

THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

From BIM to VR

-The design and development of BIMXplorer

MIKAEL JOHANSSON



Department of Civil and Environmental Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

From BIM to VR

-The design and development of BIMXplorer

MIKAEL JOHANSSON

ISBN 978-91-7597-449-1

© MIKAEL JOHANSSON, 2016

Doktorsavhandlingar vid Chalmers tekniska högskola

Ny serie nr 4130

ISSN 0346-718X

Department of Civil and Environmental Engineering

Chalmers University of Technology

SE-412 96 Gothenburg

Sweden

Telephone: +46 (0)31-772 1000

Chalmers Reproservice

Gothenburg, Sweden 2016

From BIM to VR - The design and development of BIMXplorer

MIKAEL JOHANSSON

Department of Civil and Environmental Engineering

Chalmers University of Technology

Abstract

The architecture, engineering and construction (AEC) industries are currently undergoing a change from a drawing-based form of information exchange to a model-based. Using the concept of Building Information Models (BIM), the content produced by architects and designers has evolved from traditional 2D-drawings to object-oriented 3D-models embedded with information to describe any building in detail. This, in turn, has opened up new possibilities of using real-time visualization and Virtual Reality (VR) as a tool for communication and understanding during the design process. However, as primarily created to describe a complete building in detail, many 3D dataset extracted from BIMs are too large and complex in order to be directly used as real-time visualization models. Because of this, it is still difficult to integrate VR and real-time visualizations as a commonly used tool during the design process. The recent introduction of a new generation of Head-Mounted Displays (HMD) has also made the situation even more challenging. Although these new types of VR devices offer huge potential in terms of realism, sense of scale and overall suitability for design and decision-making tasks, they are also far more demanding when it comes to real-time rendering performance.

In order to address the current situation this thesis contributes with the design and evaluation of a new software application that provides a simple interface from BIM to VR. Following a design science research approach this application has been developed in order to fulfil a set of requirements that has been identified as important in order for VR and real-time visualization to become an everyday used tool for design and communication during the building design process. Along that path, three new technical solutions have been developed:

- An efficient cells- and portals culling system automatically realized from BIM-data.
- An efficient approach for integrating occlusion culling and hardware-accelerated geometry instancing.
- An efficient single-pass stereo rendering technique.

The final system – BIMXplorer – has been evaluated using several BIMs received from real-world projects. Regarding rendering performance, navigation interface and the ability to support fast design iterations, it has been shown to have all the needed properties in order to function well in practice. To some extent this can also be considered formally validated, as the system is already in active use within both industry and education.

Key words: Building Information Modeling, BIM, Virtual Reality (VR), Real-time rendering, Head-mounted display (HMD).

Acknowledgements

First of all, I would like to thank the people that were with me at The Visualization Studio at Chalmers Lindholmen – the place where all of this work really started! Claes and Börje – I hope that you are enjoying your time in retirement! Special thanks go to Mattias Roupé, my friend and colleague with whom I have had very close collaboration during this entire project. We have had a lot of fun over the years and I especially value your way of always seeing the potential and opportunities instead of problems in any given situation.

I would also like to thank all my other colleagues at the Construction Management department for their support, especially Mikael Viklund Tallgren and my main supervisor, the amazing Petra Bosch. Huge thanks also to my examiner Christian Koch as well as my academic writing coach, Christine Räisänen.

I would also like to acknowledge my former examiner, the late Professor Per-Erik Josephson, who made it possible for me to become a PhD student in the first place.

Last, but certainly not least, I would like to thank my family and friends for their company and support, especially the three most important people in my life, Jessica, Noah and Esther – thank you for all the love and happiness you give me every day!

Göteborg, August 2016

Mikael Johansson

Appended papers

Paper I:

“Efficient Real-Time Rendering of Building Information Models”

Johansson, Mikael; Roupé, Mattias. In *Proceedings of the 2009 international conference on computer graphics and virtual reality (CGVR09)*, July 13-16, 2009, Las Vegas, Nevada, USA, Pages 97-103.

Paper II:

“Real-time visualization of Building Information Models (BIM)”

Johansson, Mikael; Roupé, Mattias; Bosch-Sijtsema, Petra. *Automation in Construction*, Vol. 54 (2015), June 2015, Pages 69-82.

Paper III:

“Integrating Occlusion Culling and Hardware Instancing for Efficient Real-Time Rendering of Building Information Models”

Johansson, Mikael. In *GRAPP 2013: Proceedings of the International Conference on Computer Graphics Theory and Applications*, Barcelona, Spain, 21-24 February, 2013, Pages 197-206.

Paper IV:

“From BIM to VR – Integrating immersive visualizations in the current design process”

Johansson, Mikael; Roupé, Mattias; Viklund Tallgren, Mikael. In *Fusion - Proceedings of the 32nd eCAADe Conference - Volume 2 (eCAADe 2014)*, 10-12 September, 2014, Newcastle upon Tyne, England, Pages 261-269.

Paper V:

“Efficient Stereoscopic Rendering of Building Information Models (BIM)”

Johansson, Mikael. *Journal of Computer Graphics Techniques (JCGT)*, Vol. 5, No. 3, 2016, Pages 1-17.

Additional publications

“Immersive visualisation of building information models: Usage and future possibilities during design and construction”

Roupé, Mattias; Johansson, Mikael; Viklund Tallgren, Mikael; Jörnebrant, Fredrik; Tomsa, Petru Andrei. In *Proceedings of the 21st International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA 2016)*, 30 March-2 April, 2016, Melbourne, Australia, Pages 673-682.

“A BIM-supported framework for enhancing joint planning in construction”

Viklund Tallgren, Mikael; Roupé, Mattias; Johansson, Mikael; Andersson, Roger. In *Proceedings of the 32nd CIB W78 Conference 2015*, October 27th-29th, 2015, Eindhoven, The Netherlands, Pages 696-705.

“An empowered collaborative planning method in a Swedish construction company - A case study”

Viklund Tallgren, Mikael; Roupé, Mattias; Johansson, Mikael. In *Proceedings 31st Annual ARCOM Conference*, Lincoln, 2015, Pages 793-802.

“Interactive navigation interface for Virtual Reality using the human body”

Roupé, Mattias; Bosch-Sijtsema, Petra; Johansson, Mikael. *Computers, Environment and Urban Systems*, Vol. 43, 2015, Pages 42-50.

“Real-Time Rendering of large Building Information Models - Current state vs. state-of-the-art”

Johansson, Mikael; Roupé, Mattias. In *Proceedings of the 17th International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA 2012)*, 25-28 April, 2012, Chennai, India, Pages 647-656.

“Using the human body as an interactive interface for navigation in VR models”

Roupé, Mattias; Johansson, Mikael. In *Proceedings of the 17th International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA 2012)*, 25-28 April, 2012, Chennai, India, Pages 79-88.

“3D-City Modeling: A Semi-automatic Framework for Integrating Different Terrain Models”

Roupé, Mattias; Johansson, Mikael. *Advances in Visual Computing, 7th International Symposium, ISVC 2011*, September 26-28, 2011, Las Vegas, NV, USA, Pages 725-734.

“Towards a Framework for Efficient Use of Virtual Reality in Urban Planning and Building Design”

Johansson, Mikael. Thesis for licentiate degree, Chalmers University of Technology, Gothenburg, Sweden, 2010.

“How can GIS and BIM be integrated?”

Johansson, Mikael; Roupé, Mattias. Poster, *The 15th International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA 2010)*, 7-10 April, 2010, Hong Kong.

“Supporting 3D City Modelling, Collaboration and Maintenance through an Open-Source Revision Control System”

Roupé, Mattias; Johansson, Mikael. In *Proceedings of the 15th International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA 2010)*, 7-10 April, 2010, Hong Kong, Pages 347-356.

“Visual quality of the ground in 3D models: using color-coded images to blend aerial photos with tiled detail-textures”

Roupé, Mattias; Johansson, Mikael. In *Proceedings of the 6th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa (Afrigraph 2009)*, February 4-6, 2009, Pretoria, South Africa, Pages 73-79.

“Virtual Reality Supporting Environmental Planning Processes: A Case Study of the City Library in Gothenburg”

Suneson, Kaj; Allwood, Carl Martin; Heldal, Ilona; Paulin, Dan; Roupé, Mattias; Johansson, Mikael; Westerdahl, Börje. *Knowledge-Based Intelligent Information and Engineering Systems, 12th International Conference, KES 2008*, Zagreb, Croatia, September 3-5, 2008, Pages 481-490.

“Virtual Reality As a New Tool in the City Planning Process”

Suneson, Kaj; Allwood, Carl Martin; Paulin, Dan; Heldal, Ilona; Roupé, Mattias; Johansson, Mikael; Westerdahl, Börje. *Tsinghua Science and Technology*, Vol. 13, No. S1, 2008, Pages 255-260.

“User’ evaluations of a virtual reality architectural model compared with the experience of the completed building”

Westerdahl, Börje; Suneson, Kaj; Wernemyr, Claes; Roupé, Mattias; Johansson, Mikael; Allwood, Carl Martin. *Automation in Construction*, Vol. 15, Iss. 2, 2006, Pages 150-165.

“Building Information Modelling for Visualisation in AEC Education”

Horne, Margaret; Roupé, Mattias; Johansson, Mikael. *The 5th conference of Construction Applications of Virtual Reality (CONVR 2005)*, 12-13 September, 2005, Durham, United Kingdom.

“From CAD to VR - Implementations for Urban Planning and Building Design”

Johansson, Mikael; Roupé, Mattias. *Digital Design: The Quest for New Paradigms (23rd eCAADe Conference Proceedings)*, 21-24 September, 2005, Lisbon, Portugal, Pages 399-405.

“From CAD to VR – focusing on urban planning and building design”

Roupé, Mattias; Johansson, Mikael. *AVR III, Conference and Workshop on Applied Virtual Reality and Open Source VR programming*, Gothenburg, Sweden, May 27-28, 2004.

Table of contents

1	Introduction.....	1
1.1	Aim and objectives.....	2
2	Background and related work.....	5
2.1	Definition of Virtual Reality (VR).....	5
2.2	Real-time rendering.....	5
2.3	Display systems.....	6
2.4	BIM and real-time visualization.....	8
2.5	Frame rate and interactivity.....	11
2.6	Rendering performance and acceleration techniques.....	12
2.6.1	Pipeline optimizations.....	14
2.6.2	Level-of-detail (LOD).....	14
2.6.3	Visibility culling.....	15
3	Research approach.....	19
3.1	Requirements.....	23
4	Summary of the papers.....	25
4.1	Paper I.....	25
4.2	Paper II.....	25
4.3	Paper III.....	26
4.4	Paper IV.....	27
4.5	Paper V.....	27
5	BIMXplorer v1.0.....	29
6	Discussion.....	33
6.1	Current state of BIM visualization.....	33
6.2	Efficient real-time rendering of BIMs.....	35
6.3	Integration of VR within the AEC field.....	37
7	Conclusions and future work.....	39
	References.....	41

1 Introduction

Real-time visualization and Virtual Reality (VR), have many applications within the Architecture, Engineering and Construction (AEC) industries (Bouchlaghem et al., 2005; Woksepp and Olofsson; Greenwood et al., 2008). With the ability to navigate freely through 3D scenes from a first-person perspective, it is possible to present and communicate ideas regarding future projects in a way that facilitates understanding among all involved parties, despite their background or professional expertise (Kjems, 2005; Westerdahl et al., 2006). For people with limited experience of interpreting traditional design documents, such as 2D drawings, the technology offers a representation that avoids misunderstanding and allows for a thorough apprehension of any type of building or facility (Mobach, 2008).

However, despite all the documented benefits that VR technology offers, it is still not used as an everyday tool during the design process. Instead, its use remains restricted to certain projects of high importance (Bullinger et al., 2010; Liu et al., 2014). In the past, this was mainly due to lack of affordable hardware offering sufficient computing power, but also the fact that the actual design was almost always performed in 2D. As a consequence, any use of VR required the time-consuming creation of a separate 3D model (Westerdahl et al., 2006; Sunesson et al. 2008), using the original design documents as a reference. As this was typically done by someone else than the architect, it severely affected a natural integration of the technology. Even when 3D CAD data was available from the actual design, it still had to be converted to a representation suitable for real-time visualization by optimizing it and adding material properties, such as textures.

Nevertheless, with the introduction of Building Information Models (BIM) within the AEC field new possibilities have emerged. Using modern modelling tools, such as Autodesk Revit or ArchiCad, the content produced by architects and designers has evolved from traditional 2D-plans and elevations to object-oriented 3D-models embedded with information to describe any building or facility in detail (Eastman et al., 2011). In theory, BIM then supports an easier and more natural integration of VR during the design process. As 3D data is available from the actual design work, there is no longer a need to create a separate 3D-model for the sole purpose of visualization.

However, as primarily created to describe a complete building in detail, many 3D dataset extracted from BIMs are too large and complex in order to be directly used as real-time visualization models (Dvorak et al., 2005; Jongeling et al., 2007; Pelosi, 2010; Steel et al., 2012; Dalton and Parfitt, 2013; Shi et al., 2015). To support interactive, real-time navigation the dataset often has to be significantly reduced or otherwise optimized – a process that may involve hours or even days to perform (Dubler et al., 2010; Liu et al., 2014). With current solutions users and stakeholder thus have to either resort to time-consuming pre-processing steps or accept that the visualization may not always be considered fully interactive or free of visual artefacts, i.e. errors. Because of this, it is still difficult to integrate VR as a commonly used tool for design and communication. Although visualizing large amounts of 3D-data in

real-time is an active research topic by itself (Yoon et al., 2008), there has been surprisingly little attention given to the specific case of visualizing large BIMs in real-time.

With the recent introduction of a new type of consumer-directed Head Mounted Displays (HMD), such as the Oculus Rift, the problem of managing large BIMs interactively has become even more relevant to solve. Although these new types of VR devices offer huge potential in terms of realism, sense of scale and overall suitability for design and decision-making tasks, they are also far more demanding when it comes to real-time rendering performance. Not only do they require a 3D scene to be rendered twice in order to produce the stereoscopic effect, but they also require a much higher update rate. Compared to typical desktop VR applications, the performance requirements have essentially increased three-folded, making an existing problem become far worse. Fortunately, this thesis has only one real purpose – to deliver a solution to this problem.

1.1 Aim and objectives

The main aim of the work presented in this thesis is to develop a software application that will allow VR to become an everyday used tool for design and communication during the building design process. As already outlined, one of the biggest obstacle for this to be realized today lies in the difficulties to directly – i.e. without any time-consuming pre-process – visualize BIMs in real-time. The main objectives are therefore to develop techniques and algorithms that allow large and complex BIMs to be directly visualized in real-time.

To better guide this process, the following research questions are investigated and considered:

RQ1: What is the current state when considering real-time visualization of BIMs?

The problem of visualizing BIMs has been highlighted in earlier studies, but it hasn't really been thoroughly investigated. The current literature contains many examples where problems of using large BIMs for visualization purposes have been expressed but often the exact details, such as what type of models, software and hardware that has been used are simply omitted. Similarly, this question also relates to the type of models that can be expected in real-world cases.

RQ2: What is a suitable acceleration technique for typical 3D building models?

There are a number of existing techniques and algorithms that can be utilized in order to accelerate real-time rendering. These have all strengths and weaknesses and a suitable choice is highly dependent on the type of 3D environment that it should be applied to. For instance, a vast, open landscape seen in a flight simulator is very different from a detailed city environment seen from the ground level when it comes to performance optimization. When considering buildings in general – which BIMs typically represent – they often feature a lot of occlusion, i.e. enclosed regions. Can we take advantage of this in an efficient manner?

RQ3: How can we take advantage of BIM-data in order to accelerate rendering?

When considering existing acceleration techniques they have primarily been developed with general 3D-models in mind, i.e. as received from CAD or DCC tools. As such, they become inherently limited by the lack of information beyond pure geometrical data. BIMs, on the other hand, contain much more information, i.e. metadata, such as detailed object properties as well as any relations to other objects. Can this additional information be utilized in order to accelerate real-time rendering?

RQ4: How can we support a natural integration of VR within the building design process?

In order for VR to become a natural and integrated part of the building design process, several barriers need to be overcome. In this context the technical ability to visualize large and complex BIMs in real-time only represents one of them. How these techniques are actually implemented with regards to accessibility, as well as usability and interface becomes equally important to consider. For instance, if time-consuming processes – although automated – are required to realize a VR session this will probably affect an integration negatively. Similarly, if the actual navigation interface is found too complex for non-experts, it will be difficult to support end-user participation.

2 Background and related work

2.1 Definition of Virtual Reality (VR)

Ever since Sutherland (1965) first articulated the term Virtual Reality (VR) it has been defined and used in many different ways. Ranging from simple environments presented on a desktop computer to fully immersive environments experienced through head-mounted displays and tracker systems, the term now means different things in various contexts. Within the scope of this thesis, VR is defined as a computer-generated visualization of spatial data that can be interactively controlled by a user and displayed on any type of screen. Furthermore, the primary application that has been considered is that of *real-time architectural walkthroughs* where users can explore and navigate through interior and exterior spaces of a virtual building model (Mobach, 2008; Liu et al., 2014).

2.2 Real-time rendering

The field of 3D computer graphics includes several techniques that aim at producing 2D images – often called frames – of three-dimensional geometric data. This is realized using a process known as 3D rendering, where the input 3D-data is converted to pixels in a 2D image based on the location of a virtual camera (Figure 1). When the resulting 2D image is displayed onto a computer screen it basically represents a window into a three-dimensional world from a specific location. The 3D rendering process can either be done in real-time or performed offline, i.e. non-real-time. When non-real time rendering is used the purpose is to produce a single image – or multiple images that are composed into an animation sequence or film – of high quality that later can be displayed on a computer screen. Depending on the size and complexity of the input 3D-data, the processing time ranges from minutes to hours or even days, but the end result will in turn be able to realistically simulate lighting, shadows, reflections and other natural phenomena. The end result, however, is a static image or film sequence that once created cannot be changed by the user – it only represents a single view of the input 3D-data.

Real-time rendering, on the other hand, takes a different approach as the process is repeated continuously. During this process, the virtual camera can be moved freely, thus giving the user the impression that he or she is travelling around in a virtual world. The actual navigation can be controlled by an input device, such as a mouse or joystick, and gives the user the ability to interact with the system. However, to make this “virtual journey” smooth and interactive, the computer has to generate a sufficient number of frames per second to prevent the user from experiencing motion sickness (Hettinger, 1992). In order to be able to perform this computationally expensive process, dedicated graphics hardware, also known as GPU (Graphics Processing Unit), has to be used. Using a technique known as *rasterization*, the GPU handles all the computations required to produce 2D image views of the 3D-data (Akenine-Möller et al., 2008). Although powerful, the GPU still has limitations on the amount of data that can be processed within a given time frame. This basically means that there always exists an upper limit on the amount of 3D-data that can be interactively visualized.

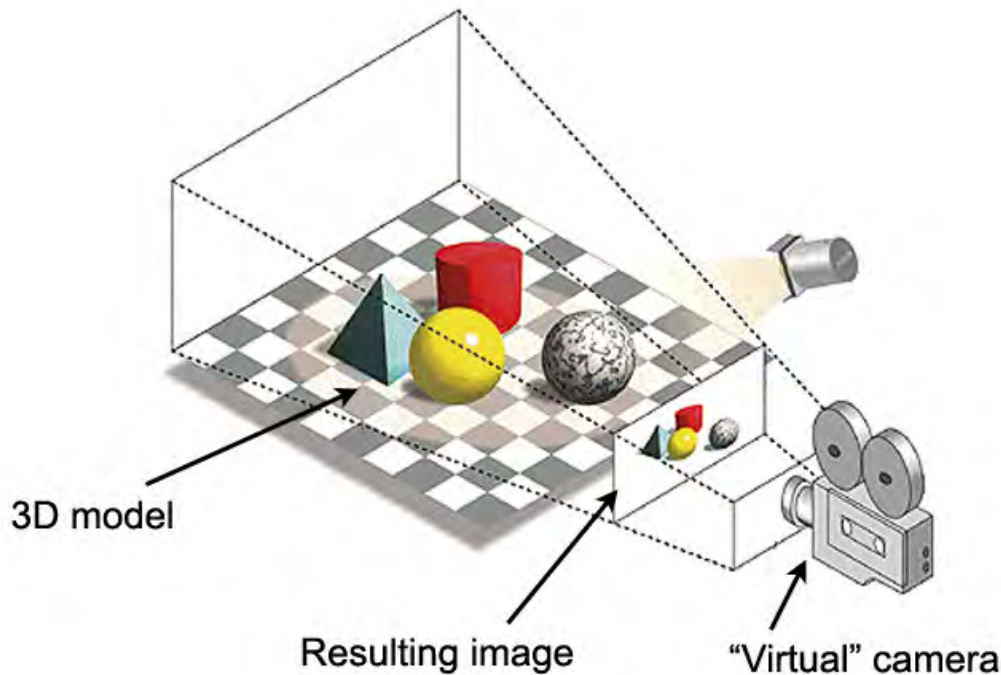


Figure 1: With 3D rendering, an input 3D model is converted into an image based on the position of a “virtual” camera.

2.3 Display systems

Although real-time rendering is the main technology for producing images of non-existing objects or environments, the result may be displayed in a wide variety of ways. Ranging from display on regular computer screens to solutions where a user is wearing a head-mounted display (HMD), the basic difference is the level of immersion they enable. Here, the level of immersion may be defined as the degree to which a user feels completely surrounded by the virtual world. When considering *Immersive VR*, the most used solutions today are either a CAVE (Cruz-Neira et al., 1992) or Head-Mounted Display (HMD) (Burdea and Coiffet, 2003). Examples of these are shown in Figure 2.

The HMD naturally supports stereoscopic vision in that it uses a small display for each eye: one for the left and one for the right. By rendering the virtual world from two slightly horizontally separated (virtual) cameras for each eye, the user experiences stereoscopic vision similar to that in real life (Kjellin, 2008). For the CAVE solution LCD shutter glasses are often used in order to provide the stereoscopic vision. These shutter glasses work by blocking one eye’s view of the screen (or wall). When an image is rendered for the left eye, the shutter glasses block the right eye view. For the right eye the process is then reversed. The switching process is synchronized with the graphics card and performed at a high frequency, thus giving the user a perceived true stereoscopic vision.

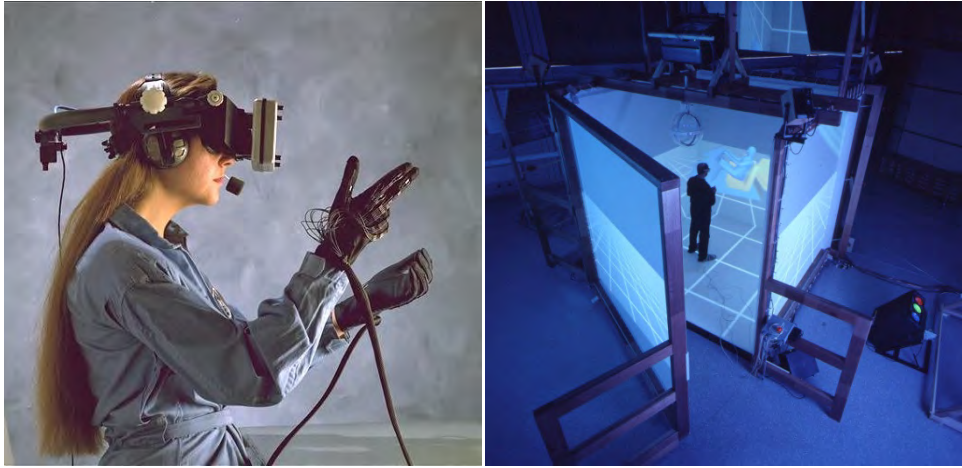


Figure 2: HMD (left) and CAVE (right) (Courtesy NASA and Chalmers)

A semi-immersive alternative to the CAVE is the *Powerwall* (Westerdahl et al., 2006). As shown in Figure 3, it is basically a small cinema screen allowing for many people to view the VR simulation simultaneously. By using stereographic shutter glasses or polarized glasses, stereoscopic vision is enabled.



Figure 3: A Powerwall solution.

Finally, *Desktop VR* (Modjeska, 2003) refers to the case when the real-time rendering is displayed on a regular computer screen or portable computer screen, without stereoscopic vision. This solution is the simplest in terms of required hardware, and by using a projector the system is also suitable for a larger audience. When using the “projector-assisted” approach

of Desktop VR, the solution thus basically becomes a small Powerwall without stereoscopic vision.

Until very recently, HMDs have been either low-cost-low-performance or high-cost-high-performance devices (Dörner et al., 2011), making them less useful in practice. However, with the introduction of a new generation of consumer-directed HMDs, such as the Oculus Rift and HTC Vive (Figure 4) this has completely changed. These devices provide a high resolution, large field-of-view as well as orientation and positional tracking ability at a very competitive price (around \$600-1000). Within the scope of this thesis, both the Desktop VR solution as well as the new generation of HMDs has been targeted.



Figure 4: Oculus Rift (left) and HTC Vive (right)

2.4 BIM and real-time visualization

A Building Information Model (BIM) may be defined as a digital representation of the physical and functional characteristics of a building. Compared to a general 3D-CAD model, a BIM is a different kind of representation since it defines not only geometrical data but also specifications and information regarding spatial relations and connections among the included components. The creation of a BIM is typically done in a modern modeling tool, such as *ArchiCAD* or *Autodesk Revit*. These systems represent each component in a building as an object with parametric properties and relations to other parts of the building. The collection of objects is not seen as a 2D-drawing or 3D-model, but is instead stored in a single database that represents the complete building. From this database it is then possible to derive different representations, such as a 2D drawing or 3D model (Figure 5). As all representations originate from the same data, any change in the database will automatically update all representations. This property makes the system especially efficient when considering revisions and updates. As a repository of information a BIM support a multitude of applications along the design and construction process, including cost-estimation, energy analysis and production planning (Eastman et al., 2011).

For the majority of BIM authoring tools the underlying data-model closely resembles that of the Industry Foundation Classes (buildingSMART, 2007). The IFC was designed to provide a universal basis for the information sharing over the whole building lifecycle (Eastman, 1999), and is the de facto standard for representing BIMs. It differs from general 3D-file formats, such as 3D Studio, FBX or COLLADA (Arnaud and Barnes, 2006), in that it represents a

building or facility with specific (virtual) building objects instead of pure geometrical entities. The IFC scheme supports a wide variety of buildings objects, such as IfcWall, IfcDoor, IfcWindow, IfcSlab and IfcRoof together with an unlimited set of properties connected to each object. Using the IfcRelation feature, any object can also relate to other objects, making it possible to form constraints and relations between building parts. For instance, a door “knows” that it is placed in a particular wall. Another major difference between IFC and general 3D-file formats is the representation of space. Every instance of an IFC-object must belong to a spatial context. Special space-enclosing structures are the sites (IfcSite), buildings (IfcBuilding), storeys (IfcBuildingStorey) and rooms (IfcSpace). Additionally, any window or door placed in a wall results in an opening element (IfcOpening) that represents the cut-out in the affected wall.

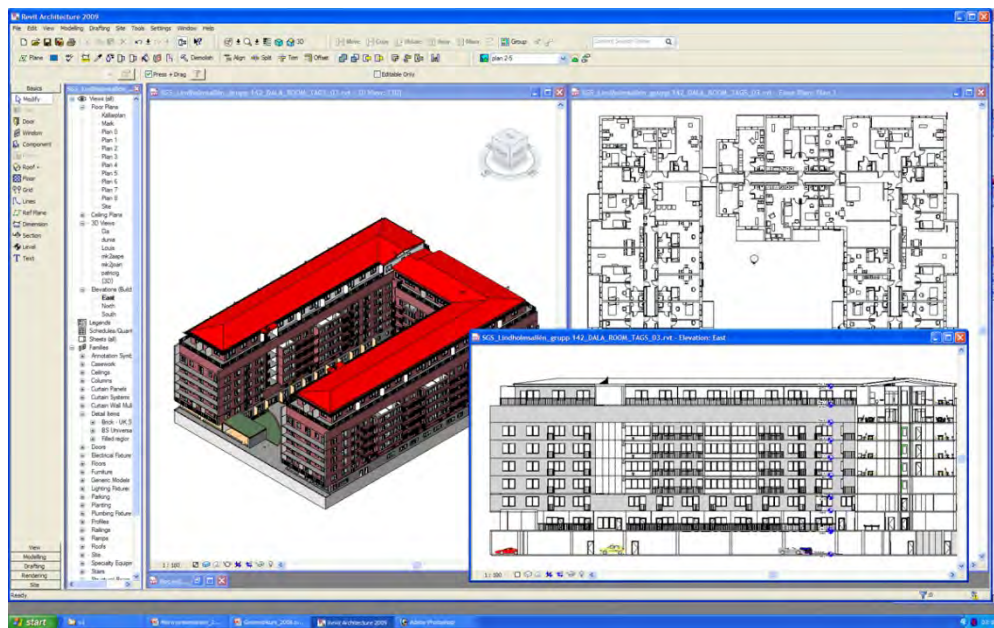


Figure 5: A BIM created in Autodesk Revit.

Although practically all BIM authoring applications follows the IFC-specification they still differ in many ways when considering the level of information they contain and the ability to extract that information. In that sense, it can be said that each application defines BIM in its own way. In order to support the whole range of possible authoring environments, the work presented in this thesis therefore defines a BIM according to the IFC-specification. That is, the acceleration techniques presented within the scope of this thesis have been designed to work with models defined according to the IFC-specification, and do not rely on any additional data or features beyond that.

The introduction of BIM within the AEC field is interesting, as it makes it possible to use a single source of data for 2D-drawings, offline renderings as well as real-time renderings and VR. In theory, this should make it much easier to integrate real-time visualizations as a design and communication tool during the actual design process. As 3D-models can be extracted directly from the BIM-systems, there is no longer a need for the additional creation of a

separate 3D-model for visualization purposes. However, in practice, this development has also introduced a new set of challenges. As primarily created to describe a complete building in detail, BIMs can be too large and complex in order to be directly used for real-time rendering (Dvorak et al., 2005; Svidt and Christiansson, 2008; Steel et al., 2012; Dalton and Parfitt, 2013). Although commonly used software tools for BIM visualization is able to directly load models regardless of size and complexity it is often difficult to achieve smooth frame rates without further processing of the input dataset or by introducing non-conservative acceleration techniques (Dubler et al., 2010). It is therefore common that a visualization session has to be preceded by an optimization step in order to make the dataset more suitable for use in a real-time environment. However, as this step has to be repeated as soon as the design changes it severely affects an efficient integration of real-time visualization as a communication tool (Dubler et al., 2010; Liu et al., 2014).

Recent times have also seen the use of so-called game engines for the purpose of real-time visualization of BIMs (Yan et al., 2011; Shi et al., 2015; Merschbrock et al., 2016). Given the image quality and immersion offered by modern computer games, game engines are often put forward as a better alternative to conventional BIM-viewers, such as Navisworks, as they offer high rendering performance and more elements of interactivity. However, although it's true that it is possible to use game engines to produce impressive visualizations, they still require a lot of manual work in order for this to be realized (Shi et al., 2015; Merschbrock et al., 2016). In Figure 6, a typical workflow from BIM to VR using game engines is illustrated (Halaby, 2015). As can be seen, there are several steps that needs to be performed, including model optimizations and other processes to provide real-time performance, e.g. occlusion bake. Even with a streamlined process as the one described in Figure 6, this can easily add up to several hours or even days, depending on the size and complexity of the BIM. As discussed previously, this is a process that needs to be repeated for every major change of the design.

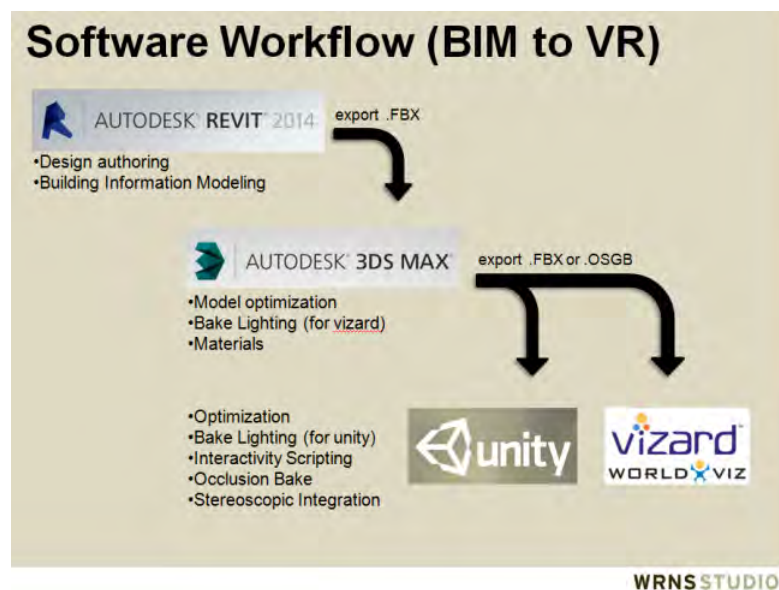


Figure 6: Typical software workflow from BIM to game engine (Halaby, 2015)

Looking forward, it could be argued that the ever increasing speed of CPUs and GPUs will solve this problem simply by brute-force performance. However, at the same time we also see that BIMs tend to become more detailed in terms of geometry and amount of objects as this field matures. In addition, new display hardware puts even higher performance demands as resolution (e.g. 4K screens) and frame rate requirements (e.g. Oculus Rift and HTC Vive) increase. As such, the problem of interactivity and real-time performance is still important to solve.

2.5 Frame rate and interactivity

An important property for any type of real-time visualization system is its ability to maintain a sufficiently high frame rate. As defined by the frequency at which new images are presented on screen it inherently affects user experience and task performance. A too low frame rate will make the system less responsive and make navigation and other interaction tasks more demanding at the same time as it greatly diminishes the sense of continuous motion. When considering a minimum frame rate, many studies have found that user performance becomes significantly reduced below 15 Hz for a number of different applications. Reddy (1997) investigated the effects of different frame rates on human performance when faced with a simple heading task in a virtual environment. During these tests low frame rate substantially degraded user performance and a frame rate of around 15 Hz was suggested to serve as a minimum requirement for a generally acceptable degree of performance. However, it was also suggested that a higher frame rate should be strived for, as this was shown to improve performance even further.

Barfield et al (1998) studied the perceived level of presence within a virtual environment as a function of input device type and update rate. Although the type of input device had limited effect, it was found that an update rate of at least 15 Hz was a critical value in order to experience a sense of presence. Regarding ease and comfort of navigation, interactivity and smoothness of motion, an update rate of 15 Hz was considered equally important. The study further revealed consistently higher ratings for all factors when the update rate was increased to 20 Hz.

In a more recent study, Claypool (2007) investigated the impact of frame rate on player performance in first person shooter games. For movement tasks it was found that an increase of frame rate from 7 Hz to 15 Hz significantly improved player performance. Unfortunately, no data were collected beyond 15 Hz. For shooting tasks, the frame rate was found to be even more important. Although the rate of improvement was steeper below 15 Hz, player performance was increased all the way up to 60 Hz.

For everyday 3D design and engineering applications the importance of maintaining a sufficiently high frame rate has also been highlighted. Experiments at The Boeing Company show that low frame rate decreases the feeling of continuous motion and that improved continuity helps user performance when searching for objects in a complex 3D model (Kasik

et al., 2002). Furthermore, empirical studies revealed that, although 10 Hz was considered useful, massive model visualization users require at least 16 Hz in order to be considered acceptable (Yoon et al., 2008).

Still, in practice, 15 Hz is not generally considered a sufficient level of interactivity. For applications such as architectural walkthroughs, 30 Hz is often recommended as a minimum frame rate in order to provide a suitable experience (Shiratuddin and Fletcher, 2006). Below this number users will typically start to experience lag to such a degree that the impression of continuous movement is lost (Herwig and Paar, 2002; Göttig et al., 2004).

Looking at the computer games industry it also becomes clear that 15 Hz is not enough to satisfy player demands. For so-called first-person shooter games (FPS) 30 Hz is generally considered the absolute minimum and many game developers even target 60 Hz in order to give players a smooth and responsive experience (Rubino and Power, 2008).

In this context it is also important to highlight the main difference between film (e.g. motion pictures) and real-time rendering in terms of frame rate. With current cinema using a standardized frame rate of 24 Hz, it may not be obvious to see the benefit of going beyond this number. However, while the rendered images represent singular moments in time, film can record motion-blurred images which effectively integrate information over time. As a consequence, smooth motion can be displayed at a comparably lower frame rate without suffering from apparent "jumps" between discrete moments in time (Rubino and Power, 2008).

Nevertheless, with the introduction of a new generation of HMDs, such as the Oculus Rift and HTC Vive the frame rate requirements have changed significantly compared to that of desktop VR. In order to provide a useful VR experience 90Hz is now considered critical (Hasan and Yu, 2015). This requirement is also different compared to that of desktop VR, where a lower frame rate may still give a sufficient experience, albeit with less fidelity. Due to a very strong connection to the tracker system, any update rate below 90Hz will produce such visual artefacts that the system essentially becomes useless.

2.6 Rendering performance and acceleration techniques

In Figure 7 the graphics pipeline is illustrated. As can be seen it is built up by a number of different stages and involves both the CPU and the GPU. The interaction between the CPU and GPU can be thought of as a client-server pair, where an application acts as the client on the CPU-side and uploads data and issues commands through the graphics driver that the GPU then processes. To take advantage of hardware-accelerated rendering an application will upload 3D positions that represent the vertices (i.e. corners) of a set of triangles to GPU memory, and then issue draw commands that specifies which triangles to draw on screen. The GPU will then transform and project each one of these triangles according to the position of a

“virtual” camera and, finally, convert it to a two-dimensional image using a process known as rasterization.

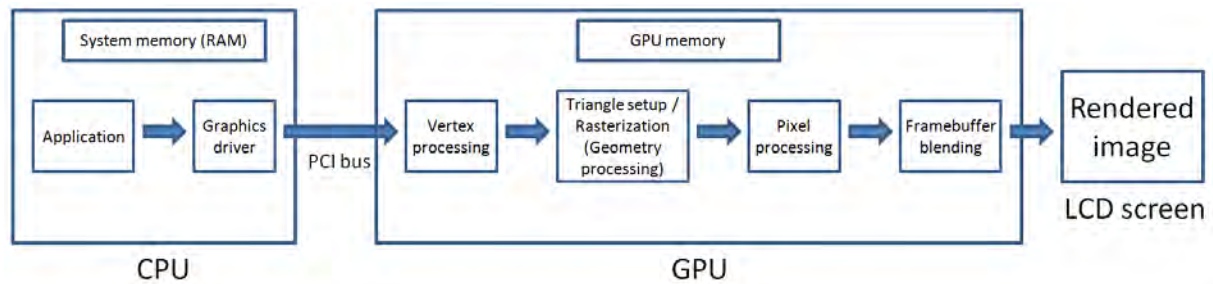


Figure 7: The graphics pipeline.

While the processes performed on the GPU used to be “fixed-function”, it is nowadays fully programmable using so-called shader programs which offer developers more flexibility when it comes to processing geometry and compute lighting. As illustrated in more detail in Figure 8 there is typically three main shader stages – vertex, geometry and fragment.

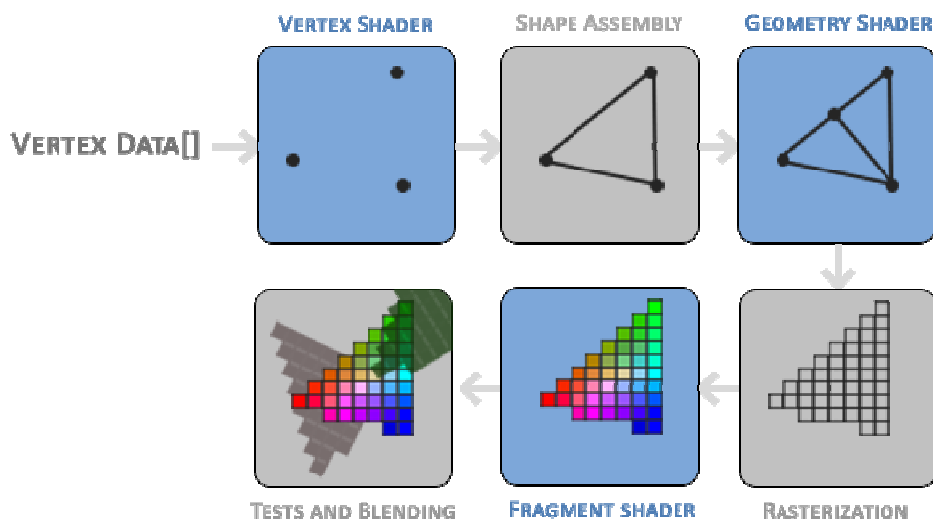


Figure 8: Different shader stages in the graphics pipeline.

During the vertex shader stage, each vertex is individually processed. This stage transforms and projects vertices to match the current view position (i.e. camera). During the geometry shader stage complete primitives (i.e. triangles) are processed. A geometry shader is optional and takes a single primitive as input and may output zero or more primitives. As such, it has the ability to amplify geometry. Finally, the fragment shader processes individual fragments (i.e. pixels) generated by the rasterization into colors that will appear on screen.

When an application takes advantage of hardware-accelerated 3D rendering “out-of-the-box”, all of the geometry of a given 3D scene has to go through at least some parts of the graphics pipeline every frame. Even with very powerful CPUs and GPUs such a scenario will therefore

always have an upper limit in the amount of geometry that can be processed at a certain frame rate. However, by taking advantage of additional acceleration techniques it is often possible to go beyond this limitation (Akenine-Möller et al., 2008). In general, these acceleration techniques can be assigned into three different categories: *pipeline optimizations* which increase performance by streamlining data flow through the graphics pipeline, *Level-of-detail (LOD)* which increases performance by reducing the geometric complexity of far-away objects and *visibility culling* which increase performance by rejecting non-visible objects. In the following subsections each one of these acceleration techniques are explained in more detail.

