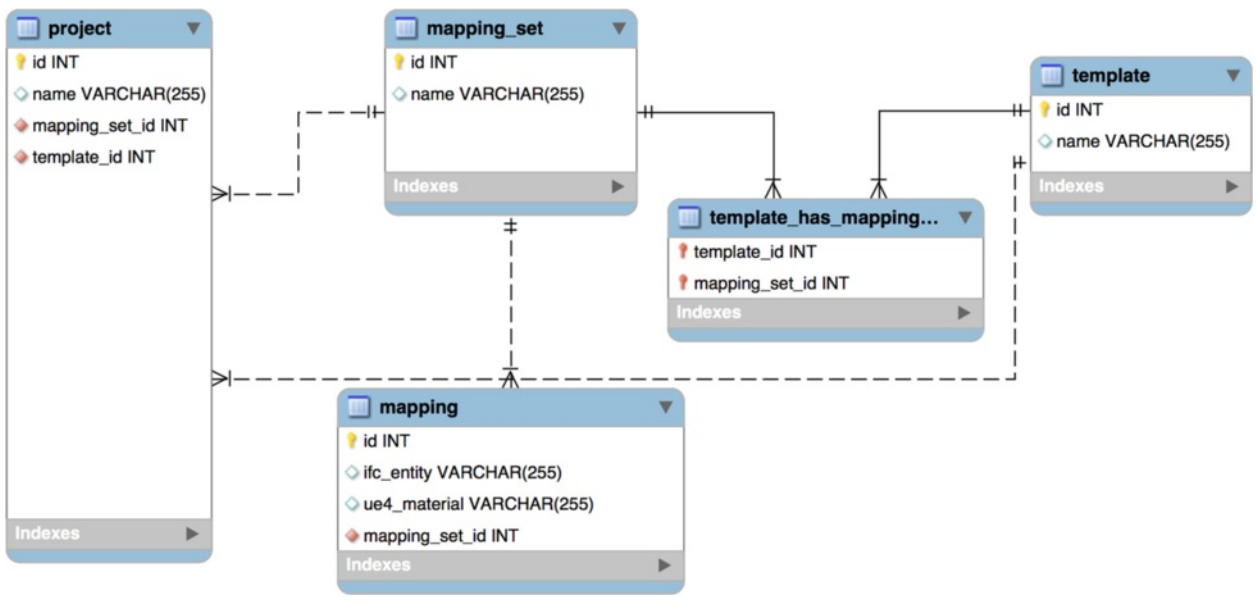

Fig. 8 Schema of material database

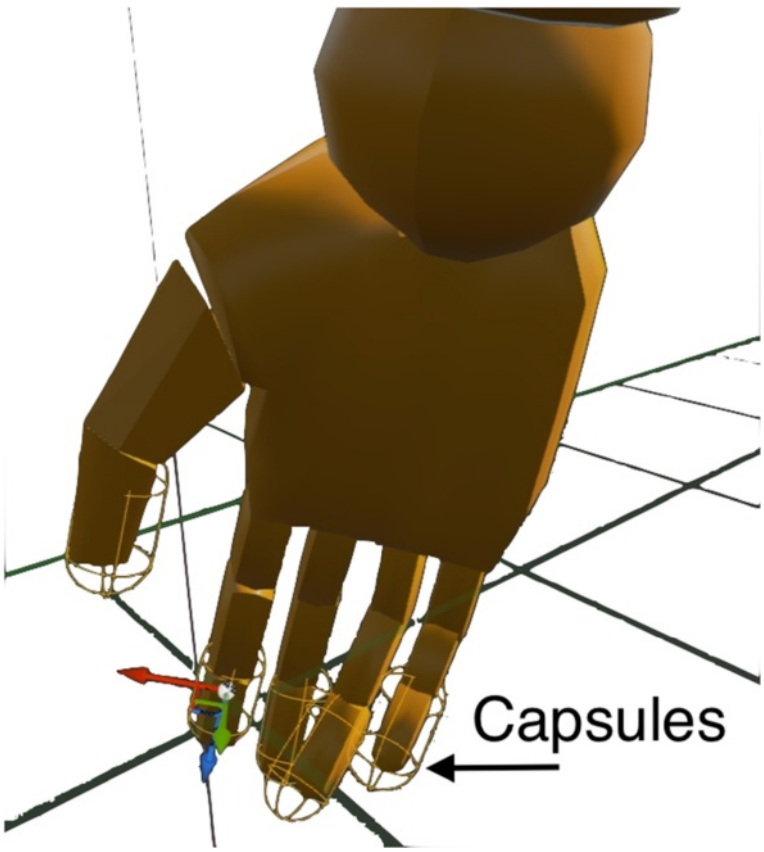

Fig. 9 Capsule collision shapes

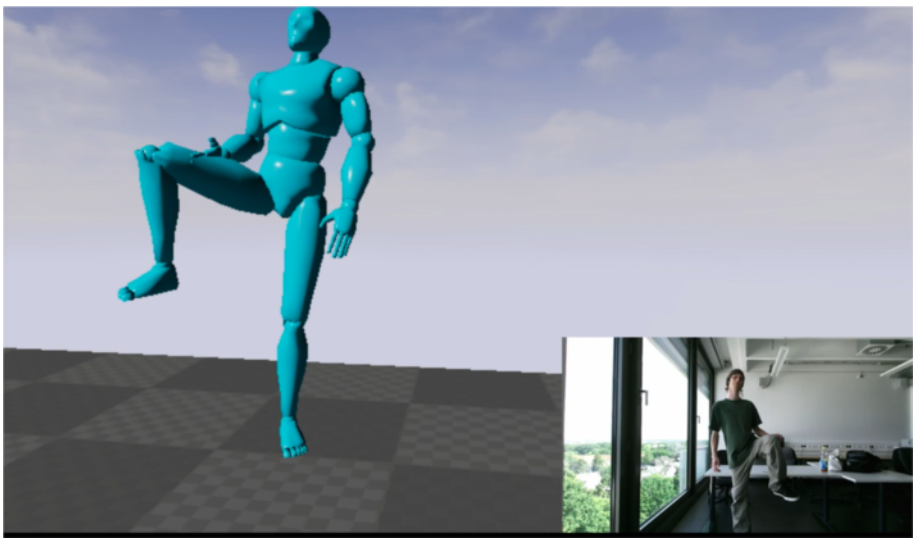

Fig. 10 Sensor fusion in testing environment

on the user's character. A Leap Motion gesture could be moving certain parts of the hand in a circle, swiping in a direction or pressing a finger forward / downward. Gesture based input can be modelled with Blueprints only, giving non-programmers a possibility to extend the logic.

As UE4 is one of the major game engines available, support for alternative HMD devices, especially the HTC Vive, is possible by using the SteamVR-API (supported from UE4 version 4.8 onwards), which aims to be a standard for interfacing with virtual reality hardware. Tool based input, such as the tracked joysticks of the HTC Vive, is beneficial when trying to grab objects, as pressing a dedicated button on the handle of the stick is easier and more reliable to detect than a grabbing gesture.

Sensor fusion

We are currently working on implementing support for the Microsoft Kinect v2 sensor into UE4. While there are plugins available, our aim is to fuse different sensor inputs together, to improve detection on hand and body postures. While the Leap Motion is good at detecting hand and finger movements, it does not detect additional body parts. Therefore we are aiming towards detecting rough body posture (legs, upper body, arms) through the Kinect system and fine tune the hand position when the Leap Motion detects one or two hands. Figure 10 shows the initial implementation inside a UE4 testing environment.

The Kinect SDK already has support for detecting skeletons, but needs some smoothing on the software side to prevent jerking of the joints on a frame-to-frame basis. Unfortunately, support for multiple Kinects on one computer to prevent detection dead zones is not possible due to runtime and hardware limitations, as the USB 3.0 bus rate would be exceeded. Using multiple computers may circumvent this, but combining the calculated output on the software side, however, will require complex calculations and calibrated sensor positions.

Scenarios

In the following segment we present some possible applications of virtual reality in conjunction with the Unreal Engine and the aforementioned VR hardware. While some of the scenarios may not be completely novel, the usage in a low cost VE with HMDs is well worth a consideration.

