Content Creation for Technical Research Centre of Finland's Virtual Reality System, Lumeportti

Tatu Harviainen
Media Lab
University of Art and Design Helsinki
Final Thesis 12/2003

Abstract

In this written part of my final thesis I describe the production work I did in co-operation with Technical Research Centre of Finland. The aim of the production work was to develop a content creation work flow and virtual environments for Lumeportti virtual reality system developed by the Technical Research Centre of Finland.

The final content creation work flow developed for the Lumeportti system consists of multiple work phases and tools. Work flow starts with a phase where the geometry of the virtual environment is modeled and ends to a phase where the building blocks of final virtual environment are compiled and packed. After this last stage the virtual environment is ready to be used in the Lumeportti system. The building blocks, i.e. the content of the final virtual environment consist of 3D geometry of the objects, textures of those objects, lighting of the environment and different functional elements placed in to the environment. Content creation work flow developed during the production work was tested and used for building several virtual environments. The main virtual environment I built during the production work was used for demonstrating the use of the Lumeportti system as a tool for architectural visualization tasks.

This written part of my final thesis starts by explaining and defining some key topics related with my production work and research area of virtual reality in general. The Lumeportti virtual reality system is described in detail to make it easier for the reader to grasp what was the technological environment I worked on. Next the different goals, challenges and problems I faced in my production work are introduced and described. The key problems of the production are stated in this chapter and next chapter explains in detail the solutions I found for these problems or in some cases why complete solutions were left unfound. Main issues here are focused on the questions of how to increase the rendered image quality in the final virtual environment, how to bind together different content creation methods used by the Lumeportti system's graphics engine and physics engine and also how to find a way to re-use already existing 3D data in the content creation work flow. Last there is a summary of the experiences gain during the production work and some speculations of what was done well and what could have been done better in the production. In this last chapter there is also some comments of what should be done in the future to push the work I did even further.

Preface

This project was conducted in close co-operation with the Technical Research Centre of Finland, VTT, my employer during the production work of this final thesis. This thesis is closely tied to the outstanding work that has been done by the people working in the Product Information Management research group developing the Lumeportti system. I would like to first thank the group manager Raimo Launonen and everyone on the research group for not only making my work possible but also making it very enjoyable. This also includes the people from Technical Research Centre of Finland's Transport and Building unit who took part to the projects linked with Lumeportti system.

I'm also very grateful for the help and information I got from Jani Lahti, managing director of Adactive Ltd. Adactive Ltd. provided also the test data for the virtual environment which played a major role in my final thesis production. Also thanks for Satu Lavinen for helping me in getting the first contact to the company.

I would also like to thank all the people at the Media Lab of the University of Art and Design Helsinki.

Last but not least I must thank my advisors on this project. Many thanks for Jukka Rönkkö who had patience and energy to help me out on so many occasions. Also thanks for Miika Tams for encouraging me and giving me ideas along the way.

And of course, I'm grateful to all the people who have helped and encouraged me during this work, but whom I'm unable to name individually. Many thanks for the support I got from my family and above all, thank you Katja.

Table of content

Introduction

Human mind is easily lured to travel to the parallel worlds and realities of imagination. Everyone of us must have felt at times the urge to leap out of the everyday life into the universe of a good book, movie or a computer game. As Janet Murray pointed out in her book Hamlet on the Holodeck [Ref. 8], the age-long desire to be able to live out a fantasy aroused by a fictional world has been intensified by a participatory, immersive medium. By this medium she means the new medium generated by the development of computer technology and telecommunication. This participatory new medium has taken long steps from text based adventure games and wireframe graphics to the extensive multi-user worlds with spatial sounds and stereographic imaging of today.

No matter how the technology evolves, the real power to take us into a journey to these dream worlds is in the hands of the artist creating the content for the medium. The real problem for the artist is still the same, how to translate the idea into the form of medium and do it so that the viewer will accept the illusion and get dwelled into it. As the technical development has made powerful interactive new medium available to us, it has also presented more demands to the content creation process. Interactive real time three-dimensional graphics require the content designer to be able to work on various tools and master new techniques of three-dimensional modeling, texturing, lighting etc. On the top of this the interactive computer software systems are still quite restrictive and demand the designer to be aware of the different technical aspects of the hardware and software systems in use in order to get satisfactory results. All this creates a need to develop the content creation tools further with usability on mind to give more freedom to the content creation.

At the moment it seems that the advantages provided by the virtual reality have only been utilized in some very tightly focused special cases, mainly in different simulator and design inspection applications. It's easy to see the vast potential of the virtual reality; being able to examine, test and use nonexistent subjects in fully immersive environments in real life or just any chosen scale. At the moment there are however some major barriers keeping us from taking the best advantage of that potential. If we could surpass the biggest problems at the moment; the high costs of virtual reality systems, usability issues and the big workloads demanded for the content creation, there might be a sudden expansion in the areas where virtual reality systems are used.

During the work on this thesis I have been fortunate to be able to work with the state of the art tools in the area of virtual reality systems and also had an opportunity to actually take part in effort of trying to solve some of the limitations related with the content creation of the virtual reality systems at the moment. This writing is an attempt to describe the work done on developing and using a content creation work flow for the virtual reality system Lumeportti. The starting point for creating Lumeportti system was an attempt to create cheaper and more generic virtual reality system, with high quality in the areas of rendered graphics, usability and physics simulation. My role and area of expertise within the Lumeportti development was the content creation side of the system.

Summary of the Chapters

First chapter of this thesis will introduce some of the key concepts and terms related with virtual reality and real time 3D graphics. These concepts and terms are mainly ones that are substantially related to the production work done in this final thesis. Key terms used in this thesis are defined and explained briefly and their relation to my thesis work are discussed. Due to the relatively short history of virtual reality research many of the terms used here are not unambiguously defined and may cause some misunderstandings if not explained and defined clearly.

In the second chapter the Lumeportti virtual reality system is introduced and explained in detail. In this chapter the hardware and software environments of the Lumeportti system are described and various development solutions are discussed. This rather detailed technical description chapter is included in this thesis for making it easier for the reader to grasp what was the technological environment I worked on.

On the third chapter the different goals, challenges and problems that I faced during the production work are listed and explained. Since my production work was a part of a bigger challenge of developing the whole Lumeportti system and there were couple of projects under which this development took place there were number of factors involved which all set additional goals and challenges for my work.

Fourth chapter describes the work done on the production and different solutions related with it. On previous chapter the different goals and problems faced during the production were lifted up and on this fourth chapter the key solution which were discovered are explained. This chapter should give the reader an overall impression of what kind of work phases and tasks the actual production work involved.

In the fifth chapter the final results from the production work are described. There the developed final content creation work flow is explained in detail. Also the virtual environments implemented for the Lumeportti system are introduced and specially the virtual environment I implemented for the pilot application of architectural visualization is described thoroughly.

In the last chapter I summarize the lessons learned and experiences gained during the production work. In this last chapter I also included some speculations of what should have been done differently during the production work or what should be done in the future to push the work I did even further.

1 Theoretical Context

Until recently virtual reality research has been focused primarily on technology and the usability issues of single virtual reality systems without the study of the content creation or general application areas. There is however a growing trend of re-focusing the research interest from pure technology of virtual reality systems towards the content and application areas of the virtual reality, in other words, to think up what can this technology be used for [Ref. 2]. This new direction of interest in virtual reality research has made the research area much more interdisciplinary and the current audience for this research has more diverse backgrounds.

For this written part of my final thesis I have gathered information mainly from the areas of current virtual reality and computer game research publications. For the content designer the tools and outcomes in virtual reality content creation process are pretty much the same as with the current real time 3D game development and quite often the best information concerning this design process comes from the computer games industry. In addition I have been able to get information directly from experts working on the area of architectural visualization and 3D computer games.

In the following paragraphs I've explained some of the key topics related with this final thesis.

1.1 Virtual Reality and Virtual Environment, the Definition and Background of the Terms

One of the first visions of the virtual reality was presented by one of the pioneers in area of computer graphics, Ivan Sutherland, 1965 in his decree where he articulated the original dream of virtual reality: "The screen is a window through which one sees a virtual world. The challenge is to make that world look real, act real, sound real, feel real." [Ref. 1]

The actual term 'virtual reality' was taken into use around mid 80's and the credit of coming up with the term has been given to Jaron Lanier. Since then there have been many attempts to define virtual reality explicitly. The debate over the various definitions however still seems to be going on at some extent. Many of the attempts to define what virtual reality is have been based on the technical characteristics of virtual reality systems. However that leads us into too limiting definitions and doesn't work universally. By my opinion Steve Bryson made a very good point in his writing about the history of the definition, that in general people are primarily interested in the effects that can be achieved with virtual reality systems, not the technology behind it. Therefore it makes sense to define virtual reality in terms of its cognitive effects. This has led to the definition of virtual reality as following:
"Virtual Reality is the use of computer technology to create the effect of an interactive three-dimensional world in which the objects have a sense of spatial presence." [Ref. 3]

In this thesis term virtual environment is used to describe the 3D model and functional elements of a scene designed for the virtual reality system. Virtual environment is displayed to the user by the virtual reality system from an egocentric point of view using real-time 3D computer graphics [Ref. 21]

1.2 Application Areas of Virtual Reality

There have been attempts to use virtual reality in many different application areas. One can find a wide range of documentation on the area of virtual reality research about virtual reality application tests from art creation to complex training tasks. At the moment the best results of using virtual reality are achieved with applications that enable the user to train some task which would be impossible otherwise [Ref. 2]. For example training of astronauts for assembly sequences that will be eventually carried out in zero gravity can only be trained effectively by using virtual reality systems. Of course different simulation systems in the areas of aeronautics and military uses have been widely used and can be considered as one area where virtual reality has been able to establish it's status. Also one area where virtual reality has proved it's usefulness is design pre-visualization in big manufacturing projects. For example automobile industry has actively developed their own virtual reality applications used for different design inspection uses with virtual models.

Even with these good results many of the intended virtual reality applications are still not what they should be for serious use. There are problems with the usability issues, image quality and content creation that should be researched and developed further. During my production work with the Lumeportti system we faced some of these common problems. Later these issues are explained in more detail.

1.3 Virtual Reality Systems

As well as tight definition of term 'virtual reality' the universal definition of virtual reality system is somewhat complicated task. There are a great number of different set ups and collections of different techniques used and categorized under the term virtual reality system. In general it can be said that most common set up would consist of display device capable of producing stereo graphic images, some haptic input device and / or tracking device for tracking the position of users head. These former three components are most basic elements, but however not all the systems titling themselves as virtual reality system, include all three components.

All of the virtual reality systems use some kind of software system which processes the input from the user and 3D computer graphics to display the virtual environment to the user. Virtual reality system capability of rendering stereo graphic images is always based on rendering two images, one for each eye, which are rendered from slightly different angles. This of course creates a need to be able to display different images to the corresponding eye of the user. For this purpose number of different kinds of alternative display devices has been developed. Often virtual reality systems are categorized according to the display devices they use. Most common categories are head mounted displays (HMD), BOOM (Binocular Omni-Orientation Monitor) devices and different kinds filter glasses which are used with normal monitors or large screens and data projectors. The latter of these are often referred as CAVE like systems.

1.4 CAVE Like Systems

The first CAVE system was developed at the University of Illinois. It was developed by Carolina Cruz-Neira, Thomas DeFanti and Daniel Sandin among some others. It was demonstrated for the public at the SIGGRAPH '92 [Ref. 6]. They had named their work CAVE, which is a recursive acronym "CAVE Automatic Virtual Environment". Since then

cave is used as a generic term to describe a virtual reality systems which use multiple projection walls as a display device. However at the moment the term CAVE has been trademarked by the company Fakespace [Ref. 22] and therefore general term CAVE system as a category has been transformed to a CAVE like system.

Since 1992 there has been active development done with the CAVE like system in various universities and research institutes. During the few last years there has been a new direction to develop new system implementations, which build upon normal PC hardware and Windows operating system [Ref. 7] instead of cumbersome Unix based systems with expensive graphical work stations. The initial costs of these new setups are significantly lower than the first CAVE like systems built upon special hardware.

Some of the current CAVE like systems have also developed into a commercial products, which are sold as ready to run packages. For example international Fakespace, Fraunhofer institute in Germany and Cyviz in Norway are selling technology derived from CAVE research.

Lumeportti virtual reality system is a CAVE like system developed by the Technical Research Centre of Finland and the details of this system are discussed later in chapter 2. Helsinki University of Technology has also invested effort in VR research and development their own CAVE like system, called EVE.

1.5 Presence and Immersion

Presence is a term used to describe the effect of user experiencing that he/she is surrounded by the virtual environment and actually being part of that virtual environment [Ref. 5].

When the differences between the virtual reality and traditional 3D computer graphics are compared the concept of presence is often seen as a key factor separating these two from each other. In virtual reality systems tracking devices are used to maintain information of the position and orientation of the users head. This information is required to maintain a sense of independent spatial presence as the user moves his head. The desired effect is that the virtual object (assuming it's not moving) stays put as the user moves around. [Ref. 3]

Besides physical presence the user of virtual reality systems can have an experience of social presence. If the virtual world is inhabited by avatars controlled by artificial intelligence or virtual characters controlled by other users the feeling of presence can be very deep even if the virtual environment itself is not that immersive.