2.6.1 Pipeline optimizations

As with any other type of pipeline, the speed at which data can flow through the graphics pipeline is inherently dictated by the slowest stage. The general idea behind pipeline optimizations is to remove the bottlenecks without reducing the amount of geometry to process. For instance, it is very often the case that an application is CPU-bound as opposed to GPU-bound. What that means is that the GPU processes data and rendering tasks at a faster rate than the CPU is able to feed it with new data and commands. In essence, the GPU becomes underutilized. One of the main reasons for this behaviour is the amount of draw commands and state changes that are made every frame (Hillaire, 2012). Rendering a set of triangles in graphics APIs such as OpenGL usually involves two main steps: (1) modifying the OpenGL states and objects in order to setup resources (i.e. textures and vertex arrays) used for rendering and (2) issuing the actual draw call to tell the GPU to render the triangles. Both these steps involve multiple calls to the graphics driver and therefore incur a certain cost (i.e. time) on the CPU-side. In CPU-bound situations it is therefore possible to improve performance by reducing state changes and draw calls. When considering state changes many of them can often be removed by sorting the objects to render based on state, such as materials and textures. By doing so, the states required for each material only has to be set once per frame (as opposed to multiple times if objects are rendered in an unsorted way). Furthermore, in order to reduce draw calls there are mainly two ways – *geometry batching* and *geometry instancing*. With batching the idea is to combine geometry that share the same state (e.g. material) in order to form larger, but fewer, “chunks” of geometry to render (Wloka, 2003). By doing so, the same amount of geometry can be rendered, but with much fewer draw calls. In contrast, geometry instancing takes advantage of the fact that many 3D scenes contain replicated geometry, such as the wheels on a car or all the chairs around a dining table (Dudash, 2007). With the instancing abilities on modern GPUs it is possible to submit a single draw call when rendering several objects that share the same geometry. By uploading each instance’s unique position to the GPU in a previous steps, they can all be transformed to their correct place during the vertex shader stage.

2.6.2 Level-of-detail (LOD)

In contrast to pipeline optimizations, the idea behind LOD is to reduce the amount of geometry that has to be drawn every frame. When objects are far away from the viewer it is

often possible to use a much simpler representation of them without affecting the visual quality too much (Luebke et al., 2002). As illustrated in Figure 9, this technique involves the creation of several different versions of an object, each one being represented by fewer triangles than the previous one. The selection of which version to use for rendering is then based on the distance to the viewer. However, although techniques exist to automate the creation of simplified versions of an object, they usually involve some sort of manual interaction in order to reach satisfying results (Garland and Heckbert, 1997).

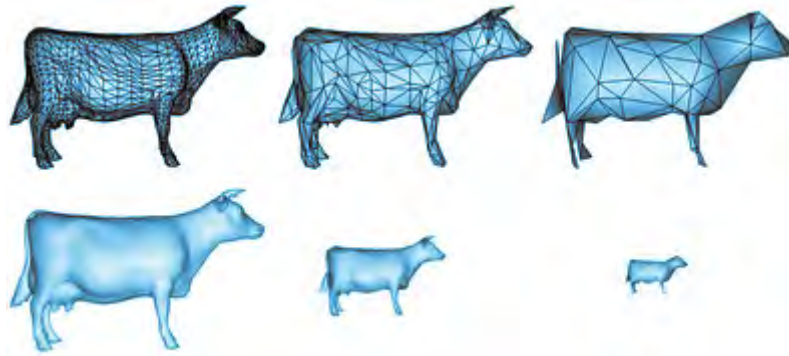


Figure 9: Illustration of LOD. The original object is replaced by simplified representations when far away from the viewer.

2.6.3 Visibility culling

As with LOD, the idea behind visibility culling is to increase performance by reducing the amount of geometry to draw. However, instead of rendering simplified objects, visibility culling tries to identify non-visible objects that don't need to be drawn at all (Cohen-Or et al., 2003). As illustrated in Figure 10, left, there will always be objects in a 3D scene that cannot be seen from a certain point of view, because they are either outside the field-of-view or hidden by other objects. The simplest form of visibility culling is *view-frustum culling*, which rejects objects that are outside the visible field-of-view (Figure 10, middle). This operation is typically not performed per-triangle but instead per-object using the objects bounding box (i.e. a box that fully encloses the object) for quick rejection. A more complex form of visibility culling is *occlusion culling* which rejects objects that are hidden by other objects (Figure 10, right). Compared to view-frustum culling this is a much more complex process as it requires computing how objects in a 3D scene affect each other. However, with the introduction of *occlusion queries* it has been possible to take advantage of the GPU to perform the actual visibility detection (Bartz et al., 1998). In essence, occlusion queries allow an application to render some geometry and then “ask” the GPU if it turned out to be visible. This way, proxy geometries (i.e. bounding boxes) can be used to test an object for visibility before it is actually rendered (Figure 11).

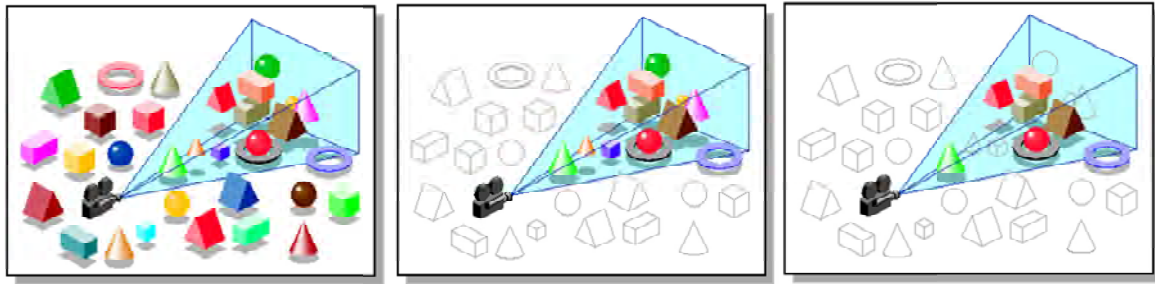


Figure 10: Illustrating no culling (left), view-frustum culling (middle) and occlusion culling (right). “Dimmed” objects are not sent to the GPU for rendering.

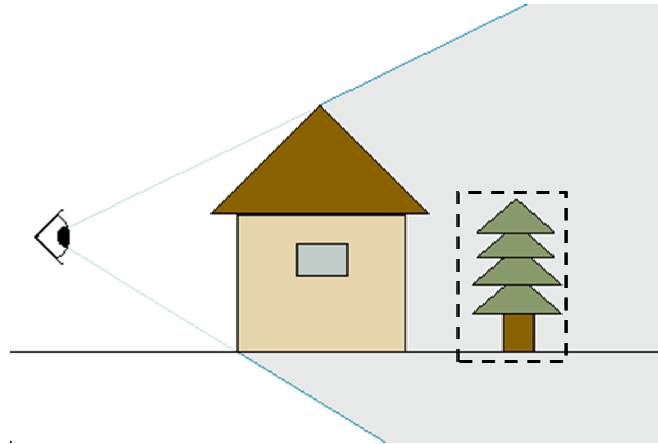


Figure 11: Using hardware occlusion queries it is possible to test the tree for visibility using a bounding box representation (dashed lines) before rendering the actual tree model.

As described above, both view-frustum culling and occlusion culling is performed *online* and therefore requires no offline pre-computations. However, there is also visibility culling techniques that works by pre-computing a *potentially visible set* from multiple regions in the 3D scene. During run-time, this set is then indexed in order to quickly obtain the objects that are *potentially* seen from a certain region in the scene (Funkhouser and Séquin1993).

A somewhat hybrid approach is *cell-and-portal culling* which primarily lends itself for use in indoor environments (Luebke and Georges 1995). As illustrated in Figure 12 it involves the creation of *cells* that are connected by *portals*. By using this data structure it is possible to restrict rendering only to objects that are in the same cell as the camera as well as objects that can be seen in adjacent cells *through* the portals.

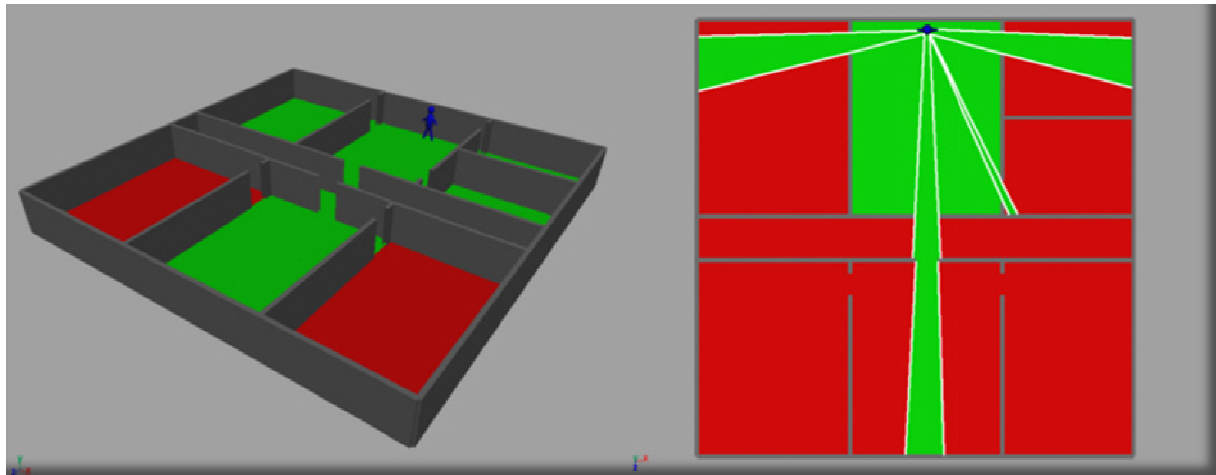


Figure 12: Cell-and-portal culling

3 Research approach

The work presented in this thesis mainly falls into the category of *technology and design science*. As such, it represents *constructive research*. As opposed to natural science, technology and design related research may be considered “artificial” in that it produces new artefacts and knowledge within a problem-solving paradigm. The concept is further explained by March and Smith (1995, p.253) who states: “Whereas natural science tries to understand reality, design science attempts to create things that serve human purposes.”

Within this paradigm, several research approaches that share a similar philosophy exists. When considering the area of Information Systems (IS), the design science research approach has mainly been popularized by Hevner et al. (2004). However, in many ways it has already been a principal approach in engineering research and computer science for a long time (Kuechler and Vaishnavi, 2008). Similarly, it is also very closely related to the constructive research approach (Pirainen and Gonzalez, 2014).

Nevertheless, regardless of specific research approach, design-oriented research is mainly concerned with the task of designing and evaluating an artefact. As discussed by Hevner et al. (2004), such an artefact needs to address an existing unsolved problem, should build on and contribute to theoretical knowledge of the problem domain and should be proven to actually improve on existing solutions or attempts to solve the problem.

Since this approach exhibit many similarities to what software developers and engineers do as part of their regular jobs, it may not be obvious what distinguishes design science research – as well as constructive research – from conventional design work. According to Hevner, there are two important differences between design research and the practice of design. First, design science involves thorough and careful use of existing theories and methods from the scientific knowledge base in order to build and evaluate the particular artefact. Secondly, it contributes to the scientific knowledge base by scholarly dissemination. As an effect of the latter, design research also tries to solve a class of problems as opposed to a specific situated problem, which is more common in design practices.

As for the actual artefacts, it has become well established within the design science field to identify four different types; *constructs*, *models*, *methods*, and *instantiations* (March and Smith, 1995; Hevner et al. 2004; Johannesson and Perjons, 2014).

Constructs are definitions and concepts that form the “language” of a domain. They are the smallest conceptual parts that make it possible to understand and communicate about various phenomena. Typical examples are the concepts of method in Java or class in the Unified Modeling Language (UML).

Models represent possible solutions to practical problems. They are sets of propositions or statements that express relationships among constructs. For instance, a database model can be used for developing a database system.

Methods are a set of steps (an algorithm or guideline) used to perform a task or solve a defined problem. Typical examples are methods for database design or a search algorithm.

Instantiations are working systems that can be used in a practice. They are realizations of an artefact in its environment, such as a database for electronic medical records or a Java program realizing a search algorithm. Instantiations can always embed constructs, models, and methods.

Furthermore, Gregor and Hevner (2013) discuss how different design science contributions, i.e. artefacts, can be classified according to the maturity of the solution, as well as the application domain. As illustrated in Figure 13, they identify four different types of contributions – *improvements*, *inventions*, *exaptations*, and *routine design*.

Improvements are new solutions for known problems. These kinds of contributions address an existing problem with a new or enhanced solution, such as one offering better efficiency, usability or utility compared to the previous state of the art.

Inventions are new solutions for new problems. These kinds of contributions involve an innovation that addresses a new and unexplored problem by offering a novel solution, such as the first X-ray machine or the first data mining system. As such, they are typically much less common than improvements.

Exaptations are known solutions extended to new problems. These kinds of contributions adapt or extend an existing solution to address a problem for which it was not intended for in the first place, such as the use of data mining in meteorology.

Routine designs are known solutions to known problems. These types of designs are often the application of existing knowledge to a well-known problem, such as the creation of a business application using best practice solutions extracted from the knowledge base. In contrast to the previous discussed types, routine designs do not offer the same opportunity to contribute to the archival knowledge base of foundations and methodologies. As such they typically do not count as design science research contributions.

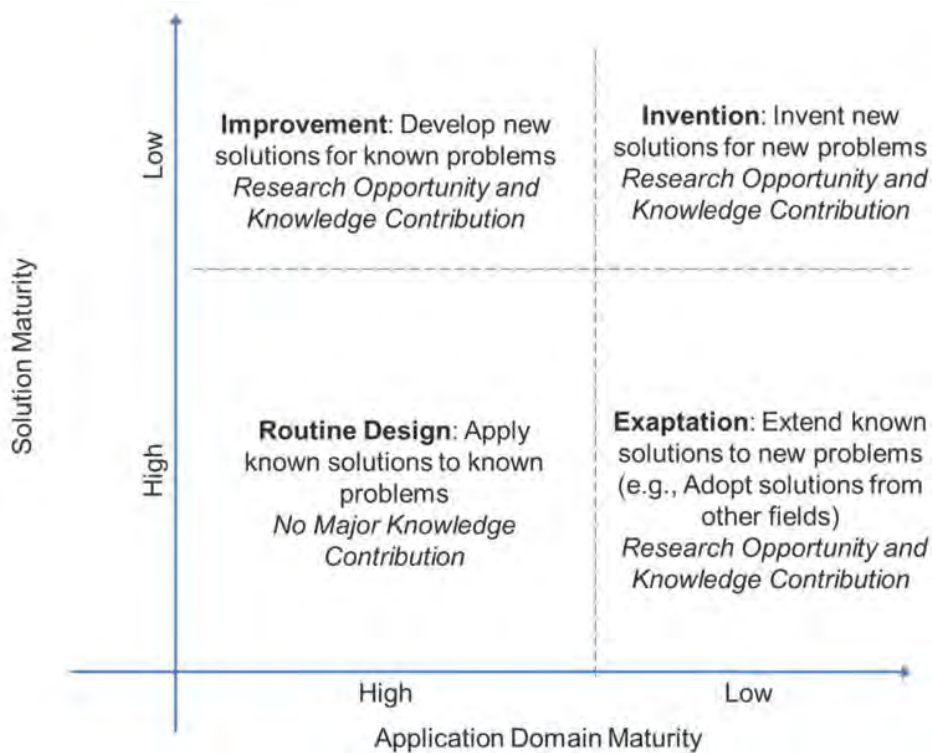


Figure 13: Design science research knowledge contribution framework (Gregor and Hevner, 2013)

When positioning the design science contribution presented in this thesis it falls within the *improvements* quadrant in that it improves on the previous state of the art with respect to efficiency as well as utility. As such, this research offers potential to contribute new knowledge to the scientific knowledgebase. As for the actual artefact, it is considered an *instantiation* in that it is a working system that can be used in practice.

When considering the actual research approach, Hevner et al. (2004) and Hevner (2007) propose a framework containing three cycles that places the design activity into a scientific framework (Figure 14). The three cycles are the *design cycle*, *relevance cycle* and *rigor cycle*. The core of the framework is the design cycle, which represents an iterative process where design alternatives are generated and evaluated against the requirements until a satisfactory design is achieved. The other two cycles connect the design cycle to the environment and to the scientific knowledge base. The relevance cycle first identifies an opportunity or an existing unsolved problem in the environment which then translates into a set of requirements that needs to be addressed by the designed artefact. An evaluation of the artefact then shows how well it meets the requirements to solve the stated problem. If it is shown to improve on existing solutions or attempts to solve the problem, the artefact is then fed back into the environment. However, the cycle repeats if the problem is only partially addressed or new problems emerge. The rigor cycle is the part that separates design science research from conventional design in a work environment (Hevner, 2007). During this cycle, the scientific knowledge base provides past knowledge to the project in order to ensure that the designs produced are research contributions and not routine designs and that appropriate and rigorous

methods are used for evaluation of the artefact. At the end of the cycle the newly developed knowledge on how to solve the identified problem is added to the knowledge base (e.g. by academic publication).

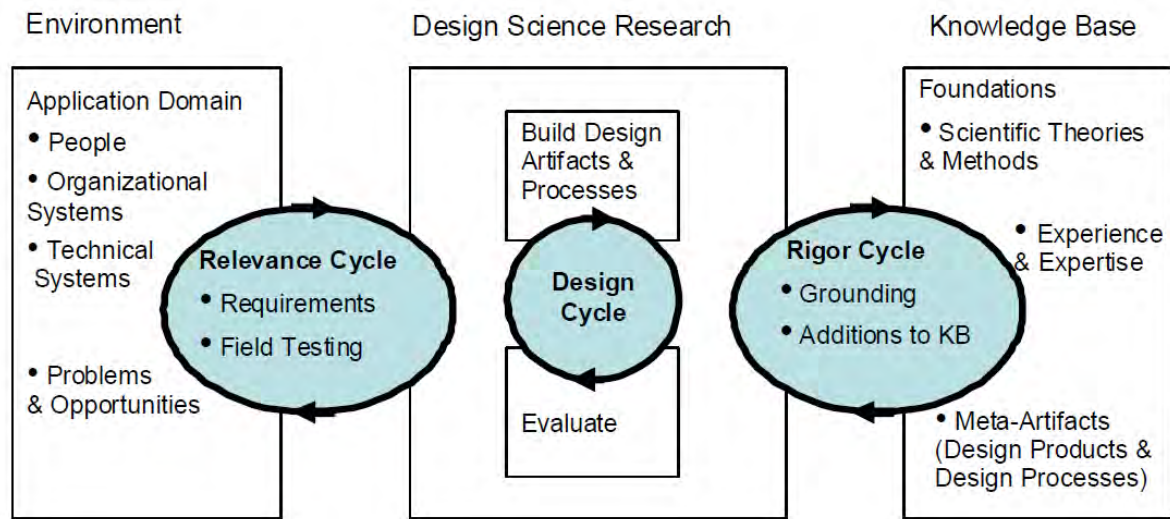


Figure 14: Design science research framework and cycles (Hevner, 2007)

In addition to the framework, Hevner et al. (2004) also outlines a set of guidelines for effective design science research. As illustrated in Figure 15 it consists of seven guidelines that highlight issues that should be addressed when performing this type of research.

When mapping the work in this thesis onto Hevners research framework, it can essentially be said that each appended paper represents a single design loop, encompassing all three cycles. Starting from the recognized opportunity (i.e. the use of BIMs for VR simulations have great potential) as well as related problems (e.g. BIMs provide a challenge to manage in real-time), an initial technology-based solution has been designed using input from the knowledge base. This solution has then been evaluated to show how well it solves the problem which essentially ends the design loop. Using the discoveries from the previous loop together with any changes in the environment and knowledge base as input, the problem formulation and evaluation criteria has then been updated and further addressed for each subsequent paper. Each design loop thus represents an incremental step towards realizing the final software application, which as of the last paper encompasses all the properties that have been identified as important in order support everyday use of VR during the building design process. The complete process will be described in more detail in Section 4, where each of the five appended papers is summarized, followed by a description of the final software application – BIMXplorer. As for the identification of required properties this will be discussed in the following subsection.

Table 1. Design-Science Research Guidelines	
Guideline	Description
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Figure 15: Design science research guidelines (Hevner et al., 2004)

3.1 Requirements

One of the most important parts of a design science project is to formulate the acceptance criteria for the ultimate evaluation of the designed artefact. As already stated, the starting point for the work presented in this thesis was the recognized opportunity of combining BIM and VR in an efficient way, as well as the related performance problems already identified in the literature and observed in an actual practise (Johansson, 2010). Thus, already from the start, the ability to provide real-time rendering performance stood out as a fundamental requirement to satisfy. However, during the course of this work new opportunities and problems have emerged which have called for updates and changes to the list of requirements that the final artefact should be evaluated against. In order to give a better understanding of how the individual contributions relate to the relevance cycle, the final set of requirements is presented in advance (i.e. before the summary of the papers) below:

The system should provide real-time rendering performance. As a fundamental feature of any real-time visualization system this requirement needs no further motivation. Still, when considering the actual definition of what real-time is, the concept becomes somewhat fluid and has to be mapped to the actual use case. As compiled from the literature, this thesis defines the *satisfactory* as well as *optimal* level of frame rate for desktop VR as 30 and 60 Hz,

respectively. However, with the introduction of a new generation of HMDs, this requirement has then been transformed into a strong 90Hz demand.

The system should support architectural BIMs taken from real-world projects. Although posed as a requirement, this should be read more as a delimitation. As of today, a BIM-based design and construction project will, eventually, include several different BIMs, each on representing a single discipline (i.e. architectural, structural, etc.). However, the primary use-case that has been considered in this thesis is that of *architectural walkthroughs*. Given VR's ability to convey scale and overall experience of space it naturally lends itself especially useful in order to study architectural qualities (Westerdahl et al., 2006; Mobach, 2008). In addition, this use-case is related to many non-professional stakeholders (e.g. clients or building end-users) who naturally have less experience in interpreting traditional design documents, such as 2D drawings. This group of people therefore have much to benefit from the use of VR in terms of communication and enhanced understanding. Thus, as reflected by the type of models that has been evaluated in the papers, this requirement (or delimitation) has primarily been posed in order to clarify that the application of visualizing structural or mechanical, electrical and plumbing (MEP) BIMs *in isolation* has not been explicitly addressed. However, this does not necessarily mean that the system cannot be used for this use-case as well.

The system should accurately reflect the input dataset. When considering acceleration techniques it is possible to resort to solutions that favour interactivity at the expense of accuracy, such as contribution or drop culling. However, seen from a scientific perspective this introduces another level of complexity when it comes to verification and evaluation. As the visualization is no longer guaranteed to truly reflect the input dataset, these types of solutions also has to be evaluated based on how well they perform in terms of accuracy. In order to not introduce this level of complexity into the evaluation, this requirement has been posed.

The system should not rely on time-consuming pre-processing steps. A viable option when considering the isolated task of providing real-time rendering performance of large 3D datasets is to perform a pre-computation step that will allow the final visualization session to run at high frame rates. However, such a solution will inherently pose itself as a potential obstacle in that an additional process is needed before any visualization session can be realized. This requirement is thus based on the simple logic that if we can omit any additional process, it will make the use of the technology more accessible and therefore easier to integrate as an everyday tool into real practise.

The system should support a wide range of users. A typical building project will involve a number of different stakeholders with different backgrounds and expertise. When considering a successful integration of VR within this setting, it is thus highly desirable that the medium can be easy to use and, ultimately, controllable by anyone.

4 Summary of the papers

4.1 Paper I

Efficient Real-Time Rendering of Building Information Models

Background and purpose

Due to a large number of individual objects and high geometric complexity, typical BIMs are not easily rendered in real-time. However, compared to a general 3D-model, a BIM defines not only geometrical data, but also information regarding spatial relations and semantics. The idea behind Paper I was to investigate if it's possible to take advantage of the additional data in order to accelerate real-time rendering.

Method

By extracting spaces (cells) and openings (portals) from a BIM we can automatically create a cells-and-portals partitioning. Using this data structure, the rendering is accelerated by rejecting objects that are not in, or can be seen from, the specific room that the viewer is currently in. To make this algorithm efficient also in outdoor cases, additional mechanisms had to be developed. These included a technique that utilizes frame-to-frame coherence and a procedure to efficiently reject non-visible exterior walls. The proposed technique was tested on two fairly large BIMs and evaluated against traditional view-frustum culling.

Results

Compared to traditional view-frustum culling, the new technique was often 10 times faster, for both exterior and interior view points, essentially making real-time rendering of large BIMs possible.

4.2 Paper II

Real-Time Visualization of Building Information Models (BIM)

Background and purpose

Paper I showed the benefit of rejecting hidden objects (i.e. cull away) with respect to real-time performance. However, the technique developed in Paper I relied heavily on specific BIM-data (i.e. spaces) being present in order to function properly. As evident from many BIMs received from real-world projects, the required data is not always present. The idea behind Paper II was to evaluate and analyze commercial BIM viewers in terms of real-time rendering performance and to evaluate more general acceleration techniques (i.e. that do not rely on specific BIM-data).

Method

Four commercial BIM viewers were in-depth analyzed in terms of acceleration techniques and real-time rendering performance. In addition, a general occlusion culling algorithm, CHC++, was implemented in a prototype BIM viewer and further refined. All viewers, including the prototype, were evaluated using four different BIMs taken from real-world projects.

Results

All four commercial viewers shared limitations in their ability to handle large BIMs interactively. The prototype viewer had no such problems. Consequently, the CHC++ algorithm was found to be a suitable acceleration technique for efficient real-time rendering of BIMs.

4.3 Paper III

Integrating Occlusion Culling and Hardware Instancing for Efficient Real-Time Rendering of Building Information Models

Background and purpose

In Paper II, occlusion culling, and more specifically, CHC++ were found to provide a suitable acceleration technique for typical BIMs. However, for viewpoints when many objects are, in fact, visible, occlusion culling alone may not always be able to guarantee sufficiently high performance. Based on the observation that typical BIMs contain many replicated objects, the idea behind Paper III was to evaluate the combination of occlusion culling and hardware-accelerated geometry instancing – a feature of modern GPUs that allow replicated geometry to be rendered very efficiently.

Method

By taking advantage of temporal coherence together with the development of a lightweight data transfer approach, occlusion culling could be performed at the object level at the same time as visible, replicated geometry can be efficiently rendered using hardware-accelerated geometry instancing. The combination of techniques was evaluated on four different BIMs taken from real-world projects.

Results

Compared to only using occlusion culling the new technique were shown to offer additional speed-ups of 1.25x-1.7x in viewpoints that represent the worst case scenarios when only occlusion culling is utilized.

4.4 Paper IV

From BIM to VR – Integrating immersive visualizations in the current design process

Background and purpose

When considering the use of immersive visualization technology within the AEC field, the introduction of consumer-directed, low-cost-high-performance HMDs devices, such as the Oculus Rift, has opened up new possibilities. Compared to previous solutions, such as CAVEs and PowerWalls, many inherent barriers, including investment costs and limited accessibility can now be broken. However, the performance demands required by stereo rendering are still difficult to satisfy without additional acceleration techniques. The idea behind paper IV was to investigate the acceleration techniques proposed in Paper II and III in a stereo setting as well as setup and evaluate a system that allowed immersive visualizations to become a natural and integrated part of the current design process.

Method

The rendering engine developed in Paper II & III was implemented as a plugin in Revit, thereby offering direct visualization from a BIM authoring environment. To support a wide range of users (i.e. from gamers to construction site workers) a simple navigation interface was developed by means of a so-called PowerPoint remote. The proposed system was tested on a BIM taken from a real world project and evaluated from three different perspectives - rendering performance, navigation interface and the ability to support fast design iterations.

Results

Compared to current immersive solutions (i.e. CAVEs and PowerWalls) the proposed system is non-expensive, portable (i.e. accessible) and has very good BIM support. Furthermore, regarding rendering performance, navigation interface and the ability to support fast design iterations, it has all the needed properties to function well in practice.

4.5 Paper V

Efficient Stereoscopic Rendering of Building Information Models (BIM)

Background and purpose

Stereo rendering is traditionally done by performing two individual and serial rendering passes – one for the left eye and one for the right eye. This was the method used in Paper IV and compared to monoscopic rendering, this setup essentially increase the number of draw calls and rasterized triangles by a factor of two. One way to remove the requirement of a second pass is by taking advantage of the geometry shader in order to duplicate and present the geometry for the left and right eye. However, even if this reduces the number of draw calls, the geometry shader typically introduces significant overhead on the GPU side. The idea

behind paper V was to explore the possibilities of using hardware-accelerated geometry instancing in order to provide a single-pass stereoscopic rendering in a split-screen stereo setup (i.e. as found in the Oculus Rift)

Method

With the instancing capabilities of modern GPUs it is possible to produce multiple output primitives from a single input, without introducing the geometry shader. As such, it becomes suitable for producing both the left and right eye view of the scene within a single rendering pass. However, the main difficulty with this approach is that current graphics API does not support multiple viewport output from the vertex shader. In the proposed technique this is solved by performing a screen-space transformation of the geometry, together with user-defined clipping planes. In addition to reduce the number of draw calls the proposed technique were shown to integrate very well with occlusion culling based on hardware-accelerated occlusion queries (i.e. as used in Paper II, III and IV). With a single depth buffer used for both the left and right eye, only a single occlusion query is ever needed per visibility test, effectively reducing the number of occlusion tests by a factor of two compared to the traditional two-pass stereo rendering technique. Furthermore, with little modifications, the new stereo instancing technique could be extended to also support the geometry instancing technique developed in Paper III.

Results

The new stereo instancing technique is very well suited for integration with occlusion query-based occlusion culling as well as conventional geometry instancing and has been shown to outperform traditional two-pass stereo rendering approach, geometry shader-based stereo duplication, as well as brute-force stereo rendering of typical BIMs on recent hardware.

5 BIMXplorer v1.0

BIMXplorer represents the final system (i.e. artefact) that is the result of the research presented in this thesis (Figure 16). In essence, this is a working software application that allows large and complex BIMs to be directly visualized in real-time, either through a traditional desktop interface (i.e. screen, mouse and a keyboard) or by using a modern HMDs such as the Oculus Rift. Many of the systems features, such as the navigation interface and the integration as a plugin within Autodesk Revit are described in detail in Paper IV. It also incorporates the acceleration techniques developed and presented in Paper II, III and V. In addition, BIMXplorer has support to directly load IFC-files through the xBIM (eXtensible Building Information Modelling) software development toolkit. Also, in order to allow for a more user-friendly navigation the system takes advantage of the PhysX SDK to support collision detection. Upon model loading, collision meshes can be automatically generated which prevents user from navigating “through” objects, such as walls and floors, in a similar fashion as modern 3D games.

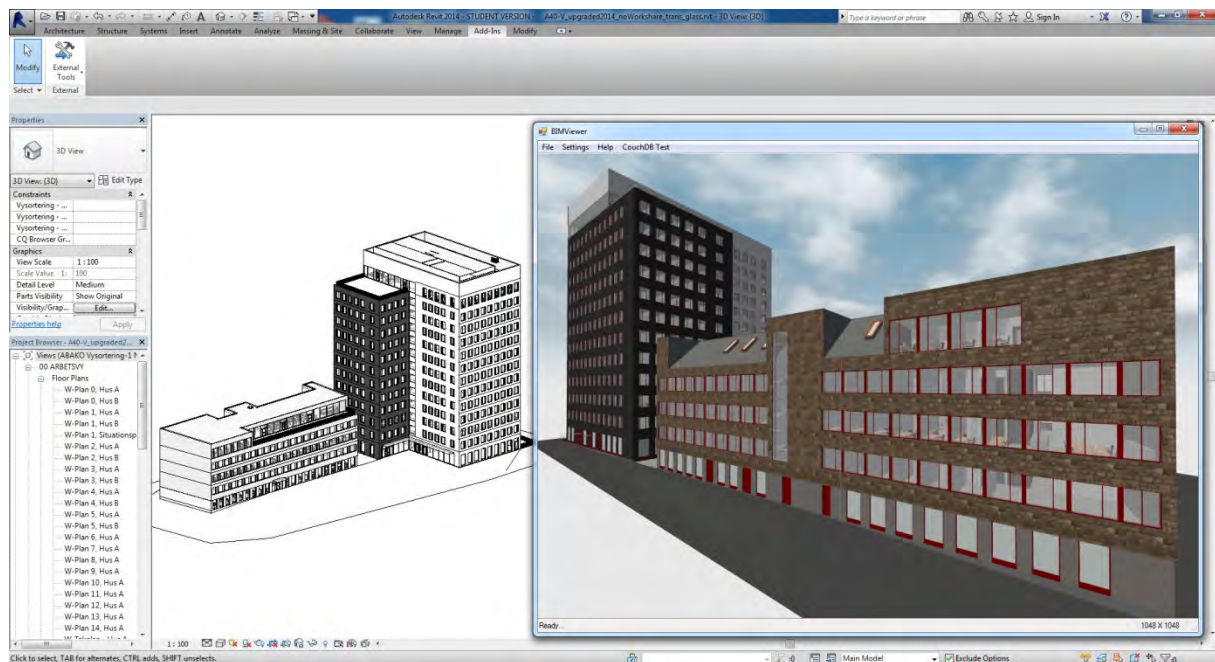


Figure 16: BIMXplorer interface as a plugin in Autodesk Revit.

To improve the visual quality BIMXplorer takes advantage of a technique known as Screen-Space Ambient Occlusion (SSAO), which calculates how exposed each point in a 3D scene is to ambient lighting (McGuire et al., 2012). Compared to a constant ambient term, this gives much better depth perception and provides clues on how objects relate to each other as seen in Figure 17.

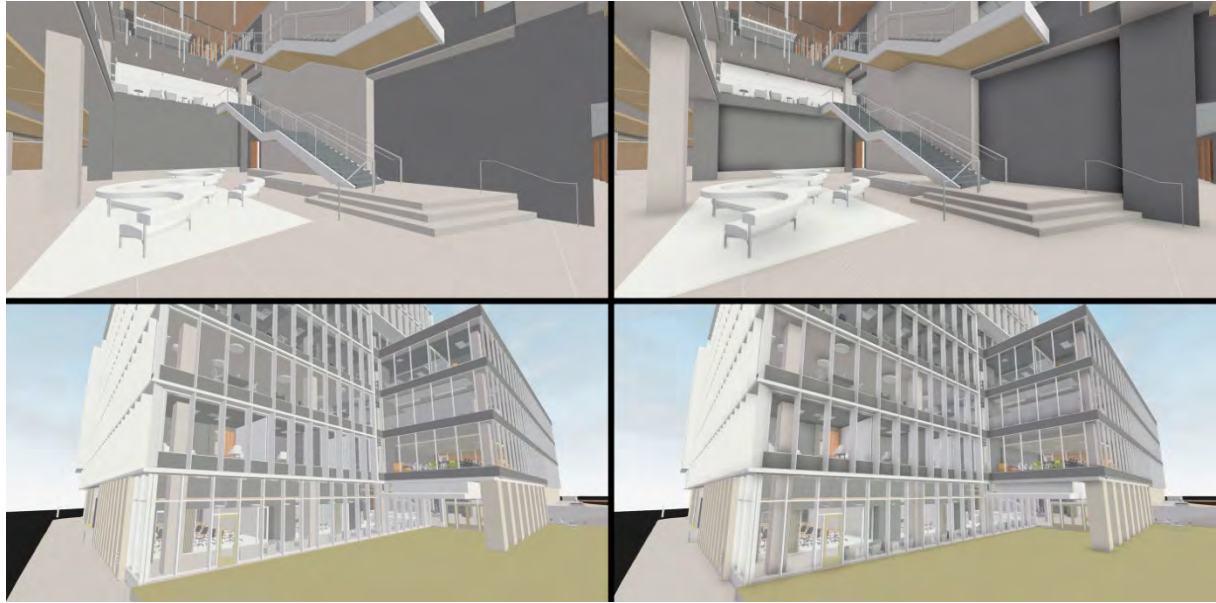


Figure 17: A BIM containing approx. 40,000,000 triangles rendered in real-time in BIMXplorer with constant ambient term (left) and SSAO (right) (Revit model courtesy of Jason Halaby, WRNS Studio).

Although not yet publicly available, BIMXplorer has already been used during several courses at Chalmers University of Technology. These courses involve the design of a suburban area as well as the design of a new university campus area featuring several new buildings created in Autodesk Revit. Throughout these projects BIMXplorer has been used as an integrated visualization tool in order to evaluate different designs and to communicate ideas among team members. At the end of these courses each team then presents their proposal by performing live walkthroughs during a final seminar (Figure 18). Being that a diverse set of BIMs have been created during these projects the courses have served as a form of continuous beta testing of the software.

In addition BIMXplorer has been in active use for over a year at NCC Construction Sweden (Jörnebrant and Tomsa, 2015; Roupé et al., 2016; Brännström and Ljusteräng, 2016) and has also been used during several projects at WRNS Studio, an architecture and planning firm located in San Francisco, California. As such, it has been proven useful in actual practise.



Figure 18: Students at Chalmers University of Technology presenting design proposals using live walkthroughs in BIMXplorer.

6 Discussion

In this section the results from the five appended papers are discussed in relation to the four research questions posed in Section 1 as well as the requirements outlined in Section 4.

6.1 Current state of BIM visualization

Going back to the initial problem statement, previous literature had already recognized the challenges of using BIMs for the purpose of real-time visualization. Still, many questions remained, such as the magnitude of the problems and how these were related to hardware and model complexity. However, based on the results from the five appended papers we can now conclude that this is, in fact, a real issue. In essence, all papers reveal this in that additional acceleration techniques – i.e. beyond that of conventional view-frustum culling or brute-force rendering – are needed in order to provide a suitable level of interactivity when rendering large BIMs taken from real-world projects. Furthermore, the in-depth analysis in Paper II shows that existing BIM-viewers are currently unable to address the problem in a satisfying manner. Also, given the huge spread in terms of rendering performance these problems can no longer only be discussed in relation to model complexity and lack of efficient hardware, but needs to include *software capacity* as an additional variable. For instance, with BIMSight any model may be seen as large and complex.

Still, it is also very important to acknowledge that several BIM-viewers have techniques to guarantee a certain level of interactivity by sacrificing correctness (e.g. drop culling). However, as identified in Paper II, the use of drop culling does not only produce an incorrect image but also gives very obvious “popping” artefacts as the priority of which objects to render constantly changes. Although no formal evaluation has been conducted as to what degree, if any, that this influences experience and usage negatively, these “popping” artefacts has been reported as very distracting in previous literature (Willmott et al., 2001; Giegl and Wimmer, 2007). As such, this thesis argues that non-conservative acceleration techniques such as drop culling are not an adequate solution to the interactivity problem.

Furthermore, when considering the actual interactivity problem as well as the corresponding requirements on frame rate, this is something that has changed during the course of this work. Although 60 Hz was initially considered an optimal level of interactivity this is no longer the case for all display systems. With the introduction of a new generation of HMDs, such as the Oculus Rift, 90Hz is now considered the absolute minimum (Hasan and Yu, 2015). Due to very strong connection to the tracker system, any update rate below 90 Hz will provide such visual artefacts that the system essentially becomes useless. As such, this requirement is different compared to desktop VR, where lower frame rates may still give an ok experience, although with less fidelity (Claypool, 2007; Rubino and Power, 2008). When also taking into account that these devices require the 3D scene to be rendered twice it should be no doubt that existing systems or techniques will not be able support HMDs in their current state.

In this context it is also important to highlight the recent trend within the AEC industry to use game engines for the purpose of real-time visualizations. Although game engines typically share the same performance problems as dedicated BIM-viewer they often have in-built tools or overall support to optimize or prepare 3D models for real-time performance. As such, it is then possible to prepare or optimize a BIM for the purpose of real-time visualization. However, this process typically involves a lot of labour-intensive, manual work making it less suitable in practise (Shi et al., 2015; Merschbrock et al., 2016).

Recent times have also seen the introduction of a couple of commercial applications, such as Revizto and Enscape, which mimics the behaviour of BIMXplorer in that they provide a plugin interface to a BIM authoring environment, e.g. Revit. These applications have not been thoroughly evaluated as part of this thesis. However, as of an initial investigation it is clear that these systems primarily rely on brute-force performance, and therefore will have difficulties to scale up to the large type of BIMs that have been evaluated in this thesis.

In relation to the above discussion it is also relevant to pose the question of *whether or not we truly need to be able to visualize large BIMs in the first place*. A simple solution to the interactivity problem would instead be to only visualize sub-sets of a complete building model, which is already a commonly used approach (Dubler et al., 2010; Shi et al., 2015). Assuming a reasonably powerful hardware it is very likely that the interactivity demands can be fulfilled as long as we restrict the visualization session to a certain region of a building. However, being forced to work with a sub-set of a complete model for the sake of performance is, by all means, a restriction. Although this thesis have only addressed actual user studies to a small degree (i.e. Paper IV), there are other recent publications that have conducted user studies with this particular system (Jörnebrant and Tomsa, 2015; Roupé et al., 2016; Brännström and Ljusteräng, 2016). Here, the ability to interactively navigate a *complete* building have been identified as a major feature in that it allows stakeholders to inspect the building as a whole and study and get an understanding of internal logistics and communication, e.g. "how many doors needs to be passed in order to get from the entrance to the lab area?". Also, the ability to have everything in a single model has been expressed as an important aspect in order to truly have a virtual representation of the actual building (Roupé et al., 2016). Nevertheless, the techniques and, ultimately, the final system – BIMXplorer – developed within the scope of this thesis naturally supports the ability to only show sub-set of a complete model.

Connecting to all of this is of course on what premises existing solutions as well as the developed techniques have been evaluated. In other words, are the models used relevant test cases? Except for the two models used in Paper I, all models have been received from real-world projects. Ranging from apartment and office buildings to more open ones, such as a library or hotel, they represent the wide variety of building types that can be encountered in practise. Also, these models have not primarily been chosen to showcase the developed technique, but instead been used as *drivers* for the development of new techniques. To better simulate a worst-case scenario two of the models (the hotel and the office building) have been

”completed” in that furniture and other interior equipment have been placed at all levels (i.e. floors) in the building(s). Among Swedish architects the general strategy today appears to be to only add interior equipment at certain levels in order to not make the models too large and complex. This, however, appears not to be the case in the US (Maller, 2011) and in order to better mimic an international situation additional levels have therefore been ”completed”.