Evacuation testing

Behaviour in case of an emergency differs from person to person. While legal requirements to provide emergency escape routes in construction are always fulfilled, they may be not necessarily the optimal solution for a safe escape. With immersive VR environments it is possible to test multiple escape scenarios safely and realistically with many types of users. The usage of first person view using an HMD and integrated sound effects enable a more realistic impression to the user than otherwise possible. Test operators can monitor the subjective perception of escape route signs and set up the scene accordingly for another run. The users' direction of view can be recorded and saved for later playback and an in depth analysis. Figure 11 shows an example of escape sign placements and visibility to the user in case of a fire outbreak. The fire objects can be placed throughout the scene and even be spawned by the engine's logic using Blueprints.

Event scripting, such as the user triggering another fire sequence upon reaching a certain position within the environment, is also possible. This technique can also be used to measure the time needed to exit the building

Fig. 11 Immersive testing of escape route visibility in UE4

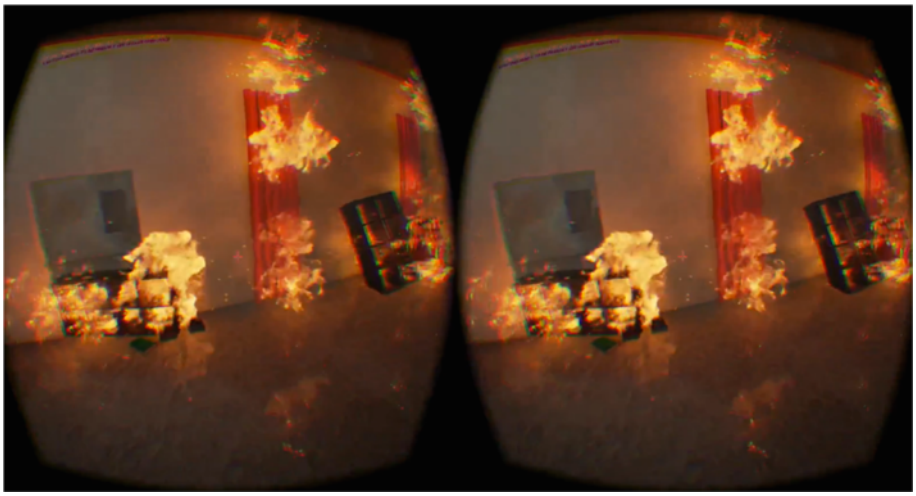

Fig. 12 Realistic lighting view of fire effects in UE4 with a HMD

safely. When setting the character's movement speed to that of a running or fast walking person, the measured time should be similar to that of a real world scenario.

The lighting calculations of the UE4 in combination with sound effects lead to a very realistic scene (cf. Fig. 12). Ideally, this testing environment can be implemented using existing IFC models and the BIMServer. Additionally, the UE4 interfaces natively with the FBX file format. Therefore, using Autodesk Revit, which is a software solution for creating and modifying BIM models, as a direct source of geometry is also possible (cf. Rüppel and Schatz, 2011). Additions to the scene are always needed in this case, as escape signs and fire outbreak locations are not automatically generated, depending on furniture and materials.

We are looking forward on streamlining the process of generating an escape scenario and automatic measuring of the escape route times, especially with participants who have no knowledge of the building layout. For testing, it should be viable to let participants experience the building from a visitor's perspective. They have entered an unknown building and walked to a certain point, but haven't been able to fully grasp all of the emergency exit routes.

Expert training

Training to control special and / or heavy machinery is a key qualification in multiple professions. However, at the beginning of such training, accidents may happen due to major mistakes. Costs of using real machinery are also not negligible. While hands-on experience is the most important part of education, certain parts can be accelerated if the user knows the control scheme beforehand.