Immersion and presence in the virtual environments are two closely linked terms. Immersion means the effect of user submerging into the virtual environment. Immersion is an experience, which can be triggered with different kinds of media, by watching a movie, reading a book or navigating a virtual environment. Immersion is a product of selectively focused attention, when user is purely interested about one medium and tries to block all the stimuli outside that medium he/she will feel submergence into it.

1.6 Multimodal Interaction and Navigation in Virtual Reality

The obvious advantage of real time 3D graphics is that it enables the user to freely explore the virtual environment and that there can be interaction between the user and virtual environment. Especially with virtual reality applications where human-computer interface often cannot be implemented by using traditional input devices; keyboard and mouse, the design of alternative conventions can be surprisingly difficult. The user interface of Lumeportti virtual reality system originally was build on user position tracking, two data-gloves and speech recognition system for command input. Navigation was mainly done by using hand gestures with data glows. Later Lumeportti system was modified to use 3dConnexions SpaceMouse motion controller device for navigation in architectural visualization application since the hand gesture navigation proved to be too complicated for average user of the system. The ease of navigation is one of the key factor behind the usability of the virtual reality system. Even though this subject is important for the system development, it's not the main scope of this thesis and therefore this subject is only lightly covered.

1.7 Stereographic Viewing and Display Devices

The development of the display devices has not been very radical if compared with the development of the computational power of computers or the rendering power of graphic display cards. Normal cathode-ray tube, a.k.a. CRT monitor is still the most commonly used display device in use and the one that provides the best resolution and color richness.

To produce a stereographic image we have to be able to render and display separate images for both eyes. Normal computer monitor can be used to view stereographic rendering by using shutter glasses, but for virtual reality purposes it's not the best solution. The computer monitor is after all quite small and cannot provide the sensation of presence and immersion as strongly as often desired.

For the purpose of stereographic rendering number of different display devices have been invented. Most common categories are head mounted displays (HMD), BOOM (Binocular Omni-Orientation Monitor) devices and different kinds filter glasses which are used with normal monitors or large screens and data projectors

There have been promises of usable head mounted displays for years, but still these devices tend to have problems preventing common use of them. There are still problems with image quality but more importantly they are still generally too expensive and they tend to cause often simulator sickness for the user.

1.8 Tracking Devices

Tracking devices allow a virtual reality system to monitor the position and orientation of the user or selected body parts of the user. Current tracking devices are based on electromagnetic, acoustic, mechanical, or optical technologies.

Besides tracking devices used for tracking orientation and position there are also devices used for tracking human postures. These devices range from whole body suits that are used for

overall motion capture to smaller devices such as data gloves for tracking the postures of users fingers.

The Lumeportti system uses both data glows and position and tracking devices. Tracking devices and how they are used in the Lumeportti system, is documented more closely on later chapters.

1.9 Real Time 3D Graphics

3D (three-dimensional) computer graphics is a term used to describe computer-generated images that represent three-dimensional space and objects rendered from a certain viewing point within that space. When the computer renders the displayed image on the fly, 10 to 60 or even more times per second, it's called real time graphics.

Thanks to the rapid development in the area of real time graphics we are now able to interact with the computer generated environments. When the computer system can accept and process input from the user in real time and at the same time update the generated image according to the collected input it gives the user opportunity to interact with the computer via the generated virtual environment.

The bottleneck for the interactive 3D world has always been the expensive processing tasks involved with 3D graphics. The real revolution on this area has been the development of the hardware graphics accelerators. At the 1998 when the second generation of personal computer graphics cards became available to the consumers, it was possible to build a single graphics processing unit chip capable of rendering over 1.5 million, textured and blended triangles per second at the cost of approximately 30 dollars per chip [Ref. 23]. Since then the graphic accelerators have been more or less standard components on personal computers and since then they have been developing at even accelerating phase.

Over the past 15 years, on the software side of 3D graphics many of the advanced rendering techniques have also passed from being purely research topics to components of practical applications. For example the algorithms used for calculating radiosity global illumination, hidden surface removal, z-buffering and ray tracing were before dismissed as too complex and demanding to have any practical value for 3D graphics. Today all of these techniques are generally used and developed even further with real time 3D graphics.

The development in the area of real time 3D graphics has also led to the common use of bigger components of graphic libraries and graphic engines for more complicated tasks. Graphics libraries are interfaces to the graphic card's hardware and they are also called application programming interface, API. At the moment most commonly used and supported by graphics cards libraries are OpenGL and DirectX.

Although the use of graphics libraries makes the programming hardware independent they are quite often still too low level for practical application development. The solution to this has been the introduction of 3D engines. 3D engines are software systems build upon the graphics libraries and they are a next abstraction level further away from the hardware. 3D engines provide the user with automated functions for many of the computer graphic's tasks that would require loads of code when using graphics libraries. These functions include for example automated commands for controlling the display window, reading and writing of files, memory handling and loading and processing of 3D content.

In addition to these two categories, graphics libraries and 3D engines, there are also software packages used with 3D graphics called middleware engines. Middleware engines could be located somewhere between the two earlier mentioned types. Middleware engines are build on the top of graphics libraries but they are not as ready to run solutions as 3D engines. Middleware engines typically have some ready tools for the content creation, but they need more complete software framework to be coded and compiled for the execution.

1.10 Virtual Reality Software and Game Engines

Term engine has been used as a general term in association with the computer games to describe the complete software application responsible for some complex task, like for example rendering graphics, simulating physics etc. Term game engine or more specifically and generally used term 3D game engine was taken into use by the general public along with the rise of the 3D first person shooter game genre in the beginning of the 90's.

There are big variations between different game engines in their operational principles and tasks which they perform. Usually the game engine has high level functions for tasks such as loading content to the game, processing user input, rendering and displaying the graphics, executing game's artificial intelligence, network operations, game physics calculations, etc. People often think that the game engines are mostly concerned with just rendering and displaying graphics of the game. However in modern game engines it's often just one task of along with many others. The part of the game engine dealing with the graphics is often called graphics engine. Graphics engine can also be a whole separate system and not just part of the complete game engine.

Current game engines are very large and complex software systems and even making just the graphics part of the engine from the scratch is a large task. Therefore in many cases it's much more practical to buy ready engine that is ready to be used, well tested and already proofed its' abilities in published games. It has been a growing trend in the game industry for the recent years that many of the game development companies build 3D games upon a game engine developed and sold by completely another company.

As well as the games have the complete software system to run the game called game engine, virtual reality systems need a similar system as well. Often virtual reality systems and 3D games are very similar when comparing the tasks which has to be performed by the core software to run the system. Especially they are similar in the way how they store and process the 3D graphics data and display this data as 3D images. Even though these two application areas have similar requirements from the software it seems that serious development of complete engines is mostly done only on the area of computer games.

In the case of the Lumeportti development it had been decided quite early on the development that game oriented 3D engine would be used for the implementation of the rendering modules. This decision had been done based on the experiences the original members of project team had from working with other virtual reality applications and especially lower level graphics libraries used traditionally in those projects.

In this thesis I've used generic term engine to describe all software systems which can be seen as a single system component or module clearly responsible for one area of system operations

no matter if it's for example responsible for rendering the graphics but also processing user input.

1.11 Factors Behind the Image Quality in Real Time 3D Graphics

By image quality of 3D graphics I mean simply an overall impression of how good the final rendered images looks in the same way as we look at a painting and have an overall impression if it's good or not. This overall impression of image quality is of course a composition of many things, some of them controlled by the designer and some of them resulting from limitations of the hardware or software used for rendering. For example the composition of the virtual environment, objects in it, textures and lighting etc. are results of the design done by the artist creating the virtual environment. Limitations derived from the hardware are for example resolution of the display device and processing power and memory in use which both restrict the complexity and richness the virtual environment may contain. Software limitations are limitations set by the 3D engine or other software used for rendering. No engine supports all the possible features available for real time 3D graphics or have capabilities to manage endless amounts of data. With virtual reality systems the subject of image quality can be bit more complex than with for example with PC games since virtual reality systems use wide range of different display devices which all may have an additional impact and restrictions to the image quality. However in the scope of this thesis I'm only interested in factors of image quality which can be manipulated and controlled by the designer of the content and therefore when I speak of image quality I only mean the image quality of rendered image without the effect which the display device in use may have to it.

It seems to me that quite often it is misleadingly thought that the image quality of real time 3D graphics is a direct result of the graphics engine used for rendering the image. Part of the factors summing up the image quality are of course direct results of the features supported by the graphics engine, but in most cases many of the graphics engine's shortcomings can be compensated by a good design and careful thinking of details when implementing the virtual environment.

I'm sure that anyone who has ever eyed through a feature list of modern 3D graphics engine is painfully aware of the overwhelming number of features related with the real time 3D graphics. In the following sub-chapters I have picked subjects of real time 3D graphics that I feel are the most important to consider when implementing the virtual environments and have biggest impact on the final image quality.

Geometry Detail

In real time 3D graphics all visible objects are constructed by number of flat planes called polygons. The number of polygons used in constructing objects geometry dictates how much detail can that geometry have. When aiming for photorealistic rendering the number of polygons needed for building the details into the geometry can raise to a very high level. Especially all the surfaces that must appear to be smoothly curved require large number of polygons to avoid the visible sharp edges between the polygons which can be seen especially in the silhouette of the objects. The limit of how many polygons the designer can use for constructing virtual environment depend on the processing power of hardware used for rendering the real time graphics and the effectiveness of the graphics engine.

Most of the modern graphics engines have features which help the designer to overcome the limits on the geometry detail by implementing some kind of level of detail algorithms or by supporting curved surfaces. Level of detail (LOD) algorithms are designed for automatically choosing an appropriate level of detail for each object depending on the viewing position. Objects which are viewed from a close distance have more polygons to show all the visible details, and objects further away have lower amount of polygons and so less details, which wouldn't be visible anyway because of the longer viewing distance. Use of level of detail algorithm usually means that the designer must make several versions of each object with various amount of polygons and the algorithm chooses the version of each object to be shown. Some of the level of detail algorithms are able to automatically reduce the number of polygons in the objects depending on the viewing distance, but the end results with the reduced geometry detail are usually quite poor and may have strange errors. Polygon reduction is quite demanding operation in the means of processing power, which means that automatic LOD algorithms may require too much processing power to be used, or at least they are more expensive than rendering the high polygon amounts without the LOD algorithm.

One solution for stretching the polygon limits has been the introduction of curved surfaces. Curved surfaces are modeled by defining a set of curves which represent the boundaries of the curved surface and then based on these curves the actual rendered surface is produced by algorithmically generating the polygons representing the surface in the run time during the real time rendering. Since the polygons are only approximating the detailed geometry and they are generated during the rendering, the number of polygons used for rendering each curved surface can be chosen freely. The use of curved surfaces make the use of level of detail algorithm much easier because the algorithm can choose the number of polygons used for rendering each object freely, and the designer doesn't have to make several version of the objects. With curved surface there is also no need for the extra memory required when having many versions of each object. Objects can be bit harder to model with curved surfaces than with polygons and this technique is only just started to emerge. However some graphics accelerator cards already support curved surfaces and can process required operations on their own dedicated processing units, which means that this new technique can grow popular very rapidly.

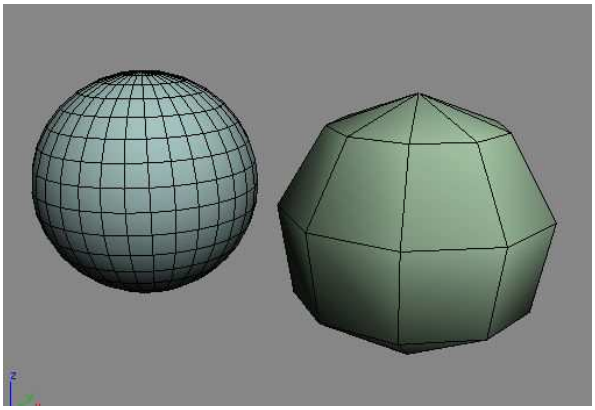


Figure 1 Effect of geometry detail

Textures and Procedural Textures

The geometry on real time 3D graphics usually doesn't mimic real world objects very well by only copying their shape. We receive a lot of information from real world objects from the outlook of their surfaces. By the outlook of the surface we are able to tell many things about

the objects, like for example what material they are made of, are they hard or soft, smooth or rough, etc. In the real time computer graphics the designer uses textures to create an illusion of real world materials.

Textures are 2D images that are projected on the surfaces of the 3D object's geometry. Textures are images which represent surface colors and patterns seen on the real world objects. They can be generated as digital paintings or by editing digital photographs. Usually the graphics engine sets some limits on the textures; they are forced to be in certain format and size. Quite often the pixel size of texture bitmaps is required to be in a powers of two (32, 64, 128 etc.) The maximum size for a single texture is usually 1024x1024 or 2048x2048 pixels.

Procedural textures are either images which are completely computer generated according some algorithmic rules or they are images which are based on normal textures described previously and transformed by some algorithmic rules. In the latter case the algorithmic rules are often used to merge several textures together with different blending effects. Procedural textures enable the designer to make much more complex textures and give the surfaces more realistic look by for example adding bumpmaps, reflections and opacity values for the surface. With 3D engines shader scripts are often used for describing the procedural textures. These shader scripts are text files which describe the mixture of different elements used for achieving a certain procedural texture.