6.2 Efficient real-time rendering of BIMs

Regarding ways to accelerate real-time rendering of BIMs two main ideas were initially considered: (1) to take advantage of the natural occlusion present in typical buildings and (2) to take advantage of the additional information (e.g. metadata) contained in a BIM. Both these ideas were explored and combined in Paper I, where information about spaces and openings was used to take advantage of portal culling (Luebke and Georges, 1995) *without* the need for any manual interaction or offline pre-processing. Although portal culling has been primarily used for indoor scenes it was shown to be efficient also for exterior viewpoints if additional techniques were added. However, even if the developed technique solved the performance problem of rendering large BIMs it turned out to be less suitable in practise. Because of the strong requirement that spaces and openings has to be correctly defined in the model, it would only be fully functional in situations where complete BIMs (i.e. complete in the sense that all spaces and openings are present) can be guaranteed. Although openings are typically generated automatically in a BIM authoring system as soon as a door or window is placed in a wall, the same is not true for spaces. Instead, spaces have to be manually added to the model as any other object - a process typically done during later stages of the design. Consequently, to support real-time visualization during *all* stages of the design process, it becomes difficult to have the main acceleration technique depend on a specific type of object - or specific data - being present in the model. Instead, a much more versatile solution could be found simply by looking at the *general characteristics* of a typical BIM – high level of geometry occlusion. As identified in Paper II, CHC++ (Mattausch et al., 2008), a state-of-the art occlusion culling algorithm, turned out to be a very good fit for real-time rendering of BIMs, essentially outperforming all existing BIM viewers on the market. Although perhaps seen as an obvious choice in retrospect, a general occlusion culling system is by far not guaranteed to always be a suitable solution. As seen from the performance results from the Navisworks occlusion culling system, it may very well perform worse than simple view frustum culling for many viewpoints. Nevertheless, CHC++ turned out to be very efficient, not only in interior, but also for exterior viewpoints, making it a very good starting point for further improvements. Still, even if the rejection of hidden objects provide a huge speed-up, there are typically several viewpoints that contain many objects that are, in fact, visible. Based on the simple logic that there is a high probability that such viewpoint contain many replicated objects, e.g. imagine all the windows seen in a hotel facade , Paper III then explored the possibility of combining occlusion culling with hardware-accelerated instancing – a feature on modern GPUs to render replicated objects efficiently. Although the use of instancing as well as occlusion culling has many examples in the literature, no known efforts had been previously made to combine these two techniques.

As seen from the results in Paper III, the idea of providing instanced rendering of un-occluded replicated objects, turned out to be a very good complement to the original CHC++ algorithm. As with the portal culling approach, the use of instancing requires specific data being present in the BIM. However, this information forms an integral part of any modern BIM authoring system – once an object of a certain *type* is added to the model it essentially becomes an *instance* of that *type*, meaning that the required data becomes available from the model. Also, the main difference compared to the portal culling approach is that the instancing technique is implemented in a way that takes advantage of replicated objects if present, but simply degrades to a standard CHC++ solution if there are no visible replicated objects. In other words, the technique does not negatively affect performance if no replicated objects are found visible.

However, with the introduction of a new type of consumer-directed HMDs, the rendering performance requirements changed. Not only became frame-rate requirements higher, but also the requirement of producing two different views every frame. As identified in Paper IV, the developed techniques were suitable also for stereo rendering. Even if two rendering passes was now required (i.e. one for each eye) the current acceleration technique was still efficient enough to support real-time frame rates. Nevertheless, as the performance demands, in terms of resolution and frame rate, became higher for each version of the HMDs it became clear that additional acceleration techniques were eventually needed. Fortunately, the concept of hardware instancing turned out to offer a solution also in this case. As identified in Paper II, the occlusion culling system was mainly CPU-bound on high end systems due to a large number of draw-calls. With a traditional stereo setup, this amount essentially doubled. Although the instancing technique (Paper III, IV) made the situation better, the amount of draw-calls still needed to be reduced in order to reach the required frame rates. The successful use of instancing then naturally sparked the idea of also using it for stereo rendering. After all, stereo rendering is essentially a process of rendering almost two replicates of the complete 3D-scene. Ultimately, this idea then became *stereo instancing* – an efficient single-pass stereo rendering technique. As seen from the results in Paper V, this also made a perfect fit for the occlusion culling system, in that not only draw-calls, but also occlusion tests became reduced by a factor of two, i.e. as compared to a traditional two-pass stereo setup.

As it would turn out, however, the concept and potential of stereo instancing had independently been recognized by developers from the game development community (Wilson, 2015). At the time of formal publication of Paper V, the stereo instancing technique was already considered a best practise within the game development industry (Vlachos, 2015). Still, the paper contributes the first detailed description of the technique and a thorough performance evaluation. For the purpose of efficiently rendering BIMs it is also the actual *combination* of the different techniques – i.e. occlusion culling, stereo instancing, conventional instancing and batching of walls – that is important in order to provide the required frame rates.

So, in perspective, the initial idea of taking advantage of the natural occlusion present in buildings as well as metadata actually turned out to be a successful approach, albeit in a different form.

Nevertheless, in this context it may also be relevant to further discuss the brute-force approach. Given that this alternative was surprisingly close to deliver sufficient frame rates for the Hotel model (Paper V), it does seem like a viable option in the near future. However, although this was true for the Hotel model, this was not nearly the case for the Student house or Office buildings. With the Hotel model only having roughly half the amount of triangles compared to the Office building, this puts things in perspective. Another aspect to consider is that with the brute-force approach essentially all of the GPU's power will be used simply to rasterize triangles. That is, even if the brute-force performance of future GPUs will be able to manage all of the dataset within an acceptable time frame, there will be less processing power left to do better shading, like SSAO, for instance. As such, it will always make sense to use additional acceleration techniques in order to better utilize the GPU's resources.

6.3 Integration of VR within the AEC field

In order for VR to become a natural and integrated tool within the design process there is more to consider than just the ability to visualize large BIMs in real-time. For instance, if time-consuming pre-processing steps or manual interactions are needed in order to support real-time performance it is highly likely that this will affect a natural integration negatively (Liu et al., 2014). However, following a design science approach, the techniques presented as part of this thesis have all been developed with the requirements and the integrational aspects in mind. Going back to the requirements that the final artefact should be evaluated against (Section 4), these can essentially be summarized as *“Being able to support instant/direct, artifact-free and user-friendly real-time VR walkthroughs of architectural BIMs taken from real-world projects on systems that exist today”*. When considering real-time performance as well as the actual definition of what real-time is the previous subsections have already discussed this. It has been shown that, when combined, the developed techniques and algorithms allow architectural BIMs taken from real-world projects to be rendered, in stereo, at more than 90 frames per second on off-the-shelf laptops. Moreover, contrary to the approach taken by several existing BIM-viewers, the developed techniques do not introduce any visual artefacts, e.g. omitting objects that should be visible.

When considering the actual integration of VR within the design process, Paper IV introduced the idea of using the rendering engine as a plugin in a BIM authoring application instead of a stand-alone application. Together with the use of a new type of consumer-directed HMD, the Oculus Rift, it essentially offers a “one button click” connection between the design environment, i.e. the BIM authoring software, and immersive VR. Compared to previous immersive solutions, like CAVEs and Powerwalls, this opens up a number of possibilities within the design process. With a low cost, portable solution that directly supports BIMs it is possible to take advantage of immersive visualizations anywhere at any time during the

design process. Although the very existence of the Oculus and HTC Vive made much of this possible, it must be highlighted that the techniques developed in this thesis are very important in order for the VR-technology to be used as an everyday tool during the design. As already discussed, the real-time rendering strategy used by any of the tested BIM viewers will not support modern HMDs. That is, even *if* stereo VR support was formally added, none of the tested viewers would be able to provide the required frame rates without introducing severe visual artefacts. Nevertheless, if we also include the use of game engines several examples can be found where modern HMDs has been used to provide immersive visualizations of BIMs. Still, as discussed, the use of game engines for the purpose of BIM visualizations currently requires additional optimization and preparation time. Even if an efficient pipeline or work procedure has been established (Halaby, 2015), this process can easily range from hours to days. In comparison, the technical contributions presented in this thesis cut this process down to, on average, 60 seconds for complete architectural BIMs. Not only does this make the use of immersive VR highly accessible in the first place, but it also supports an *active* use during fast design iterations.

Furthermore, as evaluated in Paper IV, the simple navigation interface makes it suitable also for inexperienced users. Especially when considering building end-users this becomes an important property. As of today user involvement is mostly restricted to reviewing traditional 2D-plans which may be difficult to fully interpret for all the different stakeholders in a project. With the ability for any type of user to freely navigate proposed designs from an immersive, first person perspective a much better understanding can often be achieved (Heydarian et al., 2015).

To what extent the technical contributions and, ultimately, the final system will pave the way for a more integrated use of VR during the design process remains somewhat an unanswered question. As evaluated against the technical requirements it has all the properties needed in order to function well in practise. Still, except for the evaluation of the navigation interface, no formal/documented user-studies have been conducted within the scope of this thesis. However, early versions of the system, as well as the final system have already been used in several other studies, where its suitability as a tool to enhance understanding and communication has been highlighted (Kreutzberg, 2015; Roupé et al., 2016; Hermund and Klint, 2016). It has also been used during several courses at Chalmers University of Technology. In addition, it has been in active use in several construction projects for over a year at NCC Construction Sweden (Jörnebrant and Tomsa, 2015; Roupé et al., 2016; Brännström and Ljusteräng, 2016). Simple logic tells us that this would not have been the case if the system was found unfit for use in real-world projects.

7 Conclusions and future work

The research presented in this thesis has contributed to a better understanding regarding the complexity and challenges involved in visualizing large and detailed BIMs in real-time. It has been shown that additional acceleration techniques are, indeed, needed in order to solve the interactivity problem and that existing BIM-viewers are currently unable to address this issue in a satisfying manner – this at the same time as a new generation of VR hardware calls for even higher performance demands.

In order to address the current situation this thesis contributes with the design and evaluation of a new software application that provides a “one-button-click” solution from BIM to VR. Following a design science research approach this application has been developed in order to fulfil a set of requirements that has been identified as important in order for VR and real-time visualization to become an everyday used tool for design and communication during the building design process. Along that path, three new technical solutions have been developed:

- An efficient cells- and portals culling system that is automatically realized from BIM-data.
- An efficient approach for integrating occlusion culling and hardware-accelerated geometry instancing.
- An efficient single-pass stereo rendering technique based on hardware-accelerated geometry instancing.

The final system – BIMXplorer – has been evaluated using several BIMs received from real-world projects. Regarding rendering performance, navigation interface and the ability to support fast design iterations, it has been shown to have all the needed properties in order to function well in practice. To some extent this can also be considered formally validated, as the system is already in active use within both industry and education.

For future work there are several different directions possible. For instance, when considering ways to improve rendering performance there is still much work to be done within the following areas:

Spatial hierarchy The culling efficiency is inherently dependent on how well the spatial hierarchy can represent the 3D scene with respect to occluding and enclosing objects, such as walls and floors. As of now, the bounding volume hierarchy (BVH) is constructed based on surface area and do not take into account any natural containment, e.g. due to occluding walls or floors. It would be interesting to see how the spatial hierarchy could be improved by taking advantage of any space objects – e.g. *IfcSpace* or *Rooms* in Revit – present in the model. Interior objects, e.g. furniture, could then be clustered per space object before feeding them to the BVH construction procedure.

Another way to improve the spatial hierarchy and, hence, the culling efficiency, would be to take advantage of oriented bounding boxes (OBB) instead of axis-aligned ones (AABB). As OBBs typically provides a much “tighter” fit around objects, the number of visibility tests that provides false negatives would decrease. However, instead of creating “true” oriented bounding boxes, it would be interesting to explore the concept of *building-oriented bounding boxes*. With the interior and exterior walls of most buildings following a local, orthonormal coordinate frame, it makes much sense to then orient all of the 3D scenes bounding boxes accordingly. This would then prevent many situations where object bounds intersect walls and becomes (falsely) detected as visible from the other side. Even if the local coordinate frame is not explicitly known, it should be straightforward to calculate it based on the normals of the wall geometry weighted against its surface area for all the walls contained in a BIM.

Occlusion culling The biggest disadvantage of using occlusion queries is the latency introduced by waiting for the result of the queries to return to the CPU-side of the application. CHC++ hides this latency fairly efficient by rendering previously visible objects during the wait-time, but it is still not an optimal solution. The readback of data from the GPU is required mainly because of the GPUs inability to feed itself with draw calls (Rákos, 2012). However, with recent extensions to the OpenGL API this restriction has been relaxed and it has been shown possible to implement an occlusion culling system mainly on the GPU (Boudier and Kubisch, 2015). Further exploring these features thus represents an obvious direction for future research.

Level-of-detail (LOD) As BIMs become even more detailed and several of them are to be visualized together, the concept of LOD needs to be considered in order to reduce the sheer amount of triangles that has to be rendered. Ultimately, in order to provide a scalable solution, this has to be done both locally, i.e. per object, as well as globally, i.e. for a whole facade or a whole building. However, neither of these tasks is trivial to address automatically – especially if it also has to be done in a short amount of time (Luebke et al., 2002; Arroyo Ohori, 2016). When considering per-object simplification it makes sense to initially focus on furniture and other interior equipment, as these types of objects often contain an excessive amount of triangles. A good starting point would probably be to first explore and evaluate the results that can be achieved by simplifying individual interior objects using *edge collapse* (Luebke et al., 2002; Melax, 1998).

Going beyond that of acceleration techniques, there is also much room for further research and development regarding suitable interaction interfaces for different types of applications – e.g. design review or information extraction on-site – as well as user-studies related to perception and spatial understanding with modern HMDs. Moreover, when looking at these things in a larger context, it would be interesting to see to what degree an integrated use of VR will affect the design process and, ultimately, the buildings that are a result of it.

References

- Akenine-Möller, T., Haines, E., & Hoffman, N. (2008). Real-time rendering. CRC Press.
- Arnaud, R., & Barnes, M. C. (2006). COLLADA: sailing the gulf of 3D digital content creation. CRC Press.
- Arroyo Ohori, K. (2016) Higher-dimensional modelling of geographic information. PhD-thesis, Delft University of Technology. Delft, Netherlands.
- Barfield, W., Baird, K. M., & Bjorneseth, O. J. (1998). Presence in virtual environments as a function of type of input device and display update rate. *Displays*, 19(2), 91-98.
- Bartz, D., Meißner, M., & Hüttner, T. (1998). Extending graphics hardware for occlusion queries in OpenGL. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* (pp. 97-ff). ACM.
- Bouchlaghem, D., Shang, H., Whyte, J., & Ganah, A. (2005). Visualisation in architecture, engineering and construction (AEC). *Automation in construction*, 14(3), 287-295.
- Boudier, P., Kubisch, C. (2015), GPU-driven large scene rendering, Presentation at the GPU Technology Conference (GTC 2015), San Jose, CA, USA.
- Brännström, E. och Ljusteräng, F. (2016) VR och VR-glasögon inom byggbranschen. Chalmers University of Technology (Examensarbete - Institutionen för bygg- och miljöteknik, Chalmers tekniska högskola, nr: BOMX03-16-08).
- buildingSMART, I. F. C. (2007). 2x Edition 3 Technical Corrigendum 1. International Alliance for Interoperability, URL: <http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/index.htm>.
- Bullinger, H. J., Bauer, W., Wenzel, G., & Blach, R. (2010). Towards user centred design (UCD) in architecture based on immersive virtual environments. *Computers in Industry*, 61(4), 372-379.
- Burdea, G. C., & Coiffet, P. (2003). Virtual reality technology (Vol. 1). John Wiley & Sons.
- Cohen-Or, D., Chrysanthou, Y. L., Silva, C. T., & Durand, F. (2003). A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*, 9(3), 412-431.
- Claypool, K. T., & Claypool, M. (2007). On frame rate and player performance in first person shooter games. *Multimedia systems*, 13(1), 3-17.

- Cruz-Neira, C., Sandin, D. J., DeFanti, T. A., Kenyon, R. V., & Hart, J. C. (1992). The CAVE: audio visual experience automatic virtual environment. *Communications of the ACM*, 35(6), 64-73.
- Dalton, B. and Parfitt, M. (2013). Immersive visualization of building information models. Design Innovation Research Center Working Paper, 6(1.0).
- Dubler, C. R., Messner, J., & Anumba, C. J. (2010). Using lean theory to identify waste associated with information exchanges on a building project. In *Proceedings Construction Research Congress/ASCE Conference*.
- Dudash, B. (2007). Animated crowd rendering. *GPU Gems*, 3, 39-52.
- Dvorak, J., Hamata, V., Skacilik, J., & Benes, B. (2005). Boosting up architectural design education with virtual reality, *CEMVR*, 5, 95-200.
- Dörner, R., Lok, B., & Broll, W. (2011). Social Gaming and Learning Applications: A Driving Force for the Future of Virtual and Augmented Reality?. In *Virtual Realities* (pp. 51-76). Springer Vienna.
- Eastman, C. M. (1999). Building product models: computer environments, supporting design and construction. CRC press.
- Eastman, C., Teicholz, P., Sacks, R., & Liston, K. (2011). BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors. John Wiley & Sons.
- Funkhouser, T. A., & Séquin, C. H. (1993). Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (pp. 247-254). ACM.
- Garland, M., & Heckbert, P. S. (1997). Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (pp. 209-216). ACM Press/Addison-Wesley Publishing Co.
- Giegl, M., & Wimmer, M. (2007). Unpopping: Solving the Image-Space Blend Problem for Smooth Discrete LOD Transitions. In *Computer Graphics Forum* (Vol. 26, No. 1, pp. 46-49). Blackwell Publishing Ltd.
- Greenwood, D., Horne, M., Thompson, E., Allwood, C. M., Wernemyr, C., Westerdahl, B. (2008), Strategic Perspectives on the Use of Virtual Reality within the Building Industries of Four Countries, *Architectural Engineering and Design Management*, Vol. 4, No. 2, pp. 85-98.

Gregor, S., & Hevner, A. R. (2013). Positioning and presenting design science research for maximum impact. *MIS quarterly*, 37(2), 337-355.

Göttig, R., Newton, J., & Kaufmann, S. (2004). A Comparison of 3D Visualization Technologies and their User Interfaces with Data Specific to Architecture. In *Recent Advances in Design and Decision Support Systems in Architecture and Urban Planning* (pp. 99-111). Springer Netherlands.

Halaby, J. (2015), *Simulating Presence: BIM to Virtual Reality*, Presentation at BAYA: Emerging Technologies in Architecture, 2015, San Francisco, CA.

Hasan, M. S., & Yu, H. (2015). Innovative developments in HCI and future trends. In 21st IEEE, International Conference on Automation & Computing (ICAC 2015), University of Strathclyde, Glasgow, UK.

Hermund, A., Klint, L. (2016) VIRTUAL AND PHYSICAL ARCHITECTURAL ATMOSPHERE. In *Proceedings of the International Conference on Architecture, Landscape and Built Environment (ICALBE 2016)*, Kuala Lumpur, Malaysia, pp. 3-4

Herwig, A., & Paar, P. (2002). Game engines: tools for landscape visualization and planning. *Trends in GIS and virtualization in environmental planning and design*, 161, 172.

Hettinger, L. and Riccio, G. (1992), Visually induced motion sickness in virtual environments, *Presence* 1(3), pp. 306–310.

Hevner, A. R. (2007). A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2), 87-92.

Hevner, A., March, S., Park, J., and Ram, S. (2004), *Design Science in Information Systems Research*, *MIS Quarterly* (28:1), pp. 75-105.

Heydarian, A., Carneiro, J. P., Gerber, D., Becerik-Gerber, B., Hayes, T., & Wood, W. (2015). Immersive virtual environments versus physical built environments: A benchmarking study for building design and user-built environment explorations. *Automation in Construction*, 54, 116-126.

Hillaire, S. (2012). Improving Performance by Reducing Calls to the Driver. *OpenGL Insights*, A K Peters/CRC Press, 353-364.

Johannesson, P., & Perjons, E. (2014). *An introduction to design science*. Springer.

Johansson, M. (2010). *Towards a Framework for Efficient Use of Virtual Reality in Urban Planning and Building Design*. Licentiate thesis, Chalmers University of Technology, Sweden

Jongeling, R., Asp, M., Thall, D., Jakobsson, P., & Olofsson, T. (2007). VIPP: Visualization in Design and Construction. Luleå: Luleå tekniska universitet. (Technical report / Luleå University of Technology; Nr 2007:07).

Jörnebrant, F. and Tomsa, P. (2015) The BIM Head Mounted Display as an integrative part of project phases A case study of applying new technologies in a construction project. Göteborg : Chalmers University of Technology (Examensarbete - Institutionen för bygg- och miljöteknik, Chalmers tekniska högskola, nr: 2015:89).

Kasik, D. J., Troy, J. J., Amorosi, S. R., Murray, M. O., & Swamy, S. N. (2002). Evaluating graphics displays for complex 3d models. IEEE Computer Graphics and Applications, 22(3), 56-64.

Kjellin, A. (2008), Visualizing Dynamics –The Perception of Spatiotemporal Data in 2D and 3D. Uppsala: Acta Universitatis Upsaliensis; Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Social Sciences.

Kjems, E. (2005), VR applications in an architectural competition: Case: House of Music in Aalborg. / I: Realitat Virtual a l'Arquitectura i la Construcció : Taller 2. Barcelona : Khora II, pp. 47-58.

Kreutzberg, A. (2015). Conveying Architectural Form and Space with Virtual Reality. Proceedings of the 33rd eCAADe Conference - Volume 1, Vienna University of Technology, Vienna, Austria, 16-18 September 2015, pp. 117-124

Kuechler, W., & Vaishnavi, V. (2008). The emergence of design research in information systems in North America. Journal of Design Research, 7(1), 1–16.

Liu, Y., Lather, J., & Messner, J. (2014). Virtual Reality to Support the Integrated Design Process: A Retrofit Case Study. in Computing in Civil and Building Engineering ASCE.

Luebke, D., & Georges, C. (1995). Portals and mirrors: Simple, fast evaluation of potentially visible sets. In Proceedings of the 1995 symposium on Interactive 3D graphics (pp. 105-ff). ACM.

Luebke, D., Reddy, M., Cohen, J., Varshney, A., & Watson, B., Huebner. R. (2002) Level of detail for 3D graphics, Elsevier.

Maller, A. (2011), Autodesk Revit Links, Groups, and Documentation: How to Make It Really Work!, Presentation at Autodesk University 2011, Las Vegas, Nevada.

March, S. and Smith, G. (1995), Design and natural science research on information technology, Decision Support Systems, Vol. 15, No. 4, pp. 251-266.

Mattausch, O., Bittner, J., & Wimmer, M. (2008). Chc++: Coherent hierarchical culling revisited. In *Computer Graphics Forum* (Vol. 27, No. 2, pp. 221-230). Blackwell Publishing Ltd.

McGuire, M., Mara, M., & Luebke, D. (2012, June). Scalable ambient obscurance. In *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics* (pp. 97-103). Eurographics Association.

Melax, S. (1998). A simple, fast, and effective polygon reduction algorithm. *Game Developer*, 11, 44-49.

Merschbrock, C., Lassen, A., Tollnes, T., Munkvold, Bjørn, E. (2016). Serious games as a virtual training ground for relocation to a new healthcare facility. *Facilities*. Vol. 34.

Mobach, M. (2008), Do virtual worlds create better real worlds?, *Virtual Reality*, Vol. 12, No. 3, pp. 163-179.

Modjeska, D. and Chignell, M. (2003), Individual Differences in Exploration Using Desktop VR, *Journal of the American Society for Information Science and Technology*, Vol. 54, No. 3, pp. 216-228.

Pelosi, A. W. (2010). Obstacles of utilising real-time 3D visualisation in architectural representations and documentation. In *Proceedings of the 15th International Conference on Computer Aided Architectural Design Research in Asia/Hong Kong* (Vol. 7, pp. 391-400).

Piirainen, K. A., & Gonzalez, R. A. (2014). Constructive synergy in design science research: a comparative analysis of design science research and the constructive research approach. *Liiketaloudellinen Aikakauskirja*, 3-4.

Rákos, D. (2012). Programmable Vertex Pulling. *OpenGL Insights*, A K Peters/CRC Press, 293-300.

Reddy, M. (1997). The effects of low frame rate on a measure for user performance in virtual environments. University of Edinburgh, Computer Systems Group.

Roupé, M., Johansson, M., Viklund Tallgren, M., Jörnebrant, F. och TOMSA, P. (2016) Immersive visualization of Building Information Models, *Proceedings of the 21st International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA 2016)* p. 673-682

Rubino, C., & Power, J. (2008). Level design optimization guidelines for game artists using the epic games: Unreal editor and unreal engine 2. *Computers in Entertainment (CIE)*, 6(4), 55.

- Shi, Y., Ferlet, E., Crawfis, R., Phillis, P., & Durano, K. (2015). 3D Hospital: Design and implement quest-based game framework for transitional training. In *Computer Games: AI, Animation, Mobile, Multimedia, Educational and Serious Games (CGAMES)*, 2015 (pp. 119-125). IEEE.
- Shiratuddin, M. F., & Fletcher, D. (2006, August). Southern Miss' Innovation and Commercialization Park: Development of a Large Scale Real-Time Virtual Reality Environment. In *Proceedings of 6th International Conference on Construction Applications of Virtual Reality*.
- Steel, J., Drogemuller, R., & Toth, B. (2012). Model interoperability in building information modelling. *Software & Systems Modeling*, 11(1), 99-109.
- Sutherland, I. (1965), Ultimate display, in W.A. Kalenich (ed), *Proceedings of IFIP Congress 2*, New York, Spartan Books, 506–508.
- Sunesson, K., Allwood, C. M., Paulin, D., Heldal, I., Roupé, M., Johansson, M., & Westerdahl, B. (2008). Virtual reality as a new tool in the city planning process. *Tsinghua Science & Technology*, 13, 255-260.
- Svidt, K., & Christiansson, P. (2008). REQUIREMENTS ON 3D BUILDING INFORMATION MODELS AND ELECTRONIC COMMUNICATION–EXPERIENCES FROM AN ARCHITECTURAL COMPETITION. In *CIB W78 25th International Conference on Information Technology: Improving the Management of Construction Projects Through IT Adoption*, Chile (pp. 231-238).
- Vlachos, A. (2015) Advanced VR rendering. Presentation at the Game Developers Conference 2015, San Francisco, California
- Westerdahl, B., Suneson, K., Wernemyr, C., Roupé, M., Johansson, M., & Allwood, C. M. (2006). Users' evaluation of a virtual reality architectural model compared with the experience of the completed building. *Automation in construction*, 15(2), 150-165.
- Willmott, J., Wright, L. I., Arnold, D. B., & Day, A. M. (2001). Rendering of large and complex urban environments for real time heritage reconstructions. In *Proceedings of the 2001 conference on Virtual reality, archeology, and cultural heritage* (pp. 111-120). ACM.
- Wilson, T. (2015) High performance stereo rendering for VR. Presentation at San Diego Virtual Reality Meetup #3, San Diego, California.
- Wloka, M. (2003), Batch, batch, batch: What does it really mean?, Presentation at Game Developers Conference 2003.

Woksepp, S., & Olofsson, T. (2008). Credibility and applicability of virtual reality models in design and construction. *Advanced Engineering Informatics*, 22(4), 520-528.

Yan, W., Culp, C., & Graf, R. (2011). Integrating BIM and gaming for real-time interactive architectural visualization. *Automation in Construction*, 20(4), 446-458.

Yoon, S. E., Gobbetti, E., Kasik, D., & Manocha, D. (2008). Real-time massive model rendering. *Synthesis Lectures on Computer Graphics and Animation*, 2(1), 1-122.

Paper I

Efficient Real-Time Rendering of Building Information Models

M. Johansson and M. Roupé

Department of Civil and Environmental Engineering
Chalmers University of Technology, Göteborg, Sweden

Abstract - A Building Information Model (BIM) is a powerful concept, since it allows both 2D-drawings and 3D-models of buildings or facilities to be extracted from the same source of data. Compared to a general 3D-CAD model a BIM is a different kind of representation, since it defines not only geometrical data but also information regarding spatial relations and semantics. However, because of the large number of individual objects and high geometric complexity, 3D-data obtained from a BIM are not easily used for real-time rendering without further processing. In this paper we present a culling system specifically designed for efficient real-time rendering of BIM's. By utilizing the unique properties of a BIM we can form the required data structures without manual modification or expensive preprocessing of the input data. Using hardware occlusion queries together with additional mechanisms based on specific BIM-data, the presented system achieves good culling efficiency for both indoor and outdoor cases.

Keywords: 3D graphics, BIM, real-time rendering

1 Introduction

In the field of architecture and building design a concept known as Building Information Model (BIM) is now becoming a reality. Using modern modeling tools, such as ArchiCAD or Autodesk Revit, the content produced by architects has evolved from simple 2D-drawings to parametric, object-oriented 3D-models (Figure 1). Compared to a general 3D-CAD model a BIM is a different kind of representation, since it defines not only geometrical data but also information regarding spatial relations and semantics. Its purpose is to represent any building or facility in detail and to allow the extraction of both 2D-drawings and 3D-models. In theory, this makes it possible to use one source of data for 2D-drawings, offline rendering and real-time rendering. However, when used for real-time rendering, many polygonal datasets extracted from BIM's are still too large in order to achieve interactive frame rates. A common solution is to introduce a culling mechanism to reject objects that do not contribute to the final rendered image. As different culling algorithms have different strengths and weaknesses, the choice of one is highly dependent on the type of scene that is to be rendered. For indoor architectural models, that naturally exhibits a lot of occlusion, a cell-and-portal partitioning [1] is

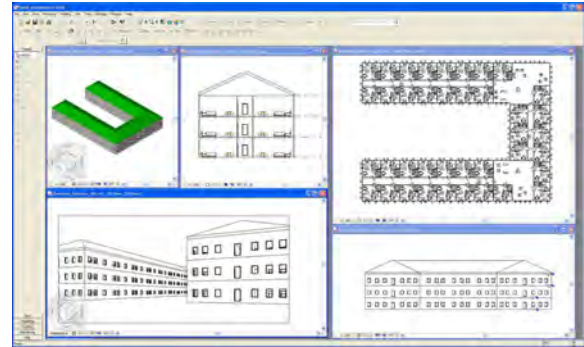


Figure 1: A BIM created in Autodesk Revit. 2D-plans, sections and 3D-models are maintained and extracted using a single database.

often used. By defining cells and portals that connect the cells, the scene is traversed starting with the cell containing the current viewpoint. An adjacent cell is processed if any portal leading into it is found to be visible. Although shown to be efficient, the cell-and-portal partitioning usually require manual interaction in order to be formed. For the purpose of quickly visualizing different proposals during the design of a building or facility, this is an unwanted step. Work has been done to enable automatic creation of the cell-and-portal partitions [2, 3], but the quality of the result when applied to a detailed building model remains unknown. Even if the creation of the partition could be solved the exterior visualization of a building model still remains an additional challenge. As the portal culling procedure was initially developed for indoor walkthroughs, it may not be efficient enough when used for an outdoor case. However, previous attempts to solve this problems were restricted by the properties of a general 3D model. With a BIM, information regarding spatial relations and semantics is accessible, which enables the problem to be treated differently.

In this paper we present a culling system specifically designed for efficient real-time rendering of BIMs. The system extends previous cells-and-portals visibility methods and by utilizing the unique properties of a BIM the required data structures can be formed without any manual interaction or expensive preprocessing. Hardware occlusion queries are used for portal visibility detection, and together with additional mechanisms based on BIM-data we achieve good culling efficiency for both indoor and outdoor cases.

The rest of the paper is organized as follows: In Section 2 related work is reviewed. Section 3 describes the IFC building model which is used as an exchange file format by BIM authoring applications. Section 4 presents our culling system in-depth and in Section 5 we present and discuss the results obtained from two test models. Finally, Section 6 concludes the paper.

2 Related work

In order to accelerate view frustum culling, Clark et al. [4] presented bounding volume hierarchies (BVH), where the scene to be rendered is organized into a hierarchical tree-structure with groups and sub-groups encapsulated by bounding volumes. Using this data-structure, entire branches of a tree can be rejected if any parent group is found to be outside the current view frustum.

Objects or geometries that pass the view frustum culling test can still be occluded by any other object or geometry in the scene. This can be detected using either object- or image-space methods. In [5] Hudson et al. presented the concept of occlusion culling with planar occluders, where a shadow frustum is constructed for each of the selected occluders. These frusta are then used to detect the invisible regions of the spatial hierarchy. This technique was later refined by Schaufler et al. [6] with occluder fusion, where several overlapping occluders were treated as a single occluder in order to reduce the number of individual tests.

The support for hardware occlusion queries (HOQ) [7] on Graphics Processing Units (GPUs) has led to a number of general occlusion culling algorithms operating in image-space [8,9,10]. With hardware occlusion queries, the GPU can be used to query the number of pixels that will end up on screen when rendering a specific set of geometries. This way, proxy-geometries can be used to test if any occlusion is present before the actual geometry is rendered. However, the technique will put additional stress on the GPU and can actually decrease rendering performance in scenes with a low number of natural occluders [11]. As of today, the most promising algorithm utilizing HOQ appears to be the Coherent Hierarchical Culling [8] introduced by Bittner et al. which exploits spatial and temporal coherence to reduce the overhead and latency of HOQ.

Indoor environments, such as those found in architectural walkthroughs, naturally exhibit a lot of occlusion. By extending the cells-and-portals technique introduced by Jones [1], Airey [12], and later Teller [2], used this feature as an advantage when precomputing from-region (cell-to-cell) visibility. Instead of precomputing a potentially visible set (PVS) from each cell, Luebke and George [13] proposed a from-point visibility calculation performed online. The method recursively traverses cells and for every visible portal the current view frustum is reduced to the screen-space bounding box of the portal geometry before traversing the

adjacent cell. Whether performed online or offline, the portal-based algorithms mainly relies on the definition of cells and portals and much research has been focused on automating the creation-process by the use of offline calculations [2, 14, 15, 16, 17]. Work has also been done in order to extend the cells-and-portals visibility method to handle outdoor environments. In [3], Lerner et al. presents a method to automatically create a cells-and-portals partitioning for urban scenes. However, their method is only applicable on simple models where building facades are represented as opaque vertical faces. As such, it cannot handle detailed building models where walls, windows and doors are represented as actual 3D-objects.

3 The IFC building model

The Industry Foundation Classes (IFC) was designed to provide a universal basis for the information sharing over the whole building lifecycle [18], and is the de facto standard for representing Building Information Models (BIM). It differs from general 3D-file formats, such as 3D Studio or COLLADA [19], in that it represents a building or facility with specific (virtual) building objects instead of pure geometrical entities. The IFC scheme supports a wide variety of buildings objects, such as `IfcWall`, `IfcDoor`, `IfcWindow`, `IfcSlab` and `IfcRoof` together with an unlimited set of properties connected to each object. Using the `IfcRelation` feature, any object can also relate to other objects, making it possible to form constraints between building parts. Another major difference between IFC and general 3D-file formats is the representation of space. Every instance of an IFC-object must belong to a spatial context. Special space-enclosing structures are the sites (`IfcSite`), buildings (`IfcBuilding`), storeys (`IfcBuildingStorey`) and rooms (`IfcSpace`). Any other spatial features, such as corridors or stair shafts are represented with the general `IfcSpace` definition. Additionally, any window or door placed in a wall results in an opening element (`IfcOpening`) that represents the cut-out in the affected wall. In Figure 2, the concept of spaces and openings is illustrated for a simple building model. According to the specification both openings and space-enclosing objects are defined as closed polyhedrons, which can be non-convex.

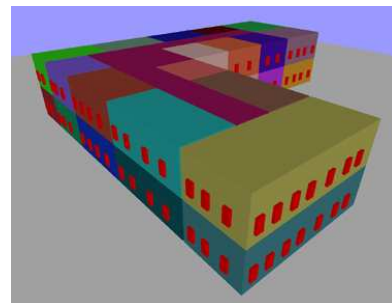


Figure 2: Geometries for spaces and openings (in red), as described by the IFC building model.

For the purpose of visibility determination the IFC building model has several advantages over general 3D-file formats. The clear definition of spaces and openings are important features as it enables information about the surroundings at any location. Together with specific knowledge regarding each objects properties and function within the building, all the necessary information to form a cells-and-portals structure without expensive pre-processing is accessible.

4 A culling system suitable for Building Information Models

The major components of our proposed system are shown in the UML diagram in Figure 3. By using the definition of spaces (cells) and openings (portals) in an IFC-file the structure can be formed during load-time without any additional calculations or preprocessing. The extraction of data from the IFC-file is performed in two steps; first we extract every individual object, second we perform an organizing step where inter-object connections are assigned according to the relation data.

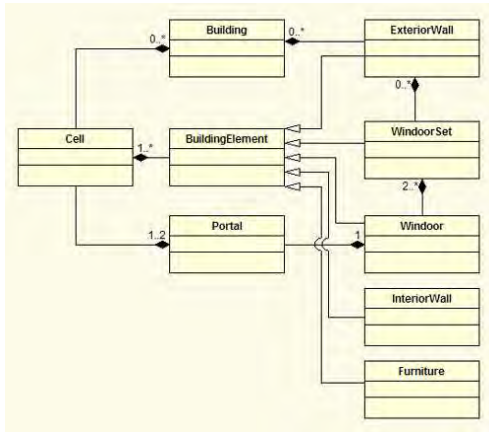


Figure 3: UML-diagram describing the major components of our system.

As seen in Figure 3, our system maintains much of the high-level structure defined by the IFC building model. The actual building object is composed of exterior walls, roofs and slabs and also has access to all the cells contained in the building. Further on, exterior walls, window sets, windowdoors and portals form a hierarchy where any portal is connected to either one or two cells. Finally, cells have access to its contained and enclosing objects. The term windowdoor refers to either a door or a window and by grouping those according to the cells that they connect to, window sets are formed.

Using this structure it is possible to perform seamless traversals starting either from the outside of the building or from a specific cell in the building. However, the high-level design itself does not guarantee the effectiveness of the culling process. In the remaining of this section we therefore

present additional mechanisms that enable the system to behave performance effective independent of view-point location.

4.1 Use of hardware occlusion queries to detect portal visibility

In a cell and portal system, portal visibility detection becomes most efficient when cells are defined as convex volumes. This feature guarantees that any geometry representing the cells boundary never occlude any portal connected to the cell. However, when using non-convex cells a portal is no longer guaranteed to be visible if it intersects the current view frustum. The situation is illustrated in Figure 4 for an L-shaped room.

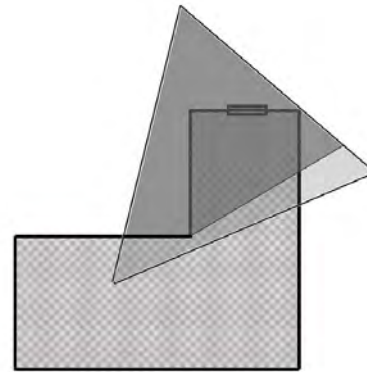


Figure 4: Undetected portal occlusion due to non-convex room shape.

The portal in Figure 4 is occluded but it will still be processed because it intersects the current view frustum. In order to reduce the number of unnecessary portal traversals the presented system takes use of hardware occlusion queries to detect portal visibility. Although this could have been solved by decomposing the cells geometries into convex regions, hardware occlusion queries offers additional benefits and require no preprocessing of the input data. By issuing a query when rendering the portal geometry we transfer the visibility detection to the GPU. This has several advantages:

- Portals can have arbitrary shapes.
- Portal occlusion will be accurately detected in non-convex cells.
- Portal occlusion due to other objects (furniture, bookshelf) in a specific cell will be detected.

Still, the use of hardware occlusion queries comes with restrictions and potential problems. The occlusion query returns the number of pixels that passes the depth-test, which then denotes whether an object is occluded or not. For this to be accurate, occluding objects must be rasterized before the object to test is rasterized. Although the actual test is fast, its result is not immediately available due to the delay between issuing a query and the actual processing on the GPU [8]. In

essence, this means that requesting the query result directly after issuing it may stall the graphics pipeline.

In order to overcome these restrictions we always render portal geometries last every frame and then we check the results of the queries in the next frame. This effectively delays portal visibility detection by one frame which could lead to a popping behavior during fast viewpoint movement. However, in our case this potential problem is reduced because the actual transparent region of a general window will always be slightly smaller than the test-geometry due to the frame of the window (This is similar for an opened door). Although this has not been detected in our tests, the case of popping behavior due to the portal detection being one frame behind could always be reduced or eliminated by scaling the test-geometry slightly in order to detect its visibility earlier.

4.2 The indoor case

When the viewpoint is located inside the building our system behaves like a traditional cells-and-portals culling system with the main difference that hardware occlusion queries are used for visibility detection of portals. For every visible portal the current view frustum is reduced to the screen-space bounding box of the portal geometry before traversing the adjacent cell. The pseudo-code in Figure 5 illustrates the portal traversal.

```
Portal::OnIntersectingFrustum(visibleCollector) {
    if(!_isBeingTraversed)
        return;

    _isBeingTraversed = true;

    if(!_hasPendingQuery)
        if(IsOccluded(_portalMesh.getQueryId()))
            _visible = false;
        visibleCollector.addOccludeeToRenderList(_portalMesh);
        _hasPendingQuery = true;
    else
        _visible = true;
        BoundingBox bb = _portalMesh.getBB();
        visibleCollector.pushScreenSpaceOpening(bb);
        if(has_cell_1)
            _cell_1.OnIntersectingFrustum(visibleCollector);
        if(has_cell_2)
            _cell_2.OnIntersectingFrustum(visibleCollector);
        else
            visibleCollector.traverseRoot();

        visibleCollector.popScreenSpaceOpening();
        visibleCollector.addOccludeeToRenderList(_portalMesh);
        _hasPendingQuery = true;
    else
        visibleCollector.addOccludeeToRenderList(_portalMesh);
        _hasPendingQuery = true;
        if(_visible)
            //Narrow frustum and traverse cell(s) as above

    _isBeingTraversed = false;
}
```

Figure 5: Pseudo-code illustrating the portal traversal when HOQ are used for visibility detection.