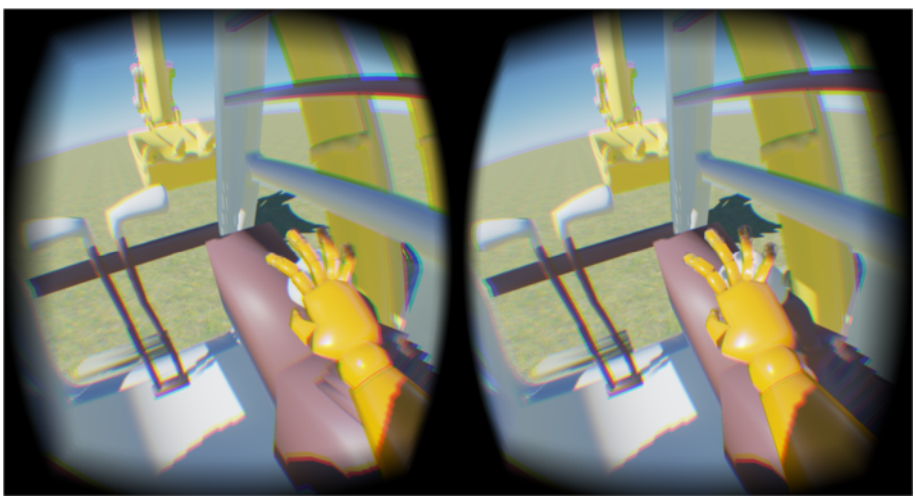

Fig. 13 Example first person view of hand interactions with Oculus Rift and Leap Motion

Figure 13 shows the stereoscopic image of the user's view with the Oculus Rift and hand detection using the Leap Motion.

Collisions of hands and controls inside the cockpit can be detected and translated to scripted events for controlling movement of the machinery. The user gets a first person impression of environmental visibility around him. This scenario requires more scripting work when applied inside the UE4, as control input and resulting actions have to be modelled. However, reusing of Blueprint components is possible, resulting in a more rapid development enabling further test cases.

Additional use cases include safety considerations when operating cranes at a construction site. The planned construction site layout can be used to model the environment. A user would take place in the cockpit of a crane, giving him the impression of limited visibility and movement range of the actual machine. Attaching objects to prepared slots on the crane supports the lifting and dropping of objects. Even the basic simulation of waypoint navigating non-player characters (NPCs) as workers on the site is possible. If a load hits any modelled object in the level, physics simulation would lead to a swinging motion. NPCs can be equipped with a defined number of hit points (health), which will drop if exposed to certain forces. A message to the user would then be presented to notify of the error and restart the scenario, again a great improvement over reality.

Accessibility validation

Accessibility planning for sidewalks or buildings needs to be accurate and mistakes at this stage may lead to expensive adjustments later on. First person testing, for example of wheel chair accessibility, is possible with the proposed VR environment.

The user can be placed into a wheel chair model and will see the designed level around him from a handicapped perspective (Fig. 14).

Using the additional hand detection and resulting arm placement enables a check for unreachable controls, such as light switches or fire alarm buttons. The physics engine prevents entering rooms that are too small to drive into.

The UE4's physics engine allows for the definition of wheeled character blueprints. While these are usually intended for simulating cars, they can also be adapted to the wheelchair. Each tire of the vehicle supports friction settings and the total centre of mass will be taken into consideration by the physics engine. Acceleration and deceleration parameters are also supported.

Feasibility / Discussion

For determining the feasibility of the approach, we have tested the performance of our imported geometry with assigned materials and the Oculus Rift DK2. The test was executed with a NVidia GeForce GTX 980, an Intel Xeon W3565 and using Windows 8.1 as operating system. The imported model consists of 4,731 IFC entities with a total of 264,131 triangles. As can be seen in Fig. 15, the overall performance is centred well around the targeted 75 frames per second (FPS). Some lag spikes exist, which occurred while doing rapid 90° to 180° turns while inside the building. The authors assume this may be due to the occlusion culling algorithm testing which faces of the triangles are visible to the user.