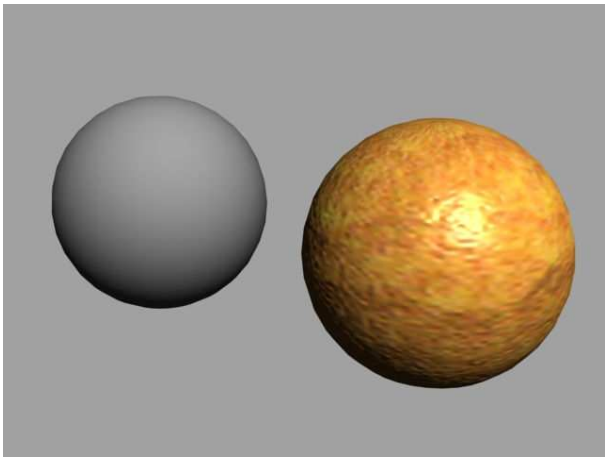


Figure 2 Effect of texturing

Lighting

Photorealism in computer graphics involves two elements: accurate graphical representations of objects and good physical descriptions of the lighting effects in a scene [Ref. 24]. Accurate graphical representation can be achieved by well designed object geometry and textures, and often good results can be achieved with tools available for the modern graphics engines. However achieving good physical description of the lighting of a scene is much more limited and difficult with current real time graphics' techniques. The problem of lighting with the real time graphics is that to achieve photorealistic results, the lighting of the scene should be calculated as a whole, because each object in the scene affects the resulting complete lighting of the scene. Lighting solutions which are able to process the complete scene are far too computationally expensive to be calculated on the fly during the real time rendering. At the moment only local lighting which is calculated per object can be calculated on the fly during

the real time rendering. The more accurate global lighting solutions can be pre-calculated and pre-rendered for the static elements of the scene before the real time rendering and stored into the texture memory. These techniques are discussed in more detail in the following paragraphs.

Local Lighting

Local lighting is the only lighting model which can be calculated on the fly during the real time rendering. Global lighting methods, which simulate the lighting of the scene as a whole give much better end result when considering the image quality but they can't be used for lighting dynamic objects or simulating dynamic lights because the lighting solution is only once pre-calculated before the real time rendering. If some object in the virtual environment can move or is animated, it's surfaces location and alignment in relation with the light sources can change for each rendered frame and hence the effect of lighting must be calculated separately for each frame. The case is the same when light source itself is moving. For these cases the global lighting can't be used. Local lighting methods are often also called dynamic lighting because they can be used in the cases where objects or light sources are moving.

Local lighting methods are processed separately for each object and they calculate only the effect of direct light emitted from the light sources reaching the surface of the object. Local lighting models can't create shadows cast by the other objects located between the surface and the light source and they can't count in the effect of light reflected from other objects. However limited the local lighting methods are, they are essential for being able to add the lighting to every object in the scene. Without lighting the objects don't even appear to be three dimensional as displayed on the following image.

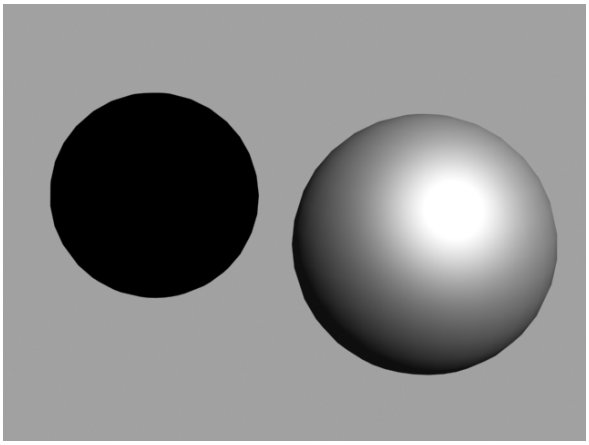


Figure 3 Effect of local lighting

Dynamic Shadows

Shadows are very important part of our way of determining the relationships and locations between objects. Without shadows the realism of a scene is lost and the spatial location of models can be ambiguous. As explained in the previous chapter the local lighting methods are the only way of adding lighting to the dynamic objects, and they are used although they can't create shadows. However shadows can be created by additional methods for these dynamic objects. At the moment the methods used for creating dynamic shadows include techniques

such as planar projected shadows, shadow mapping and shadow volumes. Dynamic shadow creation methods are becoming increasingly popular and standard features of modern 3D graphics engines. The hardware manufacturers creating the graphics cards used in personal computers have been adding components on their products during the last few years which are dedicated for making these dynamic shadow generation techniques more effective.

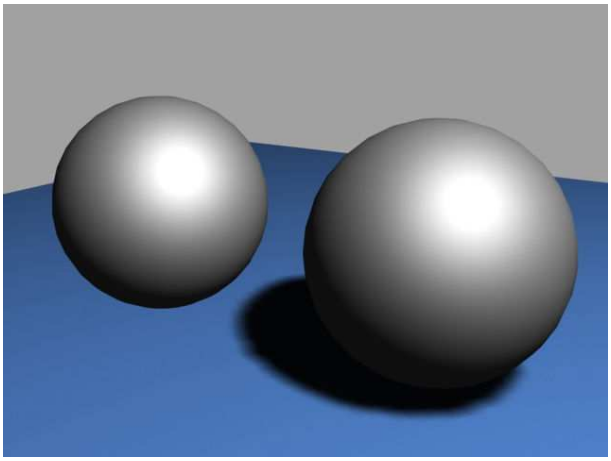


Figure 4 Effect of dynamic shadows

Global Illumination

Global illumination is a term used of lighting techniques which are trying to simulate real world lighting more accurately by constructing the lighting solutions for the whole scene and counting in the effect of indirect light bouncing of the object's surfaces. In general global lighting is divided into two categories, ray tracing and radiosity, although neither of them is actually a full global illumination solutions alone [Ref. 9]. At least at the moment ray tracing is not generally well suited for use in real time graphics and it's not supported by general graphics engines. Radiosity method is better suited and quite often used technique with real time graphics.

In radiosity method the lighting is constructed by calculating the light energy emitted by the light sources and the light energy reflected by the surfaces. Calculating the lighting using radiosity is computationally very expensive but once the lighting has been calculated it can be stored and used independent of the viewing point as long as the light sources and objects in the scene remain static. Lighting created by radiosity method is viewpoint independent because radiosity method only calculates the effect of diffuse lighting and doesn't count in the specular high-lights created by reflections of light from smooth and hard surfaces.

Lighting calculated by the radiosity method can be stored in textures or vertexes of the objects. When using textures to store the lighting, the lighting can be added directly to the textures of the scene, or alternatively additional textures called lightmaps can be used. Lightmaps are added to the original base textures of the objects during the run time. If the lighting is stored in the vertexes, every vertex of the object is resigned with a light value which corresponds the light intensity and color at that vertex and these values are distributed smoothly on the surface and added to the original textures during the real time rendering.

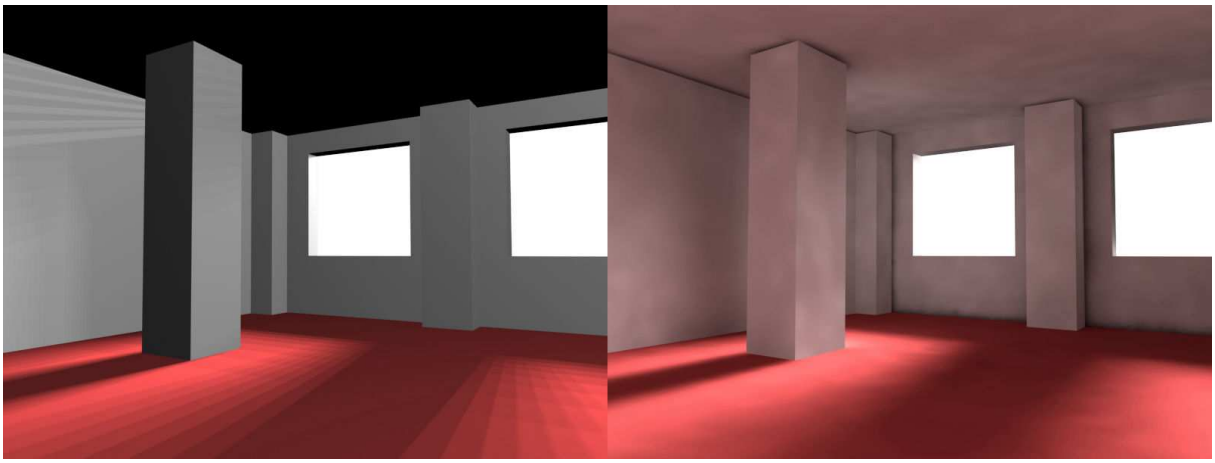


Figure 5 Effect of global illumination

1.12 Virtual Environment's Geometry Representations

In 3D computer graphics the geometry of the objects can be represented in various data structures and stored in data files. These geometry's data files can have different formats and it's up to the graphics engine which of the existing data formats it supports. In general the data formats of geometry representations can be divided into two main categories, polygonal and parametric formats.

In polygonal format all the surfaces of the objects are constructed by using small planar triangles called polygons. Currently polygonal format is the working horse of the real time 3D graphics. Fast evolution of real time 3D graphics is much a result of being able to render polygons more efficiently [Ref. 9]. On personal computers this has been achieved by using hardware accelerated graphic cards. These graphic cards have their own processing units optimized for processing and rendering polygonal data.

However suitable for real time visualization the polygonal geometry is not suitable format of representing object modeled for different engineering needs. Computer aided design, usually referred as CAD, is a term used for modeling process done with dedicated modeling tools, also referred as CAD –tools, in the different areas of engineering design. For example AutoCAD is one popular and widely used CAD program, often used for example in architectural design. CAD software emphasizes exact measuring and easy printing of instructions and blueprints. The three-dimensional geometry designed in CAD software is in parametric format, which means that the surfaces of objects are represented by parametric equations and therefore only define the outlines and measures of the surfaces rather than actual surface.

When we want to visualize these objects designed in CAD software we have to transform the parametric data to a polygonal format. This is quite often the case with virtual reality applications where we want to visualize some design solutions and man made environments. This data format conversion from parametric to polygonal data not very easy task. During my production work the Lumeportti system was tested for architectural visualization and during this development we had a fair amount of experiences with these problems.

The problems and incompatibility issues between different CAD formats and transformations from parametric to polygonal format are well known, but yet still unresolved at large scale. At the moment a big effort for finding solutions to these problems is done by the Web3d

Consortium, former VRML Consortium. Web3d Consortium has recently founded a CAD 3D Working Group, much by the drive from Intel and other industry's big players who also take part into the research work. CAD 3D Working Group has stated it's main motivation as following; "To enable technical cooperation between vendor companies to help resolve the issues of sharing and viewing of CAD/ 3D data via a common file format and viewing engine." [Ref. 25]

1.13 Content Creation Work Flow of a Virtual Environment

By content creation work flow I mean in this written part of my final thesis the process of designing and implementing the virtual environment that will be experienced by the end user of the virtual reality system. This process of content creation starts from virtual environment's 3D geometry creation and ends to the point where the final file structure concealing the virtual environment is transferred and loaded into the graphic rendering engines and is ready to be displayed to the end user of the virtual reality system. Since we are using 3D engine for rendering originally designed and used for computer games, our content creation work flow doesn't differ much from the content creation done when developing a modern 3D computer game's environments if only comparing tools and methods used for the job. In computer game industry this process of content creation is often called level design and within computer game industry this content creation work flow is more explored and better documented than within the research community of virtual reality research.

Content creation work flow can be broken down into smaller parts conveniently by examining the different design phases. Usual design phase sequence could be for example the following; initial design, geometry creation, texturing, lighting, defining functional elements of the environment, exporting files and building the final file structure. I've explained here shortly each of the main phases. Of course the work flow depends much on the 3D engine and content creation tools in use, so this model presented here is just one possible work phase sequence.

Initial Design of Virtual Environment

At the beginning of the virtual environment design it should be considered what is the implemented virtual environment meant to be used for. When designing the virtual environment starting from the usability point of view, it is easier to decide what level of realism is required, what kind of functionalities are required and can the geometry be imported from some source or will it be implemented from a scratch. Also the level of strived realism and functionalities should be designed to be in the balance with the resources that are available for the project.

Geometry Creation

When creating virtual environment designer is constantly solving a problem of how to build visually rich and realistic environment with minimum amount of data. Limits in 3D engine and processing power of computers running the virtual reality system restrict the amount of geometry, textures, light sources and all the other elements used for creating the virtual environment. Designer of the virtual environment has to be constantly aware of these

limitations and often is forced to compromise the quality of the designed environment for the sake of faster performance of the virtual environment.

Geometry, i.e. all the graphical objects in the virtual environment are shaped by designer. Shaping can be done by either directly editing the geometry of the object i.e. modeling or by some indirect way when the actual geometry is generated algorithmically. Direct modeling is the usual method for geometry creation and algorithmic generation is used only in some special cases. Most usual case where algorithmic generation is used, is the generation of large landscape environments, where the shape of ground surface is created according to heightmap images or for example by some fractal methods.

For direct editing of the geometry there are two main categories of tools available. These two modeling tool categories are level editors and general 3D modeling applications. Level editors are developed for the content creation of some specific 3D engine and they are closely tied with the file formats and functionalities used by that 3D engine. Typically popular and widely used 3D game engines like for example Quake and Unreal engines, have their own level editors. Problem with the level editors when used for virtual environment creation is that typically they are too game oriented and don't support geometry import from alternative sources.

