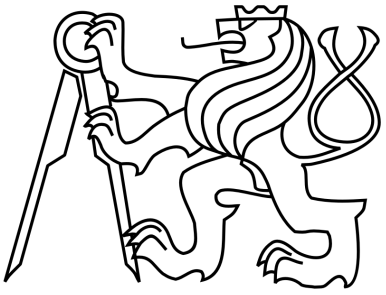

Czech Technical University

Faculty of Electrical Engineering

Department of Computer Graphics and Interaction

Open Informatics - Computer Graphics and Interaction

Master's Thesis

Architecture visualizer for distributed VR systems

Author:
Bc. Josef Kortan

Supervisor:
Ing. Zdeněk Trávníček

May 12, 2014

Declaration of Authorship

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.
I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000 Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

Signed:

Date:

"I would like to say thank you to Zdeněk Trávníček for helping me to move on and especially for inexhaustible advice in C++, design patterns and sharing all the information connected to CAVE problematics and his favourite CAVELib. Catherine Nicholson, Milan Kratochvíl, Jan Volný and Zuzana Kořínková for proofreading this thesis. František Pecháček for helping me to get the view on CG industry from artistic perspective. My family for supporting me whatever I decide to do and of course to all my friends who survived me writing this thesis and are still my friends. Thank you."

Josef Kortan

Abstract

The main topic of this thesis is an architectural visualization in distributed VR systems. It is focused on Cave automatic virtual environments. The final output is a visualizer prototype using real-time raytracing techniques with use of the NVIDIA® Optix framework. This thesis also discusses 3D rasterization techniques, because they are still indispensable parts of a real-time architectural visualization, according to a research made at the beginning.

Czech Translation

Hlavním tématem této diplomové práce je vizualizace architektury v distribuovaných VR systémech se zaměřením na CAVE. Výsledkem této práce je vizualizační protopyp založený na technologii NVIDIA® Optix využívající metody vrhání paprsků v reálném čase. Práce se ovšem také zabývá resterizačními technikami vizualizace, které jsou podle poznatků získaných během tvorby této práce stále nepostradatelnou součástí architektonické vizualizace.

Contents

List of Figures

List of Tables

Abbreviations

AV	**A**rchitectural **V**isualization (Archviz)
CAVE	**C**ave **A**utomatic **V**irtual **E**nvironment
GPGPU	**G**eneral-**P**urpose **C**omputing on **G**raphics **P**rocessing **U**nits
CUDA	**C**ompute **U**nified **D**evice **A**rchitecture
PT	**P**ath **T**racing
IBL	**I**mage **B**ased **L**ighting
HDR	**H**igh **D**ynamic **R**ange
HDRI	**HDR I**maging
OSL	**O**pen **S**hading **L**anguage
PTX	**P**arallel **T**hread E**X**ecution
BDPT	**B**i **D**irectional **P**ath **T**racing
PT	**P**ath **T**racing
MLT	**M**etropolis **L**ight **T**ransport
GI	**G**lobal **I**lumination
ERPT	**E**nergy **R**edistribution **P**ath **T**racing
IIM	**I**nstitute of **I**nter **M**edia
API	**A**pplication **P**rogramming **I**nterface
HW	**H**ard**W**are
DB	**D**ark **B**ulb
FOV	**F**ield **O**f **V**iew
SM	**S**hadow **M**apping
SV	**S**hadow **V**olumes
VRUT	**V**irtual **R**eality **U**niversal **T**oolkit
SG	**S**cene**G**raph
OC	**O**cclusion **C**ulling

LOD	**L**evel **O**f **D**etail
BVH	**B**ounding **V**olume **H**iearchy

Chapter 1

Architecture Visualization

The topic of AV is in most cases based on modeling and rendering architecture in order to create lifelike images (or videos). The final visualization of an architecture project depends on its purpose. It is obviously a different situation when you try to sell a project or to experiment with it for educational purposes.

The architects typically have a different approach how to present their work. Some of them produce images on their own using a mixture of 3D modeling software, mostly

FIGURE 1.1: Example of AV from Treevillas series. Source [1]

with postproduction software, such as Photoshop (this case mostly concerns students). Big studios have a different approach; for them it can be more productive to hire a company providing visualization solutions. When I asked Jan Tůma (architect collaborator on this project) about the rules for a good AV, the answer was quick:

"There are no rules."

As an example I have chosen the visualization presented in figure 1.1, which was created by Jacinto Monteiro (METRO CÚBICO DIGITAL[1]). To an untrained eye, it is difficult to see that the picture was completely made as a 3D visualization. A few years ago it would have taken hours to render an image like that, but thanks to GPGPU computing we are able to produce images like this one in seconds [2].

We can divide AV into the following categories:

- still renderings
- walk-through and fly-by animations (movies)
- virtual tours
- realtime 3D renderings
- panoramic renderings
- light and shadow (sciography) study renderings
- renovation renderings (photomontage)

This thesis is mostly concentrated on realtime 3D renderings for CAVE.

1.1 AV in CAVE

CAVE is an immersive virtual reality system for visualization. In the world there are many variations of this environment with projectors (or LCD panels) usually directed

[1] METRO CÚBICO DIGITAL is one of the companies providing full 3D AV solutions for clients all over the world. You can see more of their work at http://metrocubicodigital.com/.

[2] For example see renderer OTOY's Octane renderer, Blender's Cycles, AAA Studio's Furry Ball, etc. which are the new generation of GPU renderers for real production.

at different faces of a cube. The viewer stands in a cube and is able to move in a virtual world by using a controller device, usually with a head tracking system. Among other disciplines, CAVE occupies an essential place in an architectural visualization. When it is used reasonably, it can help the society to create a better place to live in. For example, it could be used to help young architects to obtain skills needed for well-designed architecture in a real environment.

1.2 Commercial Solutions

There exist many commercial solutions which are, to a greater or lesser extent, associated with visualization of any type of data, including AV. As the first one has been chosen TwinMotion which is a commercial solution used for architectural visualization by French company KA-RA. The next one is a popular game engine Unity from Danish company Unity Technologies. The last chosen one is the Brigade Engine which is a completely ray-traced solution for a game development and has its origins in Jacco Bikker's project Aurauna [2].

1.2.1 TwinMotion 2

TwinMotion (figure 1.2) is an AV software originally developed in 2005 by French company KA-RA (founded by architects Raphael Pierrat and Laurent Vidal). Its engine is developed in DirectX and it is designated for Windows desktop platforms. This engine uses real-time techniques such as SSAO, depth of field, HDR environment mapping, support for realtime shadows and many post-production effects, such as color correction, god rays, etc. The engine works on a click-and-drop basis and an user can interact with the whole scene graph of a project. TwinMotion also supports standard 3D file formats (.obj, .dae, etc.) and collaborates with Autodesk Revit. Twinmotion®2 also offers a large library of vegetation, different landscapes, indoor/outdoor furniture and cars. Twinmotion®2 also includes a sound library, because its creators believe that sound is one of the main factors influencing the complete perception of the AV [3],[4], [5], [6].

FIGURE 1.2: Twinmotion®2 screen. Source [4]

1.2.1.1 Licensing

KA-RA offers a student edition for a non-commercial academic use for 49 EUR. A network and a single user license cost 2,420 EUR.

1.2.1.2 CAVE Support

I have sent an email to the company enquiring whether they have any unannounced plans for CAVE environment support, but I have not received any response yet.

1.2.2 Unity Engine

Unity (figure 1.3) is a multi-cross-platform game engine with built-in platform for CG developed by Danish Copenhagen based company Unity Technologies. It is mainly used for developing games for desktop platforms, consoles and mobile devices. However, it is also increasingly used for different kinds of visualization, including architectural visualization. Unity engine supports bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), dynamic shadows using shadow maps, render-to-texture and full-screen post-processing effects.

FIGURE 1.3: Unity 3 development environment. Source [8]

Unity uses Direct3D (Window, Xbox 360), OpenGL (Mac, Window, Linux, PS3), OpenGL ES (Android, iOS) and proprietary APIs (Wii). Unity supports many file formats for 3D modeling softwares: 3ds MAX, Maya, Softimage, Blender, modo, ZBrush, Cinema 4D, Cheetah3D. The engine is originally written in C/C++ language and supports UnityScript (a derivation of JavaScript), C# and Boo (a Python-based language) for scripting. The shaders can be written in ShaderLab which is a special language supporting GLSL, Cg shaders or the old fixed-function pipeline. Since August 3rd, 2013, Unity allows indie developers (using Unity Free) to use realtime shadows, but only for directional lights.

Examples of an architecture visualization can be found at [7].

1.2.2.1 Licensing

Unity is provided in two versions, Unity Free (free for indie developers) and Unity Pro (commercial purposes). Unity Pro costs 1500 USD, compared to the Unity Free version (which is available for educational, academic, non-profit and commercial organizations with the total annual budget for the entire entity not in excess of 100,000 USD [10]).

FIGURE 1.4: Example of Unity getReal3D addon developed by Mechdyne. Source [9]

Unity Pro supports features like render-to-texture, occlusion culling, global illumination and special post-production effects.

1.2.2.2 CAVE support

There exist many 3rd party Unity plugins for CAVE. One of them is getReal3D (Figure 1.4) for Unity [11], developed by Mechdyne[3].

MiddleVR for Unity is another 3rd party Unity plugin, developed by Sébastien Kuntz's company "i'm in VR". This addon supports only Windows platform. [12]

1.2.3 Brigade Engine

Brigade engine (Figure 1.5) is an engine which currently connects many people from the CG world. The core of the engine is most influenced by the work of Jacco Bikker and collaborators from Delft University [13], [14].

Brigade is now owned by the Los Angeles company OTOY which has bought New Zealand based company Refracting Software (popular for their Octane GPU unbiased path-tracing renderer developed in Optix framework) and in the second half of 2014

[3]Producer of CAVELib; C library currently used in IIM for managing CAVE projections.

FIGURE 1.5: Screenshot from Brigade 2. Source: OTOY

they are planning to release a cloud computing game technology based on the new 3rd generation of Brigade engine within Amazon Cloud technology [15], [16].

Compared to the two previous engines, Brigade is unique in the way that it is not using a standard GPU rasterization, but it uses a real-time path tracing algorithm to achieve the photorealistic result. Brigade collaborates with many 2D and 3D file formats, for e.g. .obj, .dae, .blend, .3ds, .png, .targa, etc. Brigade also offers a flexible system of physically-based materials that can produce glossy, diffuse, specular (transparent or mirror) and emissive results with normal maps support.

1.2.3.1 CAVE Support

There is no known information about CAVE support. The company is currently focused on the online web cloud technologies for real-time path-tracing visualization.

Chapter 2

Background Information and Theory

In this chapter are presented different topics characterizing basic principles connected to AV. There is provived point of view from rasterization and physically based perspective.

2.1 Rendering Methods

According to my research of the current situation in real-time VA environment, there exist two basic methods that developers use for rendering. The engines are mainly based on GPU rasterization using APIs, such as OpenGL or Microsoft DirectX. On the other hand, GPGPU computing is taking more and more space in real-time visualizations with the use of computationally intensive physically based rendering methods using APIs, such as CUDA and OpenGL. These techniques are often combined to take advantages of both methods.

In the following paragraph are described the basics of path tracing algorithms. A standard 3D rasterization will not be described here. For more information about 3D rasterization, I recommend you the following resources [17], [18], [19].

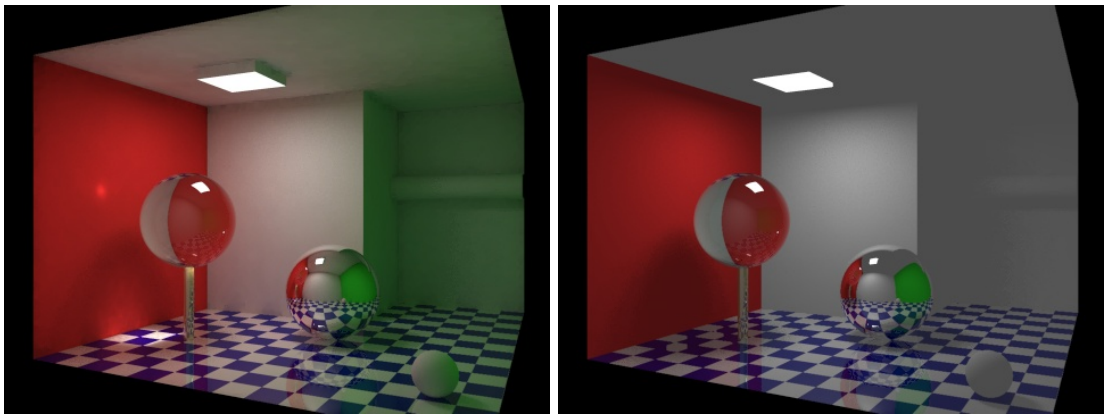

FIGURE 2.1: Comparison between GI and local illumination. Source [20]

2.1.1 Global Illumination Techniques

According to [20], GI is expression for a group of algorithms used to produce more realistic lighting in 3D rendering. These algorithms take into account light coming directly from a light source (direct illumination), but also add computation for computing rays reflected by other surfaces in a scene (indirect illumination). See the comparison in figure 2.1.