Depending on building layout and location of doors and windows large objects can sometimes be visible through more than one portal. In order to not send these objects to the renderer more than once a frame stamp is used.

4.3 The outdoor case

When the viewpoint is located outside the building a large number of portals can be intersecting the view frustum at the same time. Even if many of them turn out to be occluded they still need to be tested for visibility. This can mean processing a lot of queries. However, our building representation enables additional optimizations to be performed in order to efficiently detect portal visibility.

Back-wall culling. Inspecting a building from the outside leads to an important observation – at every exterior view-point we choose, there will always be exterior walls that are not directly visible.

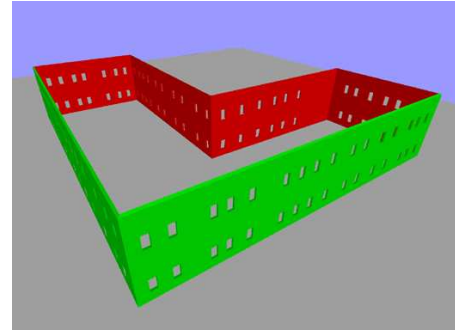


Figure 6: Exterior walls in red are facing away from the viewpoint and are considered back-wall culled.

As seen in Figure 6, several of the exterior walls are facing away from the viewpoint wherever we are located. When a roof or slab is present, these walls will not be directly visible from the outside (they may be visible through a cell, but this will be handled by the cell traversal). The concept is similar to back-face culling (but for a complete object) and requires the exterior side of the wall to be known. The information is not directly accessible from the IFC-file, but can be calculated. A wall object, as described according to the IFC specification, also has a polyline representation describing the centerline of the wall. The centerline is used to create two rays, perpendicular to the wall. One of these rays will intersect the cells mesh and therefore denote the interior side of the wall. The other one is then used as the exterior side direction.

Hierarchical traversal. The back-wall culling technique is especially powerful for a building with convex footprint as it enables fast rejection of every non-visible exterior wall. For a building with non-convex footprint, however, front-facing exterior walls can still be occluded. Because of this, the building hierarchy is used to optimize the visibility test when the viewpoint is located outside the building. The procedure is similar to certain parts of the CHC

algorithm [8], where visibility changes are propagated up- and downwards in the hierarchy. Unless any window set has been classified as visible, front-facing exterior walls are always rendered with an occlusion query. The result of the query will indicate whether the wall should continue testing for visibility in the next frame or if the tests should be performed at a lower level in the hierarchy. In Figure 7, pseudo-code for selected components of our system illustrates the hierarchical traversal. For curtain walls and frameless windows the procedure becomes slightly different and could use an enlarged window bounding box or the portal geometry instead.

Reuse of visibility classification. Although the hierarchical traversal optimizes the process of detecting visible portals, a situation could still arise when many portals are in fact visible. Repeatedly testing these portals for visibility every frame is an unnecessary waste of queries as many visible portals are likely to stay visible over a period of time. Therefore, when a portal is found to be visible it is considered to stay visible for a number of frames. A random offset value is used to schedule the next query. This way, a portal visible in frame n will only be tested for visibility in frame $n + ro$, where ro is a random offset value. For our test scenes, where the facades typically contain a lot of windows, we have used a random value between 1 and 50 for ro .

4.4 Rendering

During traversal, objects are not directly sent to the renderer, but instead placed in different queues. We use three queues:

- **DrawQueue** – Objects that has been classified as visible are placed in this queue.
- **DrawAndTestQueue** – Objects that should be tested for visibility using the actual object representation are placed in this queue.
- **TestQueue** – Objects that should be tested for visibility using a proxy-representation are placed in this queue.

After the full traversal has been performed, the different queues are sorted based on material/shader and then sent to the renderer in the above order. This simple procedure minimizes costly state changes and assures that occludees are always rendered after any potential occluders.

5 Results

We have tested our proposed system on two different Building Information Models (see Figure 8); one ten story building with a rectangular footprint (5,684 objects and 9,570,486 triangles) and one three story building with a U-shaped footprint (3,489 objects and 4,845,090 triangles). Both tests were conducted on a laptop with a 2.16 GHz Intel Core CPU, 2GB of memory and a Ge-Force Go 7950 GTX graphics card. The screen resolution was set to 1920 x 1200 pixels. The models were created using Autodesk Revit Architecture 2009 and exported to the IFC-file format

```
Building::OnIntersectingFrustum() {
    for(every exterior Wall w)
        if(w intersects frustum AND NOT back-wall culled)
            w.OnIntersectingFrustum();
}

Wall::OnIntersectingFrustum() {
    if(any WindowSet is visible)
        Wall::Render();
    for(every WindowSet ws intersecting frustum)
        ws.OnIntersectingFrustum();
    else
        if(hasPendingQuery())
            if(isOccludedBasedOnQuery())
                for(every WindowSet ws intersecting frustum)
                    ws.SetToInvisible();
                Wall::RenderWithQuery();
            else
                Wall::Render();
                for(every WindowSet ws intersecting frustum)
                    ws.OnIntersectingFrustum();
            else
                Wall::RenderWithQuery();
}

WindowSet::OnIntersectingFrustum() {
    if(any Window is visible)
        for(every Window wd intersecting frustum)
            wd.OnIntersectingFrustum();
    else
        if(hasPendingQuery())
            if(isBoundingBoxOccludedBasedOnQuery())
                for(every Window wd intersecting frustum)
                    wd.SetToInvisible();
                    _parentWall.CollectChildrenVisibility();
            else
                for(every Window wd intersecting frustum)
                    wd.OnIntersectingFrustum();
                    _parentWall.CollectChildrenVisibility();
            else
                WindowSet::RenderWithQuery();
}

Window::OnIntersectingFrustum() {
    if(_portal is visible)
        Window::Render();
        _portal.OnIntersectingFrustum();
    else
        if(hasPendingQuery())
            if(isOccludedBasedOnQuery())
                _portal.SetToInvisible();
                _visible=false;
                _parentWindowSet.CollectChildrenVisibility();
            else
                _visible=true;
                Window::Render();
                _parentWindowSet.CollectChildrenVisibility();
                _portal.SetToVisible();
                _portal.OnIntersectingFrustum();
        else
            if(_visible)
                Window::Render();
                _portal.OnIntersectingFrustum();
            else
                Window::RenderWithQuery();
                _portal.SetToVisible();
                _portal.OnIntersectingFrustum();
}
```

Figure 7: Pseudo-code for selected components illustrating the hierarchical traversal.

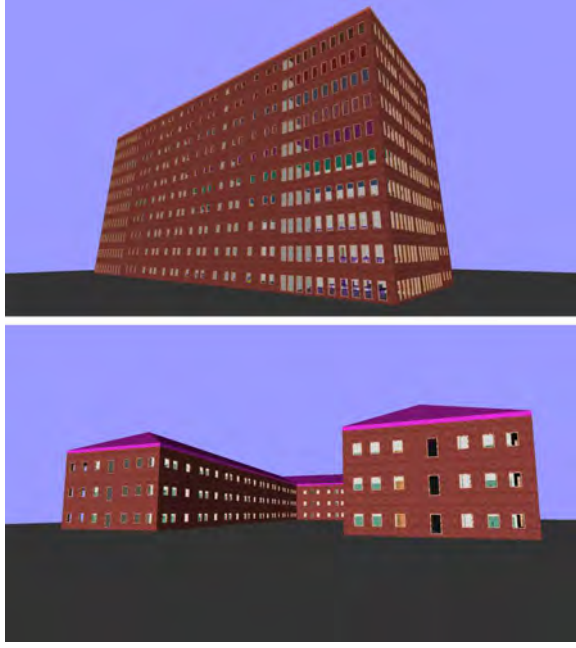


Figure 8: Top : The ten story test model. Bottom : The three story test model.

For each test scene we have constructed two walkthroughs; one exterior and one mainly interior. We have measured the frame times for rendering with view-frustum culling (VFC) only and our cells-and-portals culling implementation (CPC). For the VFC case the building hierarchy is treated as a bounding volume hierarchy.

5.1 Ten story building

In the exterior walkthrough we start at the ground level (Figure 8) and fly up above the top level while orbiting around the building. The second walk-through is a shorter sequence at the fifth level of the building. Figure 9 presents the different frame times measured for the walkthroughs of the ten story building. As can be seen, our implementation is often more than ten times faster compared to view-frustum culling. However, more important is the observation that our implementation is almost always faster than VFC, regardless of view-point location. The only exception is the special case when we are located inside the building in front of a window and looking outside. The scenario appears at four times in the interior walkthrough and the frame times here are equal or actually slightly lower when only VFC is used. Still, in such a case, only a limited number of objects are rendered and therefore never stress the overall graphics performance.

5.2 Three story building

In the exterior walkthrough we are following a path around the building while the view direction is oriented towards the building. We use the footprint of the building, offset in the exterior direction, to construct the path. In the second walkthrough, we follow a path on the second floor of the building. In the end of the sequence we exit through one

of the windows and fly across the yard to the other side of the building. Figure 10 presents the frame times measured for the different walkthroughs of the three storey building. Also in this case, our implementation is often more than ten times faster compared to view-frustum culling. Figure 11 shows how effective our implementation is compared to only VFC.

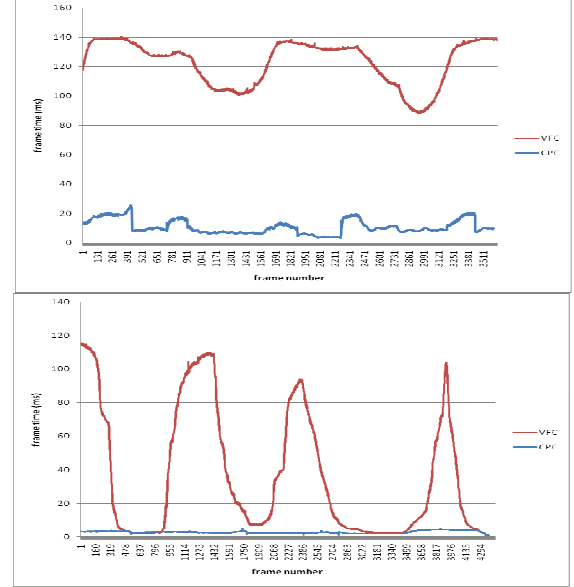


Figure 9: Frame times for the exterior (top) and interior (bottom) walkthrough of the ten story building model.

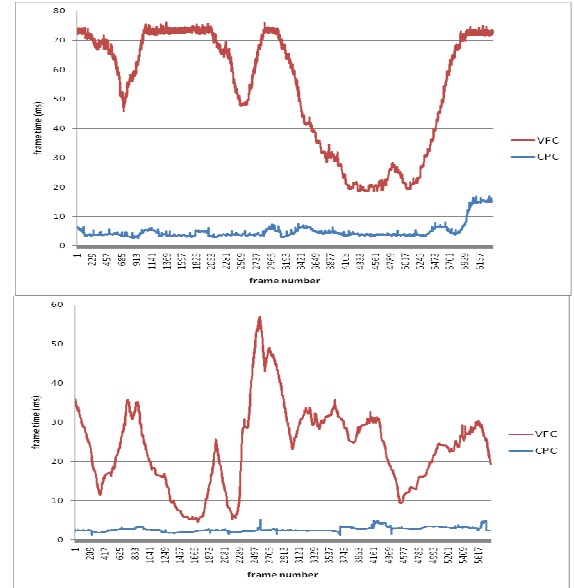


Figure 10: Frame times for the exterior (top) and interior (bottom) walkthrough of the three story building model.

6 Conclusions

We have presented a culling system specifically designed for Building Information Models. By using the unique properties of a BIM the required cells-and-portals

structure can be formed without manual interaction or expensive preprocessing of the input data. Compared to only view-frustum culling, our culling implementation was often more than ten times faster in our test scenes. This includes both indoor and outdoor cases. Moreover, our system always performs better than VFC (except for the special case explained in Section 5.1). Although cells-and-portals systems are primarily used for indoor environments, we have shown that they can be very efficient also in outdoor cases if additional mechanisms are used. Especially the back-wall culling technique, as it is an efficient method to quickly reject large parts of the scene that is hidden to the viewer.

In the future we want to investigate the possibilities to enhance the performance of our system even further by using additional BIM-data to handle level-of-detail management during rendering.

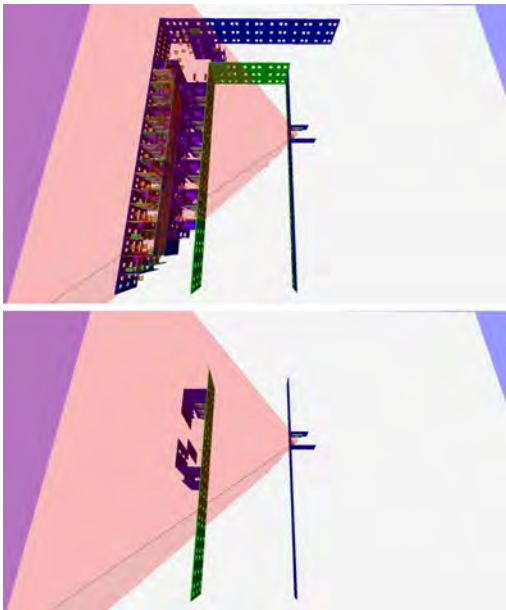


Figure 11: Comparison of view-frustum culling (top) and our culling implementation (bottom).

7 References

- [1] C.B. Jones, “A New Approach to the ‘Hidden Line’ Problem”, *The Computer Journal*, vol. 14 no. 3 Aug. 1971.
- [2] S.J. Teller, C.H. Sequin, “Visibility preprocessing for interactive walkthroughs”, *Computer Graphics Proceedings of SIGGRAPH 91*, 25(4), 61–69, July 1991.
- [3] A. Lerner, D. Cohen-Or, Y. Chrysanthou, “Breaking the Walls: Scene Partitioning and Portal Creation”, *Pacific Graphics*, 2003.
- [4] J.H. Clark, “Hierarchical Geometric Models for Visible Surface Algorithms”, *Communications of the ACM*, vol. 19, no. 10, pp.547-554, October 1976.
- [5] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, H. Zhang, “Accelerated Occlusion Culling Using Shadow Frustra”, in *13th Annual ACM Symposium on Computational Geometry proc.*, pp. 1–10, 1997.
- [6] G. Schaufler, J. Dorsey, X. Decoret, F.X. Sillion, “Conservative volumetric visibility with occluder fusion”, *Proceedings of SIGGRAPH 2000*, pages 229-238, July 2000.
- [7] M. Craighead, “GL NV occlusion query”, *OpenGL extension specification*, http://www.opengl.org/registry/specs/NV/occlusion_query.txt
- [8] J. Bittner, M. Wimmer, H. Piringer, W. Purgathofer, “Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful”, *Computer Graphics Forum (Eurographics 2004)*, 23, 3, pp. 615-624, 2004.
- [9] D. Sekulic, “Efficient occlusion culling”, In *GPU Gems*, pages 487-503, Addison-Wesley Professional, 2004.
- [10] H. Chih-Kang, T. Wen-Kai, C. Cheng-Chin, Y. Mau-Tsuen, “Exploiting Hardware-Accelerated Occlusion Queries for Visibility Culling”, *IEICE Transactions*, 88-A(7), 2007-2014, 2005.
- [11] J. Staffans, “Online Occlusion culling”, Thesis Pro Gradu, Department of Information Technologies, Faculty of Technology, Åbo Akademi, Åbo 2006.
- [12] J.M. Airey, “Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations”, PhD thesis, UNC Chapel Hill, 1990.
- [13] D. Luebke, C. Georges, “Portals and mirrors: Simple, fast evaluation of potentially visible sets”, In *ACM Interactive 3D Graphics Conference*, Monterey, CA, 1995.
- [14] S. Teller, “Visibility computations in densely occluded environments”, PhD thesis, University of California, Berkeley, 1992.
- [15] D. Meneveaux, K. Bouatouch, E. Maisel, R. Delmont, “A new partitioning method for architectural environments”, *Journal of Visualization and Computer Animation*, 9(4), 195–213, 1998.
- [16] D. Haumont, O. Debeir, F. Sillion, “Volumetric cell-and-portal generation”, *Computer Graphics Forum*, 22(3), 303–312, 2003.
- [17] S. Lefebvre, S. Hornus, “Automatic cell-and-portal decomposition”, Technical Report 4898, INRIA, 2003. <http://artis.imag.fr/Publications/2003/LH03/>
- [18] C.M. Eastman, “Building Product Models: Computer Environments Supporting Design and Construction”, CRC Press, 1999.
- [19] R. Arnaud, M.C. Barnes, “COLLADA: Sailing the Gulf of 3d Digital Content Creation”, AK Peters Ltd, 2006.

Paper II



Real-time visualization of building information models (BIM)



Mikael Johansson*, Mattias Roupé, Petra Bosch-Sijtsema

Chalmers University of Technology, Department of Civil and Environmental Engineering, Construction Management, SE-412 96, Gothenburg, Sweden

ARTICLE INFO

Article history:

Received 17 March 2014

Received in revised form 22 December 2014

Accepted 10 March 2015

Available online 28 March 2015

Keywords:

Building Information Modeling
BIM

Real-time visualization

Real-time rendering

ABSTRACT

This paper highlights and addresses the complexity and challenges involved in visualizing large and detailed Building Information Models (BIM) in real-time. The contribution of the paper is twofold: (a) an in-depth analysis of four commonly used BIM viewers in terms of real-time rendering performance and (b) the development and validation of a prototype BIM viewer specifically designed to allow real-time visualization of large and complex building models. Regarding existing BIM viewers our results show that they all share limitations in their ability to handle large BIMs taken from real-world projects interactively. However, for the same test models our prototype BIM viewer is able to provide smooth real-time performance without sacrificing visual accuracy. By taking advantage of an efficient visibility determination algorithm, our prototype viewer restricts rendering efforts to visible objects only, with a significant performance increase compared to existing BIM viewers as a result.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

In recent years, there has been a shift from vision to realization regarding the use of Building Information Models (BIM) within the architecture, engineering and construction (AEC) industries. Using modern modeling tools, such as Revit Architecture, ArchiCAD or Tekla Structures, the content produced by architects, designers and engineers has evolved from traditional 2D-drawings, sketches and written specifications to parametric, object-oriented 3D-models embedded with information to describe any building or facility in detail. As a digital representation of the physical and functional characteristics of a building, a BIM serves as a repository of information supporting a multitude of applications along the design and construction processes, including cost-estimation, energy analysis and production planning [1]. As all of the data is available in 3D, the concept of BIM further fosters the use of real-time visualizations as a tool to communicate ideas and share information among and between different stakeholders in a project. Currently, several different BIM viewers – both commercial and free – are available for the purpose of interactive presentations, walkthroughs and design reviews. During these sessions the interactive 3D visualization model becomes a common frame of reference supporting a shared understanding across interdisciplinary groups. Architects can explore design options in real-time together with the client while engineers have the ability to explain the assembly order of complex structural details for steel workers – all made possible from a single source of data.

However, as BIMs are primarily created to describe a complete building in detail, many 3D datasets extracted from them provide a challenge

to manage in real-time [2]. A fundamental feature of any type of real-time rendering system is to provide interactive and real-time updates. Failure to do so can reduce the benefit or even defeat the purpose of using the technology in the first place. A too low or fluctuating update rate will make navigation and other interaction tasks more demanding and may also cause participants to lose orientation or even feel sick [3]. In order to serve as a platform for efficient communication it is therefore important that the visualization software can deliver sufficient rendering performance to provide a smooth and interactive experience, even for large and detailed BIMs.

In this paper we highlight and address the complexities and challenges involved in visualizing large BIMs interactively. The contribution of our work is twofold. First we present our findings from analyzing four commonly used BIM viewers – DDS CAD Viewer, Tekla BIMsight, Autodesk Navisworks and Solibri Model Viewer – in terms of real-time rendering performance. We have used several BIMs received from real-world projects as test cases and show that all viewers share limitations in their ability to handle large and detailed building models. Secondly, we have developed a prototype BIM viewer to evaluate modern algorithms and strategies for real-time rendering of large 3D-models. Given the limitations found in the existing viewers, our work aims to contribute with a solution that provides both accuracy and interactivity, even for large and detailed BIMs.

The remainder of the paper is structured as follows: In the next section we review related work. Section 3 outlines the methodology and in Section 4 we present our findings from analyzing the existing BIM viewers in terms of real-time rendering performance. In Section 5 we motivate the choice of acceleration technique and provide implementation details for our developed prototype BIM viewer,

* Corresponding author. Tel.: +46 31 772 11 32; fax: +46 31 772 19 64.

together with a performance analysis of it. Finally, Section 6 concludes the paper.

2. Related work

2.1. BIM and real-time visualization

With the introduction of BIM the use of real-time 3D visualization as a communication tool has become more accessible. As 3D data can be extracted directly from the design authoring environment, there is no longer a need to create a separate 3D model for the sole purpose of visualization. However, as evident from several recent studies this development within the AEC field has also introduced new challenges. Very often models become so large and complex that they exceed the capacity of the computer which causes problems in viewing the models real-time [4–6]. Even for partial models, commonly used software tools for BIM visualization would either fail to load the 3D data or be unable to render it in real-time [2]. Furthermore, [7] discussed that even the latest hardware would quickly become overloaded, forcing architects and designers to ‘wipe out’ parts of the models to be able to work with them. Given the complexities involved in manipulating large BIMs interactively, a common work-around today is to split up the main model in different sub-models [8,9]. Nevertheless, besides restricting visualization sessions to sub-sets of the complete project, this approach typically introduces additional modeling work, as not only the initial, but also any revisions of the model need to be broken up [10]. When considering handheld and portable devices, similar issues have also been raised [11]. Given that these devices are typically equipped with much less powerful hardware the problems in terms of interactivity naturally become even worse.

However, although the challenge of visualizing large BIMs in real-time has been recognized, there are still many uncertainties surrounding the topic. For instance, none of the studies cited above, except [2], mentions any information regarding the size of the models that have been used. Instead of actual figures specifying number of individual objects or polygons, models are referred to as ‘large’, ‘complex’ or ‘of high level of detail’. Similar observation can be made regarding hardware as none of the studies except [11] provide any description of the systems in use. Thus, without any further specification of model complexity or hardware present, it becomes very difficult to either compare the findings or map them to other circumstances. Moreover, as the actual problems encountered are only vaguely described, it becomes equally difficult to understand the magnitude of them. Without using a suitable metric, such as frame rate, statements like ‘sluggish model manipulation’ and ‘viewing problems’ are hard to transform into more concrete knowledge other than that the problem exists. The current problem space also seems to be restricted to only two variables – model size and computing power. As such, any limitation in the software’s ability to efficiently utilize available computing power is not considered. In fact, none of the studies above except [2] and [10] even make any references to the actual software used. Since the studies mention that a problem exists, but do not specify details concerning hardware, software, size of models or present more explicit information concerning the problem this article highlights that currently there is a gap within the AEC research literature. So far, the challenge of visualizing large BIMs in real-time has merely been identified and is far from being addressed or even properly analyzed. Although visualizing large amounts of 3D-data in real-time is an active research topic by itself [12], there has been surprisingly little attention given to the specific case of visualizing large BIMs in real-time. Notable exceptions include recent approaches to take advantage of cloud computing to leverage sufficient rendering performance on mobile devices [13,14]. However, although this represents an interesting future research direction, current solutions either suffer from low image quality [13] or are unable to provide real-time performance [14]. Recent studies have also advocated the use of so-called game engines to visualize BIMs in real-

time [15,16]. The arguments put forward is that typical game engines, besides providing high rendering performance, offer the ability to add more elements of interactivity to the visual simulation. Still, these types of demonstrators typically use very small models. As such, they are not representative for BIMs received from real-life projects. Similar limitations also apply to the work presented in [17], where specific BIM-data, such as spaces and room definitions, were utilized in order to provide high rendering performance for large BIMs. Although the algorithm developed was successfully evaluated on large test models, these were not taken from real-life projects. Based on our own observations, the data required for the algorithm to be fully functional is not always present in models received from real-life projects.

In the current study we address the observed gap within the research literature in two ways: first, we report on the current state within the AEC industries regarding real-time visualization of BIMs by providing an in-depth analysis of the rendering capacity offered by commonly used BIM viewers. Secondly, by evaluating and implementing recently developed algorithms for efficient real-time rendering of large 3D-datasets in a prototype BIM viewer, we address the challenge of visualizing large BIMs and provide a report on what is currently possible using recent technological advancements. In the following subsection we continue our review of related work, focusing on our metric for real-time performance – frame rate.

2.2. Importance of interactivity and frame rate

An important property for any type of real-time rendering system is its ability to maintain a sufficiently high frame rate. Although this number is highly dependent on the context, there seems to be support for a threshold of around 15 Hz for a number of different applications. For simple heading tasks [18], movement and shooting tasks in first person shooter games [19] as well as overall ease and comfort of navigation in virtual environments [20], user performance has been shown to be substantially degraded when the frame rate goes below 15 Hz. When considering everyday use of 3D design and engineering applications, similar observations have been made. Experiments conducted at The Boeing Company show that low frame rate decreases a subjects feeling of continuous motion and that massive model visualization users require at least 16 Hz in order to be considered acceptable [21,12].

Still, even if 15 Hz represents a minimum frame rate in terms of acceptable performance or experience, it is generally not considered to be a preferred or satisfactory level. For many applications 30 or even 60 Hz is often advocated. In [22] 30 Hz was found to be a minimum satisfactory frame rate for an interactive visualization of a proposed university research and technology park. Below 30 Hz users would start to experience lag and imperfect renditions. For other urban and architectural visualizations similar observations has also been reported by [23] and [24], where frame rates below 25 Hz were found to produce a jerky experience where the impression of continuous movement was lost. The aim for higher frame rates is also apparent in modern 3D computer games. 30 Hz is often considered minimum in order to give players a smooth and responsive experience, and game level design becomes inherently dictated by this target [25].

In line with the above research we chose to define the *minimum*, *satisfactory* and *optimal* levels of frame rate as 15, 30 and 60 Hz, respectively. For the tests and analysis presented in this paper we will use these numbers as a metric for interactivity and real-time performance.

2.3. Acceleration techniques for real-time rendering

Even if the performance of central processing units (CPUs) and graphics processing units (GPUs) has increased tremendously during the last years there is always an upper limit in the amount of 3D-data that can be interactively managed by any system out-of-the-box. Fortunately, a number of acceleration techniques exist that allow us to

go beyond this limit. Following the notation used by [26] and [27], these techniques can basically be assigned into three categories; pipeline optimizations, level-of-detail (LOD) and visibility culling (i.e. exclusion of non-visible geometry). In the following subsections we review each one of them.

2.3.1. Pipeline optimizations

In order to effectively utilize available hardware an application must be able to feed the GPU with data and rendering tasks at a sufficiently high rate. If this is not the case, the application is likely to become CPU-bound [26,28]. This is a commonly used term to emphasize that the bottleneck is not the GPU but instead the CPU. In such a scenario the GPU becomes underutilized and the only way to increase rendering performance is to either reduce the CPU's workload or spend available cycles more efficiently. However, depending on the given situation this is often possible by arranging and processing geometry data in a different way. One aspect that is considered fundamental for static geometry is to take advantage of buffer objects to store geometry data directly in GPU memory [29]. This will allow an application to upload all geometry data to the GPU once, instead of transferring it from CPU RAM for every frame. Even for scenes with moderate amount of geometry this can have a huge impact on performance as the burden on both the CPU and the graphics bus is reduced [30].

Performance may also be further improved by batching together geometry from several objects in order to reduce the number of draw calls [28]. This is because each draw call is associated with a fixed CPU-cost, regardless of size (i.e. number of triangles). By forming larger, but fewer, objects, the same amount of geometry can be rendered but with less CPU-cost. For similar reasons it is also often beneficial to sort objects by material properties before rendering [31]. This is because changes of the rendering state typically constitute a significant cost at both the CPU- and GPU-side. By arranging draw order based on the objects state, the same amount of geometry can be rendered, but with far less state changes. For 3D scenes with many replicated objects, it is also possible to reduce CPU overhead by taking advantage of hardware-accelerated geometry instancing [32]. With this concept, multiple copies of the same geometry can be rendered with a single draw call. The actual transformation of each instance is then performed on the GPU.

However, although these optimizations are efficient per se, their common characteristic is that they do not reduce the amount of data that has to be processed by the GPU for every frame. As such, pipeline optimizations are not indefinitely scalable.

2.3.2. LOD

Compared to pipeline optimizations, a more scalable approach to consider is LOD. With LOD, the main idea is to reduce the complexity of a 3D object representation when the object is far away from the current viewpoint. In such a situation the object becomes small on screen and a less detailed representation can be sufficient in order to give the same visual impression. The simplified version of the object is often created by reducing the number of triangles, replacing geometric features with textures or a combination of both [33]. Regardless of simplification strategy the end result is an object that is less stressful for the GPU to process. However, applying LOD per-object may not always lead to improved rendering performance. As object-based simplifications typically do not reduce the number of draw calls, this will only lead to increased rendering performance if the application is GPU-bound [34,35]. For CPU-bound scenarios it is therefore more suitable to consider hierarchical LOD (HLOD), where spatially coherent objects are simplified together instead of individually [36]. The approach is similar to geometry batching, but also takes into account level of detail. Individual objects are combined either before or after simplification and, if applicable, the process can be done recursively in a hierarchical fashion. When viewed from a distance multiple objects can then be replaced by a single object that

incorporates less detailed versions of the replaced ones. Compared to per-object LOD, the benefit of this approach is thus that both object complexity and number of draw calls are reduced.

2.3.3. Visibility culling

With visibility culling the idea is to improve rendering performance by only processing geometry that is potentially visible. Often, only a subset of a complete 3D-dataset is visible from any given view point, and if an application can quickly reject invisible geometry the overall performance will be naturally increased. As illustrated in Fig. 1, this category of acceleration techniques typically includes three different approaches – *view-frustum culling*, *back-face culling* and *occlusion culling* [37]. *View-frustum culling* rejects objects outside the camera's view frustum, *back-face culling* rejects polygons that face away from the viewer and *occlusion culling* rejects objects that are hidden by other objects. Because of their efficiency and straightforward implementations, both back-face culling and view-frustum culling are present in nearly every real-time rendering application today. View-frustum culling is traditionally performed by the application on the CPU and in order to reduce computational workload visibility testing is not done per-triangle but instead per-object using a bounding box as proxy geometry. By arranging the 3D-scene in a spatial data structure, such as a bounding volume hierarchy, the process can also be further accelerated by performing the tests hierarchically [38]. In contrast, back-face culling is almost exclusively performed on the GPU. Because this feature is implemented in hardware, an application can enable and take advantage of it with a single graphics API call. Occlusion culling, on the other hand, is a much more complex process as it requires computing how objects in a 3D-scene affect each other. Although many different approaches exist, the most commonly used today perform occlusion tests (either on the GPU or the CPU) by rasterizing a proxy geometry (e.g. a bounding box that encloses the “real” object) against a z-buffer that represents scene geometry that is assumed to be, or has been previously identified as, visible. Based on the result of the occlusion test with the proxy geometry, the “real” geometry is then rendered. As occlusion culling can greatly reduce the number of objects that have to be processed on the GPU, it is a viable acceleration technique for real-time rendering of building models and will be further discussed in Section 5.

The common requirement when considering culling is that it should be conservative, i.e. it should not cull (discard) objects that are visible. However, there are also approaches that are non-conservative, such as contribution culling [39]. The rationale for contribution culling is that objects that are small on screen contribute less to the rendered image and can be removed in order to improve performance. This is typically implemented by estimating the objects screen-size (in pixels), and if below a user defined threshold the object is discarded. A more aggressive form of this concept is drop-culling [40], where not only the objects contribution, but also frame rate, is taken into account. Often based

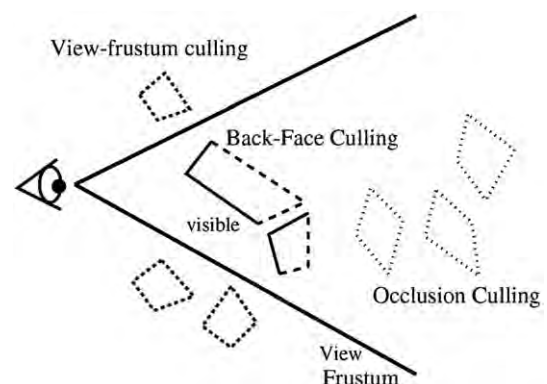


Fig. 1. Illustration of different culling techniques (Source: [37]).

on an extended heuristics taking size, screen-size and distance to view-point into account, low-priority objects are then “dropped-out” in order to maintain a user defined frame rate. However, besides presenting a visualization that is incorrect (missing information), this acceleration technique will typically also produce very distracting visual artifacts. As object priority becomes different for each view-point, objects will constantly “pop” in and out during navigation. Because three of our tested viewers were found to use drop-culling as their primary acceleration technique we will review this aspect in more detail in the following section.

To conclude our discussion of different acceleration techniques and to provide a better overview we present all of them together with their main characteristics in Table 1.

3. Methods and materials

In the study we first analyze and compare the performance of four existing BIM viewers – DDS CAD Viewer 8.0 and 10.0, Tekla BIMsight 1.9.1, Autodesk Navisworks 2015 and Solibri Model Viewer 9.0. Secondly we discuss and relate our developed prototype to these existing BIM viewers. In order to analyze and compare the different viewers (including our own prototype) in terms of real-time rendering performance, four different BIMs were used. In Fig. 2, the different test models are rendered in our Prototype viewer. These models were taken from real-life projects and represent either planned or existing buildings. All four models (Library, Student house, Hospital, and Hotel) were created in Autodesk Revit Architecture 2012 and exported to the Industry Foundation Classes (IFC) file format [41]. Although the Hotel model contains some structural elements they are primarily architectural models. As such, no Mechanical, Electrical or Plumbing (MEP) data is present. However, all models except the Hospital contain furniture and other interior equipment. In Table 2, related statistics for all models are presented. In this table, the total number of objects (i.e. windows, doors, walls) as well as triangles contained in each model is presented. In addition, the total number of geometry batches is presented, where a batch refers to a part of an object (i.e. a typical window has two batches – one for the frame, and one for the glass).

Table 1
Different acceleration techniques for real-time rendering and their main characteristics.

Acceleration technique	Pros	Cons
VBOs	Reduce CPU workload. Reduce memory transfer over the bus.	None
Geometry batching Hardware instancing	Reduce CPU workload. Reduce CPU workload.	Can decrease culling efficiency. Increase GPU workload. May complicate culling.
Level-of-detail (LOD)	Reduce GPU workload.	Requires the existence or (manual) creation of simplified models.
Hierarchical LOD	Reduce CPU and GPU workload.	Difficult to implement for general 3D scenes without manual interaction.
View frustum culling	Simple to implement. Reduce CPU and GPU workload.	None
Back-face culling	Simple to activate. Reduce GPU workload.	None
Occlusion culling	Can greatly reduce the number of triangles that needs to be rendered. Can reduce both CPU and GPU workload.	Non-trivial to implement. Actual speed-up is very implementation-dependent.
Contribution culling	Simple to implement. Reduce both CPU and GPU workload.	Non-conservative. May introduce “popping” artifacts.
Drop culling	Reduce both CPU and GPU workload. Allow for real-time navigation regardless of scene complexity.	Non-conservative. Often severe “popping” artifacts.

To measure the actual performance, Dxtory was used [42]. Dxtory is software that “hooks into” the graphics driver in order to measure frame rate. The frame rate (in frames per second) is then displayed as a numerical value in the viewport of the currently active viewer. As we have implemented different performance metrics in our prototype BIM viewer we were also able to verify the accuracy of Dxtory which shows no overhead in practice. To further analyze the behavior of the different viewers we have also used gDEBugger [43] and RenderDoc [44], which allows calls to the graphics API to be traced, and GPU-Z [45], which reports the GPU utilization level. All the performance tests were performed on two different computers, one high-end workstation (WORKSTATION) and one laptop (LAPTOP). The workstation was equipped with an Intel i7 3.06 GHz CPU, 6 GB of RAM and an Nvidia GeForce GTX 570 GPU running Windows 7 x64. The laptop was equipped with an Intel i7 1.9 GHz CPU, 4 GB of RAM and an Nvidia GeForce GT 620 M GPU running Windows 8.1 x64. According to the specifications these two systems offer vastly different capacity in terms of both CPU- and GPU performance. However, as evident from a number of Super PI tests [46] we found the single-thread CPU performance to be fairly equal on both systems. For all of the tests the viewport size was set to 1280 × 720 pixels, with anti-aliasing deactivated, and a camera field-of-view of 75°. While performing the actual tests, no other application except the currently tested BIM viewer has been running. In order to further minimize any potential performance interference, real-time anti-virus protection has been disabled and the Power Option mode in Windows has been set to “High performance”. On both systems the Nvidia graphics driver version 344.75 has been installed. In the graphics driver setting, the Power management mode has been set to “Prefer maximum performance”.

In order to make a fair comparison of the different viewers in terms of performance, any acceleration technique that results in an incorrect visualization is turned off. Specifically, Navisworks and Solibri as well as DDS CAD Viewer implement drop culling in order to guarantee interactive frame rates during navigation. As previously stated this is a non-conservative acceleration technique that simply stops drawing objects once a certain rendering time has been reached. Although the rejection of objects appears to be based on its relative “importance” for the current view (size, type and distance to viewpoint) the approach does not guarantee a correct visualization and often results in severe visual “popping” during navigation. Fig. 3 shows an example of this in Navisworks for the student housing model when the desired frame rate has been set to 20 Hz. As can be seen many objects that are visually important are omitted. During navigation the negative effects of this technique is even more pronounced as the selection of objects to “drop-out” constantly changes. In Solibri the behavior is similar, however, instead of specifying a target frame rate the users specify a maximum number of objects that are allowed to be rendered during navigation. The effects of enabling drop culling in Solibri are shown in Fig. 4 for the Hotel model. Here we have specified the maximum number of rendered objects to 9000, which corresponds to a frame rate of 20 Hz on the WORKSTATION system. In addition to show the visual artifacts encountered during navigation, Fig. 4 also highlights the problems of using object size as a heuristic for visual importance. Although small objects generally contribute less to the final image, this observation is highly dependent on context. The façade of the Hotel building is covered with a number of dark stone plates. As the individual components are small in relation to other objects in the model they are given a low priority by the drop culling algorithm. However, together they form a vital part of the visual impression and, hence, omitting them greatly affects the esthetics of the building.

In the latest version of DDS CAD Viewer (10.0), the effect of drop culling is similar, as seen in Fig. 5 where the desired frame rate has been set to 10 Hz. However, contrary to both Navisworks and Solibri it is not possible to turn it off. Because of this, we instead perform all of our tests on the previous version of the viewer (8.0), where it is possible to turn it off.

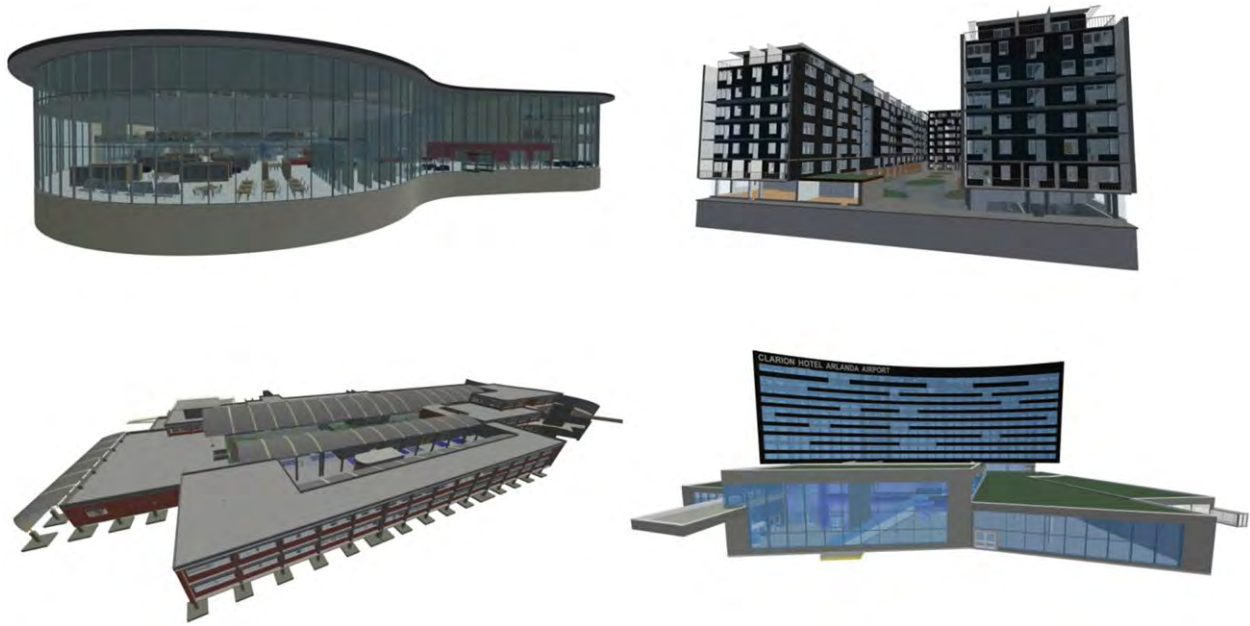


Fig. 2. The different test-models: Library (top-left), Student house (top-right), Hospital (bottom-left) and Hotel (bottom-right), visualized in our prototype BIM viewer.

For all of the tests we have focused on the worst case scenario in terms of rendering performance. For the viewers that only take advantage of view frustum culling (Solibri, BIMsight and DDS), this simply means any viewpoint where the complete model is visible. With Navisworks, on the other hand, the worst case scenario is more dependent on the particular viewpoint, as it implements occlusion culling. We therefore represent our worst case scenarios as the lowest frame rate that we have encountered during several interactive navigation sequences within each one of the four models. Still, even for Navisworks, the worst case scenario becomes an exterior viewpoint when any of the tested models are fully contained within the view frustum.