General 3D modeling applications are often used with more middleware type of 3D engines. For example 3ds Max, Lightwave, Maya and SoftImage are popular general purpose 3D modeling applications used for building game levels and other real time content. General purpose 3D modeling applications are usually easier to use, or at least generally better known, than level editors and they support wide range of import data formats. Drawbacks of these applications when used for virtual environment creation is that data export for the 3D engine is more complicated than with level editors. Level editors also guide the designer more to make suitable geometry for the real time rendering than general 3d modeling applications which are not originally designed for real time geometry creation. Dedicated level editors also give better visual feedback of what the final environment will look like compared to the general 3D modeling applications which often require constant testing on the final 3D engine which can be quite time consuming process.

The difference in geometry creation when creating virtual environment and not game level is that for virtual environment the content is often already modeled for some other purposes. In virtual environment scenarios it is often desired to be able to manipulate and test objects manufactured and existing in real world. Objects manufactured mechanically are quite often designed and modeled in some CAD software and therefore direct use of these CAD models for virtual environment geometry would ease the work load needed for content creation. As I explained earlier the CAD format data is not generally suitable directly for the real time visualization and therefore the designer must convert the geometry so that it can be used with 3D engine. Usually level editors don't support data import and therefore only general purpose 3D modeling applications can be used in this case.

If the 3D engine in use support some level of detail optimization method and requires that the designer models the different level of detail models manually, they should also be done during this stage of content creation.

Texturing

Texturing is the work phase where textures and other surface material related parameters are applied to the geometry. Texturing may seem like a easy job, but in reality it is an art form. Very subtle changes in object's materials used with well finished textures can make a great difference to the final image quality and realism of the virtual environment. In many cases I have learned that base textures done by editing digital photographs taken from real world surfaces give the best end results. Since digital photographs have all the small details that are hard to create artificially they give more realistic look to the virtual environment's surfaces.

Lighting

As explained earlier there are few different lighting techniques which can be used with real time 3D graphics. No matter what techniques are used, the lighting should be designed as a one complete element of the virtual environment and different outcomes with different techniques should be tested carefully. If realistic image quality is desired from the virtual environment, some global illumination solution should be used to create the lighting to the static elements of the virtual environment. Dedicated level editors tend to have their own radiosity lighting methods implemented as integral part of the level editor. If some general 3D modeling tool is used for content creation, the radiosity has to be done with some radiosity tool available and then find a way to add the radiosity solution to the final scene. Lighting has a big impact on the mood of the virtual environment and it can be effectively used for controlling the user's point of interest in the virtual environment.

Adding Functional Elements

Functionalities and in some cases different physics attributes are applied last to the designed virtual environment. Different functionalities can vary from simple animation of an object in the scene to more complicated functional element, like for example an elevator buttons which can be used to control an elevator in the virtual environment or light switches etc. These functional elements are placed in level editor or general modeling tool as just location points where the 3D engine will place the functionality. Functional elements can only be tested in the final virtual environment run by the virtual reality system and therefore testing of these elements can be quite time consuming especially if the compilation of the data from edit format to the final run time format of the virtual reality system takes long.

Exporting and Building the Final File Structure

After the virtual environment has been created with the content creation tools in use, it must be compiled to the file formats used by the virtual reality system. In many cases this compilation process can have many different steps and it can be the real bottleneck for the design process and testing of the virtual environment. With some virtual reality systems used for product inspection, a small adjustment in the environment geometry can cause extra work of several days just because of many time consuming data conversions and packaging operations needed to transform the data to a format which can be used by the virtual reality system. When choosing tools for virtual environment creation the ease of compilation and packing should be considered quite carefully.

2 Lumeportti Virtual Reality System

Lumeportti is a CAVE –like virtual reality system that has been designed and built by the Product Information Management research group at the Technical Research Centre of Finland. During my production work the Lumeportti system was under development and the development was still ongoing when I finished my production work. This chapter describes the Lumeportti system as it was at the time I worked with it and there may be some changes done to the Lumeportti system after my production work that are not documented here.

Initial goals of Lumeportti development were that the system should be build based on standard PC hardware, the software architecture of the system should be as modular as possible and that the software architecture should be build by using available already existing software components. In this chapter I describe the different parts of the Lumeportti system's hardware and software setup.

2.1 Overview

As a virtual reality system, Lumeportti is a basic CAVE –like system. It has two large 2000 mm x 2000 mm projection screens. Screens are used for rear screen projection and they are made of translucid material which is special made so that it will conserve the lights polarization. The images are projected to the screens with normal LCD data projectors, using two projectors for each screen. The system could be easily extended to have more screens on sides or to have a top screen as a roof to the side screens. It would be even possible to make the floor to be one screen, but that requires some special construction or the use of normal front projection from above.

For the user to attain the stereographic effect there must be a way to display different images for both eyes. The Lumeportti system produces stereographic images by using polarization filtering. Each screen uses two projectors, one is displaying the image for the user's right eye, and the other projector is displaying the image for the left eye. Both of these images are polarization filtered, which is done by fitting polarization filters to the front of the projector's lens. The user is wearing eye glasses that also have polarization filters. The polarization filters on the glasses separates the two overlapping images projected to the screen and makes only one image visible for each eye.

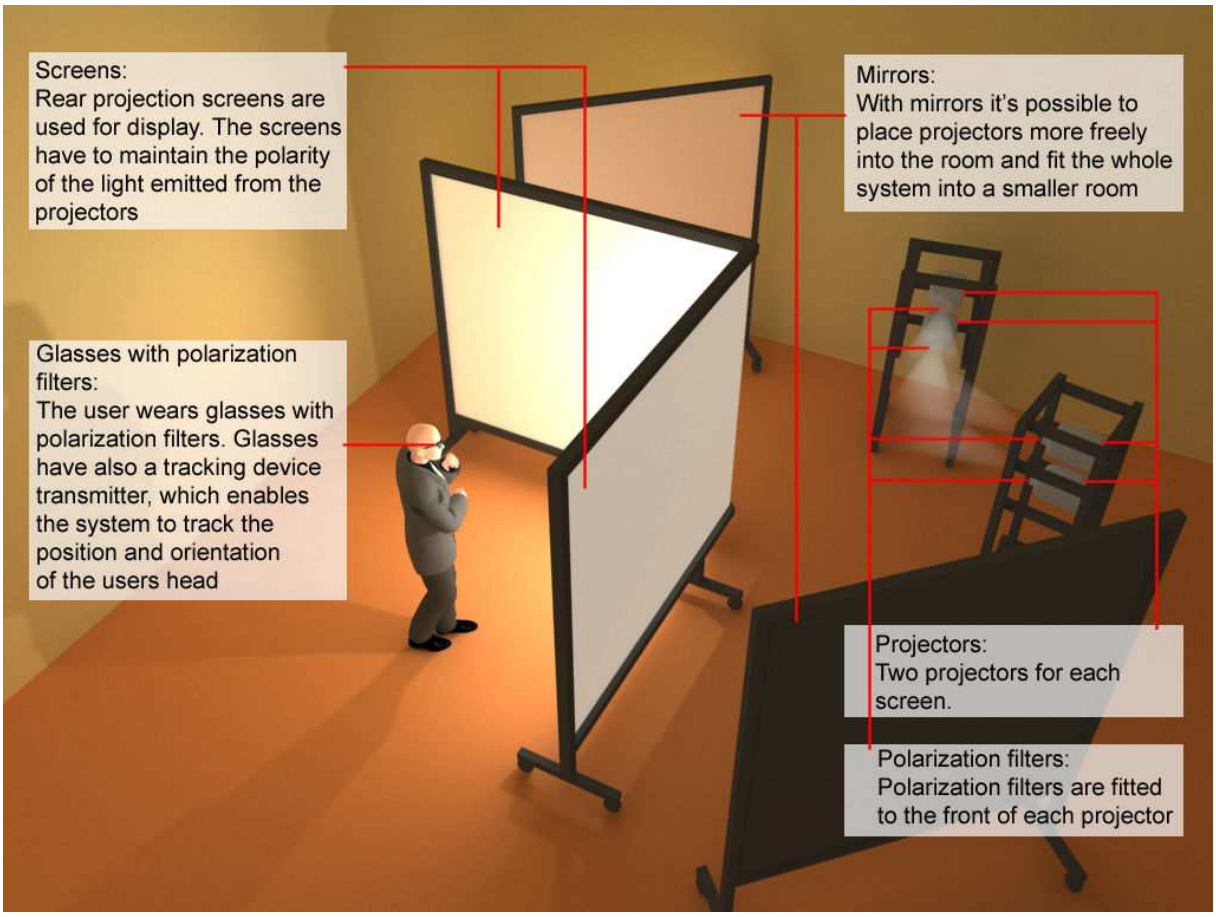


Figure 6 Lumeportti virtual reality system setup

2.2 Lumeportti Hardware

Lumeportti system's hardware is composed of computers, display screens, video projectors, tracking devices, data gloves and spatial sound system. Each of these components is selected bearing in mind relatively low costs and interoperability.

The system is using two rear projection screens for display and two video projectors for each screen to have a stereographic ability. For each of these four video projectors there is a dedicated rendering computer. These four computer used for rendering are standard PCs with display cards normally used and optimized for 3D computer game graphics. Rendering computers run on Microsoft Windows 2000 operating system. Currently each of the rendering computers has approximately the following setup:
- 2 GHz Intel Pentium 4 processor
- 512 MB Ram memory
- NVidia GeForce 4 Ti-4600 128 MB Display card

Besides the four rendering computers the system has one computer dedicated for physics engine, one computer for main server and one laptop used for user input tracking. There is also a possibility to connect additional computers for audio engine and user input handling, currently they are however done in separate processes in the main server.

The whole system is controlled by the main server, which connects to the other PCs via high speed ethernet cabling and a switch. The main server setup is otherwise identical to the rendering PCs except that main server has 1024 MB of RAM memory.
A separate laptop is used for processing data from data gloves and audio input device. This laptop could be attached to the user in cases where more freedom of movement is desired.

The main tracking device used in this setup is IS-600 infrared / ultrasonic tracker. Tracking device is used for tracking the position of users head and also the position and orientation of the users hands.

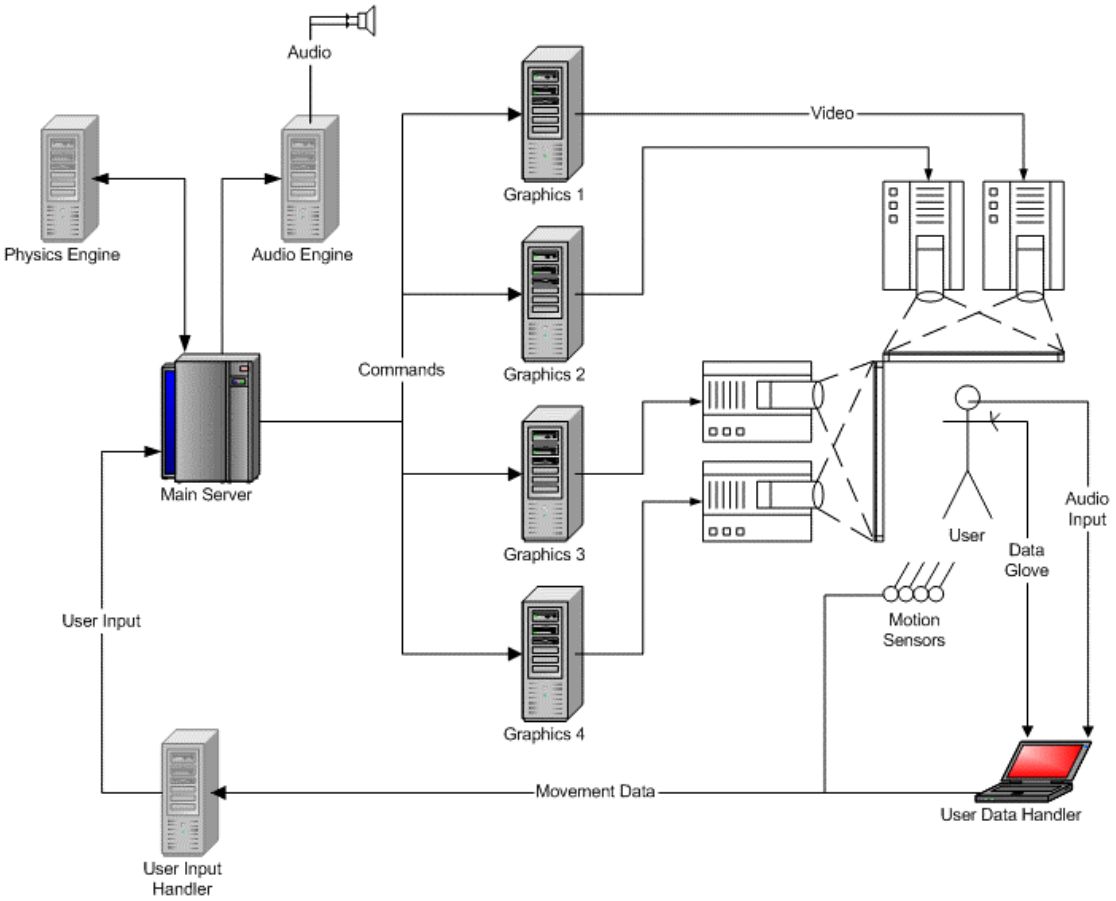


Figure 7 Lumeportti hardware component setup (original image by HUT Lumekirves project group)

2.3 Lumeportti Software

The software environment of the system has been developed as flexible as possible and all hardware dependencies have been avoided as far as possible. The software architecture of the system is modular which means that all the subsystems like for example graphics rendering and physics calculations, have each well defined interfaces and communicate with each other only through these interfaces. Use of these interfaces has made it possible to distribute the software architecture so, that the separate modules run on different processes and can be executed on separate computers connected to the system. Due to this modular architecture it's

for example possible to relatively easily replace physics engine with another one by totally different manufacturer. This flexibility also means that each of the single hardware component is relatively easy to replace or update.