The images rendered using these techniques often appear more realistic than those using only direct illumination. However, the computation of such algorithms is computationally more expensive. Thanks to the new era of GPGPU devices, the situation is getting better and the game developers and other users are everyday closer to producing more realistic sceneries at frame-rates sufficient for real-time visualization.

There has been a large amount of research into GI algorithms. For an introduction I would personally recommend Shirley and Keith Morley's book [21], which is rather old but gives a good basics in GI. For more advanced algorithms, Physically Based Rendering [22] by Pharr, Humphreys and Hanrahan is great; it received an Academy Award (Oscar) for Technical Achievement in January 2014 [23] (and it is the first book ever that received Oscar recognition).

Last year a work by Robin Hub investigating GI algorithms [24] was also published. We also should not overlook Dietger van Antwerpen's thesis "Unbiased physically based rendering on the GPU", in which he proved that it is possible to implement algorithms such as Bi Directional Path Tracing (BDPT), Energy Redistribution Ray Tracing (ERPT) and Metropolis Light Transport (MLT) in CUDA.

FIGURE 2.2: Example of an interior produced by path tracing method. Source: OTOY

2.1.1.1 Path Tracing

Samuel Lapere [25] [1] gives a simple explanation like this: "Path tracing is a global illumination extension to the ray tracing (original Whitted ray tracing is unable to compute caustics; indirect illumination (color bleeding, ambient lighting) and produces soft shadows) algorithm for achieving photorealistic image results. It traces many rays (samples) per pixel in different directions and then takes the average value to calculate final color of each pixel. The amount of rays per pixel and their spatial distribution in a pixel sample are important to minimize noise and aliasing. When a ray hits a surface, a new ray is traced from that hitpoint in a random direction until the max depth is reached (or until a Russian roulette-like mechanism stops shooting the ray)."

It can basically simulate almost every known material (see figure 2.2), including participating media like fog, clouds, sub-surface scattering, etc. PT has also problems with caustics generation and there exist better methods to generate them, for example a progressive photon mapping.

[1] Brigade Engine developer and real-time photorealistic rendering enthusiast having a popular blog: http://raytracey.blogspot.cz/.

2.2 Materials

One of the most important aspects of the architecture visualization is the need of a material library. In real world, the architects use a "limited" amount of materials for their buildings. In the virtual environment the situation is different, because they can do with the virtual architecture's appearance basically whatever they want. Of course, material libraries for 3D modeling systems are also limited and architects should gain some basic knowledge to create their own unique materials (shaders).

This is a citation from CG architectural visualization artist Ronen Bekerman [26]:

"The real world is a highly reflective place! Pure diffuse is pretty hard to come by actually, whereas in the CG world it is very easy to diffuse your scenes to the death."

Basic principles used for material simulation are well-known techniques, such as Phong and Blinn shading accompanied by added textures. Lately the CG community started using additional techniques, such as bump mapping, normal mapping, parallax mapping or displacement mapping to achieve more realistic look.

Phong and Blinn are the simplified shading models derived from the more complex lighting models, such as bidirectional reflectance distribution function (BRDF) (see examples in figure 2.3) originally proposed in Kajiya's The Rendering Equation paper from 1986 [27] describing the basic principles of light ray interaction within space and materials. David Immel and his collaborators also produced very similar work in the same year [28].

The rendering equation is in most cases good enough for VA, but it is important to note that there are more complex techniques (functions) describing the light interaction with the material, such as bidirectional scattering distribution function, more known as a sub surface scattering (SSS). The Brigade engine is supporting SSS, but it is still a matter of an experiment.

I also came across an interesting article about real material library (Figure 2.4) in New York [30]. Over the time they have collected over 7000 different materials of material designs. Something similar also works in a virtual space, but companies do not want to create one global database and the materials (shaders) are provided within the renderer solution. This increases the troubles with material compatibility in our system, because

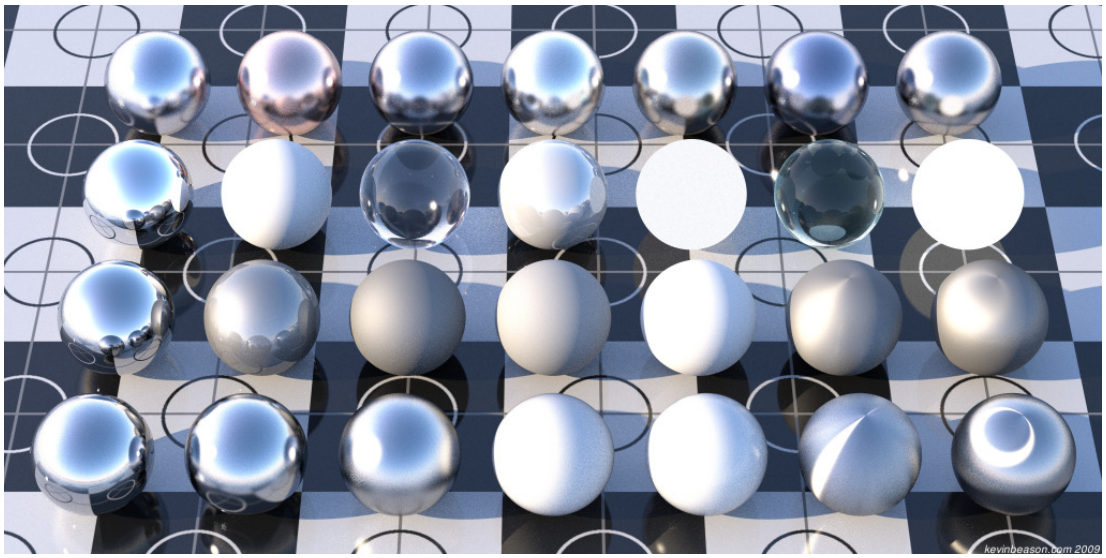

FIGURE 2.3: Example of BRDF materials with HDR lighting. Source [29]

FIGURE 2.4: Material Library in New York. Source [31]

every architect has a different aproach (3D software, renderer, ...) to achieve the final AV.

2.2.1 Shading Languages

To define the materials in OpenGL (DirectX), shading languages, such as HLSL (Cg) or GLSL, are used. In physically based renderers there is a different situation and it depends on an implementation of a particular renderer and a BRDF material function is often used.

Nowadays, OSL (Open Shading Language) is playing an important role for the material definitions. OSL has originally been developed by Sony Pictures Imageworks for their in-house movie renderers. We cannot say that it is a competitor to the GPU based shading languages, but it is still used more and more for describing materials, lights and pattern generation.

See [32] for more information. OSL is currently supported in Arnold Renderer, Blender, Cycles, Autodesk Beast and one of the most popular commercial renderers for architects, V-Ray (Chaos Group has announced its support in upcoming V-Ray 3.0 [33]).

2.3 Lighting

Without lights, there would be nothing to see. The basic lights used in computer graphics are directional lights, point lights, spot lights, etc. More information about them can be found in any good CG book [18]. It is also good to remember that these light models are far away from the reality where basically every light could be seen as an area light. In our context, the light is one of the aspects which can rapidly change the mood of the AV and can also have a negative impact on the client side.

2.3.1 Image Based Lighting

One of the standard methods to simulate environment in real-time graphics is to use cube maps. This technique is not popular in AV nowadays. IBL with HDR images is often used instead. As Paul Debevec writes: IBL is the process of illumination scenes and objects (real or synthetic) with images of light from the real world. It evolved from the reflection-mapping technique. IBL requires to have omnidirectional photographs recorded in high dynamic range. There are more references and free HDR domes textures available in [34]. IBL method is the one which can produce AV images without any additional light system from the architects and can reduce the additional pre-production of architectural models. See Figure 2.5.

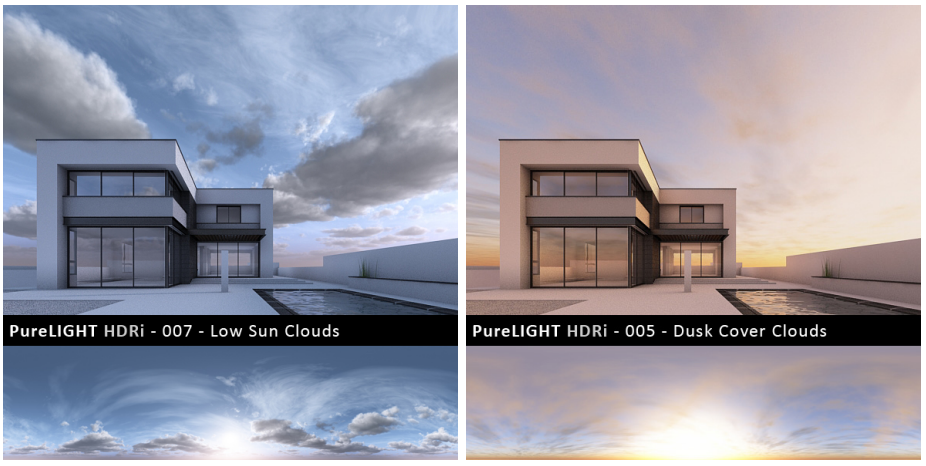

FIGURE 2.5: Example of IBL technique with diferent HDR sky dome panoramas. Source [35]

2.3.2 Shadows

Shadows are a large topic in the real-time computer graphics. In a physically based rendering there is basically no effort to create them due to the principle of the ray tracing algorithms, but in 3D rasterization the shadows are created by many evolved techniques.

2.3.2.1 Shadow Mapping

SM (projective shadowing) was introduced by Lance Williams (inventor of mip-mapping) in 1978 (article Casting curved shadows on curved surfaces). Shadow mapping involves projecting a shadow map on a geometry and comparing the shadow map values with the light-view depth of each pixel [36] (see figure 2.6). The final result highly depends on the shadow map resolution. A frequent problem of shadow mapping is an aliasing or shadow continuity glitches in its simple implementation. Many techniques have evolved to fix these problems. See [37] where is an overview of different shadow mapping techniques. For large architecture scenes there is one obvious problem with rendering the whole scene geometry into the shadow maps and sometimes the resolution is not large enough [38].

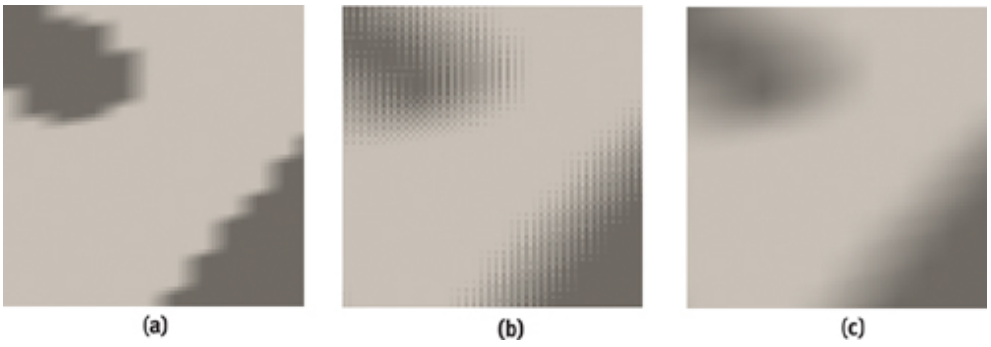

FIGURE 2.6: Illustration of anti-aliasing techniques. 1, 4, 16 samples per pixel. Source [36]

2.3.2.2 Shadow Volumes

The concept of SV was introduced by Frank Crow in 1977. It uses an accurate geometry model describing the region occluded from the light source. A shadow volumes technique often uses a stencil buffer for implementation and the geometry from light is computed on CPU. SV is more accurate and computationally expensive compared to SM. This method became popular for its usage in the video game Doom 3 [39].

I would like to refer to Daniel Šimek's master thesis [38]. He made a research of different shadow techniques and he has implemented defferred shading method into VRUT system.

2.3.3 Scenegraph

A scenegraph is a hierarchical graph data structure which is often used in 3D computer world to store entities: lights, geometry objects, materials, etc. SG is a structure that arranges a logical and often (but not necessarily) a spatial representation of a scene. The definition of a scenegraph is fuzzy, because every implementation of scene graphs is unique for a particular application. This means that there is no consensus as to what a scene graph should be [40].

As an example I have chosen Open Scene Graph (competitor to OpenSG) as a representant for the 3D rasterization engines and for demonstration of acceleration methods for faster rendering used by OSG.

OSG is probably the most popular open-source scene graph API. It is designed to run under different operating systems and even on mobile platforms using CMake build system.