Drawing all of the 264,131 triangles at once was not possible while maintaining a steady 75 FPS to the user, but is also not necessary. As the user will be most of the time inside the building, which is the key application of our proposal, the integrated algorithms will only draw visible (not occluded) surfaces in the view frustum of the

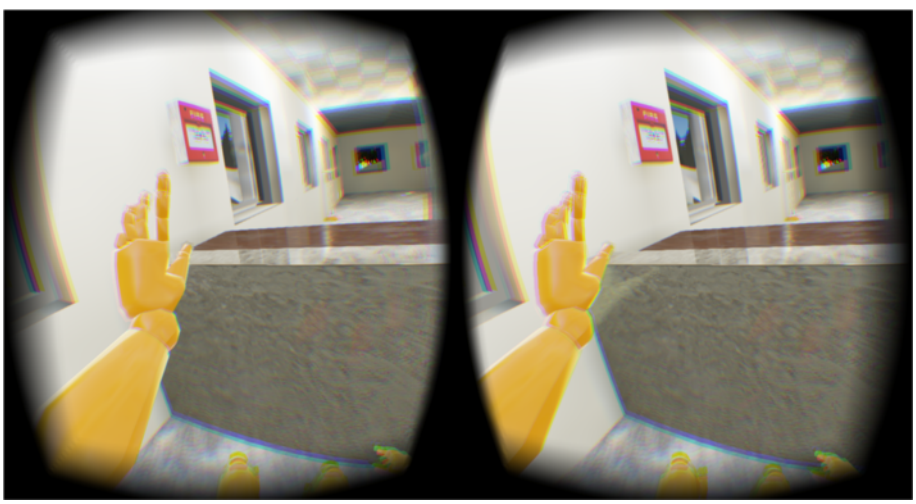

Fig. 14 Simulated wheelchair inside a building

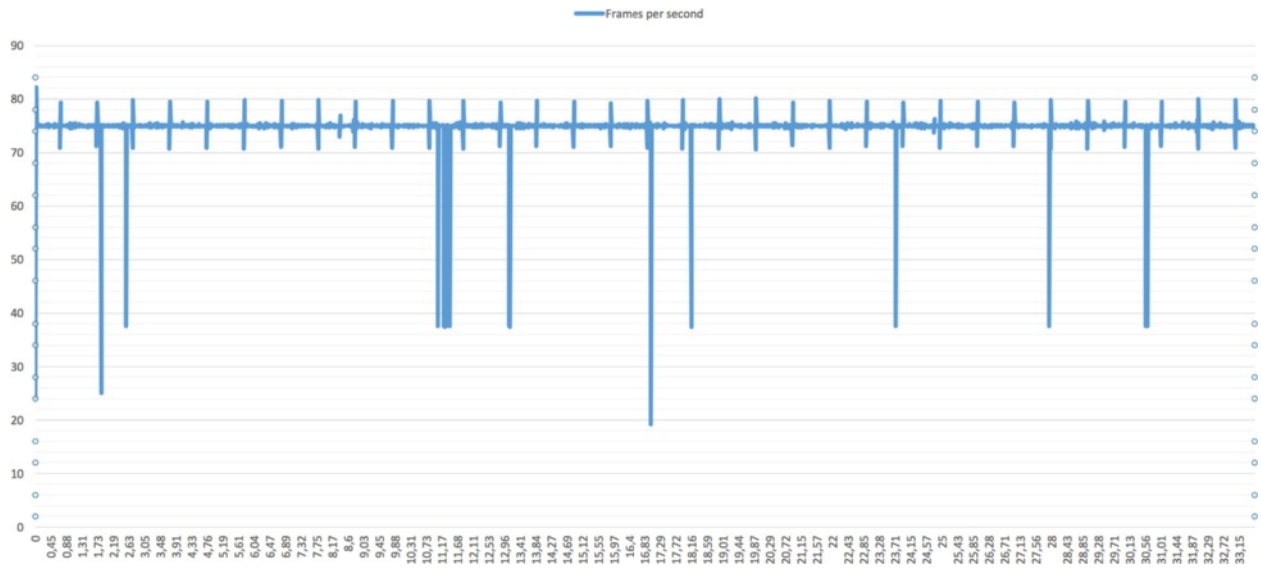

Fig. 15 Frames per second for example IFC model

camera. Therefore, the amount of triangles to be processed is drastically reduced and the performance needed for displaying an immersive interior is easily fulfilled. Figure 16 shows the general approach of the occlusion culling process (Hudson et al., 1997). The camera is creating a view frustum that extends from the viewpoint into the 3D scene. Everything inside this volume has to be potentially rendered by the UE4, but as some objects occlude other entities farther away (in view direction) an intelligent approach has to be chosen on testing which parts are visible. Therefore, after finding all occluders in the vicinity of the camera, shadow volumes can be cast from them. Everything inside this volume is culled and can be skipped when rendering the scene, using the hardware's z-buffer.