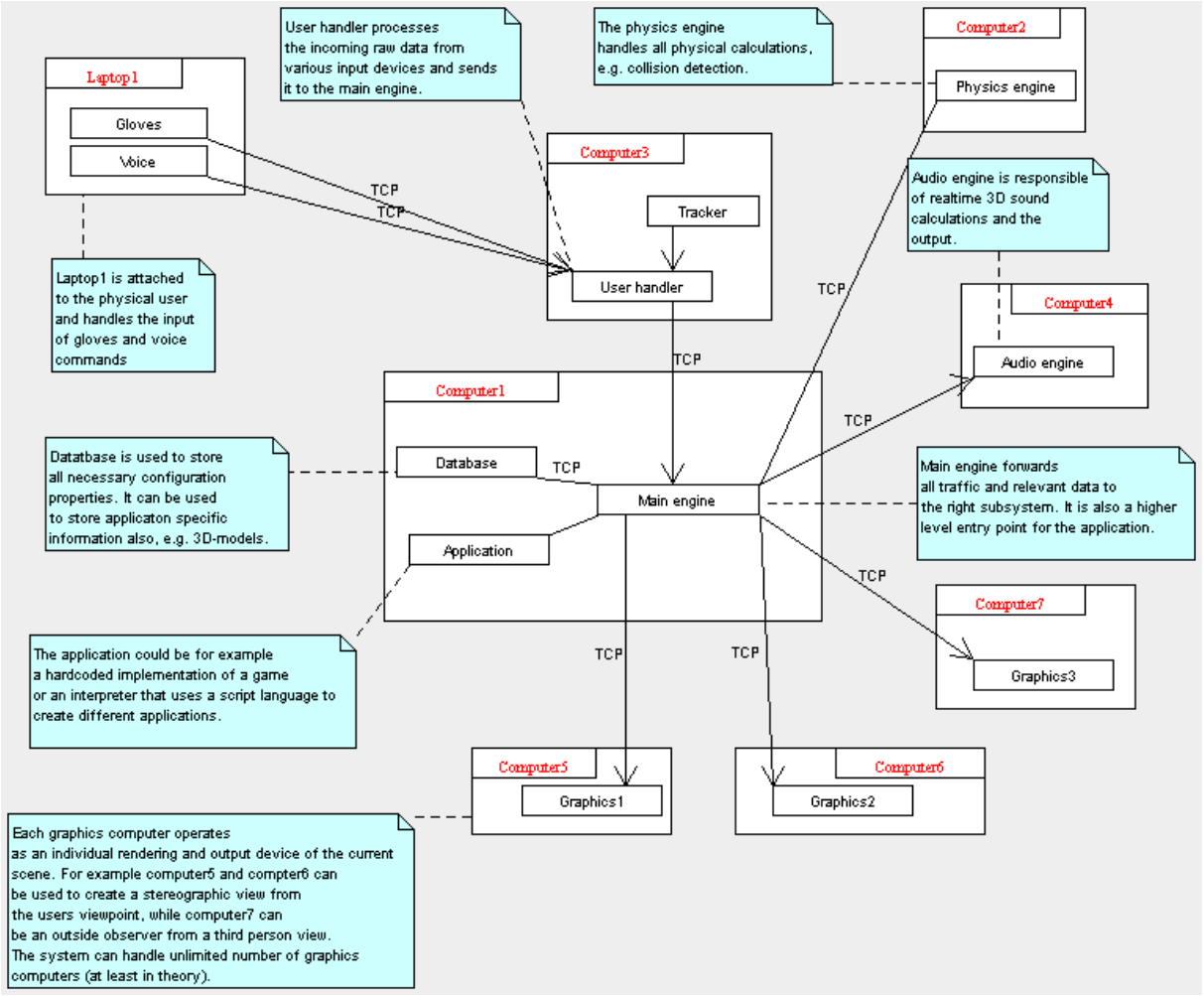


Figure 8 Lumeportti software architecture

3 Starting Points and Challenges of the Production

Before I started the work on Lumeportti system I had earlier worked on a Technical Research Centre of Finland's VIRIKE project, where a large outdoor virtual environment visualization was build. On that project Blueberry 3D engine [Ref. 26] was used for visualizing approximately 10 times 10 kilometer area of a historical landscape. On the beginning of this project I had done an extensive evaluation and comparison study of different available commercial and open source 3D engines. This study was used for selecting 3D engine used in the VIRIKE project. Due to this earlier work on 3D engines I had an adequate knowledge about 3D engines and real time graphics before I started to work on the Lumeportti system.

When I first started working with Lumeportti system it was in a development stage where the hardware system was build and core software architecture used for connecting different software modules was implemented. At that time the software used for rendering on graphics computers and synchronization of these different modules was under development. The goals and challenges drawn forth by the development of the Lumeportti system had it's impact to my production work. The goals and challenges that were raised form the projects under which the Lumeportti system was develop had as well their own effect on the challenges and problems that I faced on my own production work. In this chapter I first explain the goals and challenges of the Lumeportti development and projects related with it and then I'll explain the main goals and challenges I faced on my own production work.

3.1 Lumeportti Development

The development of the Lumeportti system was started since compared with the prices of commercial CAVE –like systems sold as turn-key solutions the approximated price of building an own system from a scratch seemed like the only affordable solution within the VIEW projects budget. From this starting point the initial goal of the Lumeportti system development was to use as much already existing, ready to be used software components and compile the hardware architecture from standard PC hardware that one can find on the selves of any computer store. Using already existing, ready to be used software components, also saved a lot of work effort that would have been required if they would have been build in house. In addition to the savings achieved by this solution it was realized that advantages achieved by the modular structure of the system were certainly an improvement over the other existing virtual reality systems. Modularity allowed construction of the system's software by different modules from different manufacturers and therefore improvements over the earlier CAVE-like systems in the areas of image quality and physics simulation could be easily achieved as a ready to be used features on the system. The big challenge of the Lumeportti development was how to fuse all the different modules together successfully.

3.2 Projects Linked With Lumeportti

Lumeportti system development was originally started under project called VIEW of the Future and later another project called PCVR was merged in to the same development work of the Lumeportti system. VIEW was an EU funded project and it aimed at developing virtual reality tools to be used for different training tasks. Under this project Lumeportti system was to be used as a training platform for assembly sequences that would be executed by astronauts building the International Space Station. In the PCVR project the focus was on researching

the different possible applications of virtual reality under the area of construction industry. One of the application possibilities to be demonstrated on the Lumeportti system was architectural visualization.

The focus in the VIEW project was on complex training tasks of different assembly sequences, which drew forth a set of requirements for the Lumeportti system. During the VIEW project a pilot application was to be build to the Lumeportti system where one assembly sequence case was presented and possible to execute by the user. The case was from assembly sequence of International Space Station, ISS, which would be executed by an astronaut in zero gravity. Implementation of this training application required the Lumeportti system to provide features for multimodal interaction and realistic and comprehensive physics simulation of dynamic objects and their interaction. In addition to this, it was desired that already existing CAD –models of the objects in the scene could be utilized on pilot case's virtual environment implementation.

In the scope of the PCVR project architectural visualization seemed obvious application area of virtual reality for several reason. Different pre-visualization techniques are already used on regular basis on different stages of architectural design process. Quite usual visualization method used with bigger projects is a pre-rendered walk through animations for demonstrating designed building's interior details and surrounding environment. Walk through animations are often aimed for public demonstrations and other design evaluations by people perhaps without experience in examining architectural plans and therefore giving them clearer and easier to grasp visualization of the design. When applying virtual reality for this purpose the architectural design could be examined in even more immersive and realistic manner. In virtual reality people can move around freely in the planned building and experience the space of it in truly three dimensional stereographic manner. In addition to this the use of virtual reality system for architectural visualization makes it possible to add and use different kinds of functional elements in the environment, people could use elevators, open doors etc. Applying Lumeportti for architectural visualization increased the requirements set for the system. The visual quality plays quite major role in the architectural visualizations and the visual quality of the rendered graphics must be enough realistic so that the representation of the design is valid compared to the building in reality. The overall quality of the graphics must also be high so that the visualized design looks good and visualization works in favor for the architectural plan in the eyes of the clients. In this project as well as in VIEW project it was desired that already existing 3D models in different CAD –format could be used directly for the content creation of the Lumeportti system.

When it was decided that I would be doing my production work of my final thesis under the PCVR project, it was agreed that I would be implementing one complete pilot environment used for demonstrating the use of Lumeportti for architectural visualization. This was the starting point for the production, but the situation changed soon and instead of building one pilot environment, it was seen that the whole content creation work flow needed some development and that I would be making content for both of the two separate projects dealing with the Lumeportti development for different uses. So both the development of the Lumeportti system and also the goals of the two projects related with it had their own impact on the challenges I met during my production work.

3.3 Goals and Challenges of the Production Work

Now when subsequently considering the production work I did, it's quite easy to see the key problems I faced and spent most of the time solving during the production. Some of the problems I faced during the production seem now trivial and some were left unresolved. Part of these problems and challenges were apparent right from the start of the project and some of them I realized only now as real problems or challenges when writing document about the production. Many of the requirements I had to consider when doing my production were direct results from the goals and challenges set by the Lumeportti development or the projects related with it, but also many of the challenges I faced during my work were purely generated by the separate area of content creation or design methods related with it. If I would try to compile a list of the problems, challenges and goals I set, faced and found out during the production, the list would look something like this:

- What sort of content creation schema would suit the needs of virtual reality system best
- How to create content for a virtual reality system
- How to reorganize and develop the content creation work flow to make it more usable and efficient
- How to design the content creation work flow to best serve the content creation when in use for creating virtual environments for architectural visualization
- How to apply functional elements to the virtual environment
- How to tie together the different content creation schemes of different Lumeportti software components
- How to utilize CAD models for the content creation
- How to increase the image quality of virtual reality system and how to provide control of factors behind image quality to the designer working on the content creation
- How to build a good virtual environment for the pilot case
- How well does the system as a whole suit the needs of architectural visualization
- How to create any of this with limited resources

4 Production Work and Key Solutions

As I explained in previous chapter the starting point for my production work was to build one pilot case on the area of architectural visualization for the Lumeportti system. However it was soon realized that the whole content creation work flow needed to be set up, which proofed to be not so trivial work and also that the work flow needed additional development to work for the purposes of content creation for Lumeportti system as intended. So my responsibilities within my final thesis production work expanded to include also the actual work of setting up and developing the content creation work flow and during this work I implemented several different virtual environments instead of one pilot environment which was also implemented. But of course before all these additional development needs were fully discovered we did a hopeful try to implement the system with the first version of the Lumeportti software components and content creation tools associated with them.

4.1 First Try; Ogre and Quake III

When I started my production work, the Lumeportti system used free open source 3D engine called Ogre. Ogre is a community project aiming to implement an efficient 3D engine with clearly designed, object oriented programming structure. The project is still on going and the engine was and still is under development. However there were some positive experiences from using this engine and also the fact that there was a source code available for this engine made it favorable candidate for the Lumeportti system's 3D engine and first version of Lumeportti system was implemented using Ogre engine for graphics rendering.

At this stage of Lumeportti development it was desired that the system could be demonstrated as soon as possible to the groups possibly interested in joining the projects related with the Lumeportti development. For this demonstration there was a need for somewhat realistic and good looking building model which could be inspected from outside and inside. We knew that Ogre engine was able to load and render Quake III levels and so it was decided that I would try to build quickly a demo environment using Quake III level editor for the content creation. Quake III is a first person shooter game developed by ID Software [Ref. 17], and it was at that time one of the most commonly used game engines for the state-of-art computer games.

I knew from the experience that it was possible to build fairly impressive game levels with the features that the Quake III engine had and from our tests with the Ogre engine it seemed that Ogre supported pretty much the same set of features in the levels as the Quake III engine. I also had some experience on using different Quake level editors and believed that the level editor was fairly good tool for the content creation. I chose to use Q3Radiant for the level editing [Ref. 18]. This tool is developed for pure game editing and widely used for content creation with Quake engine based games. It has a quite a good set of tools, but the ideology is that the level is build entirely within this tool and there is no way of importing environment geometry modeled on other applications to the editor. The editor is also forcing set of specific rules to the environment geometry that are needed to optimize the render process of the 3D engine.

During the couple of weeks I worked with this first demo I got pretty familiar with the level editor tool and was able to work with it quite efficiently. My aim at this stage was to create as fast as possible somewhat decent looking environment that could be used as a demo of Lumeportti system navigation and stereographic display. Since I'm not an architect and was

working with a tool not really familiar, my starting point for this environment was to try to build something simple as possible that still had a strong feeling of scale and space. For this purpose I sketched quickly sort of generic modern office building layout. As soon as I had a vague idea sketched I started to work with the level editor, building quick mock ups and then trying different modifications. By this trial and error method I was able to put up a scene that was used in the demo. Here's a couple of screenshots of the final demo scene.