Modern GPUs are able to proceed millions of vertices per second. Some applications have such a large amount of geometry that standard methods used in GPU pipeline (hidden-surface removal) are not able to render at a sufficient frame rate and we must use additional culling techniques [41].

OSG is using two main techniques: Occlusion Culling and Level of Detail.

2.3.3.1 Occlusion Culling

OC is a technique that eliminates objects invisible from the viewer, because they are blocked by other objects (before they are loaded into the GPU pipeline). OpenGL or DirectX would render the scene correctly thanks to the Z-buffer algorithm, but it is not able to recognize an occlusion, since it is processing every object (triangle) separately. OSG remembers the position of the viewer and support different algorithms. [41] There are several techniques for this scene management using quadtree, octrees, bvh structors or bsp trees. These techniques help to sort the geometry in the visibility order according to camera (viewer) position and we can determine whether the geometry belongs to the scene or not. Before such techniques can be used, there is an additional step which is called view frustum culling. This process removes the geometry if it does not belong to the current camera's view frustum. The calculation of the viewing frustum is carried out on the CPU, based on the camera projection parameters [42].

View frustum culling only disables the renderers for objects that are outside the camera's viewing area, but does not disable anything hidden from view by overdraw [43]. See figures below 2.7 and 2.8.

2.3.3.2 Level of Detail (LOD)

OSG has a level of detail node whose children have different geometry complexity. According to a rendered area of a geometry object on the screen, OSG can dynamically change the level of detail of an object on the screen and save GPU computation. This,

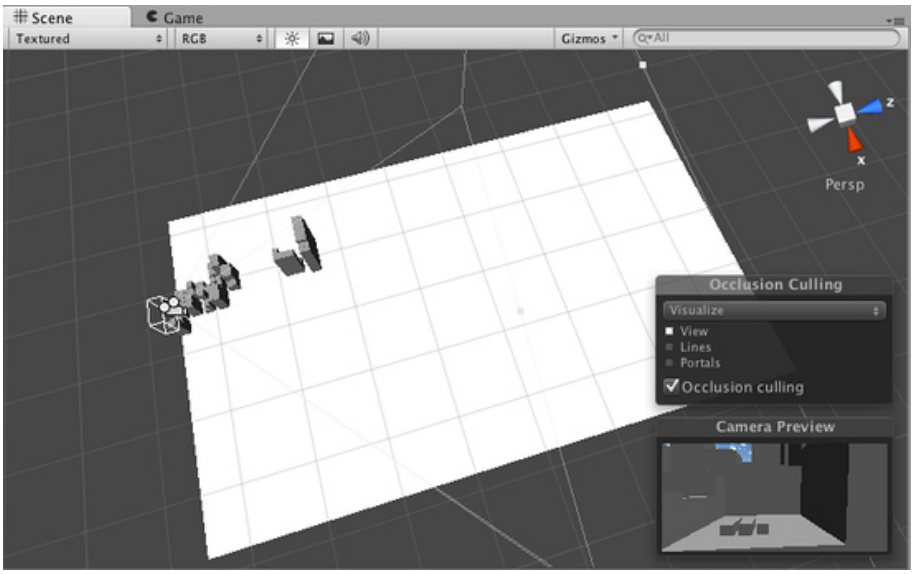

FIGURE 2.7: Scene from Unity Pro with occlusion culling technique. Source [43]

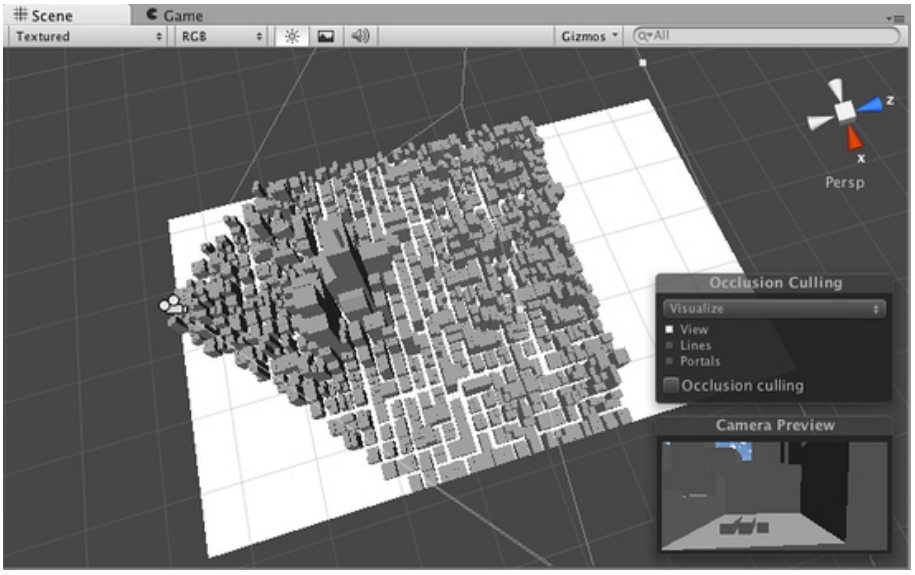

FIGURE 2.8: Scene from Unity with occlusion culling technique. Source [43]

of course, needs additional work of preparing 3D models in different geometry complexities [41].

2.3.4 Acceleration Data Structures

One of the main problems in a physically based rendering world is to achieve fast computing of the intersection with the geometry. I have chosen two basic data structures used in different varieties in nowadays rendering software. We can not exactly say which

one is the best and it is highly dependent on the geometry distribution in a rendered scene.

In the work [44] a reference to [45] is provided where three different acceleration data structures (uniform grids, kd-trees and BVHs) were compared and analyzed for GPU ray-tracing. The authors reported that the uniform grids were only superior for uniformly populated scenes which is not the case for VA, since we cannot standardize the inputs from architects. They concluded that BVHs were better for coherent rays, such as primary rays and shadow rays, while kd-trees could be more efficient on average for incoherent rays. Kd-tree needs much more memory than BVHs also.

2.3.4.1 Bounding Volume Hierarchy

In our context BVH is a tree based data structure used for geometric objects encapsulation. All these objects (triangles) are wrapped in bounding volumes (often spheres, axis-aligned bounding boxes, oriented bounding boxes, ...) and stored as leafs of the data structure. In a leaf there is usually approximately 5-10 geometric objects in a group. The number of objects depends on the particular implementation. BVH is also used for collision detection computation in physics engines [46], [47].

2.3.4.2 K-d Tree

K-d tree is a spatial data structure and can be seen as a generalization of octrees and quadtrees data structures. K represents the number of subdivided dimensions (usually 3). Instead of simultaneously dividing space in two (quadtree) or three (octree) dimensions, the k-d tree divides space along one dimension at a time (usually along x, then y, then z) [47], [48].

Chapter 3

Design and Implementation

At first, the history and situation of the CAVE visualization in IIM is described, before any improvements of the current visualizer are demonstrated. Also, in this introduction part a discussion which has led us to the current design and improvements in the Dark Bulb is presented.

3.1 History of AV in IIM

Since 2007, when the CAVE was built, there were a few projects, more or less related to AV in the CAVE placed in IIM. These projects were usually created by students in a short time during one semester period using standard OpenGL API (mostly old version OpenGL style). Almost in every case they were based on a simple shading model using Phong/Blinn shading with the addition of diffuse textures accompanied by pre-baked shadows in a texture.

These projects are not suitable for our needs, because they were made usually just as a DEMO without any plan for the updates in the future (Snowman, Physical Balls, FEL Model based on Quake engine).

Because of the need to have a proper visualizer used for different purposes (point cloud visualisation, VRML models, Wave Front Obj models), a basic CAVE Framework was developed. CAVE Framework (CF) was originally made for OpenGL API and has been constantly developed and improved. Since 2011 CF is based on the C++11 syntax and

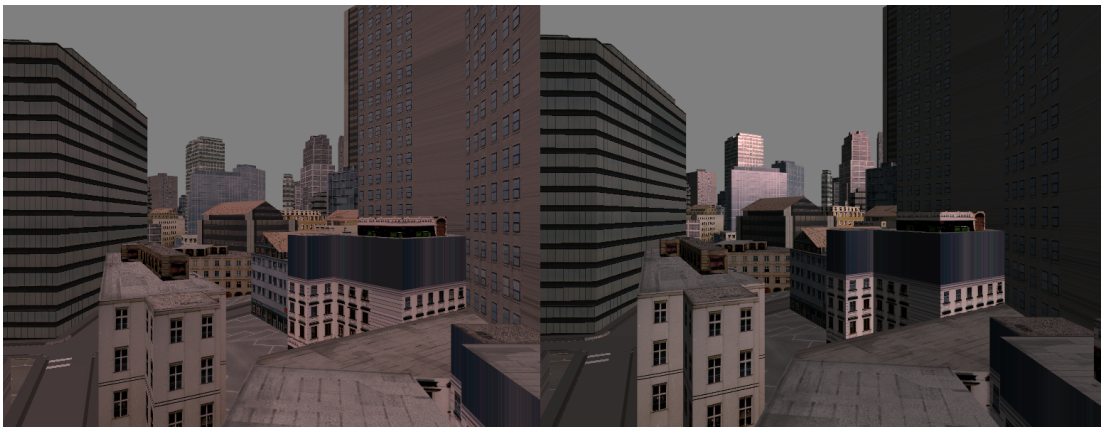

FIGURE 3.1: Phong shading vs. diffuse model in CE_cinegrid02 model.

uses new OpenGL 3.3 (and higher) style with fallback for older versions. In a version approximately a year old there was a support for .vrml, .obj and .ply formats with a basic emission shader and the texture support.

Since the August 2013 we have been mostly improving CF based on the need of Faculty of Architecture which is planning to use the CAVE for educational purposes and I have become a member of a research team.

3.2 Beginning of a Dark Bulb Project

At the beginning we have started to think about the future of the CAVE framework, making research and simultaneously improving current visualizer in order to follow the needs of architects collaborating with IIM.

One of the first improvement was made in a logical way how the CAVE framework works. The old version was tightly dependent on the OpenGL renderer and there was no elegant way how to add any new type of a renderer (DirectX, Optix Renderer, ...). Thanks to the visitor pattern a solution was created. It is modular and it is able to create a new type of renderer without any modification of the core functionality.

After this step, the aim of this work was concentrated on visual aspect of the OpenGL renderer and on improvement of the shader collaboration with the framework. The most significant difference brought the implementation of Phong shading model for directional and spot lights which where missing. See the difference at figure 3.1

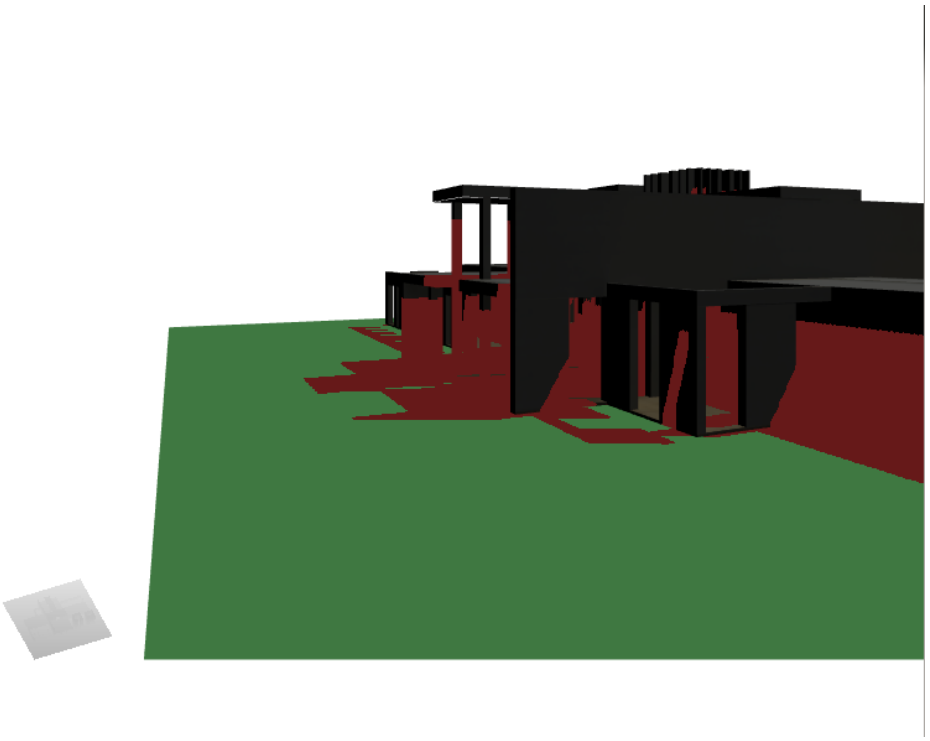

FIGURE 3.2: Basic implementation of shadow mapping technique.