In addition we have selected two interior viewpoints in all of our test models. The first one of these, VP1, is taken from the entrance in the public buildings (Hotel, Library and Hospital) with the view direction pointing horizontally “into” the building. For the student model, VP1 is taken from a room on the “outer perimeter” of the building on the fourth floor, with the view direction pointing “into” the center of the building. For VP2 we have selected highly (Hotel, Hospital and Student house) and moderately (Library) occluded viewpoints, primarily to be able to determine how well any of the viewers could take advantage of the fact that only a subset of the scene is actually visible. Regardless of culling strategy, the interior viewpoints represent neither the worst nor the best case in terms of performance. However, with the exception of BIMsight, they do allow for a fair comparison between the tested viewers. Due to a surprisingly low, hard-coded camera field-of-view (around 30°), the view frustum is much narrower in BIMsight, than compared to the other viewers. This has the effect that it does not contain an equally large amount of objects. As such, the performance results for the interior viewpoints in BIMsight become much better than they would have been with the field-of-view that was chosen for

the tests (75°). In Fig. 6, one of the selected view points (VP1) is shown for each of the models.

4. Performance analysis and comparison of existing BIM viewers

The results from our performance tests are presented in Fig. 7 for each of the four test models (From top to bottom: Library, Student house, Hospital and Hotel). For each model, the performance in the worst case scenario as well as for viewpoint one (VP1) and viewpoint two (VP2) is presented for all existing viewers on both systems. The frame rates presented are consistent (i.e. non-fluctuating) for the different viewpoints and have been re-verified during several different test rounds. As both Navisworks and Solibri offer different acceleration features, we present the performance results from these two viewers in two versions. For Solibri this includes the default setting (Solibri) as well as with “Optimize Rendering Speed” enabled (Solibri OPT). For Navisworks the results for both CPU Occlusion Culling (Navis CPU) as well as GPU Occlusion Culling (Navis GPU) are presented. For all viewers we have performed the tests with the graphics driver settings “Threaded optimization” enabled, as well as disabled. The results presented in Fig. 7 are for the setting that provided the best performance for each viewer. For Solibri and Navisworks that turned out to be “enabled”. For BIMsight and DDS, on the other hand, that turned out to be “disabled”.

When reviewing the results in Fig. 7 it becomes clear that on the WORKSTATION system, Solibri with “Optimize Rendering Speed” enabled (Solibri OPT) offers the best overall performance. This holds for all four test models and the difference compared to the other viewers is quite significant. For the worst case scenarios, where each model is completely visible, Solibri is able to deliver speed-ups of 1.3×–44× compared to that of the other viewers. However, the benefit of the optimization feature is clearly not consistent across different systems. In fact, enabling it on the LAPTOP system actually results in a major performance decrease for the Library and Student House models.

Focusing on the results from Solibri without optimization enabled, we instead find that it share similar overall performance as Navisworks with CPU occlusion culling (Navis CPU). The only clear exception is for the Library model, where Solibri provides much better performance on both systems.

Table 2
Number of triangles, objects and geometry batches for each of the four test-models.

Model	# of triangles	# of objects	# of geometry batches
Library	3,685,748	7318	11,195
Student house	11,737,251	17,674	33,455
Hospital	2,344,968	18,627	22,265
Hotel	7,200,901	41,893	62,624

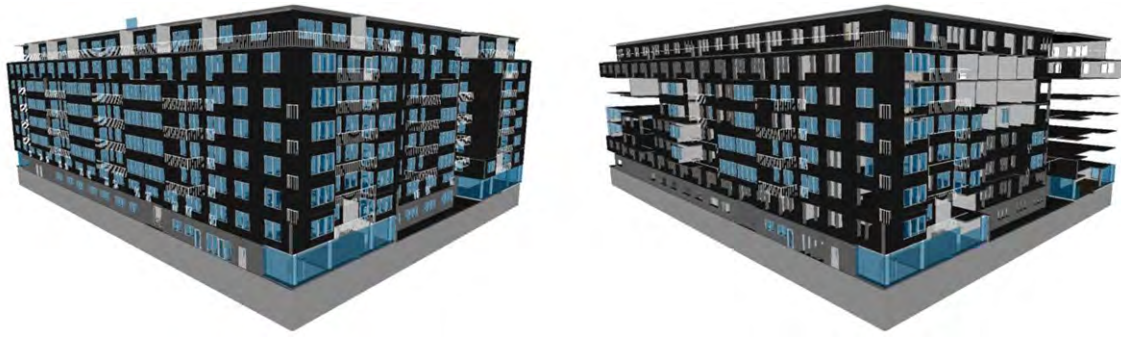


Fig. 3. Visual artifacts (right) when forcing 20 Hz in Navisworks on the WORKSTATION system.

However, perhaps more interesting is that the worst case performance offered by the other two viewers (DDS and BIMSight) is way below our minimum requirement of 15 Hz for all test models. Even for the interior viewpoints these viewers share the problem of reaching our lowest level of interactivity. When considering the worst case scenarios, equally low performance numbers also apply to Navisworks with GPU occlusion culling (Navis GPU). For all models on both systems the worst case performance is far from reaching 15 Hz. On the other hand, for highly occluded viewpoints (e.g. VP1 and VP2 in the Hotel and Student House models, as well as VP2 in the Hospital model), Navisworks with GPU occlusion culling is actually very close to deliver the best performance of all viewers.

Given the huge spread in rendering performance among the tested viewers it becomes clear that the individual rendering implementations must differ. Although the use or non-use of occlusion culling stands out as one obvious difference, it doesn't fully explain the huge variation. In order to investigate this aspect further we analyzed the different viewers using gDEBugger and RenderDoc. To our surprise we found that neither BIMSight nor DDS CAD Viewer takes advantage of vertex buffer objects (VBOs) to efficiently render geometry. Even if VBOs have been a standardized feature of OpenGL since 2003, BIMSight still use vertex arrays to draw geometry. Because of additional data transfer for each draw call, the performance penalty of this approach is naturally higher as evident by the low performance in our tests. The case of the DDS CAD Viewer, on the other hand, is more complex as it uses display lists [47]. A display list is a group of rendering commands that has been compiled and stored in GPU-memory for later execution. As with VBOs, this approach considerably reduces data transfer for each draw call and should lead to a similar performance. However, according to our test results this appears not to be the case. For reasons that are unknown the performance of the DDS CAD Viewer is instead similar to, or actually lower, than that of BIMSight.

In contrast, both Solibri and Navisworks were found to use VBOs. In the case of Solibri, one VBO is used per object. Furthermore, with

optimization enabled, Solibri takes advantage of hardware-accelerated geometry instancing. As explained in Section 2.3.1, this is a feature of modern GPUs that allows several instances of a replicated geometry to be rendered with a single draw call, thereby reducing CPU overhead. However, as evident from our performance results, this optimization technique appears to only provide a consistent performance increase on a high-end system, such as the WORKSTATION.

The use of VBOs in Navisworks, on the other hand, differs substantially. Navisworks pre-allocates a number of VBOs of similar sizes (between 1200 and 1500 triangles each) and then fills these with triangles from the model based on spatial locality. That is, a VBO in Navisworks does not necessarily correspond to the triangles of a particular object, but instead to a “chunk” of nearby located triangles. Nevertheless, contrary to what one would expect we also found that Navisworks re-uploads all, or at least major parts, of the VBO content every frame, even if there is no change in visibility. As re-uploading large amounts of data to GPU memory every frame is prone to affect performance negatively, it is thus very likely that the performance of Navisworks could be substantially increased by skipping this step. Other than to support streaming of geometry data (to support models of sizes that exceed available GPU memory) we cannot explain the choice of such a strategy.

To give a better overview of our initial findings we present viewer-specific information regarding any acceleration techniques that are used in Table 3. The following abbreviations are used (and combined) in the table: VFC for View frustum culling, DC for Drop culling, OC for Occlusion culling, and HAGI for Hardware-accelerated geometry instancing.

Still, even if Solibri and Navisworks with CPU occlusion culling were found to offer the better performance among the existing viewers, none of them were able to provide sufficiently high frame rates for all models on both systems. In fact, relating to our satisfactory requirement of 30 Hz, Solibri is only able to guarantee enough interactivity for one of the tested models (Library). For the other three models, only two of

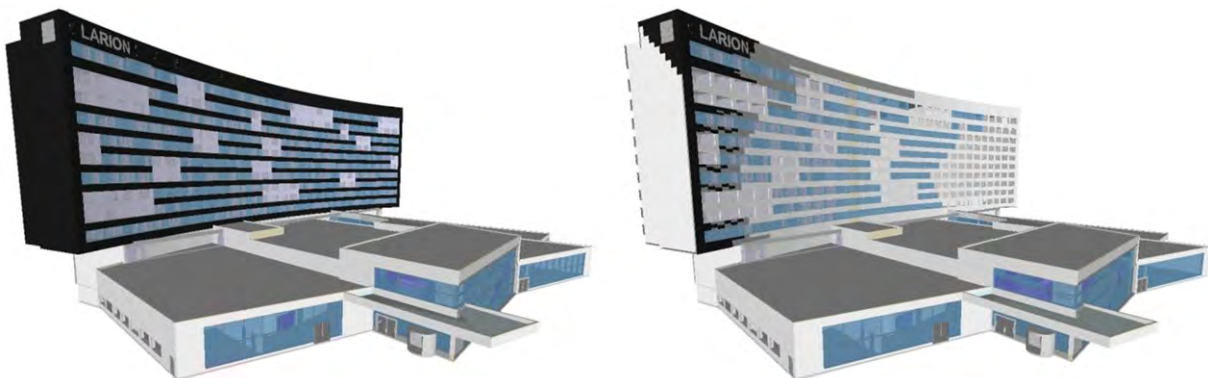


Fig. 4. Visual artifacts (right) when forcing 20 Hz in Solibri on the WORKSTATION system.

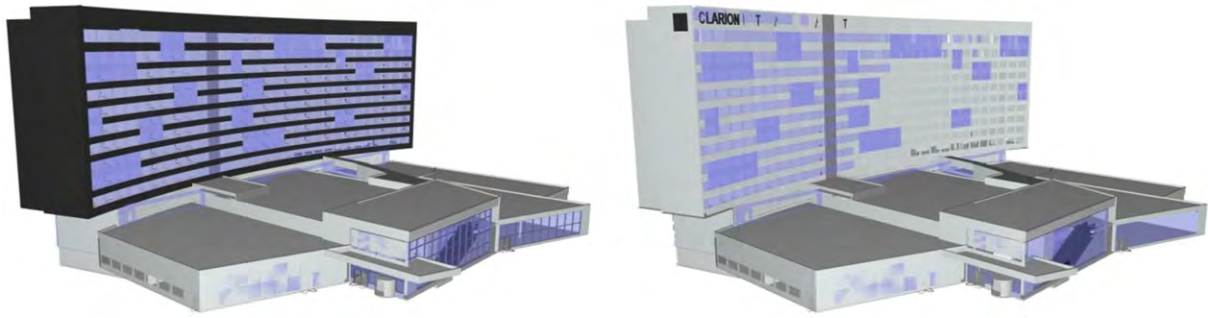


Fig. 5. Visual artifacts (right) when forcing 10 Hz in DDS CAD Viewer 10.0 on the WORKSTATION system.

them (Hospital and Student house) barely pass the minimum requirement of 15 Hz. Even with optimization enabled on the WORKSTATION system, these two models could only reach 30 Hz for the interior viewpoints. Although these models can be considered large and detailed, they are far from being a worst case scenario, even today. Taking future directions for the use of BIMs into account it should be safe to assume that even larger models will be used tomorrow. As such, a more performance efficient solution for real-time visualization of BIMs is required. To further understand the reason behind the performance results and to find the actual bottlenecks we provide a more detailed analysis of both Solibri and Navisworks in the following subsections.

4.1. Detailed analysis of the Solibri Model Viewer

When taking a closer look at the results from the Solibri Model Viewer (without optimization enabled) in relation to model statistics, it becomes clear that the performance on the WORKSTATION system is more affected by object count than by triangle count. In Fig. 8 we illustrate this observation by comparing the rendering performance with the *inverse* object count ($1/\text{object count}$) and *inverse* triangle count ($1/\text{triangle count}$). Here, the different models are arranged based on increasing performance and, as can be seen, the rendering performance is almost inversely proportional to object count for the four different test models. When considering triangle count no such relation can be seen. In fact, if performance were dictated by triangle count, we

would have gotten completely different results. The strong relation between object count and frame rate thus points in the direction that, for all four test models on the high end system, the Solibri Viewer is CPU-bound due to the large number of draw calls. To further confirm this GPU-Z was used to measure the actual GPU load, which showed no more than 30% GPU utilization for any of the models on the WORKSTATION. Due to the large number of individual objects in relation to the total number of triangles (i.e. low triangles-per-object ratio), the time spent on processing draw calls on the CPU is thus greater than the time the GPU needs to draw the actual geometry. Because of this the GPU is mostly idle and the rendering performance becomes entirely dictated by the number of draw calls that has to be processed by the application.

On the LAPTOP system the performance results for the Hotel and Hospital model are equal to that of the high end system. Even with a less powerful GPU these models are still CPU-bound, as evident from an analysis with GPU-Z. Furthermore, with rendering performance being equal to that of the high end system, the results confirm that both systems have similar (single-thread) CPU capacity. However, for the Library and Student House model the Solibri viewer is instead GPU-bound, as evident by lower rendering performance compared to that of the high end system and a GPU utilization at 100% according to GPU-Z. For these two models the triangles-per-object ratio (504 and 664, respectively) are high enough to make the less powerful GPU the bottleneck.

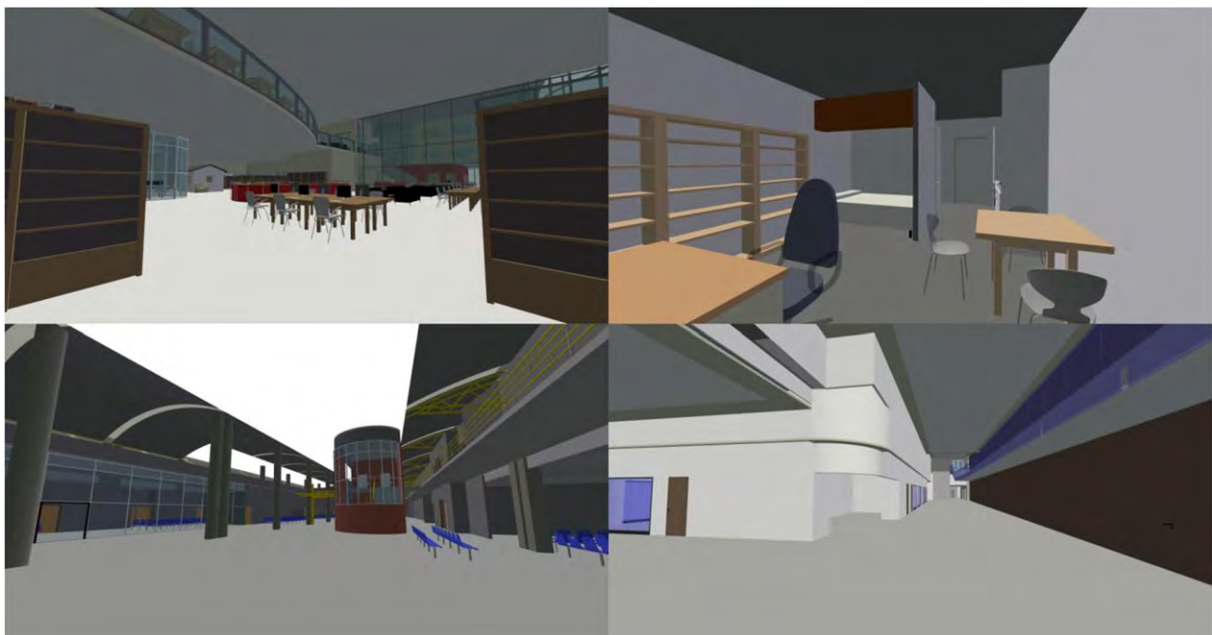


Fig. 6. Interior viewpoints (VP1) for the Library (top-left), Student house (top-right), Hospital (bottom-left) and Hotel (bottom-right), visualized in our prototype BIM viewer.

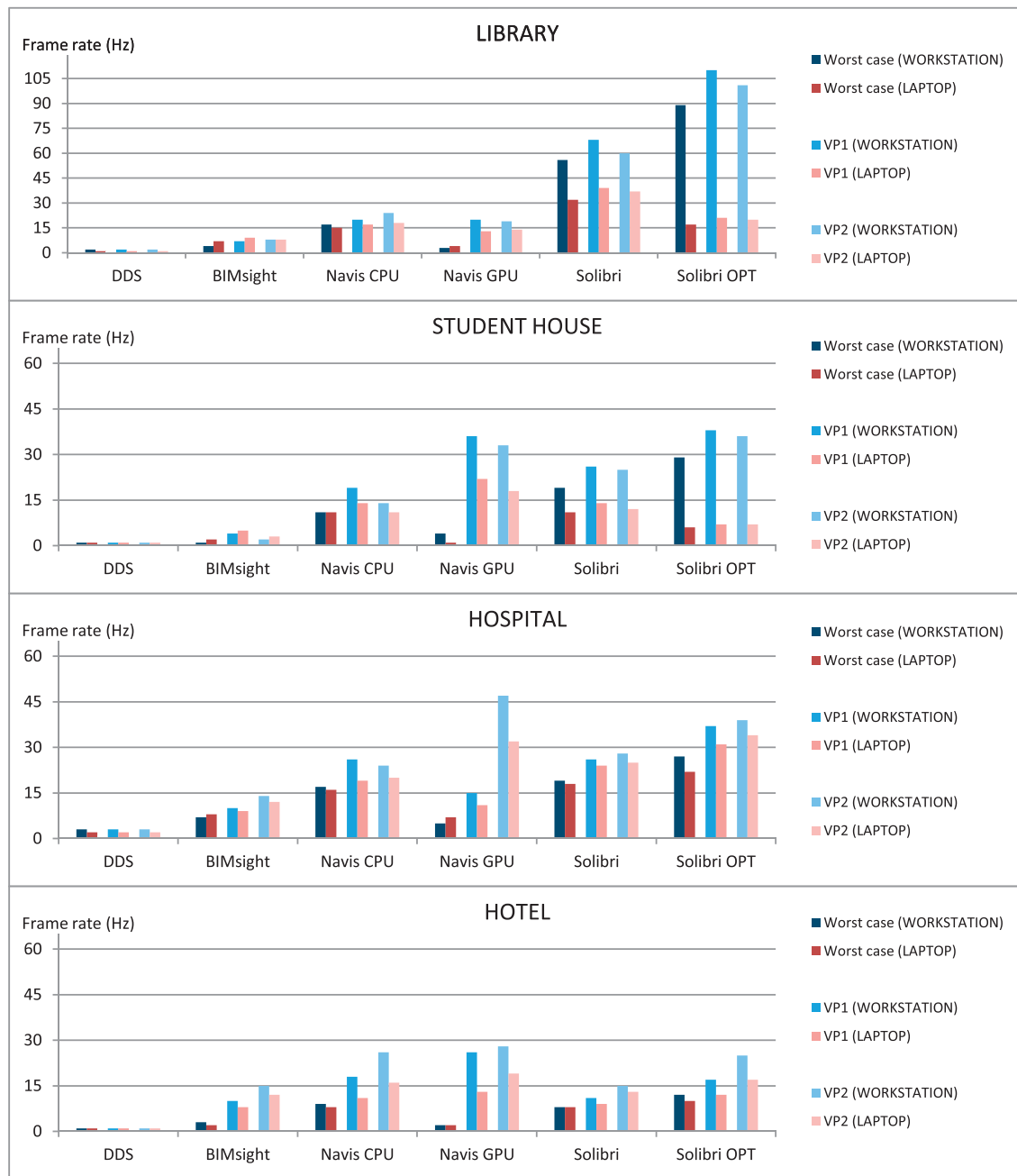


Fig. 7. Rendering performance (in Hz) for the four existing BIM viewers for the (from top to bottom) Library, Student house, Hospital and Hotel model, respectively.

These tests have highlighted the importance of keeping a low draw call count (i.e. number of objects) in order to achieve high rendering performance. At approximately 20,000 draw calls per frame, the corresponding CPU-load alone will limit the performance to around 15

Table 3

Vertex data type and acceleration technique used in the tested viewers (VFC = View frustum culling, DC = Drop culling, OC = Occlusion culling, HAGI = Hardware-accelerated geometry instancing).

BIM viewer	Vertex data	Acceleration technique
Solibri 9.0	VBOs	VFC DC (optional) HAGI (optional)
Navisworks 2015	VBOs	VFC DC (optional) CPU OC (optional) GPU OC (optional)
BIMsight 1.9.1	Vertex arrays	VFC
DDS 8.0	Display lists	VFC DC (optional)
DDS 10.0	Display lists	VFC DC

FPS. Even with an infinitely fast GPU the performance would not change. At the other end of the spectrum, even 7000 draw calls are too much in order to reach an optimal frame rate of 60 Hz. These numbers, of course, assume a system with similar CPU performance as the ones tested. Depending on GPU and triangles-per-draw-call ratio, the performance may then be further reduced. For the GPU on the LAPTOP system the breakpoint when this occurs was found to be somewhere between 172 and 504 triangles per draw call. For the WORKSTATION system, on the other hand, 664 triangles per draw call appeared to be far from the GPU's ultimate capacity. In practice this means that the number of triangles per object could be substantially increased without affecting rendering performance.

With this analysis at hand, it becomes clear why Solibri also offers an optimization feature that takes advantage of hardware-accelerated geometry instancing. As typical building models often contain much replicated geometry, such as identical windows, doors and furniture, it

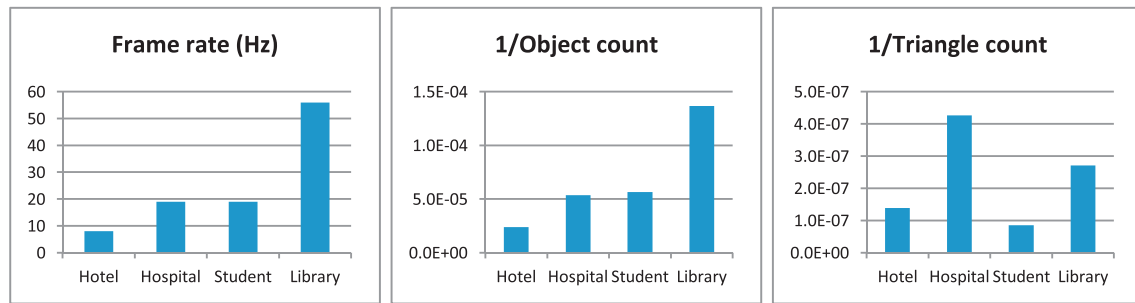


Fig. 8. Worst-case rendering performance (left), inverse object count (middle) and inverse triangle count (right) for the four different test models visualized in Solibri on the WORKSTATION system.

almost seems as a perfect fit in order to reduce the number of draw calls and corresponding CPU-load. Our performance results on the WORKSTATION system verify this for all four models. With optimization enabled, the speed-up is between $1.5\times$ and $1.6\times$ in the worst case scenarios. However, the use of instancing also puts higher burden on the GPU, as evident by a decrease in performance for two of the models (Library and Student house) on the LAPTOP system. As these two models are GPU-bound (as opposed to CPU-bound), the use of instancing does not provide any benefit, but instead only increase the work that has to be done on the GPU, with a performance reduction as result. We can thus conclude that the use of instancing can give a significant performance gain, but only if the application is CPU-bound.

4.2. Detailed analysis of Navisworks

With Solibri mainly implementing pipeline optimizations to provide efficient rendering, the different strategies used could be extracted through a graphics API debugger together with model statistics. However, with Navisworks taking advantage of occlusion culling, a detailed analysis becomes much more difficult. Even in the case when the actual occlusion tests are performed on the GPU (Navis GPU), the logic that decides which tests to perform, as well as how the rendering engine should react to them, is implemented on the CPU. As we are only able to trace calls to the graphics API, a potentially huge part of the respective algorithms therefore becomes “hidden”. For instance, in the case of the CPU occlusion culling system, no information can be extracted regarding how many occlusion tests that are performed or even if the culling is performed during the same frame as the rendering.

Even so, there is some interesting information that we are able to extract when analyzing Navisworks in RenderDoc. In Table 4, the frame rate, number of draw calls performed, as well as triangles rendered is presented for the different occlusion techniques in the worst case and VP1 viewpoints for the Student House model on the WORKSTATION system. In addition, we have added the data when no occlusion culling is enabled (i.e. only frustum culling). For both number of triangles and number of draw calls, we have further separated the data in those that represent “real” geometry (Regular) and those that represent bounding boxes used for the occlusion tests (B-box). As reported by RenderDoc, every bounding box draw call also has a corresponding occlusion

query (i.e. a single occlusion query is issued for every bounding box draw call).

As can be seen in Table 4, the number of triangles rendered in the worst case scenario is almost equal, regardless of occlusion culling method. The frame rate and total number of draw calls, on the other hand, shows huge differences. Although the number of regular draw calls (i.e. that represent the “real” geometry) are fairly equal, the number of bounding box draw calls performed for the GPU occlusion culling technique show a surprisingly high number (13,800). Given that all of the 11,200,000 triangles contained in the Student model can be rendered at an almost equal frame rate without any occlusion culling at all (Frustum), the huge number of occlusion tests (i.e. draw call + occlusion query) thus appears to be the main reason for the low performance. In comparison, the CPU occlusion culling technique stands out as offer equally good culling efficiency, but with far less overhead.

However, for the highly occluded viewpoint (VP1) the GPU occlusion culling technique shows more promising results. Compared to CPU occlusion culling, the culling efficiency is much better (i.e. it does not overestimate the amount of visible geometry), and although the number of occlusion tests are still fairly high the overhead becomes manageable, as evident by higher frame rate.

To summarize our analysis of Navisworks, we have been able to determine that the available occlusion culling systems greatly reduce the amount of geometry that needs to be rendered in any given viewpoint. However, because of an inefficient implementation, the GPU-based approach suffers from significant overhead and mainly becomes useful only in interior or highly occluded areas. In comparison, the CPU-based approach stands out as having, not only lower, but also fairly constant overhead. As such, it becomes more useful in practice.

Through our detailed analysis of Solibri Model Viewer and Navisworks we have gained a better understanding of their inner workings and been able to identify the main bottlenecks in terms of rendering performance. In the case of Solibri we have also been able to construct a number of metrics as to how it will behave for different BIMs on systems similar to the ones tested. Together, these results form a baseline when considering different approaches to improve rendering performance. In the following section we further address

Table 4

Rendering statistics for the Student model rendered in Navisworks for two different viewpoints using three different acceleration techniques on the WORKSTATION system.

Viewpoint	Technique	# of draw calls			# of triangles			Frame rate
		Regular	B-box	Total	Regular	B-box	Total	
Worst case	Frustum	4600	–	4600	11,200,000	–	11,200,000	3
	CPU OC	517	–	517	1,240,800	–	1,240,800	11
	GPU OC	463	13,337	13,800	1,111,200	160,044	1,271,244	4
VP1	Frustum	2477	–	2477	5,944,800	–	5,944,800	6
	CPU OC	263	–	263	631,200	–	631,200	19
	GPU OC	30	2160	2190	72,000	25,920	97,920	36

the current situation by analyzing potential acceleration techniques for our prototype BIM viewer.

5. Development and validation of our prototype BIM viewer

As discussed in Section 2, there are a number of techniques that can be utilized in order to accelerate real-time rendering. These have all strengths and weaknesses and a suitable choice is highly dependent on the type of 3D environment that it should be applied to. For our prototype BIM viewer we opted for occlusion culling as the primary acceleration technique. Given that BIMs typically exhibit a lot of occlusion, this appeared as the most logical choice. Although we initially also considered LOD and geometry batching, these techniques were ruled out for practical reasons. Geometry batching would complicate any form of applications that require visibility to be controlled per-object, such as 4D simulations, and the use of LOD would require the existence of simplified 3D-models for all objects. Furthermore, LOD would only be helpful if the application was GPU-bound. In a similar way, hardware-accelerated geometry instancing would potentially reduce CPU-overhead, but also make culling less efficient. In comparison, occlusion culling becomes a much more versatile approach. As objects can be rejected before sent to the GPU, both number of triangles and draw calls are reduced. Assuming that the process of identifying hidden objects does not introduce additional overhead, occlusion culling therefore becomes beneficial in either CPU-bound or GPU-bound scenarios. Within this category of acceleration techniques several different algorithms exist that are mainly differentiated by whether they require time-consuming offline computations or not [37,48]. For the purpose of review sessions that require the visualization to be initiated on-demand, an offline solution is clearly not the most suitable approach. For our prototype BIM viewer we have therefore chosen to implement what is generally considered current state-of-the-art in terms of online visibility detection – the latest version of the Coherent Hierarchical Culling algorithm [49].

5.1. Coherent hierarchical culling

Occlusion queries is a feature of modern GPUs that lets any application query the number of pixels that will end up on screen when rendering a specific set of geometries. This way, proxy-geometries can be used to test if any occlusion is present before the actual object is rendered. However, the use of occlusion queries introduces latency in the rendering pipeline which may lead to a decrease in performance if used naively. The initial Coherent Hierarchical Culling algorithm (CHC) [50] makes use of temporal and spatial coherence in order to reduce this latency. The state of visibility from the previous frame is used to initiate queries in the current frame (temporal coherence) and by organizing the scene in a hierarchical structure (i.e. bounding volume hierarchy) it is possible to test entire branches of the scene with a single query (spatial coherence). While traversing a scene in a front-to-back order, queries are only issued for previously invisible interior nodes (i.e. groups of objects) and for previously visible leaf nodes (i.e. singular objects) of the hierarchy. The state of visibility for previously visible leaves is only updated for the next frame and they are therefore rendered immediately (without waiting for the query results to return). The state of visibility for previously invisible interior nodes is important for the current frame and they are not further traversed until the query results return. By interleaving the rendering of (previously) visible objects with the issuing of queries, the algorithm reduces idle time due to waiting for queries to return.

Still, for scenes with a low level of occlusion, the initial version of the algorithm can actually decrease performance (compared to only using frustum culling). To address this, CHC++, a revised version of the algorithm was developed [49]. Although the core ideas remain the same, CHC++ introduces several optimizations which make it perform very well even in situations with low occlusion. Most notably, the

improved version addressed the problem of redundant state changes due to the interleaved rendering and querying. Instead of directly querying a node, it is appended to a queue. When this queue reaches a certain size, the rendering state is changed to querying and an occlusion query is issued for each node in the queue.

For our prototype BIM viewer the revised version of the algorithm has been implemented, as outline in the following section.

5.2. Implementation

Our prototype BIM viewer is written in C++ and uses OpenGL 4.3 as graphics API. It is compiled as a 64-bit, single-threaded application. To be able to load BIMs through the IFC file format we initially used IFC Engine DLL [51], which is a C++ toolbox with functionality to access 3D-geometry and object properties expressed in IFC files. However, IFC Engine DLL only provides a single “chunk” of geometry per object. That is, even if a window created in a BIM authoring software typically consists of two different chunks of geometry – one for the frame, and one for the glass – it is only possible to get a single, combined chunk of geometry for the window from IFC Engine DLL. Because of this restriction we instead implemented our own file exporter in Revit through the Custom Export API. That allows us to get the 3D geometry correctly separated by material for all types of objects. Although we extract the geometry directly from Revit's internal database, we have verified that the scene complexity (i.e. number of objects and triangles) is close to identical as when using the IFC Engine DLL for all test models.

Upon scene loading, we construct a bounding volume hierarchy (BVH) according to the Surface Area Heuristics (SAH) [52]. In this hierarchy the leaf nodes represent the individual building components, such as doors, windows and furniture. Although all rendering traversal work primarily utilizes the BVH, we also maintain a “traditional” scene graph based on the transformation logic among the objects in the 3D scene. For each object we construct one VBO per material. That is, a typical window object will be represented by two VBOs, one for the frame geometry and one for the glass geometry. We take advantage of instancing at the memory level – replicated objects of a certain type use the same VBO, but are transformed by a unique transformation matrix. That is, each instance still requires a unique draw call. Transparent geometry is collected during scene traversal and sorted back-to-front before rendered in a final pass with alpha blending enabled.

The implementation of CHC++ is based on the description and accompanying code presented in [53], which is written by the same authors who wrote the original paper. As such, it serves as a valid reference implementation. We use all the proposed “features” of the original algorithm except for the multiqueries optimization. The purpose of multiqueries is to group invisible (but spatially non-coherent) nodes that are likely to remain invisible, and then use a single occlusion query for each such group in order to reduce the total number of queries. However, as we did not observe any obvious performance benefit from this feature we have disabled it during the tests presented in this article.

A major requirement in order to provide an efficient implementation of CHC++ is the ability to render axis-aligned bounding boxes with low overhead. Being primarily based on OpenGL 2.1 (with extensions), the code presented in [53] uses OpenGL “Immediate Mode” (i.e. glBegin/glEnd) for rendering bounding boxes, and vertex arrays for rendering “tight” bounding boxes (i.e. bounding volumes that are represented as an aggregation of its children's bounding boxes at a particular depth in the hierarchy). As these represent deprecated features in OpenGL 4.3, another solution was thus needed for our prototype viewer. Initially we tried to store every bounding box of the hierarchy as a single VBO in GPU memory. However, due to a large number of buffer binds during rendering (i.e. one for each bounding box to test), this affected performance negatively. Instead we take advantage of the fact that any axis-aligned bounding box can be represented as a unit-cube (i.e. a cube with length 1 centered in origo)

with a corresponding translation and scale. Every bounding box in our spatial hierarchy therefore also maintains a 4×4 transformation matrix, which represents the combine translation and scale that is needed in order to form it from a unit-cube. During every *querying phase* (as explained in Section 5.1) a single VBO, representing the geometry of a unit-cube, is used to render every individual bounding box. Before performing each individual draw call, the unique “unit-cube transformation matrix” is submitted to the vertex shader as a uniform variable. In comparison to both the “Immediate Mode” approach as well as storing a single VBO per node of the BVH, the unit-cube approach provided a much more performance efficient solution.

To also enable efficient rendering of tight bounding boxes we further extended the unit-cube concept. In essence, to support tight bounding boxes, multiple unique bounding boxes needs to be rendered. We do this by submitting an array of unit-cube transformation matrices as a vertex shader uniform, followed by an instanced draw call. When performing instanced rendering in OpenGL an internal counter is available in the vertex shader, which advances for each iteration. Based on the value of the internal counter we fetch the correct matrix from the array and transform the unit-cube. That is, by taking advantage of hardware-accelerated instancing, we can render any tight bounding box with a single draw call.

5.3. Performance evaluation of our prototype BIM viewer

For the performance evaluation of our prototype BIM viewer we have used the same test models and interior view points as described

in Section 3. As we utilize an efficient occlusion culling algorithm the performance is highly dependent on the view point we chose. We therefore represent our worst case scenarios as the lowest frame rate that we have encountered during several interactive navigation sequences within each one of the four models. In Fig. 9 we present the frame rates recorded for the worst case scenario, viewpoint one (VP1) and viewpoint two (VP2) for all models on both systems. To better illustrate the performance gain offered by our prototype we have also added the results from the Solibri viewer (the results from the optimized version for the WORKSTATION and from the default version on the LAPTOP). As can be seen the speed-up in the worst case scenarios is quite significant, ranging between $2.2\times$ – $5.7\times$ and $1.7\times$ – $5\times$ for the WORKSTATION and LAPTOP, respectively. That is, even for the Library model, which doesn't exhibit as much occlusion, our prototype viewer is close to offer a doubling of the rendering performance compared to Solibri. In this context it is also important to note that Solibri does not support multiple materials per object. If a similar strategy had been used for our prototype viewer, the natural reduction of draw calls could have made the difference in performance even bigger.

More important, however, is the actual performance numbers. When considering the worst case scenarios our viewer clearly passes or is just below our optimal level of interactivity of 60 Hz for three of the models on both systems. Although our prototype is not able to provide equally high figures for the Hotel model, it can still guarantee a worst case performance of at least 30 Hz on either system (41 Hz on the WORKSTATION), which equals our satisfactory requirement.

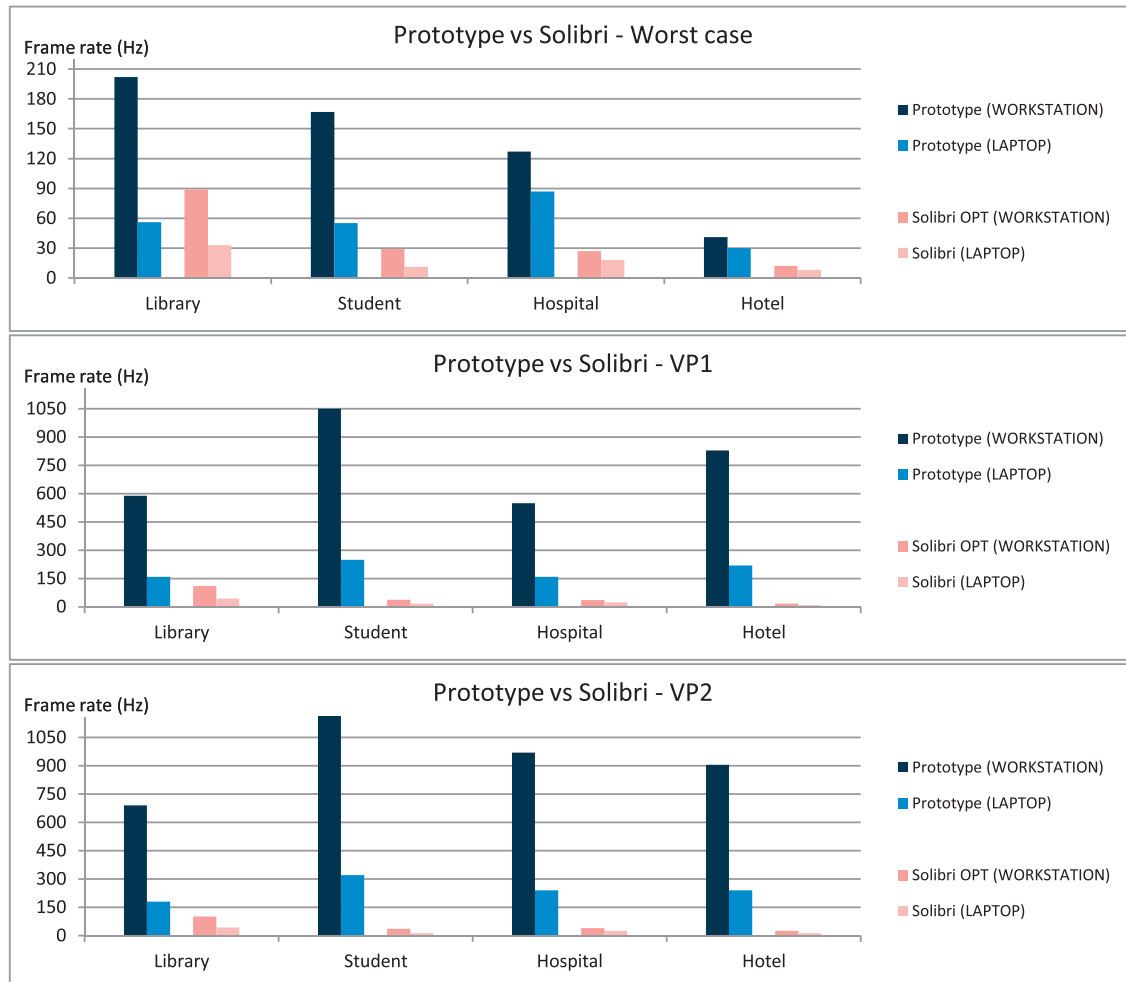


Fig. 9. Comparison of rendering performance between our prototype viewer and Solibri for the worst-case scenario (top), VP1 (middle) and VP2 (bottom) for the four test models on both systems.

For the interior viewpoints (VP1 and VP2 in Fig. 9) the performance offered by our prototype viewer is even more impressive. Here, the occlusion culling system can take advantage of the fact that only small subsets of the complete models are actually visible. For any of the interior viewpoints in either model the frame rates are consistently above 450 and 150 Hz for the WORKSTATION and LAPTOP, respectively. Consequently, our prototype viewer is able to guarantee very smooth and responsive navigation inside all of the four models on both systems. In comparison to Solibri these numbers corresponds to speed-ups between $5.3\times$ – $48\times$ and $3.5\times$ – $26\times$ for the WORKSTATION and LAPTOP, respectively.

In Fig. 10, we provide additional information about the rendering performance obtained with our prototype viewer. Here, the frame rates are presented for a pre-defined camera path for each one of the four test models. During these sequences the camera is following a path around each building while the view direction is oriented towards the center of the building. As each model is completely visible throughout the whole navigation sequence, albeit from different directions, it serves as a representative worst case scenario. In addition to give a clear picture of the performance characteristics provided by our prototype viewer, these graphs tell us that a models triangles-per-object ratio affect performance even when occlusion culling is utilized, as evident by much greater performance difference between the LAPTOP and WORKSTATION for the Library and Student house models. However, in comparison to our analysis of Solibri, a further discussion regarding CPU- or GPU-boundedness on the different systems becomes a bit more complex. This is because with GPU-based occlusion queries, rendering is no longer only a unidirectional process (i.e. the CPU is feeding the GPU with draw commands), but instead also incorporates elements of bidirectional communication (i.e. the CPU is issuing an occlusion query and then later requests the result of it back from the GPU). That is, not only might the GPU be waiting for the CPU, but also the other way around. In essence, this means that it is difficult to get 100% GPU utilization. To make things even more complex, final

performance and level of GPU utilization doesn't necessarily go hand in hand. As only a subset of a scenes triangles might have to be rendered (due to the culling), performance can still be high even if the GPU isn't fully utilized. That is, it might be "worth" to be CPU- or wait-bound, performance-wise.