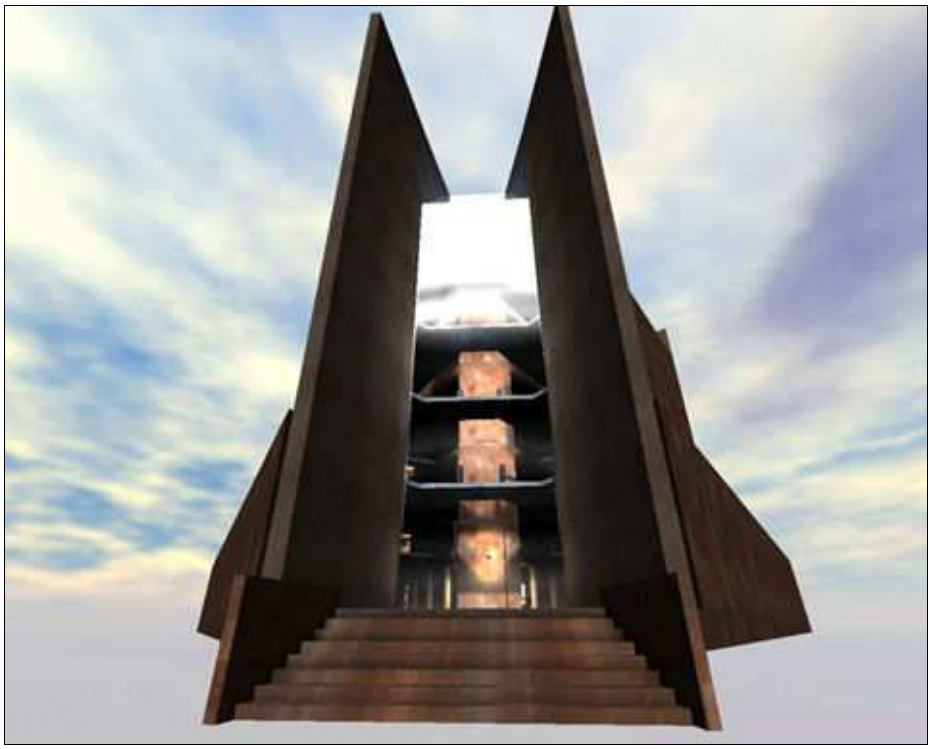


Figure 9 Building created with Quake III level editor

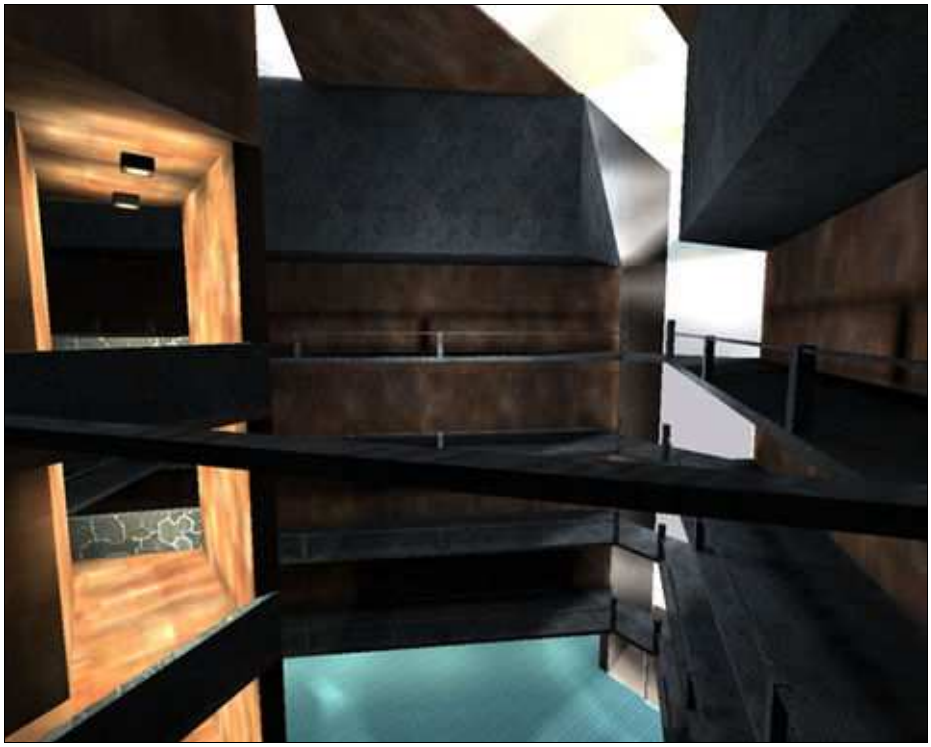


Figure 10 Building created with Quake III level editor

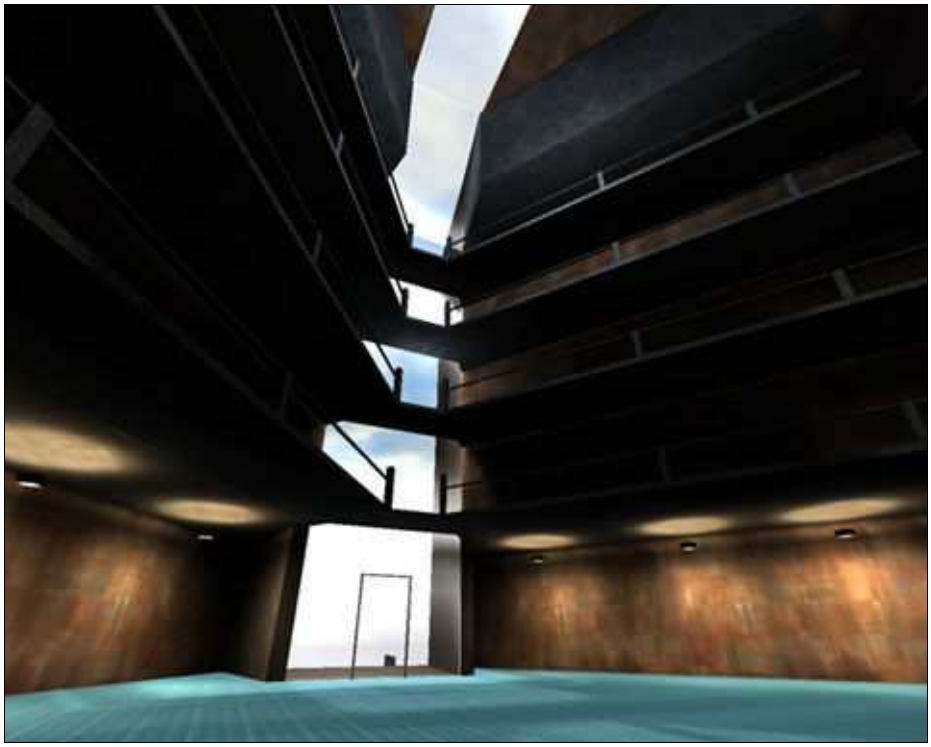


Figure 11 Building created with Quake III level editor

Figure 12 Building created with Quake III level editor

If not counting in the time what it took me to learn to use the level editor, it took me approximately a week and a half to model the building to the point seen on the pictures. If I would have wanted to work on this model to finish it, it would have been a full month of work, or even more.

During this implementation of the demo, it was noticed that the features of the engine and level editor were strictly designed with certain kind of first person shooter game genre in mind and level editor was really not suitable for building architectural environments, since it lacked tools for accurate modeling or measurements, which turned out to be a severe limitation when trying to create accurate architectural model. Also the modeling tools were in general all quite vague about the measures and finesse and lacked the features needed for fine tuning smaller details. A lot of time used for modeling was wasted on checking and correcting rules set by the BSP file format which was used by the engine. Also the fact that there were no means to import models or CAD drawings made with other programs to the editor meant that this model of content creation would not ever be ideal for architectural visualization. In addition to these limitations the lighting features of the editor were too insufficient and the result they produced were not quite realistic.

During this stage of the production I learned that there had been research already done in studying how to apply 3D game engine for architectural visualization. Martin Centre for Architectural and Urban Studies [Ref. 19], at the University of Cambridge has done a research project where they studied the possibilities of real time architectural visualization using Quake II game engine [Ref. 20]. Their results from the project indicated that interactive visualization did offer advantages in understanding the overall structure, especially when communicating with people without architectural education. As well as having good results on the aid this model of visualization gave to the users they also ran into problems caused by the engine and content creation tools. The final results in their research project indicated that

game engines with emphasis on the very narrow game genre were not after all very suitable for use in virtual reality systems.

In addition to these earlier mentioned problems the people working on the software of the Lumeportti system found out that they could not dynamically load objects in to the Ogre scenes build with Quake level editor, which made it impossible to connect the physics simulation to the environment. This and shortcomings on the image quality and lighting forced the people working on the Lumeportti development to face the fact that the Ogre engine was not perhaps the best choice for rendering the graphics on the Lumeportti system. It was decided that Synaptic Soup's freshly released Cipher engine would be fitted to replace the Ogre engine in the Lumeportti system. This was a lucky turn for the work I did on the content creation side, because it gave much more possibilities in re-organizing the whole content creation process and gave much more freedom for me to modify the use of different tools on the content creation work flow. For the rest of the time I worked on my production work, Cipher was the rendering engine used in the Lumeportti system.

4.2 Solution; Increasing image quality

Cipher is a 3D engine released by Synaptic Soup. Synaptic Soup was a development company of videogames and videogame's technology founded by few ex Electronic Arts employees in 2000. Synaptic Soup released their first version of quickly developed 3D game engine Cipher on May 2001. The work at the Technical Research Centre of Finland for fitting the Cipher engine to the Lumeportti began on October 2002. The Cipher 3D engine had impressive feature list and it's performance seemed to be quite competent.

The whole ideology of Cipher was to be more generic middleware type of engine rather than full blown game engine for just one specific game genre like for example the Quake engine. Cipher relied strongly on using Discreets 3ds Max for content creation. The whole 3D object and environment modeling was to be done in the 3ds Max and then that data would be transferred for real time rendering with number of data conversions. For our needs this model of content creation fitted well since 3ds Max had an option of importing already existing models in various data formats. If we would like to use Lumeportti system for architectural visualization we would have to be able to work with various data formats and for that use 3ds Max was one of the best tools since it supports wide range of 3D data formats and there is available more import / export plug-ins for not so well known formats.

From the experiences with the Ogre engine we also knew that if we wanted to model and manage realistic architectural environments, our editor would have to offer better tools for modeling than average game level editors do. The graphics quality that the Cipher engine rendered was also better than the render image quality of Ogre engine. However when the Cipher engine was put in to the test, it was realized that the image quality with the default features of the content creation tools provided with the engine were not high enough for the use in the areas we were trying to apply the Lumeportti system for. The biggest problem was the lack of tools for radiosity lighting. Since the Lumeportti system was going to be used for architectural visualization, radiosity lighting was going to be in a key role when considering the realism and image quality of the rendered environment especially with indoor scenes.

Discreet had just earlier released version 5 of the 3ds Max and it had radiosity lighting and render to texture as new features implemented for that release. I decided to try to apply these

3ds Max features to the real time models of the Lumeportti system and have the radiosity lighting done in 3ds Max.

The radiosity lighting done by 3ds Max was very good in image quality, the core functionalities of this feature are extracted from Lightscape, which is a lighting tool often used in traditional architectural visualization. The render to texture feature of 3ds Max made it possible to calculate the radiosity lighting solution and make this lighting a part of every objects material as an additional texture layer. In the lighting textures the pixel colors are interpreted as reflected light intensity and color. This additional texture layer is called lightmap.

3ds Max render to texture feature can render only one texture per object, so the designer of the scene has to make sure that the environment consist of reasonably sized objects. Rendering one lightmap per object also means that there must be multiple texture mapping channels in the object. In order to be able to export these additional mapping channels I had to write couple of custom plug-in functions to the 3ds Max software. 3ds Max includes a scripting language called max script, which is relatively easy to use and yet powerful sort of programming tool. Using this scripting language I implemented functions for resetting the materials that the 3ds Max assigns to the object when render to texture feature is used. Another plug-in function had to be implemented for exporting shader scripts that included the information of different texture layers of the objects in a shader language which the Cipher engine could understand. Later I merged these two macro scripts into one function which did the both actions at the same time.

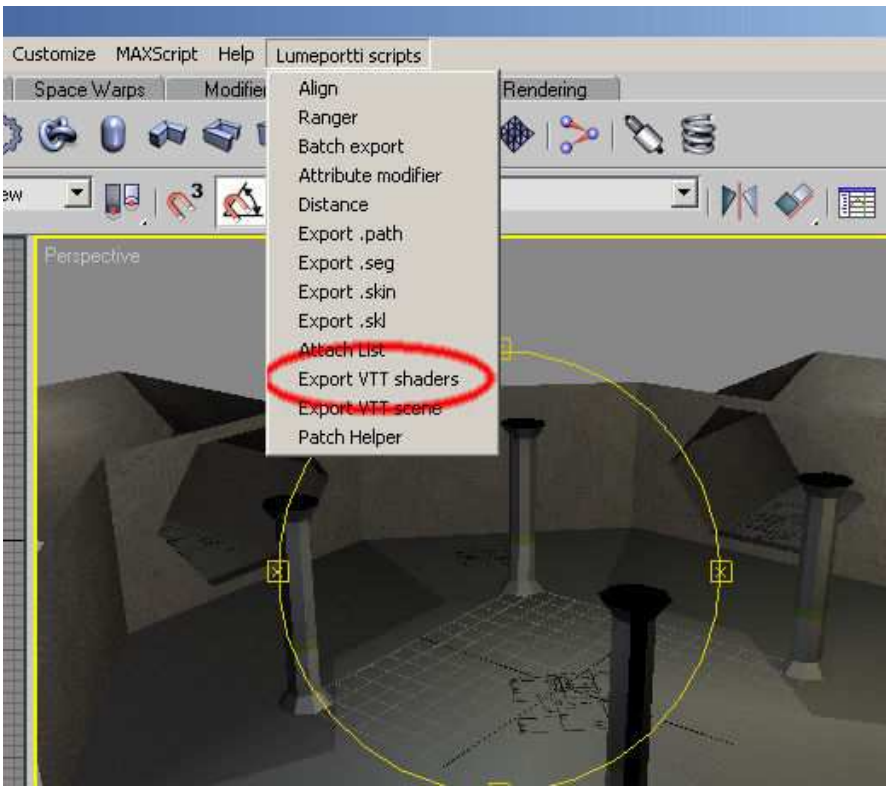


Figure 13 Added macro script function in the 3ds Max's user interface

After I had the Max scripts finished and running I found out that the Cipher's file converter didn't work as expected. File converter handled multiple texture mapping channel faulty. This

problem was however quickly corrected by Synaptic Soup's product support. After these adjustments the first test scene was build quite fast.