Until January 2014 we had been also learning and testing an algorithm for creating more realistic shadows than the pre-baked shadows in textures. A basic 2-pass shadow maps model was implemented, but the result was not such as elegant solution as it was expected, since there were different 3D models provided and it was necessary to adapt the resolution of the light map to achieve the sufficient results 3.2.

In January the following road map (see figure 3.3) was prepared. It includes possible improvements based on the techiques from Chapter 1 and Chapter 2.

Basically, two options how to continue was considered.

The first considered solution was to build the whole pipeline for the architects on a commercial software based on the rasterization rendering techniques. For example to use Unity Engine with one of the plugins used for CAVE environments. Also, Twinmotion developers was contacted about the CAVE support problematic, but no respond have not come yet.

The second solution was to continue in the development of the current CAVE framework based more or less on the roadmap already presented. The problem was to decide which way to go (rasterisation vs. physically based rendering), since we were just two

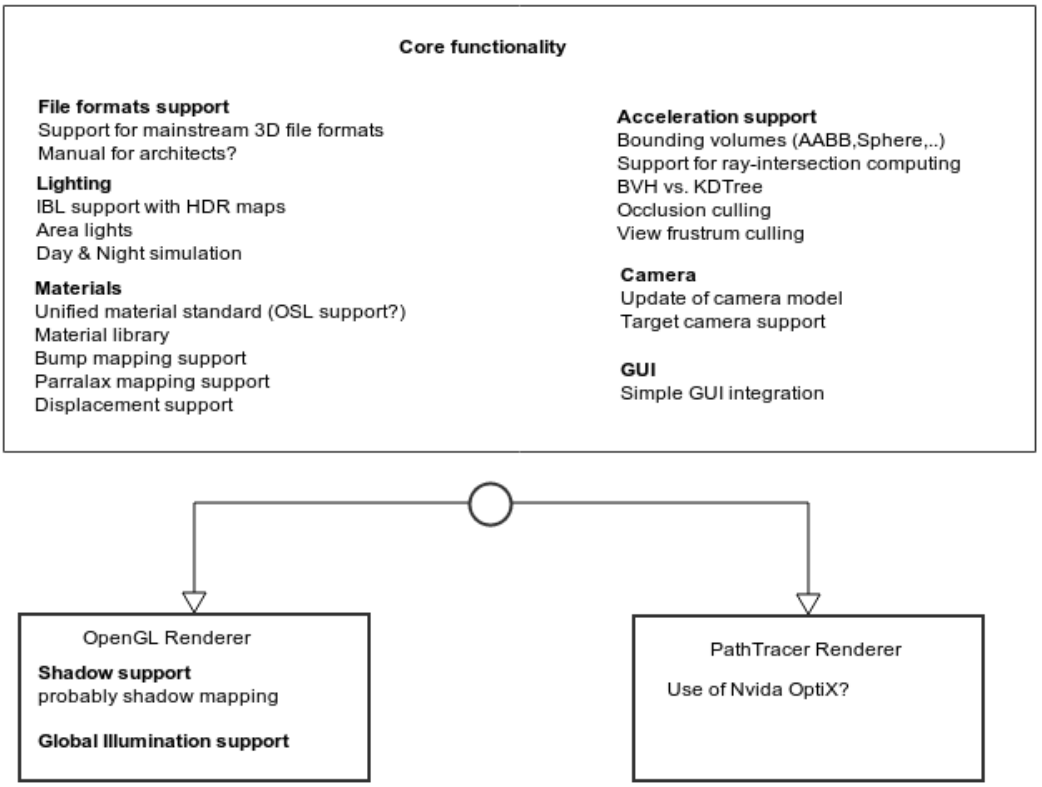

FIGURE 3.3: Draft made in January 2014 for possible improvements.

developers. After a few discussions with Jaroslav Sloup (supervisor of A4M39GPU), it was decided to experiment and to try to build a solution on the real-time ray tracing computations with the use of CUDA and NVIDIA® Optix framework. Because we think that currently this is the way to go and after seeing proofs delivered from Brigade we would like to have similar results on the multiprojections environments as well as at one screen.

3.3 Data Formats Compatibility

One of the key features which had to be solved for the AV was to support many data formats from different 3D modelling softwares. We have also decided to concentrate on the 3D formats used for a static representation of the models without any information for the animation.

After the research we decided to add a support for an Open Asset Import Library which is well-known in a 3D developer community for its compatibility with the most

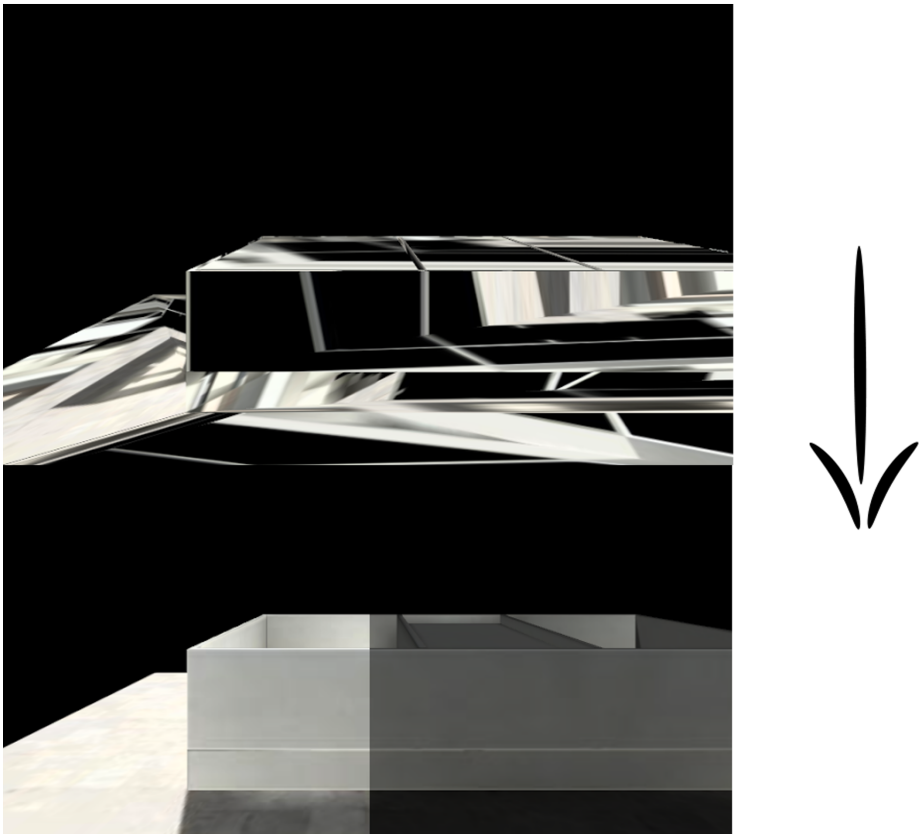

FIGURE 3.4: Improvement in a texture compatibility. On the top there is a result with the original .obj loader and on the bottom there is an Assimp obj. loader result. The author of the model is Jan Tůma.

of the mainstream 3D data formats. Assimpt does not support .vrml format which was sometimes also used for visualization in the CAVE. For that case there is a VRML loader from the previous version.

During the time of the DB development, we were mostly concentrating on the tests of WaveFront .obj formats produced from 3Ds Studio Max and Blender exporters.

One of the features which saved a lot of time was a better texture interpretation and compatibility with .obj (respectively mtl format) data format. In the figure 3.4 there is illustrated the difference between the original .obj loader and the assimp .obj loader.

3.4 Dark Bulb Framework

In this section we are going to introduce the important improvements which has been done to the original CAVE framework. First of all, the NVIDIA® Optix is introduced with the short description how the current Optix Renderer works. Than we are going to

take a closer look at the problematics of a perspective off-axis projection and to explain the sampling process and creating the final image.

But at first we should have a look at a general structure of a current DB Framework version in figure 3.5.

3.4.1 NVIDIA® Optix

OptiX is a programmable ray tracing framework for software developers to rapidly build the ray tracing applications that yield extremely fast results across NVIDIA® GPUs with a conventional C programming. Unlike a renderer with a prescribed look, or being constrained to certain data structures or a language limited to rendering, the OptiX engine is extremely general - enabling software developers to quickly accelerate whatever ray tracing task they wish and execute it on widely available hardware - all license free [49]. This is the original description provided by NVIDIA® for the Optix on the official webpage. Since the time we have decided to use this framework for our development I can personally say that it is no marketing joke and everything written is true (at least for me and the version 3.1).

Since the version 3.5, which has been released on the 6th of March 2014, there is an upgrade for the licence and NVIDIA® has special rules for the commercial products using Optix Framework.

3.4.2 Optix Renderer

Current version of the Optix renderer is based on the Whitted ray-tracer style. It supports a diffuse reflection, specular reflection and a refraction with the combination of a color get from the diffuse texture.

The renderer has also implemented HDRI IBL using .exr format (HDRI maps can be mostly seen on the examples in the Testing section) with the combination of a directional light. This was sufficent for a proof of concept but we are planning to add more types of light (spot light,point light, area light, day and night system).

We are also ready to a add bump mapping support, but we have skipped this feature for the current tests.