Nevertheless, based on the graphs in Fig. 10, we can conclude that the Library and Student models are primarily GPU-bound on the Laptop system. This is evident by the huge differences compared to the results on the workstation system, as well as 90% GPU load reported by GPU-Z. On the Hospital and Hotel models we also see that a faster GPU will provide better performance. However, especially in the case of the Hotel model, the difference compared to the workstation system is not by as much as for the other two models. For the Hotel model we can therefore conclude that it is primarily GPU-bound, but very close to the point of instead being CPU-bound because of a large number of draw calls (due to many objects being, in fact, visible).

The models on the workstation system, on the other hand, appears to be primarily CPU-bound, as evident by a GPU utilization level of around 60% for the Library and Student house, and only 30% for the Hospital and Hotel models, as reported by GPU-Z. However, as mentioned previously, it makes more sense to look at the GPU utilization level in relation to frame rate instead of focusing on whether the application is CPU-bound or not. More specifically, for viewpoints where we have a low GPU utilization level at the same time as performance is low, there is potential to increase performance by utilizing the GPU better.

Together, the results from the performance tests of our prototype BIM viewer confirm our assumption that occlusion culling is a suitable acceleration technique for real-time visualization of typical building models. To some degree, the benefit of using this type of acceleration technique for highly occluded interior viewpoints was already demonstrated by our performance tests with Navisworks. However, by taking advantage of a highly efficient algorithm, such as the CHC++, we have shown that the utilization of occlusion culling does not need to

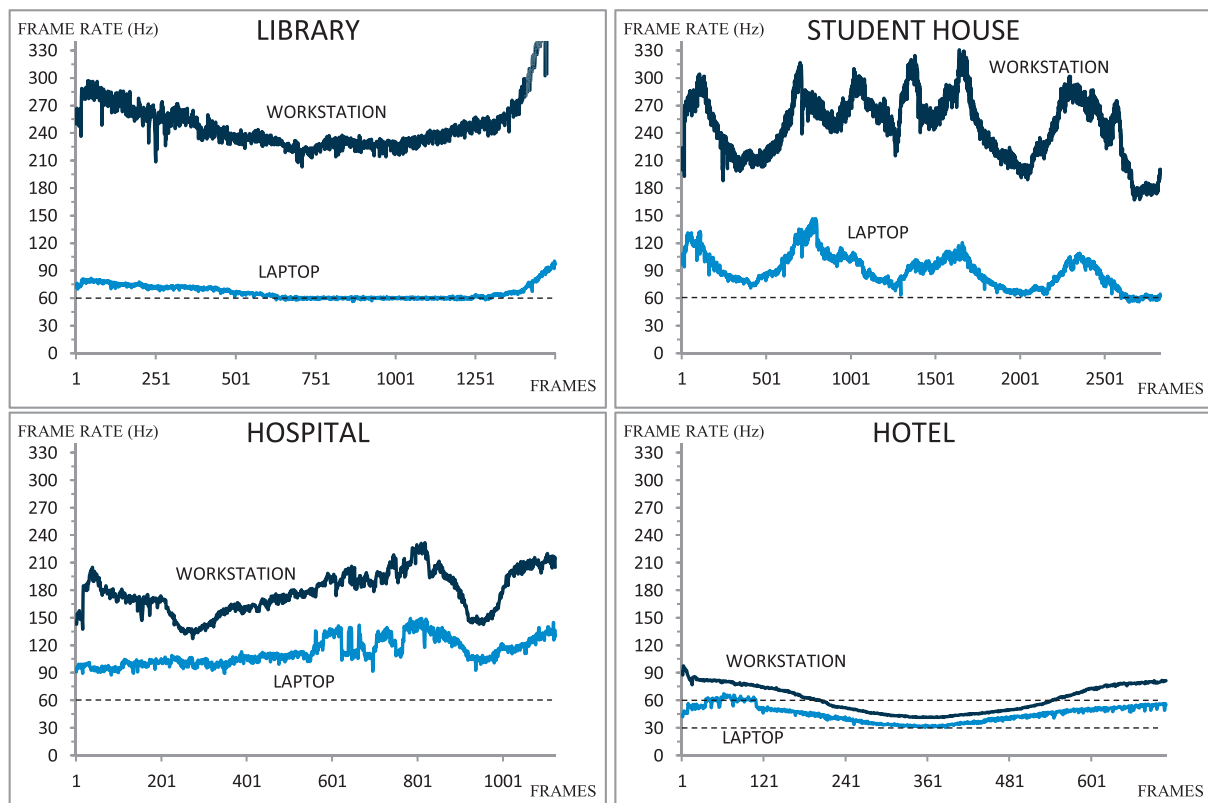


Fig. 10. Rendering performance (Hz) of our prototype viewer for the pre-defined animation sequence around each model on both systems.

suffer from additional overhead and therefore becomes suitable also for exterior viewpoints.

6. Discussion and conclusions

In this paper we have highlighted and addressed the complexity and challenges involved in visualizing large BIMs. The contribution of the paper is twofold: (a) an analysis of commonly used BIM viewers in terms of real-time rendering performance and (b) the development and validation of a prototype BIM viewer that is able to handle large and detailed building models. Below these two contributions are discussed in more detail.

Regarding the analysis of existing BIM viewers, an obvious observation from our tests was the huge difference in rendering performance. Even when omitting the extremely poor results from the DDS CAD Viewer, the difference between the fastest and the slowest viewer range between $1.6\times$ and $22\times$ for any of the viewpoints on either system. These findings thus highlight the need to add *software capacity* as a distinct variable to the problem space currently surrounding the topic of BIM visualization. While previous studies mainly have attributed visualization-related problems to either *model complexity* or lack of sufficient *hardware* our tests clearly show that a viewer's ability to effectively utilize available hardware can become a far more important factor. For the same models, different viewers will produce user experiences that range from totally unacceptable to satisfactory as far as rendering performance goes. A selection of BIM-viewer will therefore directly affect how well design review sessions or client presentations can be performed. In addition, our in-depth analysis shows that the very concept of *model complexity* and *hardware* also needs to be revised. Although it might be reasonable to assume that the performance offered by any of the viewers would mainly be affected by a model's triangle count and the computational power of the GPU, our tests instead show that object count and CPU performance is equally important to consider. In fact, in the majority of test, the CPU, and not the GPU, turned out to be the bottleneck. Because of a strong interdependence between a model's triangles-per-object ratio, CPU- and GPU performance, seemingly different systems may thus behave surprisingly similar depending on model characteristics. Although we initially expected our WORKSTATION system to offer much more performance than the LAPTOP, our results from the Solibri viewer were near-identical on both systems for two of the models. Due to a low triangles-per-object ratio these models became CPU-bound on both systems and with current graphics APIs being primarily single-threaded, the added number of CPU cores on the WORKSTATION system did not make any difference. Although the optimization feature (instancing) in Solibri was able to reduce the CPU-dependency to a certain degree, the performance results were still very close on both systems for two of the models. Similar behavior also became apparent from our tests with Navisworks, as the worst case performance results were very close to being identical for all models on both systems with CPU occlusion culling. Consequently, regardless of viewer, it becomes difficult to discuss either *model complexity* or *hardware* without knowing the details of their respective counterparts.

The test from Navisworks further revealed that different occlusion culling strategies can have vastly different effects. As such, it becomes equally difficult to discuss and compare different acceleration techniques based on what category they belong to. In every situation, regardless of strategy, the details must be known.

However, despite huge differences in rendering performance, none of our tested viewers were able to provide a sufficiently high frame rate for all models. In fact, even Solibri, who turned out to offer the best performance, could only guarantee a satisfactory level of 30 Hz for one of the tested models. For the other three models the frame rates in the worst-case scenarios were either below or just above our minimum requirement of 15 Hz. Even with optimization enabled, these models could not be guaranteed to reach 30 Hz. In this context

it is of course important to acknowledge that both Navisworks and Solibri implement functionality to guarantee a certain frame rate during navigation. Nevertheless, as realized through the rejection of objects that should be visible, this approach severely impacts both accuracy and visual fidelity (e.g. see Figs. 3 and 4). Hence, our analysis clearly shows that all of the existing BIM viewers share limitations in their ability to provide accurate, real-time visualizations of large and complex BIMs. As these models tend to become even larger and more detailed at the same time as increasingly faster GPUs currently fail to solve the problem, there should be no doubt that these limitations need to be addressed.

Our second contribution is the development of a prototype BIM viewer that solves these particular limitations. Our prototype BIM viewer was able to provide sufficiently high frame rates for all models on both systems without the need to use non-conservative culling strategies (i.e. contribution or drop culling). By taking advantage of a state-of-the-art occlusion culling algorithm, the rendering efforts are restricted to visible objects only, with an overall increase in performance as a result. Compared to Solibri, the prototype viewer provides a $1.7\times$ – $5.7\times$ speed-up in the worst case scenarios and $3.5\times$ – $48\times$ for the internal viewpoints. Our tests have thus demonstrated the efficiency of the CHC++ algorithm and confirmed our hypothesis that occlusion culling is a suitable acceleration technique for typical building models. Because the speed-up comes from rejecting hidden objects we also expect it to scale well when adding structural and MEP objects to the models. As these objects typically become obscured by other objects the performance penalty should be marginal.

Future work will look more into the use of occlusion culling in combination with other solutions. We noticed from our HOTEL model that occlusion culling efficiently rejects hidden objects, but still some exterior viewpoints provided by the HOTEL model contained a large number of visible objects that had to be rendered. In such scenarios it becomes important to reduce the cost of draw calls in order to reach an optimal level of 60 Hz on systems like the WORKSTATION. We believe that this could be addressed by taking advantage of more modern features from the graphics APIs, such as hardware-accelerated instancing. Recent updates to both Direct3D and OpenGL have exposed much functionality in order to reduce driver overhead and draw call cost. Exploring these features represents an obvious direction for future research.

Ultimately, however, more scalable approaches, such as LOD, also need to be considered as indicated by our results from the LAPTOP system. For very large and detailed building models or in situations when several of them are to be visualized together, the sheer amount of geometry that needs to be rendered on screen is likely to go beyond the capacity of any GPU, even when occlusion culling and strategies to reduce the cost of draw calls is applied. In addition, all of the geometry and texture data may not fit into video memory (GPU RAM) or even central memory (CPU RAM). In order to address such scenarios, both LOD and data streaming techniques need to be further investigated.

References

- [1] C. Eastman, P. Teicholz, R. Sacks, K. Liston, *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors*, John Wiley & Sons, New Jersey, 2008.
- [2] J. Steel, R. Drogemüller, B. Toth, Model interoperability in building information modelling, *Softw. Syst. Model.* 11 (2012) 99–109.
- [3] P. Yuan, M. Green, R. Lau, A framework for performance evaluation of real-time rendering algorithms in virtual reality, *VRST '97 Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, ACM, New York, 1997, pp. 51–58.
- [4] T. Lehtinen, Increasing Integration in Construction Projects: A Case Study on a PPP Project Adopting BIM, *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2012*, CRC Press, 2012. 439–446.
- [5] K. Svidt, P. Christiansson, Requirements on 3D Building Information Models and Electronic Communication: Experiences from an Architectural Competition, *CIB W78 25th International Conference on Information Technology: Improving the Management of Construction Projects Through IT Adoption*, Chile, 2008, pp. 231–238.

- [6] S. Paavola, H. Kerosuo, T. Mäki, J. Korpela, R. Miettinen, BIM Technologies and Collaboration in a Life-cycle Project, *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2012*, CRC Press, 2012, 855–862.
- [7] U. Plesner, M. Horst, Before stabilization: communication and non-standardization of 3D digital models in the building industry, *Inf. Commun. Soc.* 16 (2013) 1115–1138.
- [8] J. Plume, J. Mitchell, Collaborative design using a shared IFC building model—learning from experience, *Autom. Constr.* 16 (2007) 28–36.
- [9] R. Davies, C. Harty, Implementing 'site BIM': a case study of ICT innovation on a large hospital project, *Autom. Constr.* 30 (2013) 15–24.
- [10] C. Dubler, J. Messner, C. Anumba, Using lean theory to identify waste associated with information exchanges on a building project, *Construction Research Congress 2010: Innovation for Reshaping Construction Practice*, ASCE, 2010, pp. 708–716.
- [11] Z. Shen, L. Jiang, An augmented 3D iPad mobile application for communication, collaboration, and learning (CCL) of building MEP systems, *J. Comput. Civ. Eng. ASCE* (2012) 204–212.
- [12] S. Yoon, E. Gobbetti, D. Kasik, D. Manocha, Real-time massive model rendering, *Synth. Lect. Comp. Graph. Animat.* 2 (2008) 1–122.
- [13] T. Chuang, B. Lee, I. Wu, Applying cloud computing technology to BIM visualization and manipulation, In *28th International Symposium on Automation and Robotics in Construction*, Korea, 2011, pp. 144–149.
- [14] B.D. Larsen, Accessing large 3D BIMs from mobile devices, *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2012*, CRC Press, 2012, 505–508.
- [15] W. Yan, C. Culp, R. Graf, Integrating BIM and gaming for real-time interactive architectural visualization, *Autom. Constr.* 20 (2011) 446–458.
- [16] S. Kumar, M. Hedrick, C. Wiacek, J. Messner, Developing an experienced-based design review application for healthcare facilities using a 3D game engine, *ITcon* 16 (2011) 85–104.
- [17] M. Johansson, M. Roupé, Efficient Real-time Rendering of Building Information Models, *Proceedings of the 2009 International Conference on Computer Graphics and Virtual Reality (CGVR09)*, Las Vegas, 2009, pp. 97–103.
- [18] M. Reddy, The Effects of Low Frame Rate on a Measure for User Performance in Virtual Environments, Technical Report ECS-CSG-36-97, Department of Computer Science, University of Edinburgh, 1997.
- [19] K. Claypool, M. Claypool, On frame rate and player performance in first person shooter games, *Multimedia Systems* 13 (2007) 3–17.
- [20] W. Barfield, K. Baird, O. Bjorneseth, Presence in virtual environments as a function of type of input device and display update rate, *Displays* 19 (2) (1998) 91–98.
- [21] D.J. Kasik, J.J. Troy, S.R. Amorosi, M.O. Murray, S.N. Swamy, Evaluating graphics displays for complex 3D models, *Comput. Graph. Appl.* 22 (3) (2002) 56–64.
- [22] M.F. Shiratuddin, D. Fletcher, Development of Southern Miss's Innovation and Commercialization Park Virtual Reality environment, *Proceedings of the 6th International Conference on Construction Applications of Virtual Reality*, Orlando, Florida, 2006.
- [23] A. Herwig, P. Paar, Game engines: tools for landscape visualization and planning, *Trends in GIS and Virtualization in Environmental Planning and Design*, 2002, pp. 161–172.
- [24] R. Göttig, J. Newton, S. Kaufmann, A Comparison of 3D Visualization Technologies and their User Interfaces with Data Specific to Architecture, *Recent Advances in Design and Decision Support Systems in Architecture and Urban Planning*, Springer, Netherlands, 2005, pp. 99–111.
- [25] C. Rubino, J. Power, Level design optimization guidelines for game artists using the epic games: unreal editor and unreal engine 2, *Comput. Entertain. (CIE)* 6 (4) (2008) 55.
- [26] T. Akenine-Möller, E. Haines, N. Hoffman, *Real-Time Rendering*, 3rd edition A. K. Peters, 2008.
- [27] M. Johansson, M. Roupé, *Real-Time Rendering of Large Building Information Models, CAADRIA 2012 – Beyond Codes & Pixels*, 2012, pp. 647–656.
- [28] M. Wloka, Batch, Batch, Batch: What does it really mean? Presentation at Game Developers Conference 2003, San Jose, California, 2003.
- [29] M.J. Kilgard, Modern OpenGL usage: using vertex buffer objects well, *SIGGRAPH Asia'08 courses*, 2008 (13:1–13:31).
- [30] P. Castelló, J.F. Ramos, M. Chover, A comparative study of acceleration techniques for geometric visualization, *Computational Science–ICCS 2005*, Springer, Berlin Heidelberg, 2005, pp. 240–247.
- [31] S. Hillaire, Improving performance by reducing calls to the driver, *OpenGL Insights*, CRC Press, 2012, 353–363.
- [32] B. Dudash, *Animated Crowd Rendering*, GPU Gems 3, Addison-Wesley, 2007, 39–52.
- [33] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, R. Huebner, *Level of Detail for 3D Graphics*, Morgan Kaufmann, 2002.
- [34] J. Terrace, E. Cheslack-Postava, P. Levis, M.J. Freedman, Unsupervised Conversion of 3D Models for Interactive Metaverses, *2012 IEEE International Conference on Multimedia and Expo (ICME)*, 2012, pp. 902–907.
- [35] I. Leitão, M.B. Carmo, Block and Quadtree based Simplification in Tiled Blocks Terrain Algorithms, *GRAPP 2009: International Conference on Computer Graphics Theory and Applications*, 2009, pp. 205–210.
- [36] J. Cohen, D. Manocha, *Model Simplification*, Visualization Handbook, Elsevier, 2005, 393–411.
- [37] D. Cohen-Or, Y.L. Chrysanthou, C.T. Silva, F. Durand, A survey of visibility for walkthrough applications, *IEEE Trans. Vis. Comput. Graph.* 9 (3) (2003) 412–431.
- [38] J. Clark, Hierarchical geometric models for visible surface algorithm, *Commun. ACM* 19 (10) (1976) 547–554.
- [39] Z. Constantinescu, Levels of detail: an overview, *Nonlinear Anal. Modell. Control* 5 (2000) 39–52.
- [40] J.L. Posada-Velasquez, A methodology for the semantic visualization of industrial plant CAD models for virtual reality walkthroughs (Doctoral Dissertation) TU Darmstadt, 2006.
- [41] BuildingSMART, IFC2x Edition 3 TC1, Available from: <http://www.buildingsmart-tech.org/specifications/ifc-releases/ifc2x3-tc1-release/summary2007>.
- [42] ExCode Dxtory, Available from: <http://excode.com/dxtory-features-en.html> 2013.
- [43] Graphic Remedy, gDEBugger Available from: <http://www.gremedy.com> 2010.
- [44] Crytek, RenderDoc, Available from: <http://cryengine.com/renderdoc> 2014.
- [45] GPU-Z, Techpowerup, Available from: <http://www.techpowerup.com/gpuz/> 2013.
- [46] wPrime Systems, P.I. Super, Available from: <http://www.superpi.net/> 2013.
- [47] D. Shreiner, *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1*, 7th Edition Addison-Wesley Professional, 2009.
- [48] J. Bittner, P. Wonka, Visibility in computer graphics, *Environ. Plan. B Plan. Des.* 30 (2003) 729–755.
- [49] O. Mattausch, J. Bittner, M. Wimmer, CHC++: coherent hierarchical culling revisited, *Comput. Graph. Forum* 27 (2) (2008) 221–230.
- [50] J. Bittner, M. Wimmer, H. Piringer, W. Purgathofer, Coherent hierarchical culling: hardware occlusion queries made useful, *Comput. Graph. Forum* 23 (3) (2004) 615–624.
- [51] RDF, DLL IFC Engine, Available from: <http://rdf.bg/ifc-engine-dll.php> 2014.
- [52] J.D. Macdonald, K.S. Booth, Heuristics for ray tracing using space subdivision, *Vis. Comput.* 6 (6) (1990) 153–165.
- [53] J. Bittner, O. Mattausch, M. Wimmer, *Game-Engine-Friendly Occlusion Culling, ShaderX7: Advanced Rendering Techniques*, Charles River Media, 2009, pp. 637–653.

Paper III

Integrating Occlusion Culling and Hardware Instancing for Efficient Real-Time Rendering of Building Information Models

Mikael Johansson

Chalmers University of Technology, Gothenburg, Sweden
jomi@chalmers.se

Keywords: Real time rendering, Occlusion culling, Hardware instancing, BIM

Abstract: This paper presents an efficient approach for integrating occlusion culling and hardware instancing. The work is primarily targeted at Building Information Models (BIM), which typically share characteristics addressed by these two acceleration techniques separately – high level of occlusion and frequent reuse of building components. Together, these two acceleration techniques complement each other and allows large and complex BIMs to be rendered in real-time. Specifically, the proposed method takes advantage of temporal coherence and uses a lightweight data transfer strategy to provide an efficient hardware instancing implementation. Compared to only using occlusion culling, additional speedups of 1.25x-1.7x is achieved for rendering large BIMs received from real-world projects. These speedups are measured in viewpoints that represents the worst case scenarios in terms of rendering performance when only occlusion culling is utilized.

1 INTRODUCTION

With the creation of Building Information Models (BIM), the content produced by architects and designers has evolved from traditional 2D-drawings to semantically-rich, object-oriented 3D-models. With all of the data available in 3D, this concept further facilitates the use of real-time visualizations in various contexts. However, as primarily created to describe a complete building in detail, many 3D datasets extracted from BIMs provides a challenge to manage in real-time without additional acceleration techniques (Steel et al., 2012).

In this context, occlusion culling has been shown to provide a suitable option (Johansson and Roupé, 2012). Given that typical building models naturally exhibit a lot of occlusion this is an efficient approach to increase rendering performance for many viewpoints. Still, for viewpoints where many objects are, in fact, visible, occlusion culling alone may not always be able to provide sufficiently high frame rates. Common examples include exterior views of whole building facades where the sheer number of draw calls, and hence CPU burden, easily becomes the limiting factor in terms of rendering performance (Wloka, 2003).

Another characteristic of typical BIMs is the frequent reuse of identical building components. As similarity tends to reduce design, production and maintenance costs, use of multiple identical components, such as doors and windows, is common in any building (Sacks et al., 2004). For viewpoints where many objects are visible it is therefore a high probability that many of these objects are identical, albeit placed at different locations. One way to take advantage of this is to utilize the hardware instancing functionality of modern GPUs. With hardware instancing it is possible to render multiple copies of the same geometry with a single draw call, thereby reducing CPU-burden for scenes with much repetition. However, even if the reduction of draw calls improves performance in CPU-limited scenarios, the GPU still has to process all the instantiated geometry. As such, culling of invisible instances is still important to reduce overall workload

This paper presents a method to integrate occlusion culling and hardware instancing in order to provide efficient real-time rendering of large BIMs. By using an efficient occlusion culling algorithm hardware instancing can be restricted to visible replicated objects only. For viewpoints when many objects are visible, hardware instancing complements the occlusion culling by providing an

efficient rendering path for visible replicated objects. The key component to realize this is an efficient dynamic hardware instancing implementation that takes advantage of temporal coherence and uses a lightweight data transfer strategy.

2 RELATED WORK

2.1 Occlusion Culling

With occlusion culling the aim is to identify and reject occluded regions of 3D scenes in order to improve rendering performance. Within this category of acceleration techniques a vast amount of research has been conducted and for a general overview interested readers are referred to the surveys provided by (Cohen-Or et al, 2003) and (Bittner and Wonka, 2003). In essence, available algorithms can be classified according to whether they require time-consuming offline computations or not.

When considering online approaches, which require no pre-computations, the support for hardware accelerated occlusion queries has provided a simple mechanism to detect visibility. With hardware occlusion queries the GPU returns the number of pixels that passes the depth test when rasterizing a given object. This way, proxy geometries can be used to detect occlusion before the actual object is rendered. However, due to the delayed processing in the graphics pipeline the result of the query is not immediately available on the CPU which makes an efficient implementation more complex. This problem was addressed with the Coherent Hierarchical Culling (CHC) algorithm, which exploits spatial and temporal coherence in order to reduce latency and overhead of the queries (Bittner et al., 2004). However, although the CHC algorithm works well in highly occluded scenes, wasted queries and unnecessary state changes makes it less reliable for viewpoints when many objects are visible. In order to reduce the number of wasted queries, (Guthe et al., 2006) proposed a method, called Near Optimal Hierarchical Culling (NOHC), based on a statistical model for occlusion probability and a hardware calibration step. Assuming proper hardware calibration their approach always performs better than view-frustum culling. In (Mattausch et al., 2008) an improved version of the CHC algorithm, called CHC++ was presented. Although the core ideas of the algorithm remain the same, the additional components introduced by CHC++ provide a significant improvement in rendering

speed compared to both NOHC and CHC. Mainly, this was achieved by introducing batching of queries as a means to reduce costly state changes.

When considering the case of rendering complex BIMs, the efficiency of the CHC++ algorithm has been recently demonstrated (Johansson and Roupé, 2012). Compared to view-frustum culling, CHC++ provided significant speedups for a number of fairly large BIMs during both interior and exterior viewpoints.

As an alternative to occlusion queries, (Hill and Collin, 2011) recently proposed a modern variant of the hierarchical z-buffer (Green et al., 1993), where all visibility tests are performed on the GPU. The state of visibility is then read-back to the CPU, so that un-occluded objects can be rendered in a single stage. However, although reported as being successfully used in recent computer games, the performance implications are still largely unknown for general 3D models. In addition, this approach requires a set of good occluders in order to initiate the z-buffer.

A problem common to practically all visibility culling methods is that of granularity. On the one hand, in order to maximize culling efficiency, we ideally want to perform visibility determination on the level of granularity provided by the individual objects contained in a 3D scene. On the other hand, for viewpoints with many visible objects, this is not an optimal organization of the 3D scene, considering the aim of keeping a low draw call count (Wloka, 2003). In this case, reduction of draw calls can often be addressed by geometry batching, where spatially coherent objects (with similar material properties) are combined into larger ones during a pre-process (Buchholz and Döllner, 2005). However, even if this process enhance rendering performance for certain viewpoints, it potentially reduces culling efficiency, and hence, performance, for other viewpoints. Besides requiring a dedicated pre-process, geometry batching also complicates the use of additional acceleration techniques applied per-object, such as level-of-detail (LOD).

The proposed method addresses this situation by performing *implicit* geometry batching. By taking advantage of hardware instancing capabilities of modern GPUs, culling can be performed at fine granularity at the same time as the amount of draw calls is reduced for viewpoints with many visible objects.

2.2 Hardware Instancing

For 3D scenes where many individual objects have to be rendered it is not uncommon that the large number of draw calls (and related state changes and buffer binds) becomes the limiting factor in terms of performance (Wloka, 2003). Given a large amount of replicated geometry, hardware instancing is one way to address this problem. The idea behind this concept is to use a single draw call when rendering multiple copies of the same geometry. By using a previously uploaded buffer or texture object containing per-instance data (i.e. transformation matrix), each instance can then be transformed to its correct location on the GPU. Typical applications that can benefit from hardware instancing include rendering of crowds and vegetation, which usually require a large number of instances at the same time as there exists much repetition. In (Park et al., 2009), (Dudash, 2007), and later (Ramos et al., 2012), examples on how to render several thousands of animated characters in real-time with the use of hardware instancing is presented. Recently, (Bao et al., 2012) presented a GPU-driven framework for rendering large forests. Hardware instancing, together with level-of-detail selection on the GPU, allow them to render several thousands of detailed trees, with shadows, in real-time.

However, even if hardware instancing reduces the number of draw calls, and hence CPU-burden, the GPU still have to process all the geometry that is instantiated. Without any type of visibility culling, this may lead to unnecessary high GPU-burden for 3D scenes with many instances. In order to limit the number of instances, (Park et al., 2009) and (Bao et al., 2012) perform view-frustum culling on the GPU.

Still, for highly occluded scenes, such as Building Information Models, view-frustum culling only allows a subset of the invisible geometry to be rejected. The proposed method addresses this problem by an efficient dynamic hardware instancing implementation. By taking advantage of temporal coherence together with a lightweight data transfer approach, occlusion culling can be performed at object level at the same time as replicated geometry is efficiently rendered using hardware instancing.

3 THE IFC BUILDING MODEL

For the majority of BIM authoring tools the underlying data-model closely resembles that of the Industry Foundation Classes (Eastman et al., 2011).

Instead of pure geometrical entities, this scheme represents a building or facility in terms of its individual building components, such as walls, doors, windows and floors. For each component a visual representation is then provided in the form of one or several geometrical entities (i.e. triangular meshes). When considering instancing, this concept is performed at the building component level. As an example, all instances of a specific window type will be considered a unique building component but share the same visual representation. For the implementation and tests presented in this paper, no additional processing of the input 3D-data has been performed except organizing it in a bounding volume hierarchy. In this hierarchy, leaf nodes represent the individual building components. As such, culling is performed at a granularity corresponding to the individual building components. However, hardware instancing is performed at a level corresponding to the geometrical entities that represent each component.

For the rest of this paper replicated components that are suitable for hardware instancing are referred to as *instanceable*. The specific geometry being instanced is referred to as the geometry *reference*.

4 ALGORITHM OUTLINE

The proposed method consists of three main steps. In order to give an overview of the algorithm all three steps are briefly discussed below.

Determine visible instances Using an efficient occlusion culling system, we inherently have access to the set of potentially visible objects in a certain frame. Based on the assumption that hardware instancing is the most efficient way to render multiple copies of the same geometry, this set is searched for replicated components. These objects are then scheduled for rendering with hardware instancing in the *next* frame.

Upload required data to GPU Given a set of visible objects to be rendered using hardware instancing, per-instance data need to be uploaded to the GPU. To reduce per-frame data transfer, an indexed approach is used: During scene loading, transformation matrices for all potential instances are uploaded to the GPU. During rendering, only a single index per instance needs to be transferred in order to locate the corresponding transformation matrix in GPU memory. Thus, at the end of each frame, data in the form of indices is uploaded to the GPU for processing during the next frame.

Render using hardware instancing At the beginning of each frame opaque instances collected during the previous frame are rendered using hardware instancing. However, for semi-transparent geometry hardware instancing introduces complexities. As correct depth ordering no longer can be maintained, an order-independent transparency rendering technique (Everitt, 2001) is needed to support hardware instancing of semi-transparent geometry.

5 INTEGRATING OCCLUSION CULLING AND HARDWARE INSTANCING

As outlined, the general idea behind the proposed method is to dynamically select candidates for hardware instancing, based on visibility knowledge provided by the occlusion culling system. For the purpose of this, any efficient occlusion culling algorithm can be used as long as it provides visibility classification for all objects in a scene. For the implementation and tests presented in this paper the latest version of the Coherent Hierarchical Culling algorithm, CHC++ has been used. This choice is based on the simplicity of the algorithm and the fact that it has already been proven to work well when applied to large Building Information Models. As such, it provides a good basis for further enhancements. In the followings subsections the details of the algorithm is presented, starting with a review of the hardware instancing API together with important aspects of the CHC++ algorithm.

5.1 Hardware Instancing API

Taking OpenGL as an example, the instancing API extends the conventional draw call by exposing the option to specify the number of times a particular batch of geometry should be rendered. In the vertex shader an internal counter (`gl_InstanceID`) is then accessible which advances for each iteration. Using the internal counter as an index, the per-instance transformation matrix can then be sourced from any type of previously uploaded array, texture or buffer object. However, the arrangement of per-instance data must reflect the fact that the internal counter advances with a fixed step. In order to render a specific set of instances with a single draw call, the per-instance data must be arranged sequentially.

5.2 CHC++

The original CHC algorithm takes advantage of spatial and temporal coherence in order to provide efficient scheduling of hardware occlusion queries. The state of visibility from the previous frame is used to initiate queries in the current frame (temporal coherence) and by organizing the scene in a hierarchical structure (i.e. bounding volume hierarchy) it is possible to test entire branches of the scene with a single query (spatial coherence). While traversing a scene in a front-to-back order, queries are only issued for previously invisible interior nodes and for previously visible leaf nodes of the hierarchy. The state of visibility for previously visible leaves is only updated for the next frame and they are therefore rendered immediately (without waiting for the query results to return). The state of visibility for previously invisible interior nodes is important for the current frame and they are not further traversed until the query results return. By interleaving the rendering of (previously) visible objects with the issuing of queries, the algorithm reduces idle time due to waiting for queries to return.

Although the core ideas remain the same, CHC++ introduced several optimizations which make it perform very well even in situations with low occlusion. Most notably, the improved version addressed the problem of redundant state changes due to the interleaved rendering and querying. Instead of directly querying a node, it is appended to a queue. When this queue reaches a certain size, the rendering state is changed to querying and an occlusion query is issued for each node in the queue. In addition, this mechanism allows an application to perform material sorting before rendering visible objects in order to reduce costly API calls.

In order to reduce the number of queries, the original CHC algorithm introduced an important optimization based on temporal coherence - A visible object is assumed to stay visible and will only be tested for visibility again after a user-specified amount of frames (typically 10-20). This optimization, together with the assumption that hardware instancing is the most efficient way to render multiple copies of the same geometry, is the entry-point for the proposed method. When an object suitable for instancing is found visible, it is scheduled to be rendered using hardware instancing in the following frame.

5.3 Data Preparation

In a static situation, where the same set of instances should be rendered every frame, per-instance data can be uploaded to GPU-memory once, and then, during subsequent draw calls, be fetched in the vertex shader based on the value of the internal instance counter. In the proposed approach, however, a dynamic behaviour is needed in order to support per-frame selection of which geometry to render using hardware instancing. As per-instance data needs to be arranged sequentially in order to facilitate a single draw call per geometry reference, this requires complete or partial updates of the shared buffer or texture every frame. In order to optimize this process an indexed approach is used, as explained visually in Figure 1. During scene loading, transformation matrices for all instanceable objects are collected and placed in a single array, denoted *M*. The location of each instance's transformation matrix within this array is recorded for later use. During rendering, a single index is then needed to locate each instance's transformation matrix within *M*. For a 4x4 transformation matrix, this approach effectively reduces the required data transfer by a factor of 16. Both the index array (*I*) and matrix array (*M*) are implemented as a Texture Buffer Objects.

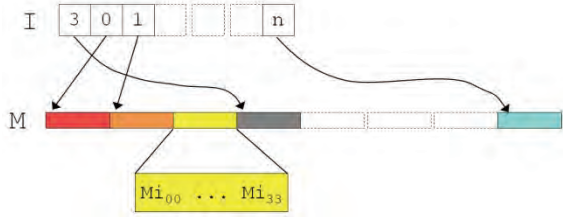


Figure 1: An array of indices (*I*) is used to locate each instance's transformation matrix, encoded in a single, shared array (*M*).

5.4 Collecting Visible Instances

Depending on the occlusion culling algorithm of use, the state of visibility for objects in a scene might be known at different stages. For instance, the GPU-based implementation of the Hierarchical Z Buffer proposed by (Hill, 2011), resolves visibility for all objects in a single phase in the beginning of a frame. The CHC++ algorithm, on the other hand, distributes this process by interleaving the rendering of objects with the issuing of queries, effectively delaying the complete visibility knowledge of a scene towards the end of the frame. In order to cope with different implementations and to provide additional time for the required data transfer, the

rendering of instanceable geometry is deferred by one frame. Thus, an object detected visible in frame *n* will be scheduled for rendering using hardware instancing in frame *n*+1. Figure 2 presents the modifications to the original CHC++ algorithm that is needed in order to implement this behavior. When a node of the spatial hierarchy is found visible, the *TraverseNode* function is called for its children (For a complete picture of the algorithm the reader is referred to the original CHC++ paper). During traversal of an instanceable leaf node the algorithm first checks if it is scheduled for rendering using hardware instancing in the current frame. If this is not the case it is rendered in a conventional way by adding it to a render queue. In a second step, it is scheduled for rendering using hardware instancing in the next frame.

```
TraverseNode(N) {
    if isLeaf(N) {
        if isInstanceable(N) {
            if N.nextInstFrameId != frameId {
                Render(N);
            }
            EnqueueForInstInNextFrame(N);
            N.nextInstFrameId = frameId + 1;
        }
        else {
            Render(N);
        }
    }
    else {
        DistanceQueue.PushChildren(N);
        N.IsVisible = false;
    }
};
```

Figure 2: Pseudo-code for the collection of visible instances. Difference to the CHC++ algorithm is marked in blue.

5.5 Data Transfer

At the end of frame *n*, a set of objects suitable for rendering using hardware instancing in frame *n*+1 has been collected. As illustrated in Figure 3, this set is sorted by geometry reference to generate a single array of indices (*I*) to upload to GPU memory. Thus, for *m* unique geometry references the array will contain *m* regions of indices. Within each region, the array is populated with indices corresponding to the location of each instance's transformation matrix in *M*. While generating the array the offset to each specific region is also recorded. This offset is needed during rendering in order to use a single indices array for all geometry references (Section 5.6).

During this stage, before the actual upload, the minimum number of instances per geometry

reference is also considered. Geometry rendered with hardware instancing uses a more complex vertex shader and require additional data transfer, which itself introduce a performance penalty. In order to gain an increase in performance the reduction of draw calls must reflect this. If the number of collected instances per geometry reference is below a user-defined parameter, N_{min} , they are not scheduled for instancing in the next frame. Instead the parameter *nextInstFrameId* (Figure 2) is set to zero on the corresponding objects in order to render them using a non-instanced approach in the next frame. For the different BIMs evaluated in this paper (Section 6), empirical tests have shown that a minimum requirement of three (3) instances per geometry reference is a suitable choice. However, ultimately, this parameter should be set per geometry reference, taking number of triangles into account.

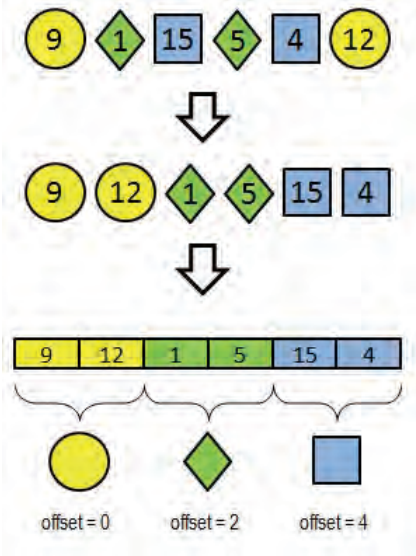


Figure 3: Collected instances (top) are sorted by geometry reference (middle) to generate the final index array and corresponding offsets (bottom). The numbers (indices) corresponds to the location of each instance's transformation matrix in M .

5.6 Rendering

As based on hardware occlusion queries, the CHC++ algorithm requires that visible objects are rendered before occluded ones in order to properly detect occlusion. Thus, in order to preserve the state of visibility, opaque instances are rendered using hardware instancing in a single step at the beginning of each frame. However, if the visibility determination system is separated from the

conventional rendering, this step can be performed at a later stage.

The actual rendering of all collected instances is performed by a single draw call per geometry reference. During this stage, the transformation matrix array (M) and indices array (I) are bound to the context. In Figure 4, GLSL code fragments from the vertex shader are shown. Here, the internal counter (`gl_InstanceID`) is used to fetch the current index from the indices array (I). This index is then used to locate the correct transformation matrix in the transformation matrix array (M). However, when invoking an instanced draw call, the internal counter will start its iteration from zero (0). As a single array is used for all indices an offset is required to define which region to fetch values from. This offset is recorded during the actual forming of the global indices array (Section 5.5), and during rendering it is supplied as a uniform per geometry reference.

```
uniform samplerBuffer M; //Matrices
uniform samplerBuffer I; //Indices
uniform int _offset;

void main()
{
    //Fetch index by offset
    int id = gl_InstanceID + _offset;

    int idx =
        int(texelFetchBuffer(I, id).x);

    mat4 OT =
        mat4(texelFetchBuffer(M, idx*4),
            texelFetchBuffer(M, idx*4+1),
            texelFetchBuffer(M, idx*4+2),
            texelFetchBuffer(M, idx*4+3));

    gl_Position =
        gl_ModelViewProjectionMatrix *
        OT * gl_Vertex;

    //-----
    //Other per-vertex calculations.
    //-----
};
```

Figure 4: Vertex shader used for the instanced rendering path (GLSL-code).

5.6.1 Semi-Transparent Geometry

Using conventional methods, semi-transparent geometry is rendered after opaque objects, in a back-to-front order, using alpha blending (Akenine-Möller et al., 2008). With hardware instancing correct order among transparent objects can no

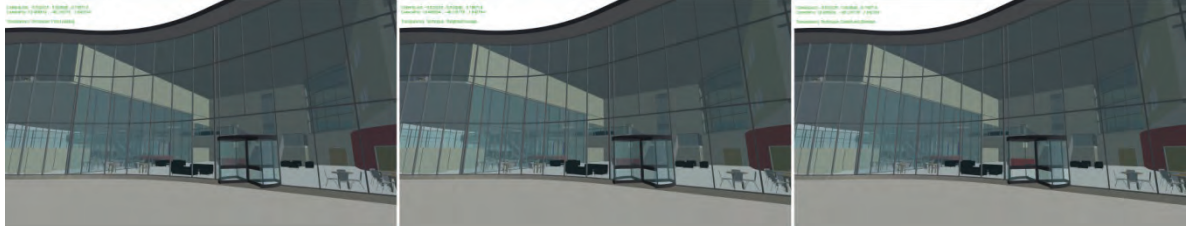


Figure 5: Transparency rendering modes: Depth peeling (left), Weighted average (middle) and sorted by object (right).

longer be preserved and, consequently, an order-independent transparency rendering technique is needed. A common technique within this category of algorithms is depth peeling (Bavoil, 2008), where transparent fragments are sorted by rendering the geometry several times, peeling off one transparent layer at a time. However, although accurate, the performance penalty of depth peeling is rather high which makes it unsuitable in practice. As a performance efficient alternative, a single pass approximation of depth peeling is suggested in (Bavoil, 2008). The technique, referred to as weighted average transparency, calculates the final color as the alpha-weighted sum of colors divided by the average alpha. When blending pixels of equal color and transparency, this technique produces correct results. However, when colors and transparency values differ too much, the result of the weighted average technique starts to deviate in terms of correctness compared to depth peeling. Still, despite being an approximation, it works very well for typical building models. As the use of transparency and color for windows and glazed structures is usually coherent within a building, the technique produces plausible results. Even in situations when many transparent layers are visible, the difference between the correct and the approximate method is hard to detect, as seen in Figure 5.

In the proposed method, both instanced and non-instanced semi-transparent geometry are rendered in a final stage each frame. For the implementation and tests the weighted average transparency technique has been primarily used. However, in the results section, the findings in terms of performance for the depth peeling approach are also reported.