Figure 14 First scene implemented for the Cipher engine

4.3 Solution; Binding Together Different Software Modules' Content Creation

One big challenge for the whole Lumeportti development was the question of how to make all the different software modules work together. This challenge also had it's impact to the development of the content creation work flow. The starting point was that the 3D engine responsible for the graphics rendering had it's own content creation scheme and physics engine responsible for the physics simulation had it's own file formats for scene representation. To make the whole content creation work easier and more usable it was desired that these two different types of contents could be designed in one tool and exported together concurrently. For this purpose also the function of the Lumeportti main engine was altered so that it managed the whole scene layout data centralized.

The physics engine that the Lumeportti system used was a CMLabs Vortex engine [Ref. 27] which is a commercially sold physics simulation engine for different kinds of real time physics simulations. The starting point of Vortex engines scene representation was that the physics models of the objects had to be described inside the code by using either some geometric primitives; box, cone, cylinder etc. or more complex freely shaped object could be stored in .obj formatted polygonal geometry files. However the use of physics objects composed of primitives is far more effective than using complex freeform meshes in .obj format when executing physics simulation. This fact ruled out the possibility that all the

objects in the scene could be just simply re-exported to .obj format and then used for physics simulation. The big problem with the Lumeportti system was that the graphical representation of the scene i.e. the Cipher scene was done in 3ds Max but there was no way of defining physics primitive models bind with the graphical objects anywhere else than doing it from inside the code. When the scenes became more complex the effort it took to define all the physics objects in the code was becoming overwhelmingly big and the correct positioning of the physic and graphic models to fit together was very difficult.

Because of these problems we decided to define and try to implement our own scene layout definition file, which could be used for defining the both graphic and physic models of the objects, object placement in the scene and relations between the both the physics and graphics objects. The Cipher engine already used similar idea on defining it's scenes structure, called .seg file. Each object in scene exported to be rendered with the Cipher engine was exported as a separate file and the .seg layout file was used to list all of those objects, their placement and transformations. This list was interpreted by the Cipher engine and all the object files mentioned in the .seg file were loaded, transformations described in the .seg file were made for the objects and then placed in the coordinate points described in the .seg file. Here's an example of the Cipher engine's original .seg file:

```
// Track Segment Reference List
// Source file: temple.max

segment_count 1

segment 01
{
           models/alcove_arch_lintel.mdl
           {
                     priority 1
                     transform 0 -1 0  1 0 0  0 0 1  160 0 0
           }
           models/alcove_arch_lintel.mdl
           {
                     priority 1
                     transform -1 0 0  0 -1 0  0 0 1  0 0 0
           }
           models/alcove_arch_lintel.mdl
           {
                     priority 1
                     transform 1 0 0  0 1 0  0 0 1  0 0 0
           }
}
```

We decided to use the basic structure of the .seg file as our starting point for our own scene layout definition file. For our own file we wanted to add also a definitions of physics objects, their placement and transformation, different kinds of physical attachment between physics objects and lighting and functional elements. To make our own scene layout file work we had to first define it's syntax to a final format so that the necessary changes could be implemented into the scene loading side of the Lumeportti software and the changes needed into to the content creation tools inside the 3ds max. The final syntax of our scene layout file was a result of comparison what could be done in the Lumeportti's software side and what could be done

in the 3ds Max side. The final syntax of the new scene layout definition file is like the following:

```
#VTT Lume Portal / VEView Scene
#Exported from 3d Max file 'testscene.max'


UNIVERSE={
           GRAPHICS_SCENE=iss.seg
           GRAVITY=0.0 0.0 0.0
           PHYSICS_PATH=/phys/
           GEOM_PATH=/models/
           SOUND_SCENE=/sounds/
           SCALE_FACTOR=0.01
END=}

BSJOINT_RELATION={
           NAME=Dummy01
           POBJECT=Box01;Box02
           ORIENTATION=0 0 0
           POSITION=35.0554,49.4649,-0.377228
           UPPER_LIMIT=1.9
           LOWER_LIMIT=1.05
           MOTORED=false
END=}

POBJECT={
           NAME=Box02
           OBJECT_TYPE=DYNAMIC
           GEOM_TYPE=BOX
           GOBJECT=objekti2
           DIMENSION=58.3026, 39.8524, 53.8745
           ORIENTATION=0 0 0
           POSITION=67.8967,0.0,-0.738008
           MATERIAL=METAL
           MASS=1.5
END=}

GOBJECT={
           NAME=objekti1
           ORIENTATION=0 0 0
           POSITION=8.48709,0.0,-1.47601
END=}

GOBJECT={
           NAME=objekti2
           ORIENTATION=-1.5708 0 0
           POSITION=63.0996,12.9151,-5.64538e-007
END=}

POBJECT={
           NAME=Box01
           OBJECT_TYPE=STATIC
           GEOM_TYPE=BOX
           GOBJECT=objekti1
           DIMENSION=59.7786, 65.6827, 64.2066
           ORIENTATION=0 0 0
           POSITION=1.47601,14.0406,-2.22839
           MATERIAL=METAL
           MASS=2.75
END=}
```

Basically our scene layout file has first header part which is used for defining global parameters for the whole environment. This initial part is under UNIVERSE header and it's content comes from an initialization text file, where user can define these parameters. After the header there is a non-organized list of graphical objects, physics objects and different kind of joints and hinges that the physics simulation can use between different physics objects. GOBJECT header is used for graphics objects and POBJECT header is used for physics objects. The implementation of the function which is used for exporting this file was only one part of the modifications I had to implement to the 3ds Max.

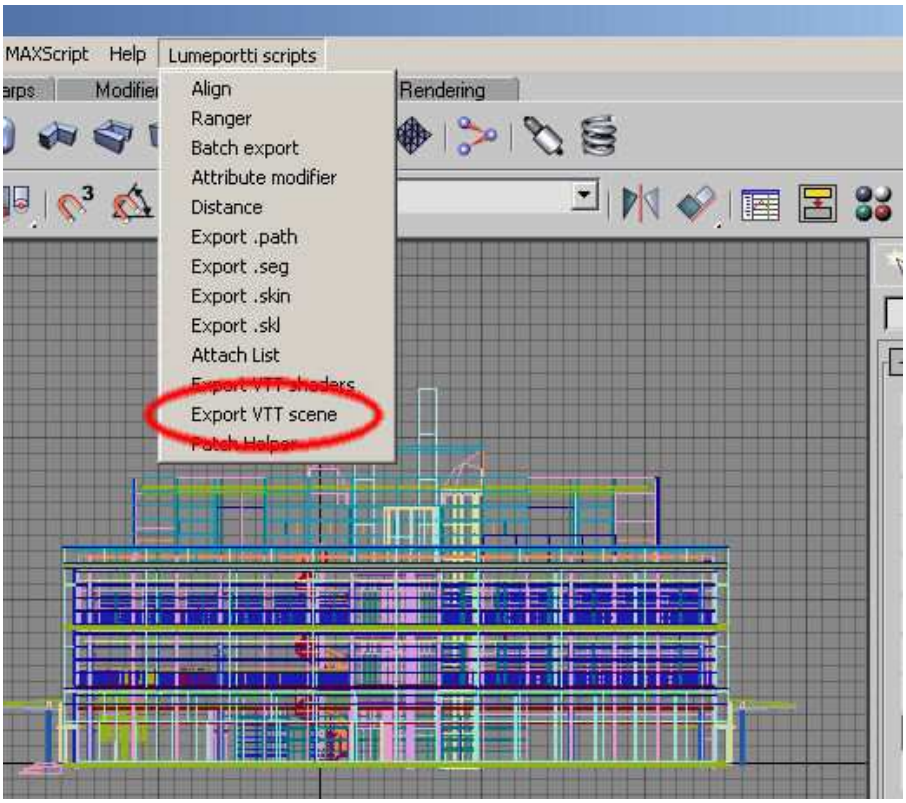


Figure 15 Added macro script function in the 3ds Max's user interface

I had to modify the original Cipher attributes modifier used for tagging graphics objects in the scene for the export and also I had implement completely new max scripted modifiers to be used for physics attributes descriptions and physics joints descriptions. In the appendix of this thesis there are examples of the max script codes I used for implementing these new features.

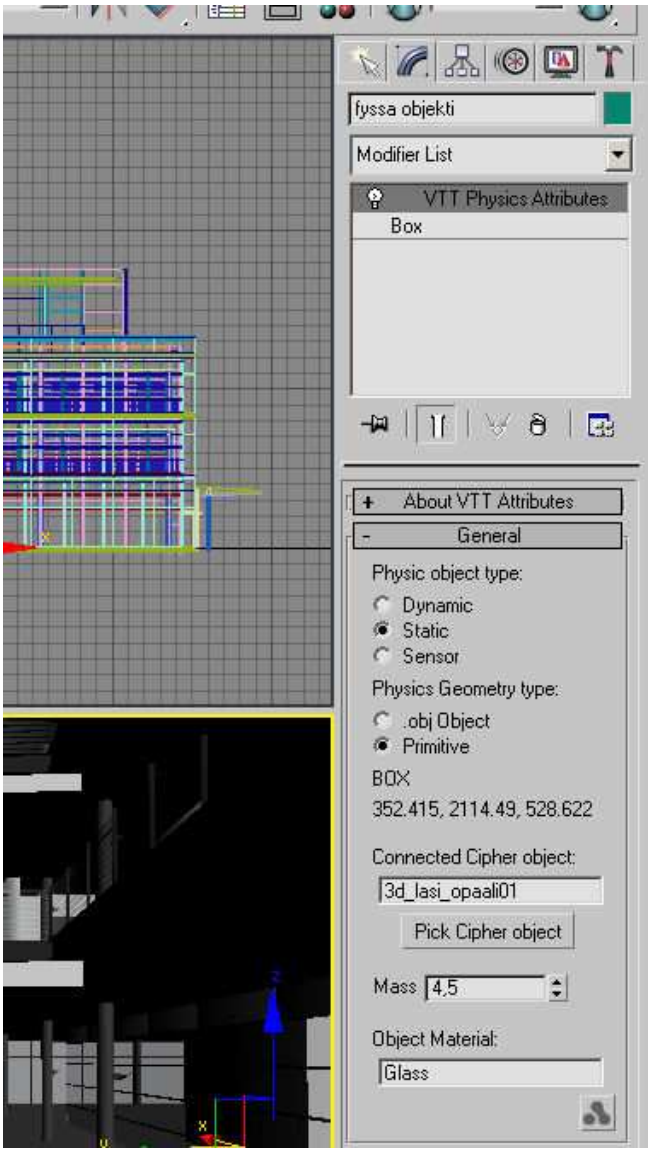


Figure 16 User interface of the implemented physics attributes modifier

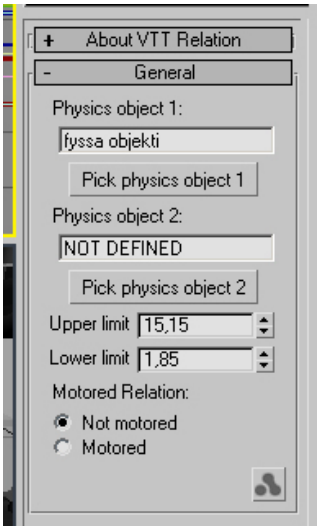


Figure 17 User interface of one of the physics joints modifier

4.4 Solution; Using CAD Data for Real Time Visualization

Subjects visualized or otherwise used as a content in virtual reality applications are very often objects which have their real world counterparts. Very often these real world counterparts are some industrially manufactured or otherwise manmade buildings, vehicles, objects etc. Quite often today such objects are already modeled with some CAD software for manufacturing process or for some other, for example marketing uses.

For a generic virtual reality software it would be a big advantage if it could use directly CAD models for content or had tools which could be used to easily convert the CAD data to a format which in turn could be used as a content in the virtual environment. Earlier in chapter 1.12 I explained the difference between parametric geometry which is used in CAD models and polygonal geometry which is used with real time 3D graphics. At the moment the polygonal format is almost the only geometry format used with the real time 3D graphics, which means that all the CAD models have to be converted somehow to polygonal format if used as a virtual environment content. This conversion from parametric to polygonal format is far from trivial task and at the moment very poorly resolved in general level.