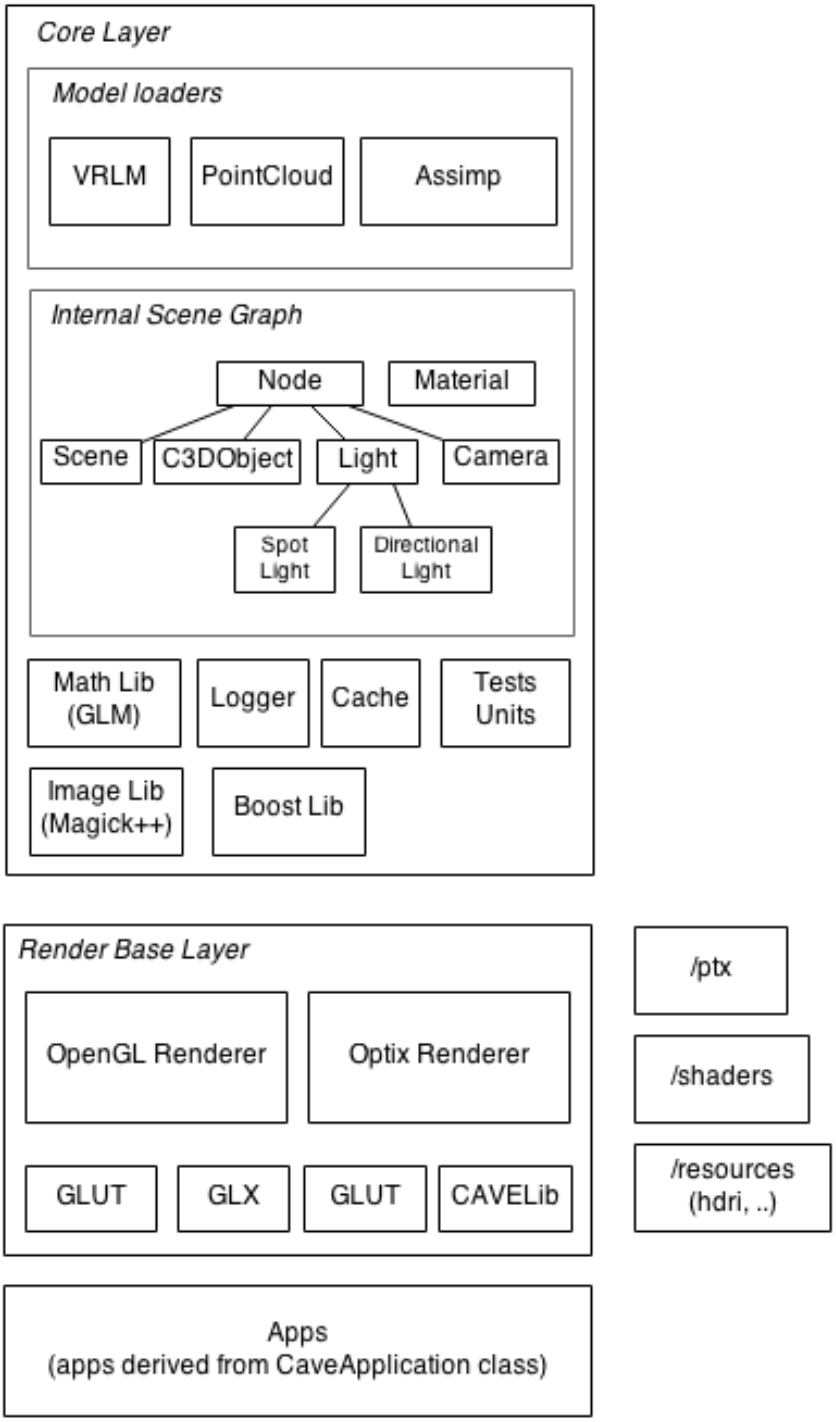

FIGURE 3.5: Simplified schema of Dark Bulb framework

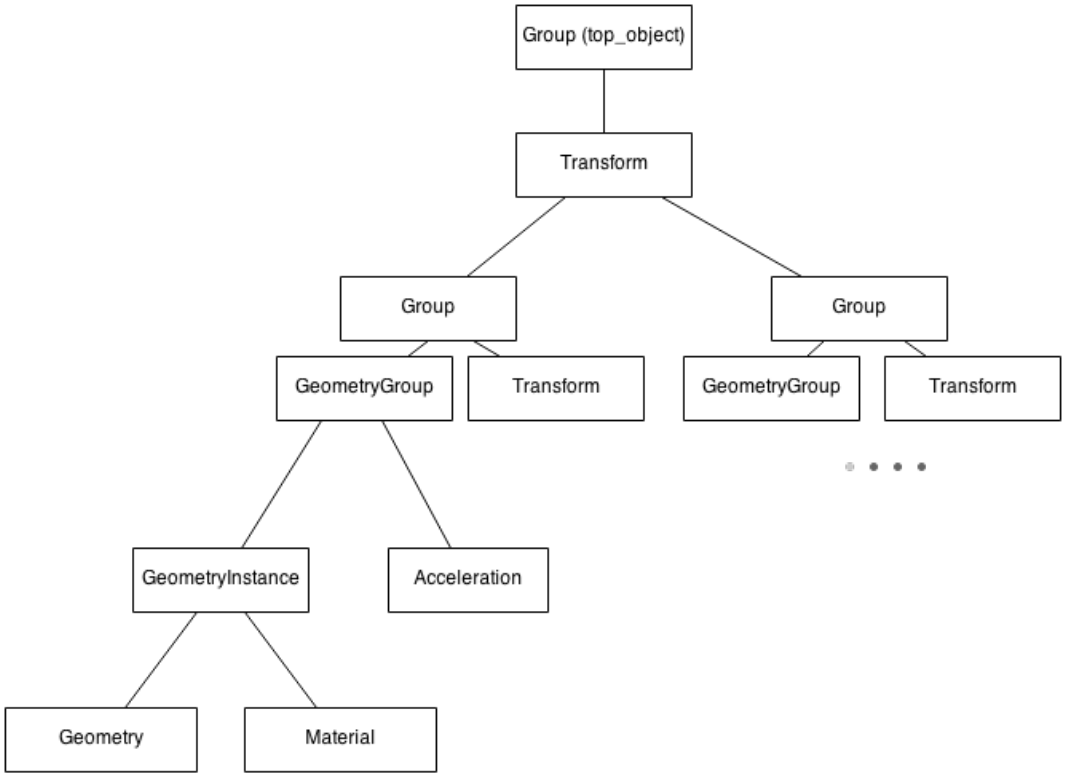

FIGURE 3.6: Scene Graph structure in Optix Renderer

Because of the structure of the NVIDIA® Optix, there is a need to re-map our internal scene graph into the form suitable for the NVIDIA® Optix renderer, since it is using its own special structures for data and accelerations. The current scene graph used for the Optix renderer can be seen in figure 3.6. As an acceleration structure Sbvh is used which is according to NVIDIA® a most sufficent structure for static scenes.

3.4.3 Perspective Off-Axis Projection

A standard situation for computing image in a ray-tracer is using some kind of the simplified model of the perspective camera projection (on-axes perspective camera projection). According to [50], these projections assume the viewer to be positioned perpendiculary in front of the view plane and looking at its center. For illustration see figure 3.7.

In virtual reality environments, however, the virtual camera often follows the tracked position of the user's head in order to create parallax effects and thus a more compelling illusion of a three-dimensional world. Since the tracked head position is not limited to the symmetry axis of the view plane, on-axis projection is not sufficient for most virtual reality environments.

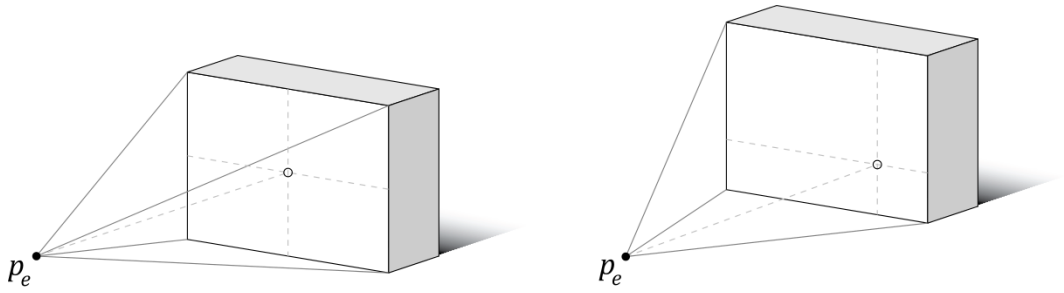

FIGURE 3.7: On-Axis projection on left. Off-Axis on the right. P_e is the position of the viewer's eye. Source [50]

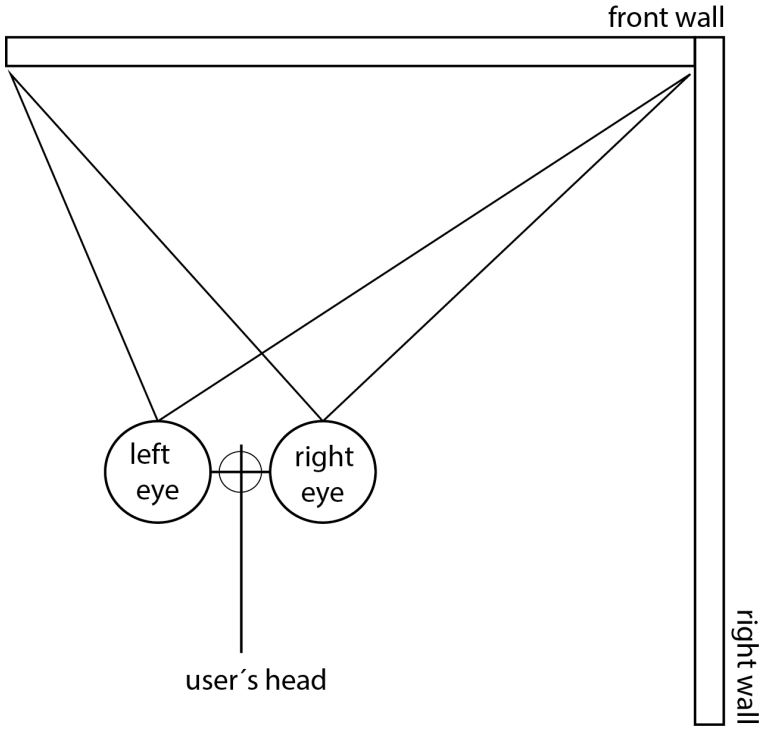

FIGURE 3.8: Off-axis projection in CAVE for front wall. User's head position is defaultly set to [0,0,0] and the left eye and right eye position is readed from CaveLib as a transformation matrix. This schema is showing the situation when the user's head position is moving from the center (notice the frustum for every eye, FOV is not symmetrical).

The difference in the CAVE (and other virtual reality environments) is that we have to take in account the viewer position of a head to achieve a 3D illusion of a space with the combination of a stereoscopic projection for every projection plane. This is also computationally expensive since we have to produce two renderered images for every projection plane.

Usually, the viewer's head position is tracked (head tracking) and the perspective projection [1] for each display is computed for a camera at the tracked position such that the viewer experiences the illusion of looking through a window into a three-dimensional world instead of looking at a flat display [51].

Currently there is no support for head tracking in our CAVE, so we suppose that the viewer's eye position is in a center of the CAVE. The illustration is in figure 3.8.

3.4.4 Progressive Refinement and Stratified Sampling

According to [52], progresive refinement is a technique which enables to compete with GPU accelerated ray tracers and to display the results much faster than with the "traditional" CPU rendering methods. Its boom was in a 80s and nowadays this method has come back [2].

Because of the need of making the GPU computation as fast as possible, especially when the user moves in a scene, there is a simple version of this method implemented on GPU. The image is iteratively changing its resolution grid according to the current frame number and a detected movement in the application. It is also possible to change the grid size operatively.

When the renderer achieves the resolution which is equal to the resolution of an output buffer, there is usually the need to sample more rays withing one pixel (standard non-real-time renderer) to prevent the result from aliasing. At this moment the renderer is not sampling more than one ray in a pixel per one frame, but rays are incrementally shot in a different pixel position over the time and doing a weighted avarage. The standard non-real-time method is called stratified sampling (sometimes jittered sampling), but in DB it is done over the time period with just one primary ray in a pixel per frame.

Illustration of this method can be seen in figure 3.9. The simplified schema of raytracer.cu producing a final render is a figure 3.10.

[1] Nice example of this method can be seen at one of the Johnny Lees's You Tube videos released 6 years ago: http://youtu.be/Jd3-eiid-Uw.

[2] It has been implemented in Pixar Render Man solution according to his developers which are also writing popular webpage scratchapixel.com [52].

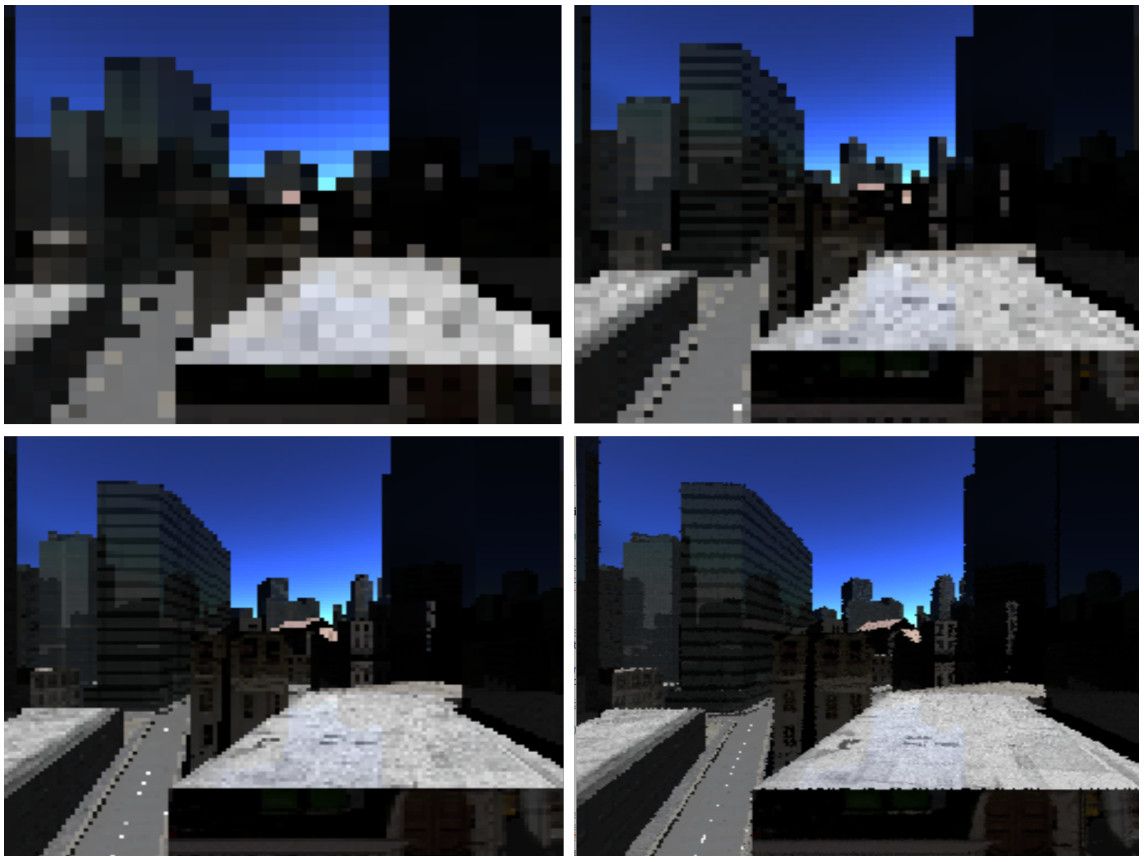

FIGURE 3.9: Demonstration of progresive refinement. Screen on the bottom-right is demonstrating stratified sampling per one frame.