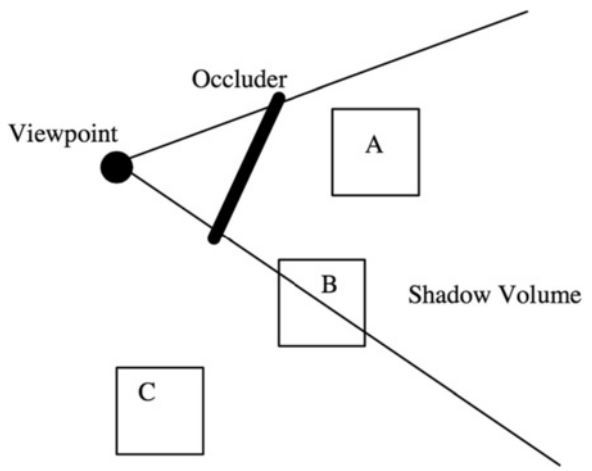

Fig. 16 Concept of shadow volume occlusion culling, source: Hudson et al. (1997)

Additional countermeasures may include another step in the pre-processing of IFC geometry by hashing the vertices coordinates and faces to store already existing or similar geometry in a map structure. Figure 17 shows the process of reusing the already processed geometry. First the vertices coordinates' and face order is concatenated into a string using delimiters. Then this string is fed into a one-way hashing function to get a resulting hash value, which can be used as a lookup key for a map structure. If the key already exists in the map, we can assume this geometry has already been processed. Therefore, the IFC entity can be cloned in UE4 for optimizing the draw calls needed on the graphics-processing unit's end. Only rotation and translation have to be applied, to reflect the different position and orientation in the 3D space. If the key does not exist in the hash map, we have to create new geometry from the vertices and faces and then store a pointer to this geometry for later usage. Then the iteration processes the next IFC entity.

With the new devices (Oculus Rift CV1 / HTV Vive) the requirement rises to 90 FPS for more immersive experiences, which should be achievable with our proposed approach. Especially as newer graphics cards in combination with Windows 10 have support for DirectX 12, which is reducing draw call overheads with an optimized graphics pipeline.

Special implementations, such as special control schemes and interactions with the 3D world still need initial development effort, as no "out of the box" solutions exist.