6 RESULTS

The proposed method has been tested on four different Building Information Models. The models were created in Autodesk Revit 2012, and all four represents planned or existing buildings (see Table 2


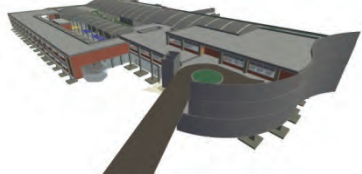

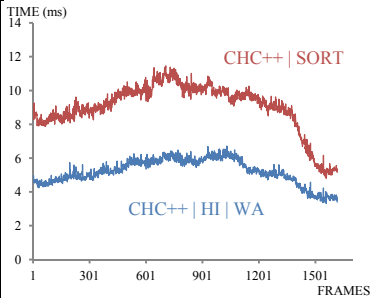
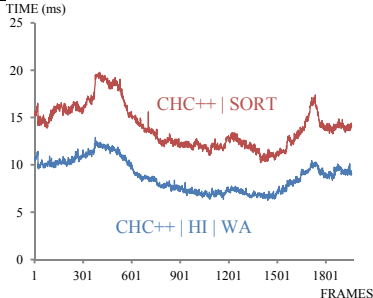
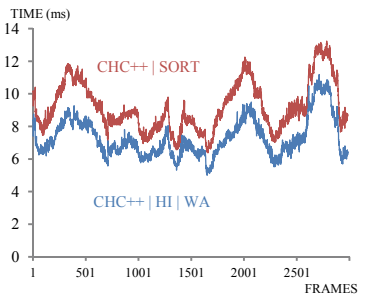
Table 1: Frame times (in ms) for view-frustum culling (VFC) and occlusion culling (CHC++) for one exterior and one interior viewpoint for each of the four test models.

Scene	INTERIOR		EXTERIOR	
	VFC	CHC++	VFC	CHC++
Library	13.1	1.9	17.2	7.5
Hospital	26.6	1.1	37.8	18.8
Student Housing	40.1	1.2	68.2	11.3
Hotel	56.3	1.4	130.2	47.8

and 3 for detailed information). For all of the tests an Intel Core i7 3.07 GHz CPU and an Nvidia GeForce 570 GTX graphics card was used. The CHC++ occlusion culling algorithm was used together with a bounding volume hierarchy built according to the surface area heuristics (Macdonald, 1990), and the screen resolution was set to 1280 x 720. Unless otherwise stated, the following parameters were used: maximum triangle count for instancing Tmax=3000, assumed visible frames Nav=20, minimum number of instances NMin=3. The weighted average technique (WA) was used for rendering semi-transparent geometry when hardware instancing (HI) was activated. Without instancing activated semi-transparent geometry was rendered using a conventional sort-by-object approach (SORT). However, for the Hotel model the performance numbers with depth peeling (DP) is also presented.

The CHC++ algorithm has previously been found to perform very well compared to only using view frustum culling for typical BIMs. These findings were confirmed for all of the test-models. Table 1 presents a comparison of frame times for one interior (highly occluded) and one exterior (same as seen in the screenshots in Table 2 and 3) viewpoint for each of the four models. As can be seen, the CHC++ algorithm provides a significant speedup, especially for the interior viewpoints. Given this, subsequent tests were focused on the worst case scenarios provided by the test models -

Table 2: Statistics for three of the test models and frame times for the proposed approach with occlusion culling (CHC++), hardware instancing (HI) and weighted average transparency (WA) compared to occlusion culling (CHC++) and conventional sort-by-object transparency (SORT) for the predefined walkthroughs.

LIBRARY	HOSPITAL	STUDENT HOUSING
		
3,291,869 triangles	1,561,972 triangles	10,857,175 triangles
7,312 objects	18,530 objects	17,666 objects
11,195 geometry batches	22,265 geometry batches	33,455 geometry batches
		

viewpoints when many objects are visible. In such situations the sheer number of objects that has to be rendered becomes the limiting factor in terms of performance. For all test models these scenarios were found in exterior viewpoints and a set of camera paths were constructed accordingly. These walkthroughs represents the worst case scenarios in terms of rendering performance when only occlusion culling was enabled. Table 2 (left) presents the frame times with and without hardware instancing enabled for the Library model. This model features a large glazing façade and consequently, many interior objects are visible at the same time when viewed from the outside. Here, the camera path provides an orbital camera movement from one side of the building to the other, while facing the center of the building. Compared to only using occlusion culling, the proposed method provides an average speed-up of 1.7x during this walkthrough. For the Hotel model the results are similar (Table 3). However, in this case the number of visible objects is mainly a result of a vast façade composed by many replicated windows, curtain wall elements and façade stones. The walkthrough sequence is similar as for the Library model, however, in the end interior, more occluded regions of the building are also visited. During these viewpoints the number of visible

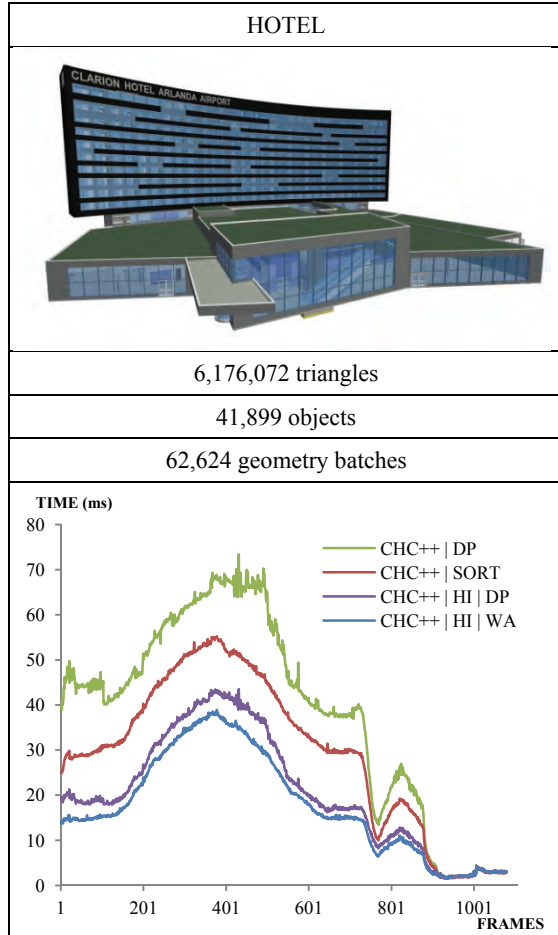
instanceable objects is low and, hence, few or none of them are rendered using hardware instancing. Still, in such viewpoints the occlusion culling system alone is able to deliver high performance and the important thing to note is that the proposed method only introduces a slight overhead, noticeable only in terms of relative numbers. For the non-interior parts of the walkthrough sequence an average speed-up of 1.7x was achieved with hardware instancing.

Table 3 also presents the performance results with depth peeling (DP). This approach guarantees a correct results but the performance penalty is higher compared to the weighted average technique. Nevertheless, compared to conventional sorting (SORT) a 1.5x speed-up was still achieved with instancing. On the other hand, when depth peeling was used in both cases (instanced and non-instanced) the average speed-up was almost 2x.

Figure 6 presents the number of draw calls with and without hardware instancing enabled for the Hotel model. This plot reveals the source of the performance gain. As can be seen, the numbers of draw calls are greatly reduced and, as a consequence, the performance is increased.

Table 2 also presents the performance numbers for the Hospital (middle) and Student Housing

Table 3: Statistics for the Hotel model and frame times for the proposed approach with occlusion culling (CHC++), hardware instancing (HI) and weighted average transparency (WA) compared to occlusion culling (CHC++) and conventional sort-by-object transparency (SORT) for the predefined walkthrough. In addition, the frame times for depth peeling (DP) are presented.



model (right). For the Hospital model an average speed-up of 1.6x is achieved with instancing. For the Student Housing model the performance gain of the proposed method is more moderate. Although an average speed-up of 1.25x is achieved, it is less than expected considering the model still has a fairly large amount of replicated components. However, compared to the other models, the animation sequence for the Student Housing model does not feature viewpoints equally beneficial in terms of instancing. First, the relative amount of visible replicated geometry is not as high and, second, the number of different geometry references is higher.

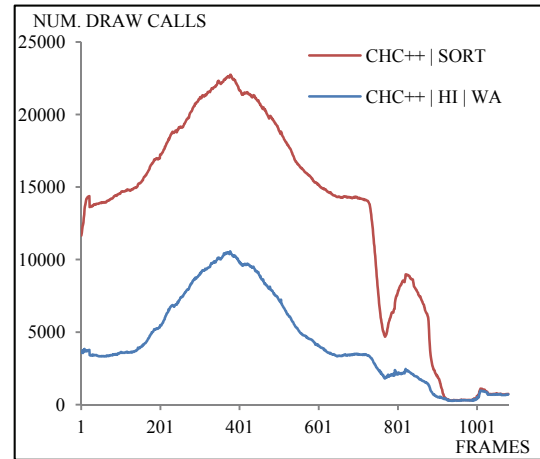


Figure 6: Number of draw calls with and without hardware instancing for the Hotel model.

7 CONCLUSION

This paper has presented a simple, yet efficient approach for integrating occlusion culling and hardware instancing. Compared to only using occlusion culling, average speed-ups of 1.25x – 1.7x were achieved for all test models in viewpoints where many objects are visible. Without dynamic hardware instancing activated, these viewpoints represents the worst case scenarios in terms of rendering performance.

The only aspect to consider a limitation is the requirement of an order-independent transparency rendering technique for semi-transparent geometry. A simple solution to remove this restriction would be to skip the use of hardware instancing for transparent geometry. Still, such geometry often possesses characteristics suitable for hardware instancing, which makes them tempting to include. For the tested models the weighted average technique was found to provide plausible results with high performance. In addition, depth peeling was shown to provide a viable option if a fully correct result is important.

For future work it would be interesting to test the proposed method together with other occlusion culling algorithms. The CHC++, although efficient, tightly integrates visibility determination and actual rendering of geometry. This puts restrictions on when collection, upload and rendering of instanced geometry can be performed. If these restrictions were relaxed, it is possible that a more efficient implementation of hardware instancing could be achieved.

Another area of further investigations would be the parameters Nlmin (minimum number of instances per geometry reference) and Tmax (maximum number of triangles for instanced geometries) for different scenes and hardware setups. Although the results show that uniform values for these parameters works in practice, it is likely that the performance could be further enhanced by letting Nlmin depend on triangle count (i.e. demanding a higher instance count for geometries with many triangles).

REFERENCES

- Akenine-Möller, T., Haines, E., Hoffman, N. (2008). Real-Time Rendering 3rd Edition. A. K. Peters, Ltd., Natick, MA, USA.
- Bao, G., Li, H., Zhang, X., Dong, W. (2012). Large-scale forest rendering: Real-time, realistic, and progressive. *Computers & Graphics*, Vol. 36, Issue 3, Pages 140-151.
- Bavoil, L., Myers, K. (2008). Order Independent Transparency with Dual Depth Peeling. Tech. rep., NVIDIA Corporation.
- Bittner, J., Wimmer, M., Piringer, H., Purgathofer, W. (2004). Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful. *Computer Graphics Forum* 23, 3, pages 615–624.
- Bittner, J., Wonka, P. (2003). Visibility in Computer Graphics. *Environment and Planning B: Planning and Design* 30, 5, pages 729–756.
- Buchholz, H., Döllner, J. (2005). View-Dependent Rendering of Multiresolution Texture-Atlases. *Proceedings of the IEEE Visualization 2005*, Minneapolis, USA.
- Cohen-Or, D., Chrysanthou, Y. L., Silva, C. T., Durand F. (2003). A Survey of Visibility for Walkthrough Applications. In *IEEE Transactions on Visualization and Computer Graphics* 09, 3, pages 412–431.
- Dudash, B. (2007). Animated crowd rendering. In *GPU Gems 3*. Addison-Wesley, pages 39–52.
- Eastman, C., Teicholz, P., Sacks, R., Liston, K. (2011) *BIM Handbook (2nd Edition) A guide to building information modeling for owners, managers, designers, engineers and contractors*, John Wiley & Sons, New Jersey.
- Everitt, C. (2001). Interactive Order-Independent Transparency. Tech. rep., NVIDIA Corporation.
- Greene, N., Kass, M., Miller, G. (1993). Hierarchical ZBuffer Visibility. In *SIGGRAPH '93*, pages 231–238.
- Guthe, M., Balazs, A., Klein, R. (2006). Near Optimal Hierarchical Culling: Performance Driven Use of Hardware Occlusion Queries. In *Eurographics Symposium on Rendering 2006*.
- Hill, S., Collin, D. (2011). Practical, Dynamic Visibility for Games. In *Gpu Pro 2*.
- Johansson, M., Roupé, M. (2012). Real-Time Rendering of large Building Information Models. In *proceedings of CAADRIA 2012 - Beyond Codes & Pixels*, pages 647-656.
- Macdonald, J. D., Booth, K. S. (1990). Heuristics for ray tracing using space subdivision. *Visual Computer* 6, 6, pages 153–65.
- Mattausch, O., Bittner, J., Wimmer, M. (2008). CHC++: Coherent Hierarchical Culling Revisited. *Computer Graphics Forum (Proceedings Eurographics 2008)* 27, 2, pages 221–230.
- Park, H., Han, J. (2009). Fast Rendering of Large Crowds Using GPU. In *Entertainment Computing - ICEC 2008 (Lecture Notes in Computer Science, 5309)*, pages 197-202.
- Ramos, F., Ripolles, O., Chover, M. (2012). Continuous Level of Detail for Large Scale Rendering of 3D Animated Polygonal Models. In *Articulated Motion and Deformable Objects (Lecture Notes in Computer Science, 7378)*, pages 194-203.
- Sacks, R., Eastman, C.M., Lee, G. (2004). Parametric 3D modeling in building construction with examples from precast concrete. In *Automation in Construction* 13, pages 291–312.
- Steel, J., Drogemuller, R., Toth, B. (2012). Model interoperability in building information modelling. In *Software and Systems Modeling*, 11, 1, pages 99-109.
- Wloka, M. (2003). Batch, Batch, Batch: What Does It Really Mean? Presentation at Game Developers Conference 2003.

Paper IV

From BIM to VR

Integrating immersive visualizations in the current design process

Mikael Johansson¹, Mattias Roupé², Mikael Viklund Tallgren³

^{1,2,3}Chalmers University of Technology, Sweden

^{1,2,3}{jomi|roupe|mikael.tallgren}@chalmers.se

This paper presents a system that allows immersive visualizations to become a natural and integrated part of the current building design process. It is realized through three main components: (1) the Oculus Rift - a new type of Head Mounted Display (HMD) directed at the consumer market, (2) a real-time rendering engine supporting large Building Information Models (BIM) that is, (3) implemented as a plug-in in a BIM authoring software. In addition to provide details regarding the implementation and integration of the different components in our system, we present an evaluation of it from three different perspectives; rendering performance, navigation interface and the ability to support fast design iterations.

Keywords: Building Information Modeling, BIM, Virtual Reality, Real-time rendering, HMD

INTRODUCTION

During the design process of a building, it is important that all the involved actors understand, participate, communicate, and collaborate with each other to obtain a high quality outcome of the design process. Hall and Tewdwr-Jones (2010) highlight the communication difficulties between the different stakeholders in the design and planning process. Communication difficulties mainly occur as a result of the different planning cultures, and because there is insufficient collaboration and information sharing during the process. The most common problem is that the information is not presented in such a way that people can understand it.

In this context, real-time visualizations and Virtual Reality (VR) have been shown to offer an effi-

cient communication platform (Bouchlaghem et al., 2005). With the ability to navigate freely through 3D scenes from a first-person perspective, it is possible to present and communicate ideas regarding future buildings in a way that facilitates understanding among all involved parties, despite their background or professional expertise. While the use of this technology has been naturally limited in the past due to lack of available 3D data from the design process, the recent introduction of Building Information Models (BIM) within the AEC field has opened up new possibilities. With the use of BIM the required 3D data can be extracted from the architect's own design-environment, instead of creating it from scratch using 2D-plans, elevations and sketches as a reference. Because of this, use of real-time visualizations has be-

come more accessible in practice.

To further enhance user experience it is commonly advocated to take advantage of immersive display technologies. Although real-time visualizations have been shown to be useful per se, stereoscopy, large screen and wide field of view all provide additional benefits. When comparing a non-immersive (monitor) solution to a four-screen (3 walls and a floor) CAVE solution, Shiratuddin et al. (2004) found consistently higher ratings for the latter regarding level of realism, ease of navigation, sense of scale and overall suitability for design and decision-making tasks.

In this context, Head Mounted Displays (HMD) also represents a viable option. Still, as available alternatives (until very recently) have been either low-cost-low-performance or high-cost-high-performance devices (Dörner et al., 2011), CAVEs and PowerWalls have emerged as the de facto standard when it comes to immersive visualizations. When considering practical applications, these types of solutions have been shown beneficial during the design of hospital patient rooms and courtrooms, as well as for design review sessions in general (Castronovo et al., 2013).

However, when considering the integration and use of immersive VR within the actual design process, the current adaptation in the AEC field still suffers from a number of limitations:

- **High cost:** Even if the cost of display and PC hardware has been rapidly decreasing over the past years, fully or semi-immersive solutions such as CAVEs or PowerWalls are still expensive (DeFanti et al., 2011).
- **Limited accessibility:** Regardless of display technology (e.g. immersive or semi-immersive), the use of a specific room or studio will naturally restrict visualization sessions to a single location. Even if situated close to the designers working environment it makes the use of VR less accessible, both physically and mentally. This immobility has also been

reported inconvenient for clients and other stakeholders (Sunesson et al., 2008).

- **Limited BIM-support:** Even if created with visualization in mind, real-time constraints and stereo rendering often require the input 3D data to be further optimized in order to be fully functional in the VR environment. When considering BIMs this process typically becomes even worse due to a large number of individual objects and high geometric complexity (Dalton and Parfitt, 2013). In addition, many BIM authoring applications have limited or missing support for materials and texture definitions when exporting 3D-data for visualization purposes (Kumar et al., 2011).

In this paper we present a solution that overcomes the above mentioned limitations and allows immersive VR to become a natural and integrated part of the design process. It is realized through three main components: (1) the Oculus Rift Head Mounted Display (HMD) - a comparably low cost device that supports a large field of view, stereoscopic viewing and physically rotation, (2) an efficient real-time rendering engine supporting large 3D datasets that is (3) implemented as a plug-in in a BIM authoring software.

A PORTABLE SYSTEM FOR IMMERSIVE BIM VISUALIZATION

Figure 1 shows the different components of our proposed system: The Oculus Rift HMD, the real-time viewer application implemented as a plug-in in Revit Architecture and a so-called PowerPoint remote control used as a navigation interface, all connected to a lightweight laptop. In the following subsections we present and discuss these components in more detail.

The Oculus Rift HMD

In order to provide an immersive visualization environment our proposed system takes advantage of the Oculus Rift HMD. The Rift is a new affordable (expected price range \$300-\$350) virtual reality device

Figure 1
System overview;
the Oculus HMD,
the Revit Viewer
plug-in and the
PowerPoint remote
control



directed at the consumer's market, mainly to provide immersive experiences for videogames. Although currently only available in the form of a Developer Kit, it is expected to be available on the broad consumer market during 2015. The device provides approximately 100° field of view, stereoscopic 3D view and includes a gyroscope, an accelerometer and a magnetometer to determine the orientation of the user's head in the real world.

As with any other stereo-providing display solution the 3d scene has to be rendered twice, once for each eye. In the case of the Rift this is implemented by means of split-screen rendering, where the left half of the screen corresponds to the left eye, and vice versa. With a full-screen resolution of 1280 x 800 pixels, this gives an effective resolution of 640 x 800 per eye.

However, although this approach to support stereo vision is conceptually simple, the actual rendering process is a bit more involved. Due to the lenses, which provide an increased field of view, a pincushion distortion of the image is introduced. To cancel out this effect, the rendering has to be done at a higher resolution, followed by a post-processing step that performs a barrel distortion. Preferably, antialiasing should also be enabled, as this greatly enhances the image quality.

So, in effect, even if the devices' resolution is "only" 1280 x 800 pixels, the actual rendering (rasteri-

zation) has to be performed at a much higher resolution (i.e. 2194 x 1371), potentially with antialiasing enabled, followed by a full-screen post-processing step. Obviously, these requirements put additionally stress on the graphics hardware, which, in turn, put high demands on a rendering engine to deliver enough rendering performance to support an interactive experience.

The rendering engine

An important property for any type of real-time visualisation system is its ability to maintain a sufficiently high frame rate. For typical desktop applications (i.e. non-immersive) 15Hz is often considered a minimum (Yoon et al. 2008), although 30 or 60 Hz is generally advocated in order to provide a satisfactory level of interactivity. However, for HMDs, such as the Oculus Rift, the minimum interactivity-demands are typically higher, as physical interaction and display update becomes much more integrated. Ultimately, a user's head movement should directly correspond to an update of the display in order to reduce the risk of potential conflicts between visual-vestibular sensory. In this context a minimum frame rate of 60 Hz is often recommended (Adelstein et al., 2003), although higher values have also been proposed (Jerald, 2010).

When considering that the task of visualizing BIMs interactively is known to be a challenge in itself (Steel et al., 2012; Johansson and Roupé, 2012),

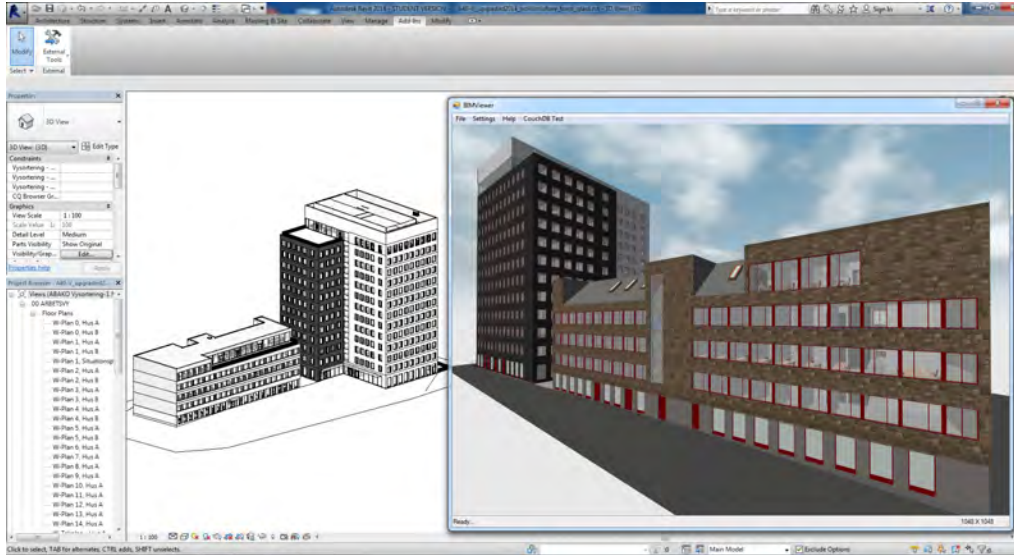


Figure 2
The viewer plug-in
interface in Revit

the frame rate requirements posed by using an HMD thus put very high demands on the rendering engine. This, especially, as the 3D dataset has to be rendered twice every frame in order to support stereoscopic vision, followed by a full-screen post-processing step.

To address these requirements we have developed an efficient rendering engine that takes advantage of two characteristics shared by typical building models - high level of occlusion and frequent reuse of identical building components. The engine, which is described in more detail in (Johansson and Roupé, 2012) and (Johansson, 2013), uses an efficient occlusion culling algorithm to restrict rendering efforts to visible objects only, and takes advantage of hardware instancing to render replicated building components efficiently. These two acceleration techniques complement each other and are essential in order to fulfil the requirements in terms of interactivity. However, we do not primarily use this rendering engine in a separate application. To support an integrated design environment we have instead implemented it as a viewer plug-in in Autodesk Revit.

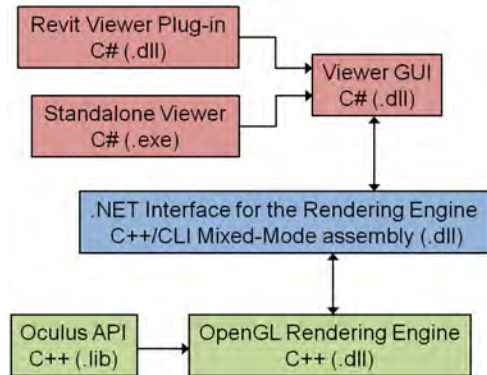
The Revit plug-in

Figure 2 illustrates how our viewer plug-in is integrated in Revit from a user's perspective. With a BIM loaded in Revit the viewer is initialized from the Add-Ins tab, resulting in the real-time 3D visualization representation becoming visible in a new window. After that a user is free to either navigate the model in a typical desktop fashion using mouse and keyboard, or connect the Oculus HMD to experience the model immersively.

From a programmers point of view the plug-in extracts the required 3D data through the Revit C# API, which exposes the entire underlying BIM database. To speed-up the data extraction process and to keep the memory footprint low, we take advantage of geometry instancing (i.e. that several identical components can share the same geometrical representation), which is an integral part of the internal Revit database. Every time a unique geometric representation is encountered for the first time, all of its data is extracted. For all subsequent cases the previously extracted geometry data is used in combination with a unique transformation.

As the rendering engine and the Oculus API is written in C++, the different software components needs to be connected through a C++/CLI bridge (Heege, 2007). The complete architecture is illustrated in Figure 3 and also shows how the GUI-module is separated from the actual plug-in, essentially allowing us to run the viewer as a standalone application on a system without Revit installed, with identical interface.

Figure 3
System architecture



Previous versions of the Revit API did not expose material data such as colors and textures, making it very difficult to use BIMs, without further treatment, for visualization sessions related to aesthetics. Fortunately, since version 2014, the API has been extended with a Custom Export API, that facilitates the extraction of material and texture data as well as texture coordinates. Because of this, it is now possible to extract complete visualization models, with materials and textures assigned, directly from the BIM authoring software.

The navigation interface

The use of an HMD makes traditional navigation interfaces, such as ones with keyboard and a mouse harder to master. As the user cannot see anything in the real world, even seemingly simple tasks, such as pressing a specific key on the keyboard or even grabbing the mouse becomes much more involved. For very experienced users that daily works with, and

navigates in, 3D models this does not necessarily pose itself as a problem, however, for people with less experience it can easily become a huge obstacle.

In order to allow for any type of user we have therefore developed a very simple navigation interface by means of a so-called PowerPoint remote control. As illustrated in Figure 4, a user can move forward or back by pressing the corresponding buttons on the remote control, with the direction of movement being decided by the user's orientation of the head.

SYSTEM EVALUATION

To illustrate the effectiveness of our proposed system we present an evaluation of it from three different perspectives - rendering performance, navigation interface and the ability to support fast design iterations. As test-model we have used a BIM received from a real-world project, a ten-story office building which is currently being built in Gothenburg, Sweden (Figure 2). The model is primarily an architectural model, with no Mechanical, Electrical or Plumbing (MEP) data present, however it does contain furniture and other interior equipment (See Figure 5). The model was created in Revit Architecture 2013 and contains approximately 4,400,000 triangles, distributed over 15,000 individual objects.

Figure 4
The navigation interface



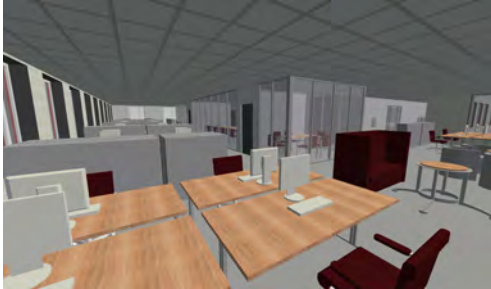


Figure 5
Interior viewpoint
at the third level of
the test-model

Rendering performance

The rendering performance test was performed on two different computers, one workstation and one laptop. The workstation was equipped with an Intel i7 3.06 GHz CPU, 6 GB of RAM and an Nvidia GeForce GTX 570 GPU running Windows 7 x64. The laptop was equipped with an Intel i7 1.9 GHz CPU, 4 GB of RAM and an Nvidia GeForce GT 620M GPU running Windows 8 x64. On both system, two different camera paths was used; one interior at the third floor of the building, and one exterior at the ground-level in front of the building. The results from these tests are presented in Figure 6. To better illustrate the performance gain offered by our rendering engine we also present performance results obtained when only view frustum culling is enabled (i.e. only discarding objects that are outside the cameras view frustum). The following abbreviations are used (and combined) in the plots: OC for Occlusion Culling, HI for Hardware Instancing, VFC for View Frustum Culling, and MSAA for 4x MultiSample AntiAliasing.

As can be seen in the plots, the use of additional acceleration techniques is vital in order to provide the required level of interactivity. With only view frustum culling enabled (VFC) it becomes very difficult to guarantee a minimum frame rate of 60 Hz, even on the workstation system. In fact, for the given camera paths, not even 20 Hz can be guaranteed on both systems.

However, with the combined use of occlusion culling (OC) and hardware instancing (HI) it is possible to fulfil the interactivity demands. The only ex-

ception appears during parts of both camera paths on the laptop system when antialiasing is enabled. Although not by much (the lowest recorded frame rates are 52 Hz and 48 Hz, respectively) it is definitely below our target frame rate of 60 Hz. Still, as antialiasing-capacity scale well with GPU performance (which is not necessarily the case with 3D model complexity due to driver overhead), we expect this particular issue to be solved by increasingly faster GPUs. This, especially when also considering that these tests were performed on a, at the time of writing, two year old laptop.

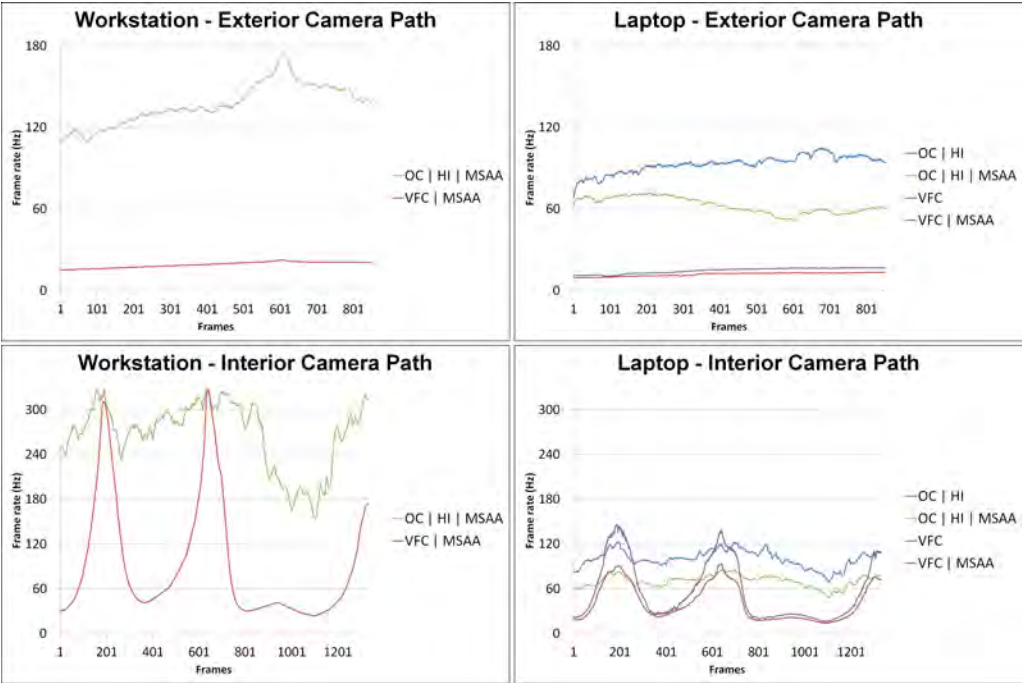
Nevertheless, without antialiasing activated, our rendering engine can provide the required level of interactivity, even on a lightweight laptop system.

Navigation interface

As part of a different, but related, research project we have performed an initial evaluation of the navigation interface with members of the on-site team that is currently erecting the real building. This group of people included the site manager as well as five construction workers from different sub-trades (piping, ventilation, sprinklers, prefab and electrical). No one, except for the site manager, had any previous experience from working with, or navigating in, 3D-models. While freely navigating and inspecting the digital representation of the building that they were currently erecting, they were asked questions regarding how they felt that this type of interface could help them extract information to support their daily work and what additional features they would like the system to have. Except for the electrical trade worker, they all expressed that this type of visual interface helped them to get a better understanding, not only in terms of specific details, but also for the project as a whole.

However, perhaps more interesting in this context, is the fact that we observed that all of them, including the electrical trade worker, were able to navigate in the model with ease. Based on our own previous experience we know that this is typically not the case when inexperienced people are faced with the

Figure 6
 Frame rates for the exterior (top) and interior (bottom) camera paths on the workstation (left) and the laptop (right) system (OC=Occlusion Culling, HI=Hardware Instancing, MSAA=4xMultiSample AntiAliasing, VFC=View Frustum Culling).



task of navigating in a 3D-model using the keyboard and a mouse (i.e. mouse-look and WASD).

Design iterations

The benefit of having the visualization environment closely connected to the BIM authoring environment becomes especially clear when considering rapid design iterations. To illustrate this we will provide two concrete examples applied to our test-model: one is the change of window types on one of the facades and the other is the removal of two conference rooms on the third floor in order to extend the office landscape area.

Although solutions have been proposed where it is possible to modify architectural models directly in an immersive environment (Schulze et al., 2014), these systems typically only support insertion and repositioning of pre-made objects or creation and

modification of simple geometry. In contrast, our examples are much more involved, as they include operations on fairly complex objects that also affect other objects. For instance, when changing windows to a type that is geometrically smaller or larger, the geometry for the host object, the wall, needs to be recomputed in order for the opening size to match the corresponding window size. Although such functionality would have been technically possible to implement in our viewer, we have instead focused on making the "conversion" from design-model to visualization-model as fast as possible. In the case of our test model, this process takes approximately 20 seconds. That is, regardless of modification, the only time needed to produce a new version of the immersive visualization will be the time required to make the actual modifications in Revit, plus 20 seconds. For the examples described above this time corresponds

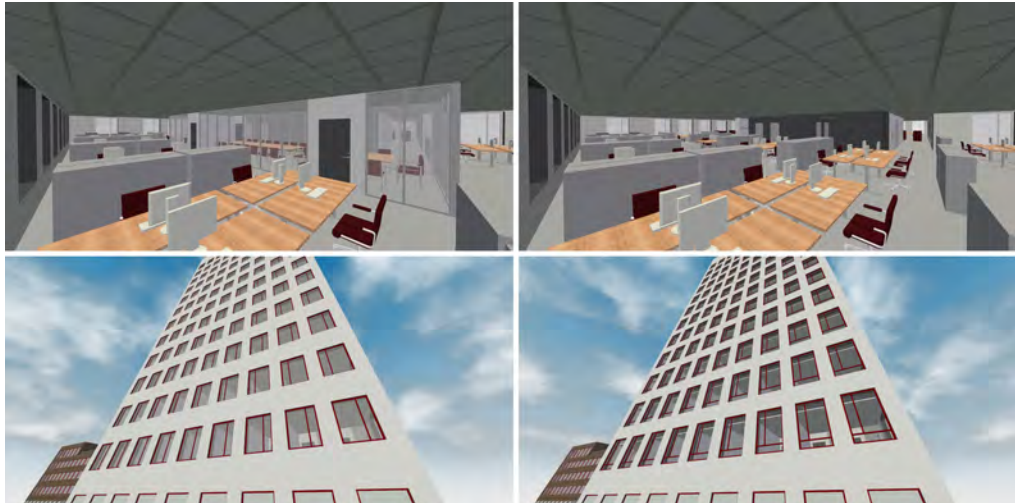


Figure 7
Before (left) and
after (right) rapid
design
modifications. Top
row illustrates
removal of
conference room.
Bottom row
illustrates change of
window types.

to 3 minutes and 2.5 minutes, respectively. In Figure 7, these two modifications are illustrated with "before and after" screenshots.

CONCLUSIONS AND FUTURE WORK

We have presented a system that allows immersive visualization to become a natural and integrated part of the building design process. By using the Oculus Rift HMD we are able to provide an immersive visualization environment without the need of a dedicated facility to host a PowerWall or CAVE installation. In addition to greatly reduce investment costs, this feature also makes the use of VR within a project become physically more accessible. As the technology is portable, clients and design team members can take advantage of immersive visualization sessions without the need to travel to a specific location.

To further address accessibility, we have developed a rendering engine capable of managing large and complex 3D datasets in real-time. As a result we can directly visualize large and complex BIMs, in stereo, without the need to manually optimize or prepare the input dataset. To support an integrated design environment this rendering engine has been implemented as a viewer plug-in in Autodesk Revit. Be-

cause of this, immersive design review sessions can be performed directly in the BIM authoring software without the need to export any data or create a separate visualization model.

In addition, we have presented an initial evaluation of the proposed system with a BIM received from a real-world project. Regarding rendering performance, navigation interface and the ability to support fast design iterations, we have shown that it has all the needed properties in order to function well in practice.

For future work we are considering several different directions, including studies related to spatial understanding with HMDs, enhancement of the interaction interface, investigation of benefits with our system in different contexts (i.e. design review, planning, on-site information extraction, etc.) as well as further research to improve rendering performance.

REFERENCES

Adelstein, BD, Lee, TG and Ellis, SR 2003 'Head tracking latency in virtual environments: psychophysics and a model', *In Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, pp. 2083-2087

- Bouchlaghem, D, Shang, H, Whyte, J and Ganah, A 2005, 'Visualisation in architecture, engineering and construction (AEC)', *Automation in Construction*, 14, pp. 287-295
- Castronovo, F, Nikolic, D, Liu, Y and Messner, JI 2013 'An evaluation of immersive virtual reality systems for design reviews', *Proceedings of the 13th International Conference on Construction Applications of Virtual Reality*
- Dalton, B and Parfitt, M 2013 'Immersive Visualization of Building Information Models', *Design Innovation Research Center working paper 6*, University of Reading, UK
- DeFanti, T, Acevedo, D, Ainsworth, RA, Brown, MD, Cutchin, S, Dawe, G, Doerr, KU and Johnson, A 2011, 'The Future of the CAVE', *Central European Journal of Engineering*, 1, pp. 16-37
- Dörner, R, Lok, B and Broll, W 2011, 'Social Gaming and Learning Applications: A Driving Force for the Future of Virtual and Augmented Reality?', in Coquillart, S (eds) 2011, *Virtual Realities*, Springer, pp. 51-76
- Hall, P and Tewdwr-Jones, M 2010, *Urban and Regional Planning*, Routledge
- Heege, M 2007, *Expert C++/CLI: NET for Visual C++ Programmers*, Apress
- Jerald, JJ 2010, *Scene-motion-and latency-perception thresholds for head-mounted displays*, Ph.D. Thesis, University of North Carolina
- Johansson, M 2013 'Integrating Occlusion Culling and Hardware Instancing for Efficient Real-Time Rendering of Building Information Models', *International Conference on Computer Graphics Theory and Applications (GRAPP 2013)*, pp. 197-206
- Johansson, M and Roupé, M 2012 'Real-Time Rendering of Large Building Information Models', *CAADRIA 2012 - Beyond Codes & Pixels*, pp. 647-656
- Kumar, S, Hedrick, M, Wiacek, C and Messner, JI 2011, 'Developing an experienced-based design review application for healthcare facilities using a 3d game engine', *Journal of Information Technology in Construction (ITcon)*, 16, pp. 85-104
- Schulze, JP, Hughes, CE, Zhang, L, Edelstein, E and Macagno, E 2014 'CaveCAD: a tool for architectural design in immersive virtual environments', *Proceedings of SPIE 9012, The Engineering Reality of Virtual Reality 2014*
- Shiratudin, MF, Thabet, W and Bowman, D 2004 'Evaluating the effectiveness of virtual environment displays for reviewing construction 3D models', *Proceedings of CONVR 2004*, pp. 87-98
- Steel, J, Drogemuller, R and Toth, B 2012, 'Model interoperability in building information modelling', *Software & Systems Modeling*, 11, pp. 99-109
- Sunesson, K, Allwood, CM, Paulin, D, Heldal, I, Roupé, M, Johansson, M and Westerdahl, B 2008, 'Virtual Reality as a New Tool in the City Planning Process', *Tsinghua Science & Technology*, 13, pp. 255-260
- Yoon, SE, Gobbetti, E, Kasik, D and Manocha, D 2008, *Real-Time Massive Model Rendering*, Morgan & Claypool Publisher

Paper V

Efficient Stereoscopic Rendering of Building Information Models (BIM)

Mikael Johansson

Chalmers University of Technology, Sweden

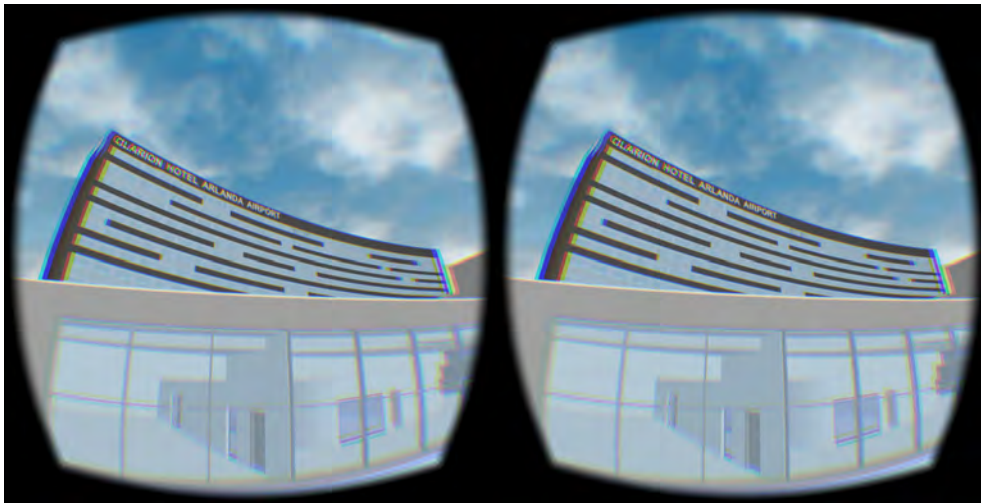


Figure 1. A building information model (BIM) taken from a real-world project rendered at more than 90 frames per second using the stereo rendering techniques presented in this paper.