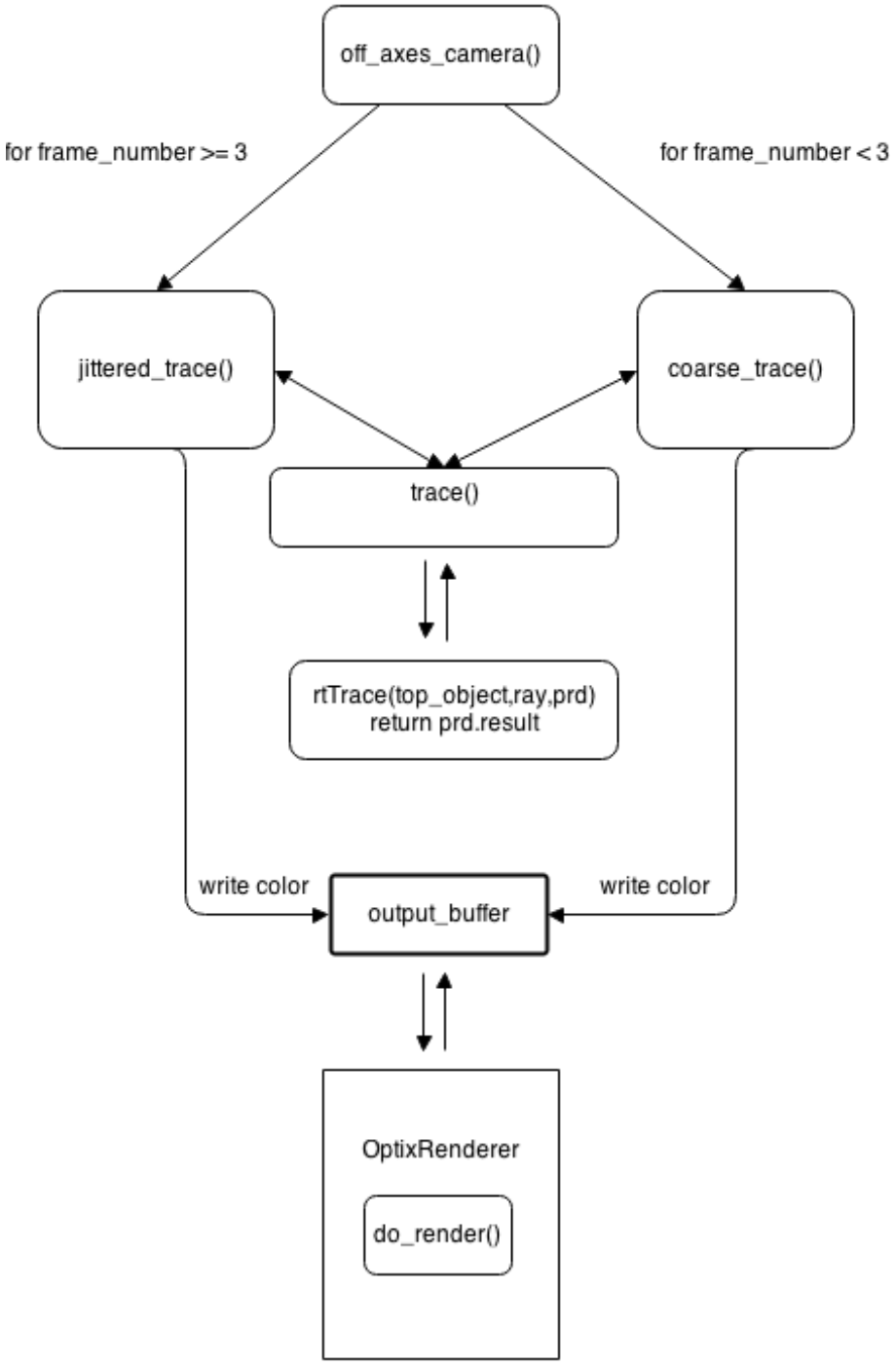

FIGURE 3.10: Simplified schema of raytracer structure.

Chapter 4

Testing

The tests have been executed during the whole development stage. We have an access for all the computing resources in IIM, but currently the development part is done mostly on the following hardware and software configurations:

- Intel® Core^{TM} i5-3570K CPU @ 3.40GHz × 4
- eVGA GTX 580 3GB
- Kingston DDR3 8GB 1600MHz CL9 XMP
- Crucial M4 SSD 120GB
- Ubuntu 13.10 (CUDA 5.0 - nvcc V0.2.1221; Optix 3.0; g++ 4.8)
- Nsight Eclipse Edition 5.0 IDE
- Nvidia Drivers 319.60

4.1 Performance Testing

We have perfomed one final test on a CE_cinegrid02 scene which has been used for many years in CAVE and consists of 150220 triangles (usually we have worked with the models from Jan Tůma having 20 000 - 40 000 triangles.).

The test has been perfomed with the different resolution grids of the output buffer (32x32, 64x64, 128x128,256x256, 512x512, 1024x1024, 2048x2048, 4096x4096) which is

read from the GPU memory and displayed through the texture being interpolated on the full screen resolution (currently we are using a default bilinear texture interpolation provided by OpenGL API). Every test of a graphics card took 1 minute. We are using an average of 10 second periods and in the table 4.1 there is presented an avarage from these 6 periods. We had also decided to skip the first 10 second period because of the additional compilation time of the shader used for the displaying of the output pixel buffer. The graphics visualization of this test can be seen at figure 4.1 page 35 .

Resolution	GTX Titan	GTX 580	GTX 690	Quadro FX 5800	Quadro 4000
32x32	84.485400	108.385200	63.629540	31.378820	52.299220
64x64	86.876520	111.579800	63.927640	29.912520	48.765560
128x128	78.509420	74.636740	48.364740	26.691020	25.715960
256x256	38.571000	28.527860	20.072100	17.808140	8.016898
512x512	10.489180	8.565752	10.602480	5.580534	2.310494
1024x1024	2.878948	2.374864	3.244628	1.371030	0.630411
2048x2048	0.850356	0.628194	0.880581	0.274134	0.165258
4096x4096	0.157429	0.162548	0.171412	error	0.000000

TABLE 4.1: Results showing performance in average FPS for various graphics cards and grid resolutions. [1] Test has been done with a CE_cinegrid02 model which has 150 220 triangles.

There has been another test made for GTX Titan 6GB and GTX 580 for Sponza model with 66 450 triangles. This was the original model from Dabrovic. The results of the measurment are shown in table 4.2 and in figure 4.5. Notice the increase of the FPS for this scene. An example of the scene render at 512x512 can be seen in figure 4.4.

Resolution	GTX Titan	GTX 580
32x32	305.951200	372.599400
64x64	294.474000	343.811400
128x128	224.831600	201.114600
256x256	88.716860	76.615160
512x512	30.758420	25.232060
1024x1024	9.647878	7.658178
2048x2048	3.029762	2.178698
4096x4096	0.756884	0.597849

TABLE 4.2: Results showing performance in average FPS for various grid sizes measured for GTX Titan and GTX 580 with the use of Sponza model (66 450 triangles).

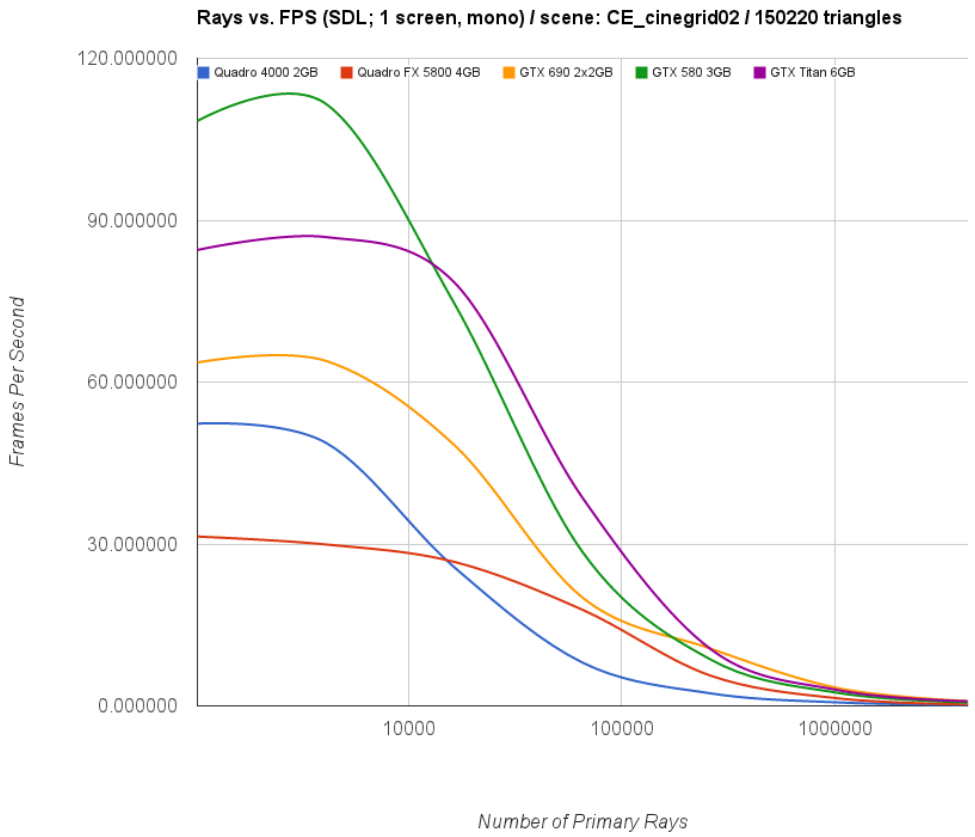

FIGURE 4.1: Results showing performance in average FPS for various graphics cards and grid resolutions in a graph.

FIGURE 4.2: Illustration of different image grid resolutions (32x32 - 128x128).

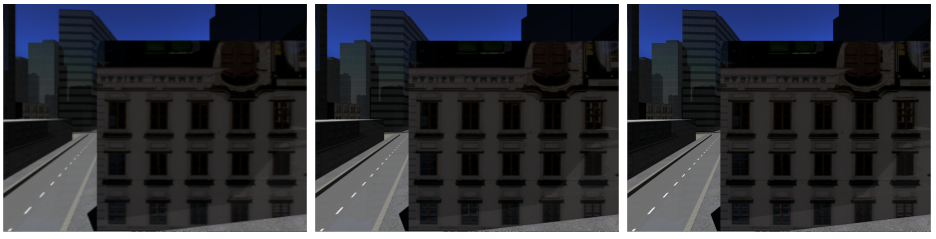

FIGURE 4.3: Illustration of different image grid resolutions (256x256 - 1024x1024).

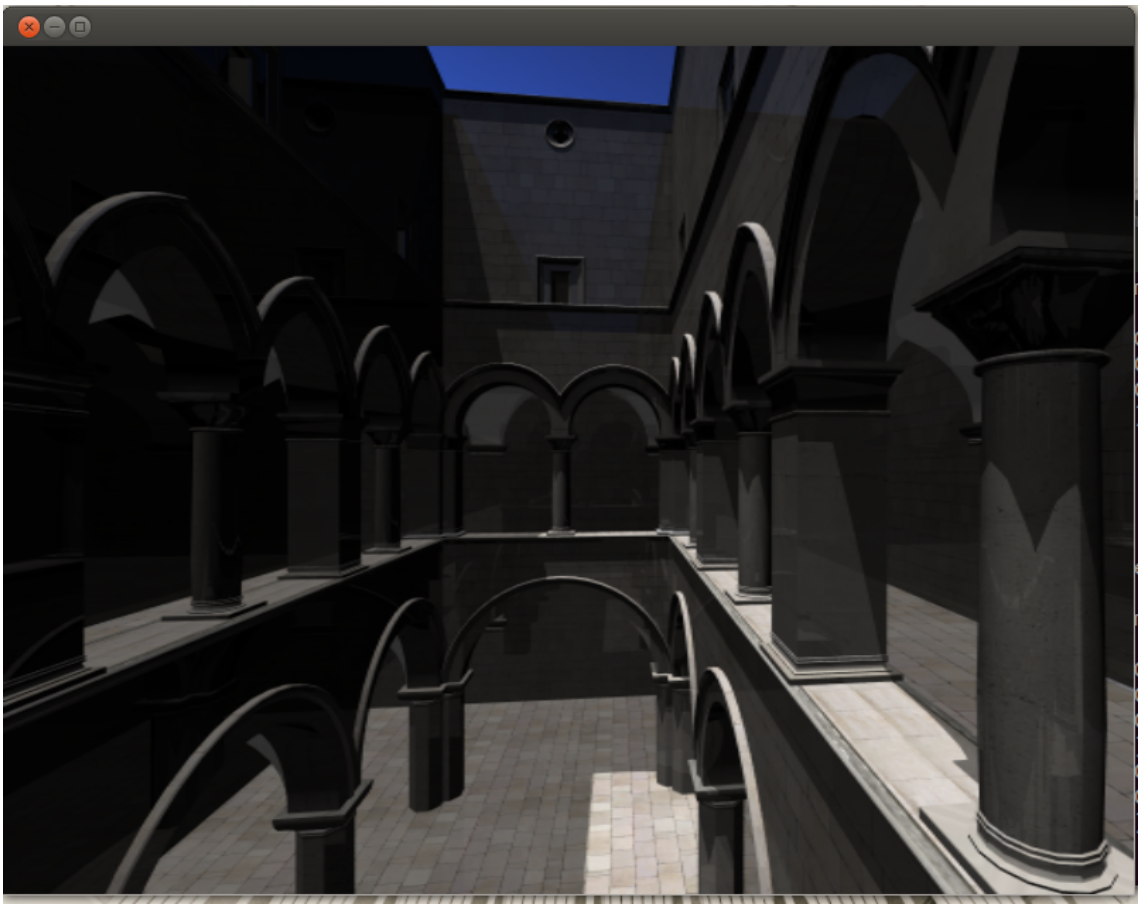

FIGURE 4.4: Sponza model renderered in resolution 512x512. FPS is approximately 25 FPS with GTX Titan.