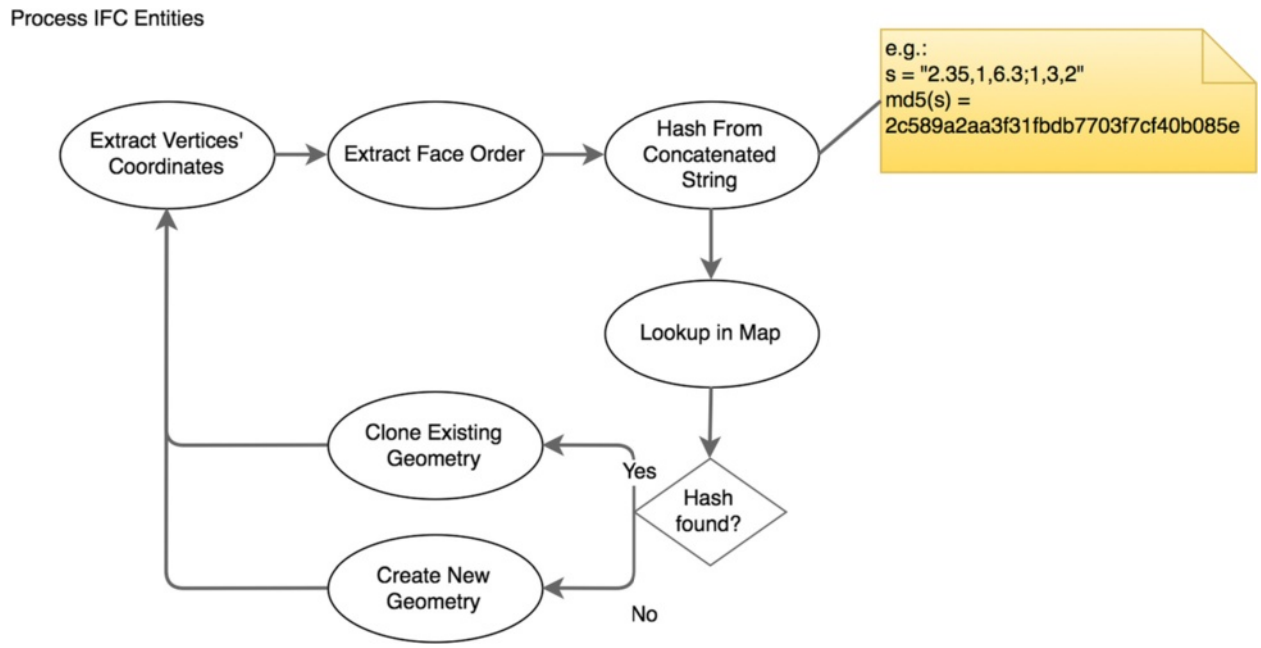

Fig. 17 Processing IFC entities with using a hashed map structure for efficient geometry usage

Complex and high level architectural scenes, such as seen in Fig. 5, still require a capable 3D artist for content creation or a developer experienced with the game engine. As immersion inside a scene does not solely depend on the realism of graphics, professional developers without deep understanding of modelling can also do fast creation of environments. Several free 3D model sites are available on the Internet, which can be used as a source of models for a scene. Full body joint detection can be implemented using Microsoft's Kinect or similar hardware. Users are then able to move every part of their body and see the results in the VE.

Hand detection using the Leap Motion is good, but tends to give false positives when not having free space in front of the user. Also, misdetection of the left and right hand are possible. This is an issue that is currently being worked on by the manufacturer and developers. Combining the Leap data with the Kinect system seems promising for now.

Regarding the fire example, we propose to further elaborate the scripting qualities of the UE4. Depending on material properties, a fire spread rate could be calculated, minimizing the set up time for the test operators and giving a more realistic environment.

As of now, several developers are porting their software to SteamVR and the HTC Vive, where the development kit preorder phase ended recently. Setting up the lighthouse trackers for the Vive and clearing a room of obstacles may be still a thing for enthusiasts, but tracking the hands with tools seems more reliable, as the optic detection of the hands is still not consumer-ready. In reasonable time, Sony will step into the market with their own solution (Morpheus headset) and this will be a big step forward for the whole VR community, as more customers are aware of the possibilities tied to VR.

Conclusion / Outlook

HMDs today are getting more useful for a wide range of applications in construction and engineering, while costing less than in the past. Modern game engines, such as the UE4, enable even non-programmers to generate logic procedures and levels for presentation. VEs enable the users to experience complex models or control schemes instead of having to comprehend a complex explanation or offline rendered 2D/3D images.

We have shown the feasibility of automating major parts in the VR creation process from BIM as a starting point and the simplification by using the proposed methodology and software, as no parallel designing process for visualization is needed. BIMServer as a central server for storing BIM data is beneficial when using the proposed workflow and plugin for UE4. Geometry does not have to be recreated and can be imported. We plan to further extend the functionality and simplify the workflow in the future to create an easy importer for VR experiences based on real building data.

Different use cases for utilizing a VE are evacuation plan testing, expert training and accessibility validation of environments. The supplied scenarios are only a small selection of what can be done using a VE for construction and engineering. Especially when using intuitive

and natural control schemes, it may be easier for users to interact with the virtual surroundings.

Also, we are looking forward to future HMDs that may include eye-tracking mechanisms. Rendering from two different point-of-views is computationally expensive, even more, when considering increasing resolutions for HMD displays in the coming years. Eye tracking can limit this impact, by only calculating high-resolution parts of the VE where the user is looking at and giving more approximate representations at the peripheral vision.