Abstract

This paper describes and investigates *stereo instancing*—a single-pass stereo rendering technique based on hardware-accelerated geometry instancing—for the purpose of rendering building information models (BIM) on modern head-mounted displays (HMD), such as the Oculus Rift. It is shown that the stereo instancing technique is very well suited for integration with query-based occlusion culling as well as conventional geometry instancing, and it outperforms the traditional two-pass stereo rendering approach, geometry shader-based stereo duplication, as well as brute-force stereo rendering of typical BIMs on recent graphics hardware.

1. Introduction

The advent of building information models (BIM) and consumer-directed HMDs, such as the Oculus Rift and HTC Vive, has opened up new possibilities for the use

of virtual reality (VR) as a natural tool during architectural design. The use of BIM allows a 3D scene to be directly extracted from the architect's own design environment and, with the availability of a new generation of VR-systems, architects and other stakeholders can explore and evaluate future buildings in 1:1 scale through an affordable and portable interface.

However, given the rendering-performance demands posed by modern HMDs, the use of BIMs in a VR setting remains a difficult task. With BIMs being primarily created to describe a complete building in detail, many 3D datasets extracted from them provide a challenge to manage in real-time if no additional acceleration strategies are utilized [Johansson et al. 2015]. In this context, a combination of occlusion culling and hardware-accelerated geometry instancing has been shown to provide a suitable option in a non-stereo environment [Johansson 2013]. This approach could be adapted to a stereo setup simply by performing two rendering passes of the scene, one for the left eye and one for the right eye. However, this would still require an additional and equal amount of rendering time, as the number of occlusion tests, rasterized triangles, and issued draw calls would increase by a factor of two.

In order to remove the requirement of a second pass, the concept of *stereo instancing* has recently gained a lot of attention within the game development community [Wilson 2015; Vlachos 2015]. Stereo instancing refers to a technique where hardware-accelerated geometry instancing is used to render left and right stereo pairs during a single pass.

In this paper, a more detailed description of the stereo instancing technique is provided together with a thorough performance evaluation using real-world datasets. Furthermore, the stereo instancing technique is used to extend the work in [Johansson 2013] in order to provide efficient stereoscopic rendering of BIMs. It is shown that stereo instancing works very well in combination with occlusion culling based on hardware-accelerated occlusion queries in a split-screen stereo setup. In addition, the use of hardware-accelerated instancing is generalized to support both replicated geometry as well as single-pass generation of stereo pairs.

2. Stereoscopic Rendering, Bottlenecks, and Acceleration Techniques

Any true stereoscopic display solution requires that the user be presented with different images for the left and right eye. In the case of the Oculus Rift, this is implemented with split-screen rendering, where an application renders the image for the left eye into the left half of the screen, and vice versa. However, due to the lenses, which provide an increased field of view, a pincushion distortion of the image is introduced. To cancel out this effect, the rendering has to be done at a higher resolution, followed by a post-processing step that performs a barrel distortion. In Figure 2, the Oculus stereo rendering pipeline is illustrated.

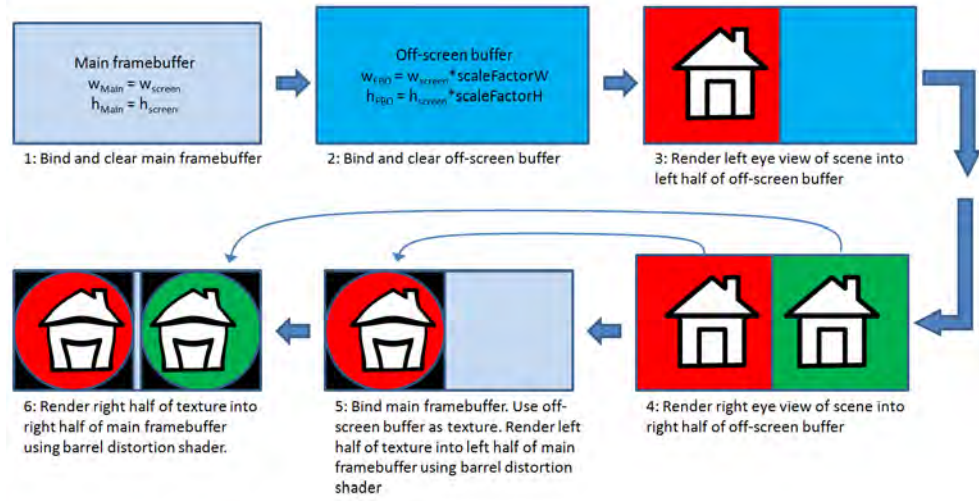


Figure 2. A typical stereo rendering pipeline for the Oculus Rift.

In addition to requiring two distinct views of the scene every frame, stereoscopic rendering targeting head-mounted displays (HMDs) typically has much higher interactivity demands compared to monoscopic, desktop applications. With the introduction of the consumer versions of both the Oculus Rift and HTC Vive, the minimum frame rate has been set to 90 Hz, which corresponds to a maximum frame time of 11.1 ms.

Traditionally, stereo rendering has almost exclusively been implemented as two independent, serial rendering passes, effectively doubling the geometry workload on both the CPU and the GPU. When considering ways to improve rendering performance in such a setup, we can generalize the situation in a similar fashion as Hillaire [2012]; if an application is CPU-bound, its performance will be mostly influenced by the number of draw calls and related state changes per frame. If, on the other hand, the application is GPU-bound, its performance will be influenced by the number of triangles drawn every frame and the complexity of the different shader stages.

As with monoscopic rendering, there are several different acceleration techniques that can potentially be utilized if the performance requirements are not met. Common examples include level-of-detail (LOD) to reduce the geometric complexity of distant objects; frustum, occlusion, or contribution culling to reduce the number of objects to render; hardware-accelerated instancing to reduce the number of draw calls when rendering replicated geometry; and geometry batching to render groups of primitives with as few draw calls as possible. An option more unique to stereoscopic rendering is to use the geometry shader to render left and right stereo pairs from a single geometry pass [Marbach 2009]. The geometry shader supports multiple outputs from a single

input as well as functionality to redirect output to a specific viewport. Consequently, a single draw call per batch of geometry is sufficient in order to produce both the left and right eye projection. However, even if this reduces the CPU-burden—by reducing the number of draw calls as well as related state changes—the geometry shader typically introduces significant overhead on the GPU.

3. Stereo Instancing

As the name implies, stereo instancing takes advantage of hardware-accelerated geometry instancing in order to produce both the left- and right-eye version of the scene during a single rendering pass. With the instancing capabilities of modern GPUs, it is possible to produce multiple output primitives from a single input, without introducing the geometry shader. To support stereo instancing, an application only needs to replace conventional draw calls (i.e., `glDrawArrays`) with instanced ones (i.e., `glDrawArraysInstanced`), providing two (2) as the instance count. Based on the value of the instance counter in the vertex shader (i.e., `gl_InstanceID` being zero, respectively one), each vertex can then be transformed and projected according to the left or right eye, respectively.

However, the main difficulty with this approach is that unextended OpenGL currently does not support multiple viewport outputs from (within) the vertex shader. Fortunately, this can be solved by performing a *screen-space transformation* of the geometry, together with *user-defined clipping planes*. The screen-space transformation procedure used in this paper is similar to that in [Trapp and Döllner 2010], but it is performed in the vertex shader instead of in the geometry shader. As illustrated in Figure 3, the extent of a single viewport in OpenGL will range from $[-1, 1]$ in normalized device coordinates (NDC) for both the x - and y -coordinates. In order to

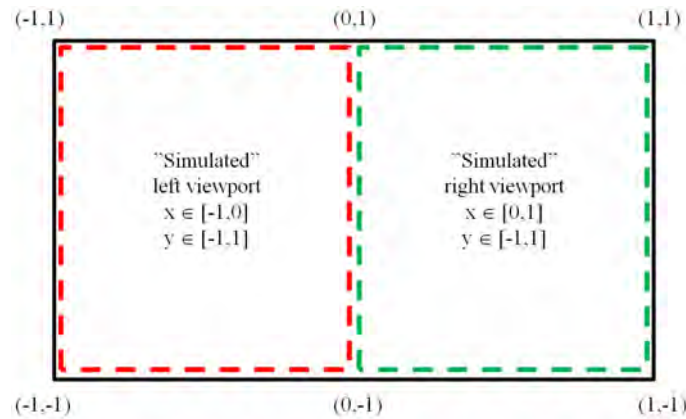


Figure 3. The extent of the simulated left (red) and right (green) viewports, as well as the real viewport (black) in normalized device coordinates (NDC).

simulate the concept of two side-by-side viewports in the vertex shader, the idea is to perform a screen-space transformation so that the x -coordinates of the left- and right-eye geometry will range from $[-1, 0]$ and $[0, -1]$, respectively, in NDC. Listing 1 provides GLSL vertex shader code for this process. A vertex $v(x, y, z, w)$ is first transformed into 4D homogenous clip space (CS) by the left or right model-view-projection (MVP) matrix. This is followed by a division by the homogenous vector component w in order to further transform it into NDC. In NDC, the x -coordinate is then mapped from $[-1, 1]$ to either $[-1, 0]$ (left) or $[0, 1]$ (right). Finally, the NDC-transformed vertex has to be transformed back to CS. In addition, a user-defined clip plane is used to restrict rendering within either one of the two simulated viewports. The code in Listing 1 assumes that a single full screen viewport has been previously defined on the client side.

```
#version 430 core
layout (location = 0) in vec3 position3;
//ViewProj matrix (left and right):
uniform mat4 viewProjMatrixLeft, viewProjMatrixRight;
uniform mat4 modelMatrix; //The Model matrix
//Frustum plane coefficients in World-Space:
uniform vec4 leftEyeRightPlaneWS, rightEyeLeftPlaneWS;
void main() {
    vec4 vertPos = vec4(position3, 1.0);
    vec4 vertPosWS = modelMatrix * vertPos;

    if(gl_InstanceID < 1) { //Left eye
        vec4 v = viewProjMatrixLeft * vertPosWS; //Transform to CS
        vec4 vPosNDC = v/v.w; //...and further to NDC
        float xNew = (vPosNDC.x-1.0)/2.0; //X from [-1,1] to [-1,0]
        vPosNDC.x = xNew;
        gl_Position = vPosNDC*v.w; //Transform back to CS
        //Additional clip plane to the right
        gl_ClipDistance[0] = dot(vertPosWS, leftEyeRightPlaneWS);
    }
    else { //Similar code as above, but for the right eye
        vec4 v = viewProjMatrixRight * vertPosWS;
        vec4 vPosNDC = v/v.w;
        float xNew = (vPosNDC.x+1.0)/2.0; //X from [-1,1] to [0,1]
        vPosNDC.x = xNew;
        gl_Position = vPosNDC*v.w;
        gl_ClipDistance[0] = dot(vertPosWS, rightEyeLeftPlaneWS);
    }
}
```

Listing 1. GLSL vertex shader illustrating stereo instancing.

Depending on hardware, the screen-space transformation and custom clipping in Listing 1 can be omitted in order to simplify the vertex shader. Both Nvidia and AMD

provide OpenGL extensions that allow access to multiple defined viewports in the vertex shader in a similar fashion as for the geometry shader (i.e., `NV_viewport_array2` and `AMD_vertex_shader_viewport_index`).

Still, as in the case with stereo duplication in the geometry shader, stereo instancing does not reduce the number of triangles that needs to be rendered every frame. As a consequence, the sheer amount of geometry that needs to be transformed, rasterized, and shaded every frame, may very well become the limiting factor.

3.1. Integration with Occlusion Culling and Conventional Instancing

In [Johansson 2013], a combination of occlusion culling and hardware-accelerated geometry instancing was used in order to provide efficient real-time rendering of BIMs. In essence, CHC++ [Mattausch et al. 2008], an efficient occlusion culling algorithm based on hardware-accelerated occlusion queries, was extended to support instanced rendering of unoccluded replicated geometry.

The original CHC++ algorithm takes advantage of spatial and temporal coherence in order to reduce the latency typically introduced by using occlusion queries. The state of visibility from the previous frame is used to initiate queries in the current frame (temporal coherence) and, by organizing the scene in a hierarchical structure (i.e., bounding volume hierarchy), it is possible to test entire branches of the scene with a single query (spatial coherence). While traversing a scene in a front-to-back order, queries are only issued for previously invisible interior nodes (i.e., groups of objects) and for previously visible leaf nodes (i.e., singular objects) of the hierarchy. The state of visibility for previously visible leaves is only updated for the next frame, and they are therefore rendered immediately (without waiting for the query results to return). However, the state of visibility for previously invisible interior nodes is important for the current frame, and they are not further traversed until the query results return. By interleaving the rendering of (previously) visible objects with the issuing of queries, the algorithm reduces idle time due to waiting for queries to return.

To integrate hardware-accelerated instancing within this system, the visibility knowledge from the previous frame is used to select candidates for instanced rendering in the current frame. That is, replicated geometry found visible in frame n is scheduled for rendering using instancing in frame $n + 1$. For viewpoints with many objects visible, the proposed solution was shown to offer speed-ups in the range of 1.25x-1.7x compared to only using occlusion culling, mainly as a result of reducing the number of individual draw calls.

When considering ways to adapt the work presented in [Johansson 2013] to a stereo setup, the properties of the stereo instancing technique offer great opportunities. As it turns out, the combination of single-pass stereo-pair generation, hardware-accelerated occlusion queries, and a split-screen setup becomes an almost perfect fit. With a single depth buffer used for both the left and right eye, only a single occlusion

query is ever needed per visibility test. That is, when a bounding box representing an interior or leaf node of the spatial hierarchy is simultaneously rendered to both the left and right viewport, a single occlusion query will report back a positive number if either one of the representations (i.e., left or right) turns out to be visible. So, compared to a two-pass approach, not only the number of draw calls and state changes but

```
#version 430 core
#extension GL_EXT_gpu_shader4 : require
#extension GL_NV_viewport_array2 : require
layout (location = 0) in vec3 position3;

//ViewProj matrix (left and right):
uniform mat4 viewProjMatrixLeft, viewProjMatrixRight;
uniform samplerBuffer M; //Matrices
uniform samplerBuffer I; //Indices
uniform int offset; //Per-geometry type offset into I

void main() {
    int leftOrRight = gl_InstanceID%2;
    float glInstanceIDF = float(gl_InstanceID);
    float instanceID_Stereo = int(floor(glInstanceIDF/2.0));
    int instance_id_offset = offset+instanceID_Stereo;
    float offsetF = texelFetchBuffer( I, instance_id_offset).x;
    float startPosF = offsetF*4.0;
    int startPosInt = int(startPosF);

    mat4 modelMatrix = mat4(texelFetchBuffer(M, startPosInt),
                           texelFetchBuffer(M, startPosInt+1),
                           texelFetchBuffer(M, startPosInt+2),
                           texelFetchBuffer(M, startPosInt+3));

    vec4 vertPos = vec4(position3, 1.0);
    vec4 vertPosWS = modelMatrix * vertPos;

    if(leftOrRight < 1){ //Left eye
        gl_ViewportIndex = 0;
        gl_Position = viewProjMatrixLeft * vertPosWS;
    }
    else{ //Right eye
        gl_ViewportIndex = 1;
        gl_Position = viewProjMatrixRight * vertPosWS;
    }
}
```

Listing 2. GLSL vertex shader illustrating stereo instancing combined with conventional geometry instancing.

also the number of occlusion tests becomes reduced by a factor of two and the culling efficiency is only marginally reduced (i.e., an object visible in either viewport will be scheduled for rendering into both).

In addition, it is straightforward to extend stereo instancing to also support conventional (dynamic) instancing as proposed in [Johansson 2013]. Listing 2 provides an example vertex shader that generalizes the use of instancing for both geometry replication as well as stereo-pair generation. For a complete picture of this approach the reader is referred to the original Johansson paper. Here, a shared array of indices (I) is used to locate each instance's transformation matrix, encoded in a single, shared array (M). Without stereo instancing enabled, the index to fetch corresponds to `gl_InstanceID` plus a per-geometry type offset (every geometry type gets a certain offset into I). With stereo instancing enabled, the instance count is increased by a factor of two, however, the composition of the index array (I) remain unchanged. First, the modulus operator (%) is used to distinguish between left and right instances, i.e., even numbers go to the left viewport and odd to the right. Second, the `gl_InstanceID+offset` is divided by two, and the integer part is used to lookup and fetch the instance's transformation matrix (the model matrix is the same for both the left and right eye). Additionally, the code in Listing 2 takes advantage of the `NV_viewport_array2` extension in order to access multiple viewports in the vertex shader, thereby removing the need for a screen-space transformation as well as custom clipping.

3.2. Batching of Non-instanced Geometry

Even with occlusion culling and the instancing-based approaches to reduce the number of draw calls, certain viewpoints may still require a large number of individual draw calls. However, many non-instanced objects in a BIM, such as walls, contain very few triangles, which make them suitable for geometry batching. In the case of wall objects, the overall low triangle count per object is a result of each wall segment—straight or curved—being considered a singular object in BIM authoring systems (i.e. Autodesk Revit or ArchiCAD). As a result, the number of wall objects represents a significant amount of the total number of objects contained in a BIM while the number of wall triangles only represents a fraction of the total amount of triangles in the model.

As BIMs contain detailed information about each individual object (i.e., meta-data), it becomes trivial to collect and batch wall geometry by material during scene loading. Although more sophisticated methods could be used to create optimal sets of wall geometry, they are simply batched by material and the centroid position of the walls bounding box (i.e., spatial coherence) in order to form clusters of approximately 7500 triangles each.

4. Performance Evaluation

The different stereo rendering techniques have been implemented in a prototype BIM viewer and tested on three different BIMs taken from real-world projects (Figure 4). All three models (dormitory, hotel, and office building) were created in Autodesk Revit 2014 and represent buildings that exist today or are currently being constructed. Although the hotel model contains some structural elements, they are primarily architectural models; as such, no mechanical, electrical, or plumbing (MEP) data is present. However, all models contain furniture and other interior equipment. In Table 1, related statistics for each model are shown. Please note both the large number of instances as well as small number of triangles per batch for the wall objects, which motivates the use of hardware-accelerated instancing as well as geometry batching of wall objects.

All the tests were performed on a laptop equipped with an Intel Core i7-4860HQ CPU and an Nvidia GeForce GTX 980M GPU with Nvidia graphics driver 361.43 in-

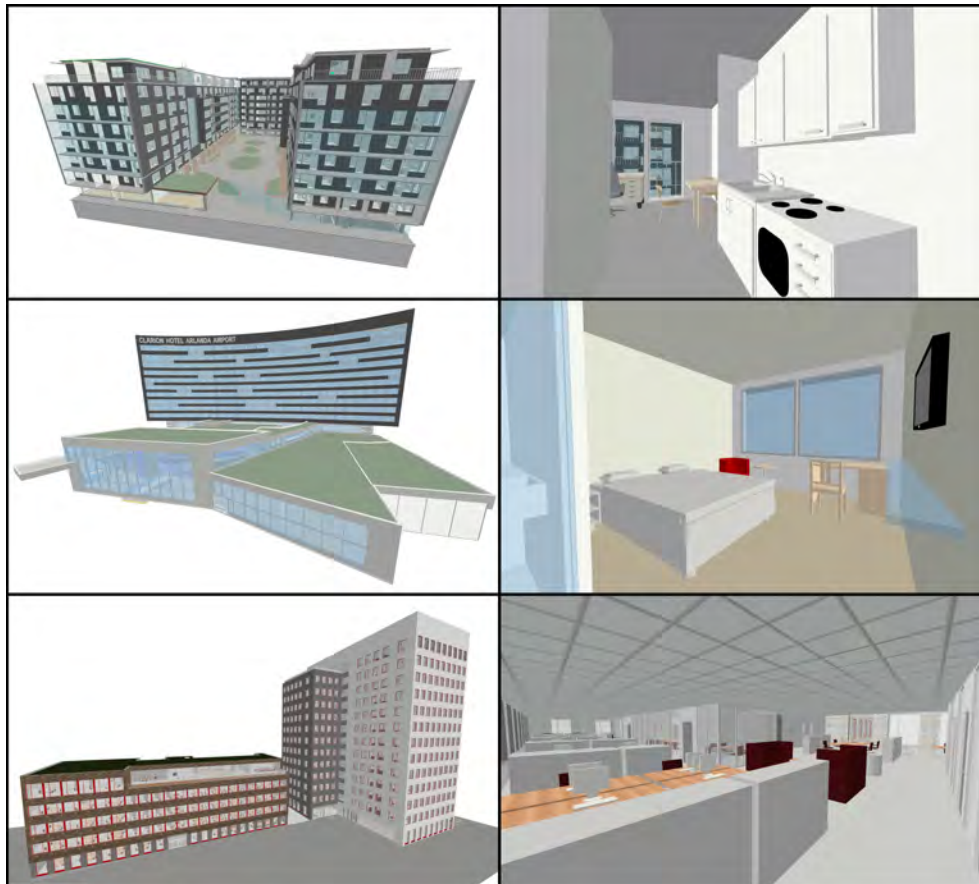


Figure 4. Exterior and interior views of the different test models. From top to bottom: dormitory, hotel, and office building.

	Dormitory	Hotel	Office
# of triangles (total)	11,737,251	7,200,901	12,454,919
# of objects (total)	17,674	41,893	18,635
# of batches (total)	34,446	60,955	26,015
# of instanced batches / total # of batches	0.60	0.56	0.81
# of wall batches / total # of batches	0.38	0.37	0.12
# of wall triangles / total # of triangles	0.014	0.04	0.004
Avg. # of triangles per wall batch	12	13	16

Table 1. Model statistics for the three different BIMs.

stalled. The resolution was set to 2364×1461 pixels (off-screen buffer) with 4x multisample antialiasing. All materials use a very simple N dot L diffuse color/texture shader.

Upon model loading, a bounding volume hierarchy (BVH) is constructed according to the surface area heuristics (SAH). Unless otherwise stated, the leaf nodes in this hierarchy represent the individual building components, such as doors, windows, and furniture. For each object, one vertex buffer object (VBO) is constructed per material, i.e., a typical window object will be represented by two VBOs, one for the frame geometry and one for the glass geometry.

The implementation of CHC++ is based on the description and accompanying code presented in [Bittner et al. 2009]. All proposed features of the original algorithm are used, except for the *multiqueries* and *tight bounds* optimizations. A major requirement in order to provide an efficient implementation of CHC++ is the ability to render axis-aligned bounding boxes with low overhead for the occlusion tests. Primarily based on OpenGL 2.1, the code in [Bittner et al. 2009] as well as [Johansson 2013] uses OpenGL Immediate Mode (i.e., `glBegin/glEnd`). In the prototype BIM viewer, a more performance-efficient *unit-cube approach* is used instead: Every bounding box in the spatial hierarchy maintains a 4×4 transformation matrix that represents the combined translation and scale that is needed in order to form it *from* a unit-cube (i.e., a cube with length 1 centred in origo). During every querying phase of the CHC++ algorithm a single VBO, representing the geometry of a unit-cube, is used to render every individual bounding box. The unique *unit-cube transformation matrix* is submitted to the vertex shader as a *uniform* variable.

For all models, two different camera paths have been used: one interior at a mid-level floor in each building, and one exterior around each building. The exterior camera paths represent an orbital camera movement around each building while facing its center. As each building is completely visible throughout the whole animation sequence, it serves as a representative worst case scenario, regardless of culling strategy.

For all models and animation paths, several different culling and stereo rendering configurations have been combined and evaluated. The abbreviations used in the

tables and figures should be read as follows: VFC for view frustum culling, OC for occlusion culling, HI for dynamic hardware-accelerated geometry instancing, BW for batching of wall geometry per material, TP for two-pass stereo rendering, GS for stereo duplication in the geometry shader, SI for stereo instancing, SI_NV for stereo instancing using the `NV_viewport_array2` extension, and BF for brute-force rendering. In the brute-force rendering case (BF) non-instanced geometry is batched together based on materials and instanced geometry is rendered using conventional hardware-accelerated instancing. Also, no occlusion culling is present. As correct depth ordering becomes difficult to maintain with the use of instancing, HI and BF render semi-transparent geometry using the weighted average transparency rendering technique [Bavoil and Myers 2008]. In all other cases, semi-transparent geometry is rendered after opaque objects in a back-to-front order using alpha blending.

4.1. Stereo Rendering with View Frustum Culling

In Table 2, average and maximum frame times are presented for the different camera paths when view frustum culling is combined with the different stereo rendering techniques. These results are mainly to illustrate the benefit of using stereo instancing without any additional acceleration strategies.

With only view frustum culling, these models are primarily CPU-bound due to a large number of draw calls in relation to the total number of triangle that are being

Stereo Rendering Technique (all modes use view frustum culling)	GeForce GTX 980M	
	Interior	Exterior
Dormitory		
Two pass (TP)	14.3 / 39.5	51.8 / 68.4
Geometry shader duplication (GS)	12.8 / 26.0	33.7 / 36.7
Stereo instancing (SI)	11.0 / 21.5	27.3 / 34.9
Stereo instancing using NV extension (SI_NV)	8.4 / 20.4	25.0 / 34.4
Hotel		
Two pass (TP)	53.0 / 82.7	91.9 / 100.8
Geometry shader duplication (GS)	29.1 / 42.0	46.3 / 49.1
Stereo instancing (SI)	29.2 / 42.2	47.1 / 49.9
Stereo instancing using NV extension (SI_NV)	28.9 / 41.5	46.2 / 48.9
Office		
Two pass (TP)	6.4 / 17.6	39.7 / 52.8
Geometry shader duplication (GS)	7.1 / 17.7	35.8 / 36.5
Stereo instancing (SI)	6.3 / 15.1	29.0 / 30.0
Stereo instancing using NV extension (SI_NV)	5.0 / 11.1	20.1 / 22.2

Table 2. Comparison of average/maximum frame time in milliseconds for different rendering techniques, models, and camera paths. Numbers in bold represent the lowest numbers encountered for each test setup.

drawn (i.e., small batch size). Because of this, stereo instancing becomes a much more efficient approach. Focusing on the exterior paths, the average frame times are reduced by 27–52% compared to a two-pass alternative. It also becomes clear that the `NV_viewport_array2` extension is preferable, not only in terms of implementation simplicity, but also in terms of performance. Although not to such a high degree, a speed-up is also recorded for the geometry shader-based approach.

Still, even if the stereo instancing technique offers a significant performance increase compared to both a two-pass as well as geometry shader-based approach, the performance is not enough in order to provide sufficiently high frame rates, even for the interior walkthroughs. Consequently, to be able to guarantee a smooth, interactive experience for typical BIMs, additional acceleration techniques are needed.

Stereo Rendering Technique (all modes except brute force use occlusion culling)	GeForce GTX 980M	
	Interior	Exterior
Dormitory		
Two pass	2.4 / 4.1	7.6 / 10.4
Geometry shader duplication	2.1 / 3.7	6.2 / 9.8
Stereo instancing	2.0 / 3.3	5.4 / 8.1
Stereo instancing (NV extension)	1.9 / 3.3	4.6 / 7.3
Stereo instancing (NV extension) + Instancing + Batching	2.6 / 4.0	5.2 / 7.0
Brute force	10.9 / 15.6	17.4 / 18.2
Hotel		
Two pass	4.0 / 5.5	27.6 / 47.9
Geometry shader duplication	3.9 / 5.3	15.0 / 25.9
Stereo instancing	3.5 / 5.1	14.7 / 25.1
Stereo instancing (NV extension)	3.1 / 4.3	14.2 / 24.5
Stereo instancing (NV extension) + Instancing + Batching	3.3 / 4.5	7.5 / 10.9
Brute force	12.0 / 13.9	11.9 / 13.1
Office		
Two pass	2.7 / 5.0	11.5 / 19.2
Geometry shader duplication	2.2 / 4.4	8.2 / 14.3
Stereo instancing	2.2 / 4.4	7.0 / 11.8
Stereo instancing (NV extension)	2.1 / 4.0	6.2 / 10.7
Stereo instancing (NV extension) + Instancing + Batching	2.3 / 4.2	6.2 / 9.0
Brute force	10.4 / 14.3	19.2 / 19.8

Table 3. Comparison of average/maximum frame time in milliseconds for different rendering techniques, models, and camera paths. Numbers in bold represent the lowest numbers encountered for each test setup.

4.2. Stereo Rendering with Occlusion Culling

In Table 3, average and maximum frame times are presented for the different camera paths when occlusion culling is combined with the different stereo rendering techniques. Focusing on the interior paths first, it becomes clear that occlusion culling alone is sufficient to provide the required frame rates. In fact, the maximum frame time recorded for any of the interior paths is never above 6 ms, regardless of stereo rendering technique. As these models features fairly occluded interior regions, the CHC++ algorithm can efficiently reduce the amount of geometry that needs to be rendered.

Still, when inspecting these numbers in more detail, it can be seen that stereo instancing is able to reduce both average as well as maximum frame times between 20–23% compared to a two-pass approach and 5–21% compared to the geometry shader-based approach. When occlusion culling and stereo instancing are further complemented with batching of walls (BW) and geometry instancing (HI), the performance results do not show the same consistency. Although the maximum frame times are consistently below that of the two-pass approach, stereo instancing is outperformed by geometry shader-based duplication for the dormitory model. Nevertheless, in this context, it is also important to note that it is the batching of walls—and not the geometry instancing technique—that is the main cause of the increased rendering time. In fact, the combination of occlusion culling, stereo instancing, and geometry instancing give equal or actually slightly better performance compared to only using occlusion culling and stereo instancing for the interior paths. However, batching of wall geometry was primarily introduced in order to reduce the number of draw calls for viewpoints when many objects are visible. As such it does not become equally beneficial for interior viewpoints.

In order to provide a better understanding of the performance characteristics, Figure 5 presents frame timings for the interior paths for the dormitory and for the office building. Most notably, these graphs show the huge difference in performance compared to a brute-force alternative (BF), regardless of stereo rendering technique. Although the brute-force approach is able to deliver frame times below 11.1 ms during parts of the interior camera paths, this level of interactivity cannot be guaranteed.

Focusing instead on the exterior paths in Table 3, the gain of the combined method becomes even more apparent. With all the techniques activated (OC+SI_NV+BW+HI), the average frame times are reduced by 32–73% compared to the two-pass approach and 16–50% compared to geometry shader-based duplication. To give a better overview of the performance behavior, Figure 6 presents frame timings for the exterior camera paths for the three different models. Here it can be seen how the different techniques complement each other depending on which model is rendered. For the dormitory and office building models, stereo instancing provides distinct performance gains compared to both two-pass rendering as well as the geometry shader-based ap-

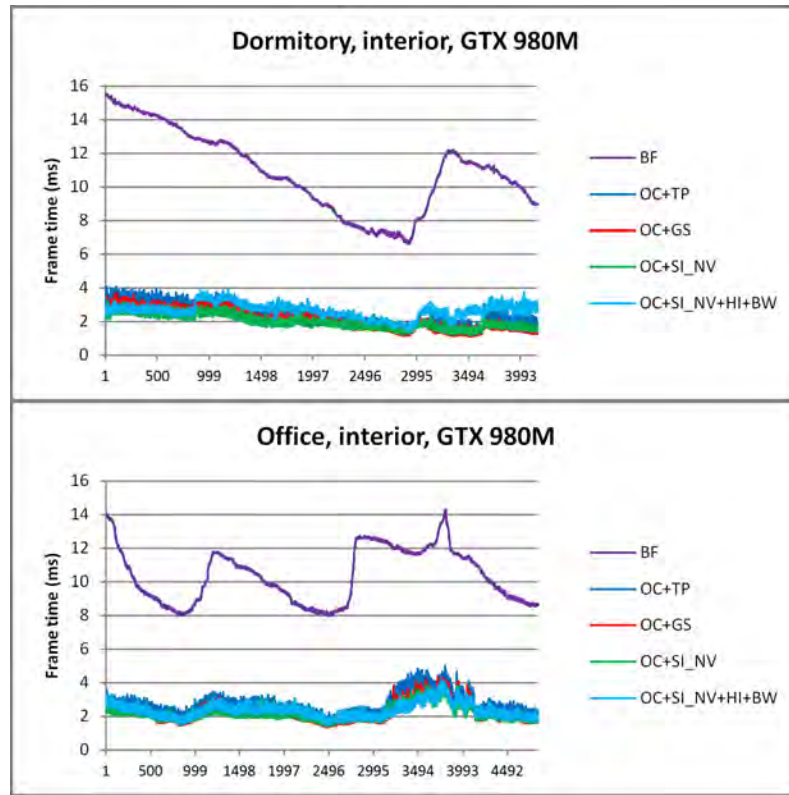


Figure 5. Frame times (ms) for the interior path of the dormitory (top) and office (bottom) buildings (BF=Brute force, OC=Occlusion culling, TP=Two-pass stereo rendering, GS=Geometry shader stereo duplication, SI_NV=Stereo instancing using NV extension, HI=Geometry instancing, BW=Batching of wall geometry).

proach. However, for these models, there is little additional to gain from conventional geometry instancing, and batching of walls. In fact, due to a high level of occlusion also in the exterior views of the student model, the average frame times are actually lower *without* geometry instancing and batching of walls activated. For the hotel model, on the other hand, the behavior is somewhat reversed. Although stereo instancing provides huge performance gains compared to two-pass rendering, the frame times are essentially equal to that of the geometry shader-based technique. Even with occlusion culling, the exterior views of the hotel model contain a huge number of visible objects and, therefore, become CPU-bound due to the large number of individual draw calls. Both stereo instancing as well as geometry shader-based duplication only address this to a certain degree. In order to reach the target frame rate, the number of draw calls has to be further reduced, which is exactly what both batching of walls and geometry instancing does. To better illustrate the individual effects of geometry instancing (HI) and batching of walls (BW), Figure 6 (middle) also presents the frame timings when these two techniques are added separately to occlusion culling

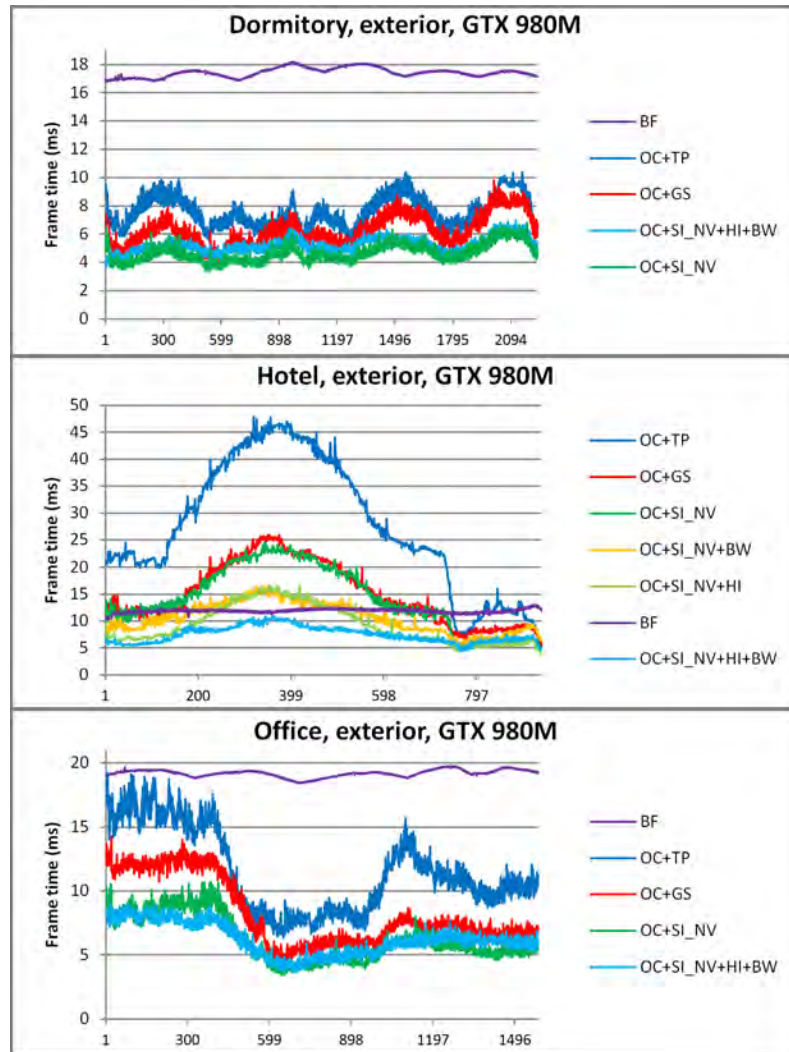


Figure 6. Frame times (ms) for the exterior path of the dormitory (top), hotel (middle) and office (bottom) buildings (BF=Brute force, OC=Occlusion culling, TP=Two-pass stereo rendering, GS=Geometry shader stereo duplication, SI_NV=Stereo instancing using NV extension, HI=Geometry instancing, BW=Batching of wall geometry)

and stereo instancing (i.e., OC+SI_NV+BW and OC+SI_NV+HI, respectively). It can thus be seen that the combination of these two techniques is required in order to reach a target frame rate of 90 Hz for the hotel model.

Another interesting aspect of the technique is that the brute-force approach is actually surprisingly close in delivering sufficient rendering performance for the hotel model. However, as seen from the results from the other two models, the brute-force alternative is clearly not scalable, especially if we also consider more complex pixel shaders. In comparison, the proposed combination of query-based occlusion culling,

stereo instancing, batching of walls, and geometry instancing is able to provide the required level of performance for all of the tested models.

5. Conclusion

In this paper the stereo instancing rendering technique has been described and further investigated. The technique is very well suited for integration with occlusion query-based occlusion culling as well as conventional geometry instancing and has been shown to outperform traditional two pass stereo rendering approach, geometry shader-based stereo duplication, as well as brute-force stereo rendering of typical BIMs on recent hardware. Although occlusion culling alone turned out to provide sufficient rendering performance for interior rendering of typical BIMs, the combination of techniques proved vital in order to guarantee required frame rates also for exterior viewpoints.

References

- BAVOIL, L., AND MYERS, K. 2008. Order independent transparency with dual depth peeling. *NVIDIA OpenGL SDK*, 1–12. URL: http://developer.download.nvidia.com/SDK/10/opengl/src/dual_depth_peeling/doc/DualDepthPeeling.pdf. 11
- BITTNER, J., MATTAUSCH, O., AND WIMMER, M. 2009. Game-engine-friendly occlusion culling. In *SHADERX7: Advanced Rendering Techniques*, W. Engel, Ed., vol. 7. Charles River Media, Boston, MA, Mar., ch. 8.3, 637–653. URL: <http://www.cg.tuwien.ac.at/research/publications/2009/BITTNER-2009-GEFOC/>. 10
- HILLAIRE, S. 2012. Improving performance by reducing calls to the driver. In *OpenGL Insights*, P. Cozzi and C. Riccio, Eds. A K Peters/CRC Press 2012, Natick, MA, 353–364. URL: <http://www.crcnetbase.com/doi/abs/10.1201/b12288-30>, doi:10.1201/b12288-30. 3
- JOHANSSON, M., ROUPÉ, M., AND BOSCH-SIJTSEMA, P. 2015. Real-time visualization of building information models (BIM). *Automation in Construction* 54, 69–82. URL: <http://dx.doi.org/10.1016/J.AUTCON.2015.03.018>, doi:10.1016/j.autcon.2015.03.018. 2
- JOHANSSON, M. 2013. Integrating occlusion culling and hardware instancing for efficient real-time rendering of building information models. In *GRAPP 2013: Proceedings of the International Conference on Computer Graphics Theory and Applications, Barcelona, Spain, 21-24 February, 2013.*, SciTePress, Lisbon, 197–206. URL: http://publications.lib.chalmers.se/records/fulltext/173349/local_173349.pdf, doi:10.5220/0004302801970206. 2, 6, 8, 10
- MARBACH, J. 2009. Gpu acceleration of stereoscopic and multi-view rendering for virtual reality applications. In *Proceedings of the 16th ACM Symposium on Virtual Reality*

Software and Technology, ACM, 103–110. URL: <http://dl.acm.org/citation.cfm?id=1643953>, doi:10.1145/1643928.1643953. 3

MATTAUSCH, O., BITTNER, J., AND WIMMER, M. 2008. Chc++: Coherent hierarchical culling revisited. *Computer Graphics Forum* 27, 2, 221–230. URL: <http://dx.doi.org/10.1111/j.1467-8659.2008.01119.X>, doi:10.1111/j.1467-8659.2008.01119.x. 6

TRAPP, M., AND DÖLLNER, J. 2010. Interactive Rendering to View-Dependent Texture-Atlases. In *Eurographics 2010 - Short Papers*, The Eurographics Association, Aire-la-Ville, Switzerland. URL: <http://dx.doi.org/10.2312/EGSH.20101053>, doi:10.2312/egsh.20101053. 4

VLACHOS, A. 2015. Advanced VR rendering. Presentation at Game Developers Conference 2015. URL: http://alex.vlachos.com/graphics/Alex_Vlachos_Advanced_VR_Rendering_GDC2015.pdf. 2

WILSON, T. 2015. High performance stereo rendering for VR. Presentation at San Diego Virtual Reality Meetup. URL: https://docs.google.com/presentation/d/19x9XDjUvkW_9gsfsMQzt3hZbRNziVsoCEHOn4AercAc. 2

Author Contact Information

Mikael Johansson
Department of Civil and Environmental Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
jomi@chalmers.se

Mikael Johansson, Efficient Stereoscopic Rendering of Building Information Models (BIM), *Journal of Computer Graphics Techniques (JCGT)*, vol. 5, no. 3, 1–17, 2016
<http://jcgt.org/published/0005/03/01/>

Received: 2015-05-06

Recommended: 2016-03-02

Published: 2016-07-25

Corresponding Editor: Warren Hunt

Editor-in-Chief: Marc Olano

© 2016 Mikael Johansson (the Authors).

The Authors provide this document (the Work) under the Creative Commons CC BY-ND 3.0 license available online at <http://creativecommons.org/licenses/by-nd/3.0/>. The Authors further grant permission for reuse of images and text from the first page of the Work, provided that the reuse is for the purpose of promoting and/or summarizing the Work in scholarly venues and that any reuse is accompanied by a scientific citation to the Work.