4.1.1 Hardware Issues

The most important factor for a sufficient FPS is the performance of graphics cards. Dark Bulb is developing on the system with NVIDIA® GTX 580 graphics card supporting CUDA Compute capability 2.0 with 512 CUDA cores and there we have proved that we are able to achieve sufficent results for one screen without any loss of real-time feeling (this of course depends on a resolution and model). For the CAVE the situation is not so feasible. Right now in IIM there is a hardware from 2007. That means 2x Nvidia Quadro FX 5800 which are not suitable for having 4 stereo (8 image buffers) screens in realtime. We can also see that for the resolutions 256x256 and higher, these cards produce the lowest perfomance in the table 4.1. This means that we have to innovate the current HW configuration to be able to produce the sufficient results at least for two screens in CAVE.

In the following figures 4.6 and 4.7 there are screenshots from the CAVELib testing and

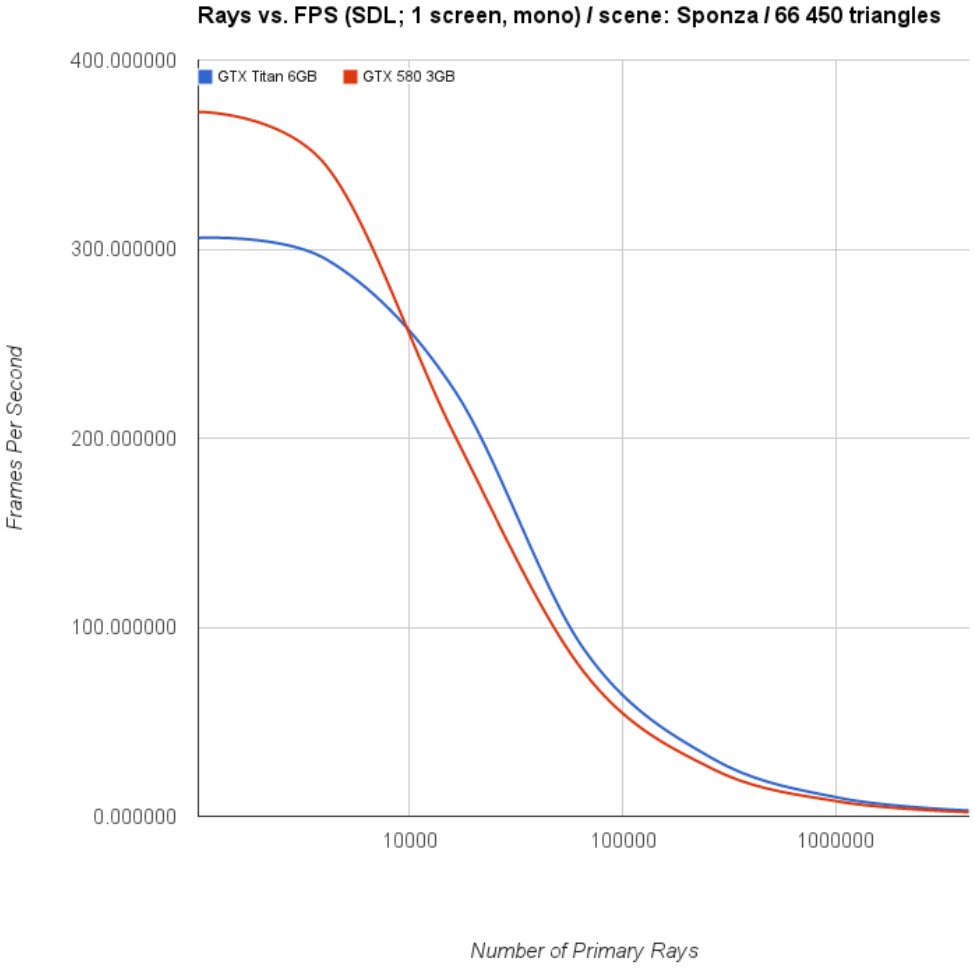

FIGURE 4.5: Graph showing performance in average FPS for GTX Titan and GTX 580 with the use of Sponza model (66 450 triangles).

CAVE tests. Figure 4.6 shows the CAVELib simulation for 3 walls in stereo projections. Figure 4.7 is photo taken during the CAVE tests with the city model (notice that black color is not black because of the age of the projector).

FIGURE 4.6: Testing of a Optix Renderer in a CAVELib for 3 screens.

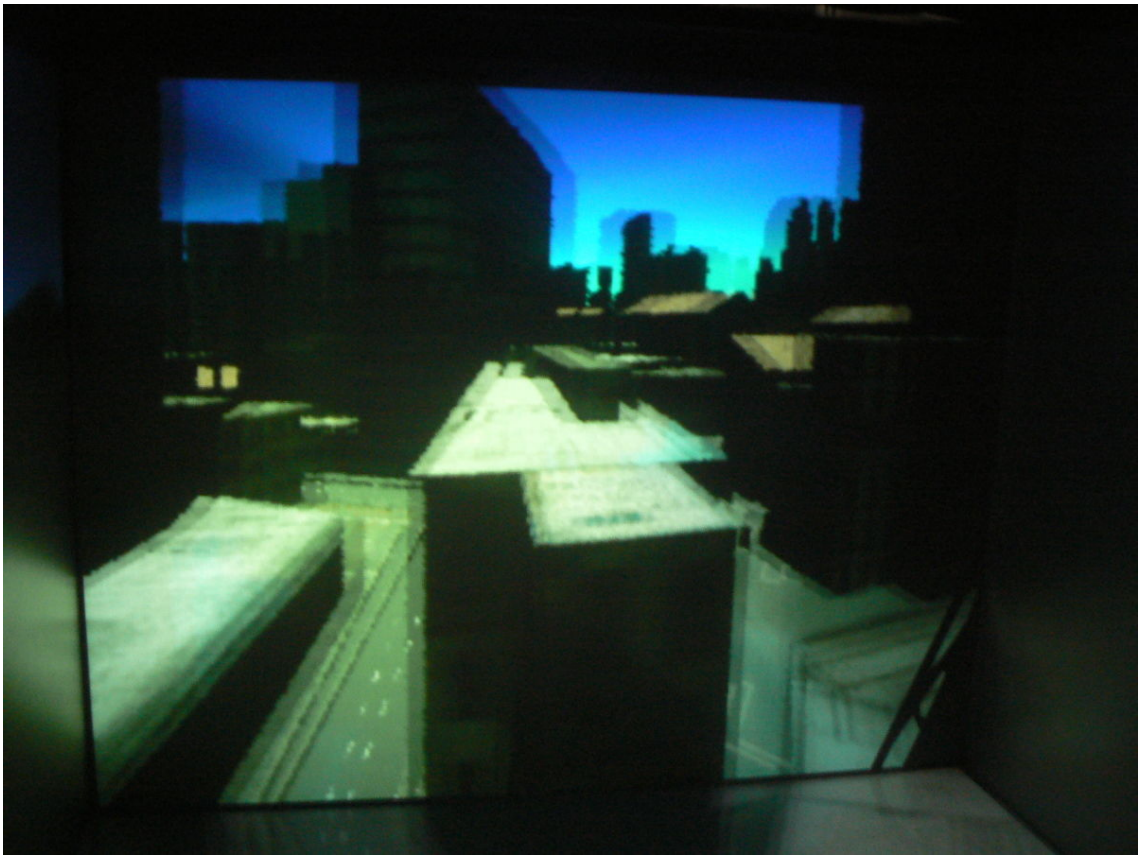

FIGURE 4.7: Photo of projection in a CAVE with CE_cinegrid02 model.

Chapter 5

Conclusion and Future Thoughts

From the analytical part of this work, many requirements on the Architecture Visualization systems were formed, especially in association with visualization environments like CAVE. The AV types and methods were in detail analyzed. Another part of the analysis brought a summary of the existing visualization methods and systems.

From these information the implementation was made and documented in Chapter 3.

As a main result of this thesis, we have changed the structure of the original CAVE framework to be more modular in order to have a variability over different rendering types (OpenGL, Optix). We improved the model loaders in order to respect the needs of architects collaborating with IIM.

Also, a ray-tracer with basic material types was implemented to achieve more realistic quality in comparision to the previous solutions. It supporst HDRI IBL, textures (includes bumb mapping), uses scene graph and it is made in a way which is extensible in the future.

We have also proved that with newer GPUs in a current CAVE HW configuration it is not impossible to have fully ray-traced solution for AV visualisation. Right now, limitation for us is the HW (it should be eliminated soon), developer resources and money.

For the future and the following stages we also have to concentrate on an optimization of our code and to experiment with the different path tracer algorithms. All the tests we have made are using the Whitted solution of raytracer with combination of progressive

refinment and the stratified sampling for anti-aliasing which is still better than the previous solution (it supports hard shadows, transparency and specularity), but the integration of real path-tracing with monte carlo would be a better option (at least it seems to be right now).

There is still some space for a better material library support and the compatibility with other 3D modelling softwares (maybe based on a OSL). The lighting system with Day and Night simulations, area lights support (this can be simulated with an emission material) and spot light support. There is also a problem how to import the light positions from the architecture models, because in a standard WaveFront .obj format there is no support for the light information. We basically want to continue with the road map provided in Chapter 3 and to concentrate on a ray-traced solution (maybe a hybrid solutions with OpenGL since we do not have a proper HW right now).

We would like also to make a better generalized controlling mechanism of a CAVE, since there is no standard solution like a computer with a mouse and keyboard controlling mechanism in combination with the graphical user interface to control diferent parameters in a real-time.

Appendix A

User Manual

In this user manual we are going to show how to export models from 3ds Studio Max which has been chosen as a reference software used by Jan Tůma and for now it is the most important one for architects using CAVE in IIM. In the future we would like to extend this manual for other software for better compatibility.

A.1 Export models from 3D Studio Max

For exporting the models from 3D Studio Max we have to take these steps if the scene is ready. Manual was made with 3D Studio Max 2014 (the manual is similar for the older versions).

For exporting WaveFront .obj format you have to ensure that your scene is ready and then follow these steps:

1. Select the Application menu and choose Export.
2. Write the desired file name and open the Save As Type drop-down list and choose gw:OBJ Exporter((*.OBJ). See figure A.1.
3. Click Save. The exporter dialog window will open. For the best results set the options to same values as in the figure A.2.

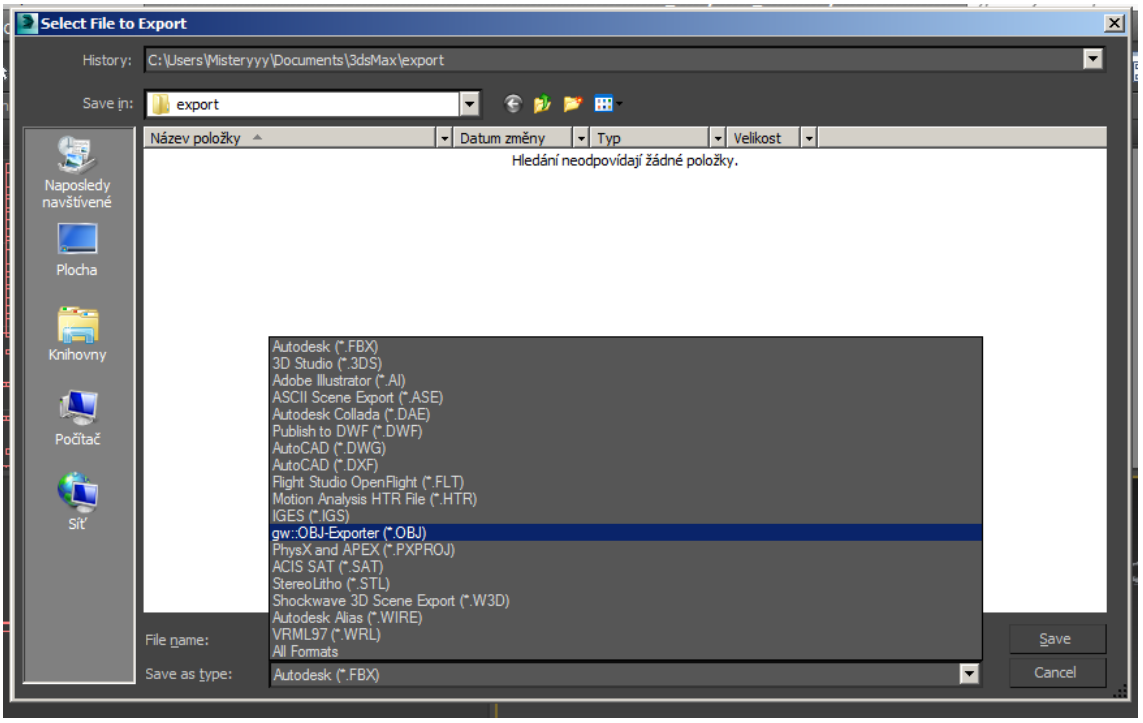

FIGURE A.1: Exporting .obj format.