Having the graphics card manufacturers change their architectures and drivers to be optimized for VR seems to be an indicator that this technology is here to stay and will be further improved in the future.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
Both authors contributed extensively to the work presented in this paper. Hilfert reviewed and analyzed the literature, developed the concept and use cases, and drafted the manuscript. König supervised the entire processes of this study. All authors read and approved the final manuscript.

Received: 31 August 2015 Accepted: 28 December 2015
Published online: 07 January 2016

References

Beetz, J, van Berlo, L, de Laat, R, van den Helm, P (2010). BIMserver. org–An open source IFC model server. *In Proceedings of the CIP W78 conference*.
buildingSMART (2015). BIM collaboration format – BCF intro. http://www.buildingsmart-tech.org/specifications/bcf-releases, Accessed: 20.08.2015
Control VR (2015). Control VR. https://www.kickstarter.com/projects/controlvr/control-vr-motion-capture-for-vranimation-and-mor, Accessed: 30.01.2015
Das, M., Cheng, J., & Kumar, S. S. (2015). Social BIMCloud: a distributed cloud-based BIM platform for object-based lifecycle information exchange. *Visualization in Engineering, 3*, 8.
Dereau, B (2015). Unreal Paris Virtual Tour. http://www.benoitdereau.com, Accessed: 30.01.2015
Edwards, G., Li, H., & Wang, B. (2015). BIM based collaborative and interactive design process using computer game engine for general end-users. *Visualization in Engineering, 3*, 4.
Epic Games (2015). Unreal Engine 4. https://www.unrealengine.com, Accessed: 30.01.2015
getnamo (2015). leap-ue4 – event driven Leap Motion plugin for Unreal Engine 4. Online: https://github.com/getnamo/leap-ue4, Accessed: 30.01.2015
Grabowski, A., & Jankowski, J. (2015). Virtual Reality-based pilot training for underground coal miners. *Safety Science, 72*, 310–314.
Hudson, T., Manocha, D., Cohen, J., Lin, M., Hoff, K., & Zhang, H. (1997). Accelerated occlusion culling using shadow frusta. In *Proceedings of the thirteenth annual symposium on Computational geometry* (pp. 1–10).
ifcopenshell (2015). open source ifc geometry engine. http://ifcopenshell.org, Accessed: 30.01.2015
Leap Motion (2014). Arm HUD VR (Alpha). https://developer.leapmotion.com/gallery/arm-hud-vr-alpha, Accessed: 30.01.2015
Leap Motion (2015), Leap Motion. https://www.leapmotion.com, Accessed: 30.01.2015
Merchant, Z., Goetz, E. T., Cifuentes, L., Keeney-Kennicutt, W., & Davis, T. J. (2014). Effectiveness of virtual reality-based instruction on students' learning outcomes in K-12 and higher education: A meta-analysis. *Computers & Education, 70*, 29–40.
Microsoft (2015), Kinect SDK v2. http://www.microsoft.com/en-us/kinectforwindows/develop, Accessed: 30.01.2015
Mumble (2015). Mumble Wiki – Positional Audio. http://wiki.mumble.info/wiki/Positional-Audio, Accessed: 20.08.2015
Oculus (2015). Oculus Rift. https://www.oculus.com, Accessed: 30.01.2015
opensourceBIM (2015). BCF-Forum. https://github.com/opensourceBIM/BCFForum/wiki/BCF-Forum, Accessed: 20.08.2015
Roupé, M., Bosch-Sijtsema, P., & Johansson, M. (2014). Interactive navigation interface for virtual reality using the human body. *Computers, Environment and Urban Systems, 43*, 42–50.
Rüppel, U., & Schatz, K. (2011). Designing a BIM-based serious game for fire safety evacuation simulations. *Advanced Engineering Informatics, 25*(4), 600–611.
Sampaio, A. Z., & Martins, O. P. (2014). The application of virtual reality technology in the construction of bridge: The cantilever and incremental launching methods. *Automation in Construction, 37*, 58–67.
Unity Technologies (2015), Unity Game-Engine. http://unity3d.com, Accessed: 30.01.2015
Virtuix (2015). Virtuix Omni. http://www.virtuix.com/products, Accessed: 30.01.2015