The setting of Scale value depends on a unit system you are working in. Leave it at 1.0 (if you set 0.1; 1 meter high object in 3ds Max will aproximatelly correspond to the height of CAVE wall in IIM).

If you have textures, do not forget to set the map-path option (Map-Export button) to enable. It will create ./map folder in your export path with textures. You can leave the other options at default.

4. Click on Export. During export a dialog window showing progress and the names of exported objects opens. When the export is ready, confirm DONE and you can use your model in the DB Framework.

Tips:

1. If you are modelling in 3ds Max, try to follow the center point in [0,0,0] and build your architecture around it. This approach will save us time to find the imported scene in the engine before we build a more robust solution.

2. Remember that not everyone has a V-Ray renderer, so there will be troubles with the special materials you set in your settings. We will do our best to fix this compatibility soon.

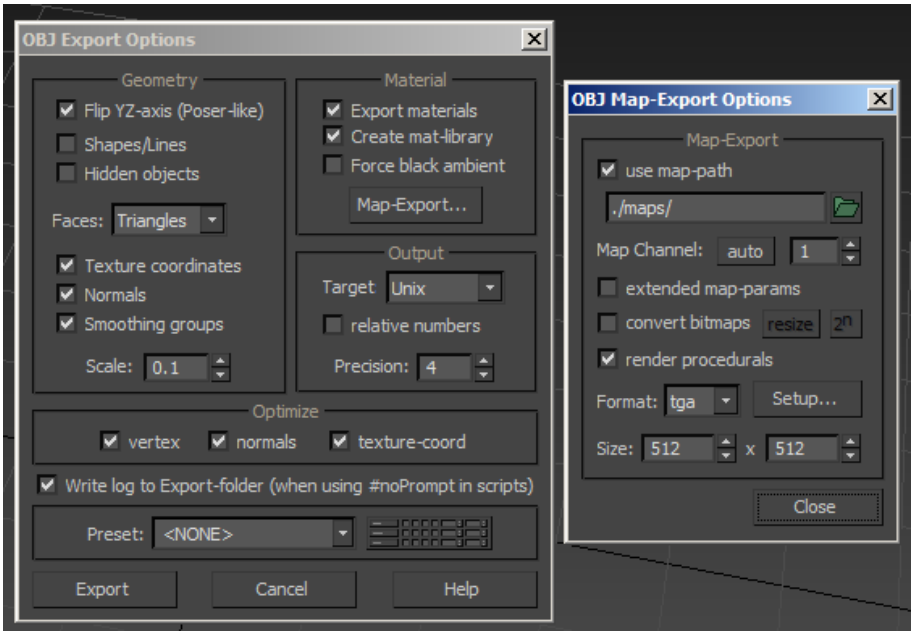

FIGURE A.2: Recommended setting for OBJExporter.

Appendix B

DVD Content

There is a directory with latex files (including figures) and exported pdf file included in DVD.

Bibliography

[1] Jacinto Monteiro's Treevillas Project Render, 2014. URL http://metrocubicodigital.com/.

[2] Jacco Bikker. Aurauna 2 official web page, 2014. URL http://arauna2.ompf2.com/. [Online; accessed 12-January-2014].

[3] Michiel Quist. Magazine 3dworld issue 142 - twinmotion®2 review, 2011. URL http://vault.3dworldmag.com.

[4] KA-RA. Twinmotion®2 official website, 2014. URL www.twinmotion.com. [Online; accessed 12-January-2014].

[5] shadows44. The rendering killer : rendu archi temps réel, 2011. URL http://www.3dvf.com/forum/3dvf/Actualites/rendering-killer-temps-sujet_358_1.htm. [Online; accessed 12-January-2014].

[6] shadows44. 3dvf - twinmotion 2 ceo interview, 2010. URL http://www.3dvf.com/dossier-710-1-interview-twinmotion-2.html. [Online; accessed 12-January-2014].

[7] Arch Virtual Website. Realtime architectural visualisation with unity3d, 2013. URL http://archvirtual.com/2010/10/21/realtime-architectural-visualization-with-unity3d/. [Online; accessed 12-January-2014].

[8] Screen of Unity 3 IDE., . URL http://elmundotech.com/2010/09/27/unity-3-game-dev-platform-available-now.

[9] getReal3D Screen, . URL www.mechdyne.com/filesimages/Software/getReal3D/getReal3D-unity-architecture_thumb.png.

[10] Unity Technologies. Unity engine eula, 2013. URL http://unity3d.com/company/legal/eula. [Online; accessed 12-January-2014].

[11] Mechdyne. Mechdyne getreal plugin website, 2013. URL http://www.mechdyne.com/getreal3d.aspx. [Online; accessed 12-January-2014].

[12] imin vr official website, 2013. URL http://www.imin-vr.com. [Online; accessed 12-January-2014].

[13] Jacco Bikker and Jeroen van Schijndel. The brigade renderer: A path tracer for real-time games. *Int. J. Computer Games Technology*, 2013, 2013. URL http://dblp.uni-trier.de/db/journals/ijcgt/ijcgt2013.html#BikkerS13.

[14] Jacco Bikker. Doctoral thesis: Ray tracing in real-time games. 2012. URL arauna2.nhtv.nl/files/thesis_jbikker.pdf.

[15] Wikipedia. Otoy, 2014. URL http://en.wikipedia.org/wiki/OTOY. [Online; accessed 22-January-2014].

[16] Bright Side of News. Gdc 2014: Otoy releases brigade photorealistic game engine into the cloud, 2014. URL http://www.brightsideofnews.com/2014/03/13/gdc-2014-otoy-releases-brigade-photorealistic-game-engine-into-the-cloudt. [Online; accessed 13-March-2014].

[17] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering, Third Edition*. Taylor & Francis, 2011. ISBN 9781439865293. URL http://books.google.cz/books?id=V1k1V9Ra1FoC.

[18] Jiří Žára Bedřich Beneš Jiří Sochor Petr Felkel. *Moderní počítačová grafika 2 vydání*. Computer Press, 2005. ISBN 8025104540.

[19] Tomas Akenine-Möller. Mobile graphics hardware, 2014. URL http://fileadmin.cs.lth.se/cs/Education/EDAN35/mGH.pdf. [Online; accessed 7-January-2014].

[20] Wikipedia. Global illumination, 2014. URL http://en.wikipedia.org/wiki/Global_illumination. [Online; accessed 22-January-2014].

[21] Peter Shirley and R. Keith Morley. *Realistic Ray Tracing, Second Edition*. Ak Peters Series. Taylor & Francis, 2003. ISBN 9781568811987. URL http://books.google.cz/books?id=ywOtPMpCcY8C.

[22] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2010. ISBN 0123750792, 9780123750792.

[23] Matt Pharr and Greg Humphreys. PBRT oficial website, 2014. URL http://www.pbrt.org/. [Online; accessed 3-March-2014].

[24] Robin Hub. Efficient methods for computing complex global illumination effects, 2014. URL https://dip.felk.cvut.cz/browse/details.php?f=F3&d=K13139&y=2014&a=hubrobin&t=dipl. [Online; accessed 3-March-2014].

[25] Samuel Lapere. Ray-tracey blog, 2014. URL raytracey.blogspot.com. [Online; accessed 18-March-2014].

[26] Ronen Bekerman. Photographic Approach in AV, 2014. URL www.ronenbekerman.com/photographic-approach-in-architectural-visualisation/. [Online; accessed 22-January-2014].

[27] James T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 143–150, New York, NY, USA, 1986. ACM. ISBN 0-89791-196-2. doi: 10.1145/15922.15902. URL http://doi.acm.org/10.1145/15922.15902.

[28] David S. Immel, Michael F. Cohen, and Donald P. Greenberg. A radiosity method for non-diffuse environments. *SIGGRAPH Comput. Graph.*, 20(4):133–142, August 1986. ISSN 0097-8930. doi: 10.1145/15886.15901. URL http://doi.acm.org/10.1145/15886.15901.

[29] Kevin Beason BRDF Examples, . URL http://www.kevinbeason.com/worklog/wp-content/uploads/2009/06/brdftestmisday.jpg.

[30] Arch Daily. The world's most epic material library, 2014. URL www.archdaily.com/262869/the-worlds-most-epic-material-library. [Online; accessed 22-January-2014].

[31] Material Library in NY., . URL http://seedmagazine.com/slideshow/material_connexion/img/material_connexion_2_1100x733.jpg.

[32] OSL. OSL Specification, 2014. URL https://code.google.com/p/openshadinglanguage/wiki/OSL_Introduction. [Online; accessed 18-March-2014].

[33] Chaos Group. OSL VRay Support, 2014. URL http://www.v-ray.com/features/new-shaders/. [Online; accessed 18-March-2014].

[34] Paul Debevec. Paul debevec home page, 2014. URL http://www.pauldebevec.com/. [Online; accessed 18-March-2014].

[35] HDRI Sky Example with Architecture, . URL http://3docean.net/item/purelight-hdri-collection-01/134408.

[36] Pellacini Bunnell. Shadow map antialiasing, 2007. URL http.developer.nvidia.com/GPUGems/gpugems_ch11.html. [Online; accessed 18-March-2014].

[37] Wikipedia. Shadow mapping, 2014. URL http://en.wikipedia.org/wiki/Shadow_mapping. [Online; accessed 22-January-2014].

[38] Daniel Šimek. Deferred shading based extensible rendering pipeline, 2013. URL https://dip.felk.cvut.cz/browse/details.php?f=F3&d=K13139&y=2013&a=simekdan&t=dipl. [Online; accessed 18-March-2014].

[39] Wikipedia. Shadow volume, 2014. URL http://en.wikipedia.org/wiki/Shadow_volume. [Online; accessed 22-January-2014].

[40] Wikipedia. Scene graph, 2014. URL http://en.wikipedia.org/wiki/Scene_graph. [Online; accessed 22-January-2014].

[41] Edward Angel and Dave Shreiner. *Real-Time Rendering, Third Edition*. Addison-Wesley, 2011. ISBN 0132545233. URL http://www.amazon.com/Interactive-Computer-Graphics-Top-Down-Shader-Based/dp/0132545233.

[42] Muhammad Mobeen Movania. *OpenGL Development Cookbook*. Pack Publishing, 2013. ISBN 1849695040. URL http://www.packtpub.com/opengl-development-cookbook/book.

[43] Unity Technologies. Occlusion culling, 2014. URL http://docs.unity3d.com/Documentation/Manual/OcclusionCulling.html. [Online; accessed 22-January-2014].

[44] Jincheng Li. Master Thesis: GPU Path Tracing. 2010. URL https://gupea.ub.gu.se/bitstream/2077/28837/1/gupea_2077_28837_1.pdf.

[45] M. Zlatuska and V. Havran. Ray Tracing on a GPU with CUDA – Comparative Study of Three Algorithms. In *Proceedings of WSCG'2010, communication papers*, pages 69–76, Feb 2010.

[46] Wikipedia. Bounding volume hierarchy, 2014. URL http://en.wikipedia.org/wiki/Bounding_volume_hierarchy. [Online; accessed 22-January-2014].

[47] Christer Ericson. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology) (The Morgan Kaufmann Series in Interactive 3D Technology)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. ISBN 1558607323.

[48] Wikipedia. K-d tree, 2014. URL http://en.wikipedia.org/wiki/K-d_tree. [Online; accessed 22-January-2014].

[49] NVIDIA®. GPU Ray Tracing, 2014. URL http://www.nvidia.com/object/gpu-ray-tracing.html. [Online; accessed 11-April-2014].

[50] Robert Kooima. Generalized perspective projection, 2009. URL http://csc.lsu.edu/~kooima/articles/genperspective/index.html. [Online; accessed 11-April-2014].

[51] WikiBooks.org. Cg programming/unity/projection for virtual reality, 2014. URL http://en.wikibooks.org/wiki/Cg_Programming/Unity/Projection_for_Virtual_Reality. [Online; accessed 11-April-2014].

[52] Scratchapixel. Adaptive progressive refinement rendering, 2012. URL http://www.scratchapixel.com/lessons/3d-advanced-lessons/adaptive-progressive-refinement-rendering/technique-description/. [Online; accessed 9-May-2014].