One big problem with the conversion from parametric to polygonal geometry is the large number of different CAD formats, which are nearly all incompatible between each other. It's very hard to find any software which could do the conversion and also supports all the needed file formats. Even bigger problem is that quite often this data format conversion is so complex task that the quality of the results is too poor to be used by any means. The conversion from parametric format to the polygonal format often causes too complex geometry objects with too much detail and high polygonal counts. Another problem is that the resulting polygonal geometry has often errors which are at times hard to detect and very slow to repair. These errors are for example missing polygons, polygons with inversed normals, overlapping polygons, isolated vertexes and so on.

During the Lumeportti development we also had our share of these problems since in couple of the application cases we were working on the content data to be used was delivered to us in CAD format. One of the initial goals of the development of the Lumeportti system and it's content creation work flow was to resolve this problem at least in some level. The main effort for finding a workable way of converting the data to a usable format was done by a student outside Technical Research Centre of Finland and the work was to be part of his final thesis. Although he was able to convert part of the objects in CAD format to a format we could use in 3ds Max the real workable general solution was never found. The objects that were converted to a format usable with 3ds Max were converted first between few different CAD – formats and then imported to the 3ds Max. The actual conversion from parametric format to the polygonal format was done when the geometry was imported to the 3ds Max, and there a lot of correcting work and optimization had to be done for the geometry. Even thought this model was not usable for the general CAD –data import to the Lumeportti systems content creation work flow, it was very valuable work since the objects that were finally converted made it possible to implement the demo applications where this data was needed.

I also tried to find some solution to this problem when working on the Lumeportti systems content creation work flow. I tried to find tools which would have produced better quality conversion from parametric to polygonal geometry or some tools which could be used for easily correcting the errors and optimizing geometry generated by the faulty conversion. It was nice to notice after I had also myself wrested with these problems without finding

solutions that the big players of the industry were as well battling with the same issues. As I explained earlier on the chapter 1.12 where I explained the basic difference between parametric and polygonal geometry, many of the big companies working on the different areas of the real time graphics or CAD have recently founded an consortium purely working on these problems. Web3d Consortium has a separate working group working to find a common data formats and tools for sharing and viewing of CAD and other 3D format data.

Also much based on the experiences from this project Technical Research Centre of Finland has started new projects dealing only with these issues of CAD / 3D formats. The projects have had very good participation from the Finish and international companies working with CAD software or using it for different purposes. It seems that there is a large scale demand for some new solution on this area. So even thought this problem area was not solved under the Lumeportti development, the work done on it has had a good impact on raising the issue.

4.5 Solution; Building the Pilot Application for the Architectural Visualization

The goal set originally for my production work was to implement a virtual environment for a pilot application on the area of architectural visualization. Although during the production work the subject for this pilot case changed and there were lot of additional development needs which were to be done to achieve the other goals set for the Lumeportti system, finally also the pilot case environment was implemented.

When I started my final thesis production work at Technical Research Centre of Finland, the international elevator manufacturer KONE corporation was taking part in to the PCVR project. It was agreed that my final thesis production work would be an implementation of one pilot case virtual environment where the subject of visualization would be a building with KONE's panoramic elevators. However soon after I had started my work, KONE withdrew from the PCVR project. At that stage it was realized that Lumeportti content creation work flow needed further development and that I would be working on that work flow development and some other pilot application subject would be searched and implemented later.

Later during the project I came across to hear about Finnish company Adactive Ltd. [Ref. 28] which is specialized in 3D modeling and visualization on the area of architecture. Adactive had also interest in the virtual reality applications and they had done some successful projects where they had build virtual reality visualizations about various buildings. Based on a meeting with Adactive's managing director Jani Lahti, we agreed that Adactive would deliver some of the models they had build to be tested with the Lumeportti system at the Technical Research Centre of Finland.

One of the models I received from Adactive was a quite large and detailed model of Finish office furniture company Martela's new Business Centre in Pitäjänmäki, Helsinki. The building of Martela Business Centre building had just been finished and it is a modern office building designed by Tommila Architects. The 3D model was very good looking and well modeled, and best of all, it was already in 3ds Max format, so there would not be any problematic file format conversion to get the model into the starting point of the Lumeportti's content creation work flow. It was agreed that the Lumeportti's pilot application about architectural visualization would be build with this model and it was agreed with Jani Lahti that Technical Research Centre of Finland could use the model to demonstrate the Lumeportti system.

Here's couple of photographs of the actual building to give some point of comparison for the results achieved by different methods of rendering.

Figure 18 Photograph of the Martela Business Center

Figure 19 Photograph of the Martela Business Center

Figure 20 Photograph of the Martela Business Center

The actual 3D model was modeled entirely in 3ds Max and lighting had been done with Discreet's Lightscape which was a commonly used lighting tool used for creating more realistic lighting solutions than what the earlier version of 3ds Max were able to produce. The file size of the original 3D model delivered from Adactive was 18,9 MB and together with the textures used for the model the file size expanded to 66,4 MB. The model consisted of 51 separate objects and it used total of over 138.000 polygons.

Here some images of the original model rendered with 3ds Max:

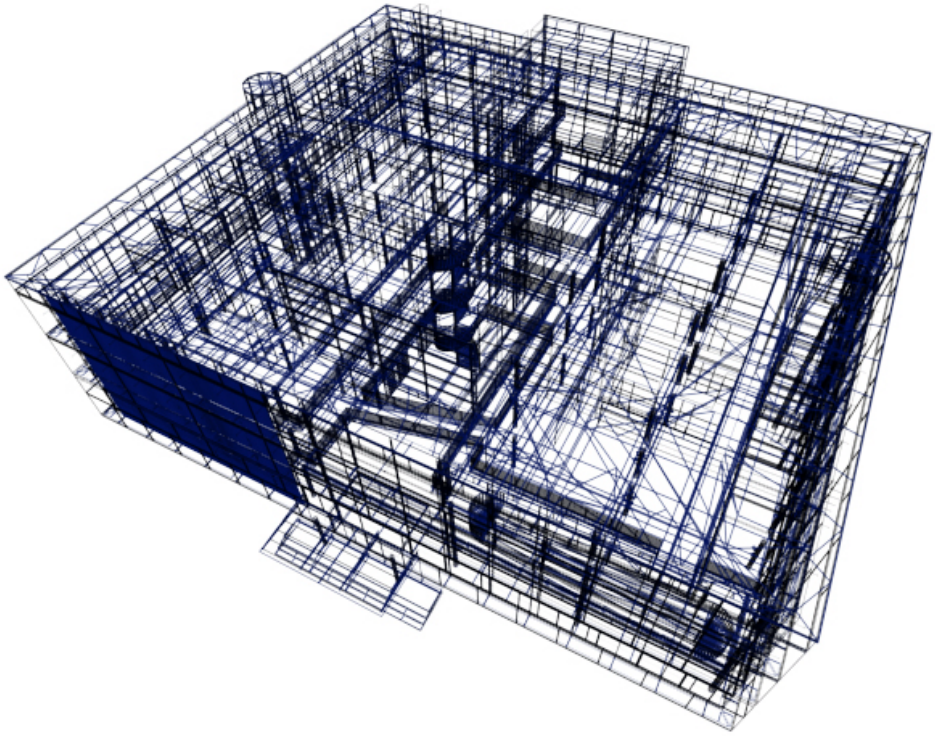


Figure 21 3ds Max wireframe rendering of the Martela model

Figure 22 Martela model rendered with 3ds Max

Figure 23 Martela model rendered with 3ds Max

Figure 24 Martela model rendered with 3ds Max

First when I got the model I tested if it would ever run on the Cipher engine. I made the needed minimal settings to the objects in the scene and exported it for the Lumeportti system. I wanted to check if the data amounts that the scene would produce were anyhow controllable and would Cipher engine be able to render the scene with a decent frame rate. The first result with the Cipher was that the scale of the whole scene was wrong and it didn't look really good since there were no textures, but the frame rate and loading times were really good, so I knew that there would not be any major technical problems that would have had to be first solved before the scene would have worked on the Lumeportti system.

After I had tested that the Cipher engine would be able to render the model without big problems, I started to modify the model to fit the requirements set by the Cipher engine and the whole Lumeportti system. I started by rendering textures to the format which could be used by the Cipher engine. The lighting of the scene had already been done with Lightscape and the results were stored by combining the lightmaps with original base textures of the scene. In this technique where lighting and base texture of surface are combined into one texture, the resulting textures are called baked textures. Earlier when I combined the radiosity lighting of 3ds Max to the content creation work flow of the Lumeportti system, I used separate lightmaps to store the radiosity light. When using lightmaps, the light values stored into the lightmaps are added on the run time to the base textures of the objects. In the alternative technique of using baked textures both the base texture and lighting are rendered into a one texture and that texture is used as normal texture of the surface. Here's an example of one of the original baked textures of the Martela model. The texture is part of the floor in the second floor. The base texture has been light brown floor tile and light and shadows caused by the sharp sun light from the windows is seen on the left side and some soft light from lamps in the ceiling is seen on the middle area of the texture.

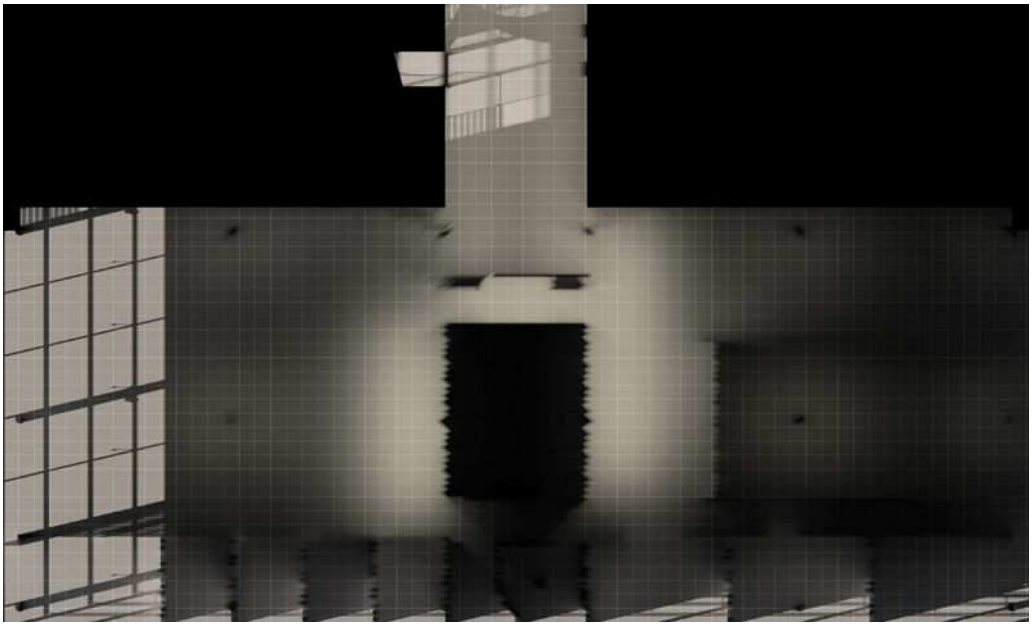


Figure 25 Original baked texture of Martela model

Since there is no way of retrieving the original base textures from baked textures or mapping parameters of the base textures I was forced to use only these baked textures when modifying the scene for the Lumeportti system. In addition the model didn't have the lighting model used for calculating the radiosity solution either, so there was no way I could have been able

to recreate the radiosity calculation with 3ds Max. The original baked textures were however very high quality, and I could have used them directly, but there were problems in their sizes. The Cipher engine demands that the pixel resolution sizes of all of the textures are in power of two (16, 64, 128, 256, ...) and the baked textures of the original model were in complete various sizes. Another problem was that some of the textures were absolutely too big to be used for real time rendering, some of them being in a resolution size of 3600 times 3600 pixels. Size limit of the maximum texture resolution is a result of 3D hardware acceleration cards used by the computer. The 3D hardware acceleration cards can only process certain sized textures and textures exceeding this limit will slow down the real time rendering significantly or textures will not be rendered.

I used 3ds Max's render to texture feature to render the baked textures again in forced resolution sizes in power of two. The overall resolution of baked textures had to be reduced at some extent to fit the memory limits of our system. I did a few tests re-renderings to find a good compromise between the quality of the textures and memory usage. After I had the textures rendered I had to modify the shader export 3ds Max script so that the exported shader script file would use baked textures instead of base textures with light maps as previously. I Also had to add a new shader script to the exported shader file because Martela model had lots of transparent glass surfaces. First I used Shader Designer tools provided with the Cipher engine to create and test a good shader script that would render somewhat realistic looking glass. After I was satisfied to the transparent glass effect I had tweaked in Shader Designer tool, I copied the shader script to the shader list of the Martela model and assigned it to all the glass surfaces of the scene.

After these tasks I tested the scene again in the Lumeportti system. The visual outlook of the scene was otherwise good, but now the errors in the model's geometry became visible. There were number of missing polygons which appeared as holes in the walls and ceilings and also there were number of overlapping polygons, especially on windows of the building model which caused annoying flickers to the real time rendering. I spent some long hours with the 3ds Max when I hunted down these errors and corrected them as well as I could. After these corrections I was satisfied to the outlook of the model in Lumeportti system. I still added a simple sky box to surround the building model to avoid rendering errors caused by the empty space. I also added a simple collision model on a top of the original building model. This collision model was to be used by the physics engine of the Lumeportti system in the physics simulation done in this environment.

Here's some pictures of the final Martela building model rendered with the Cipher engine.

Figure 27 Martela model rendered by Cipher 3D engine

Figure 28 Martela model rendered by Cipher 3D engine

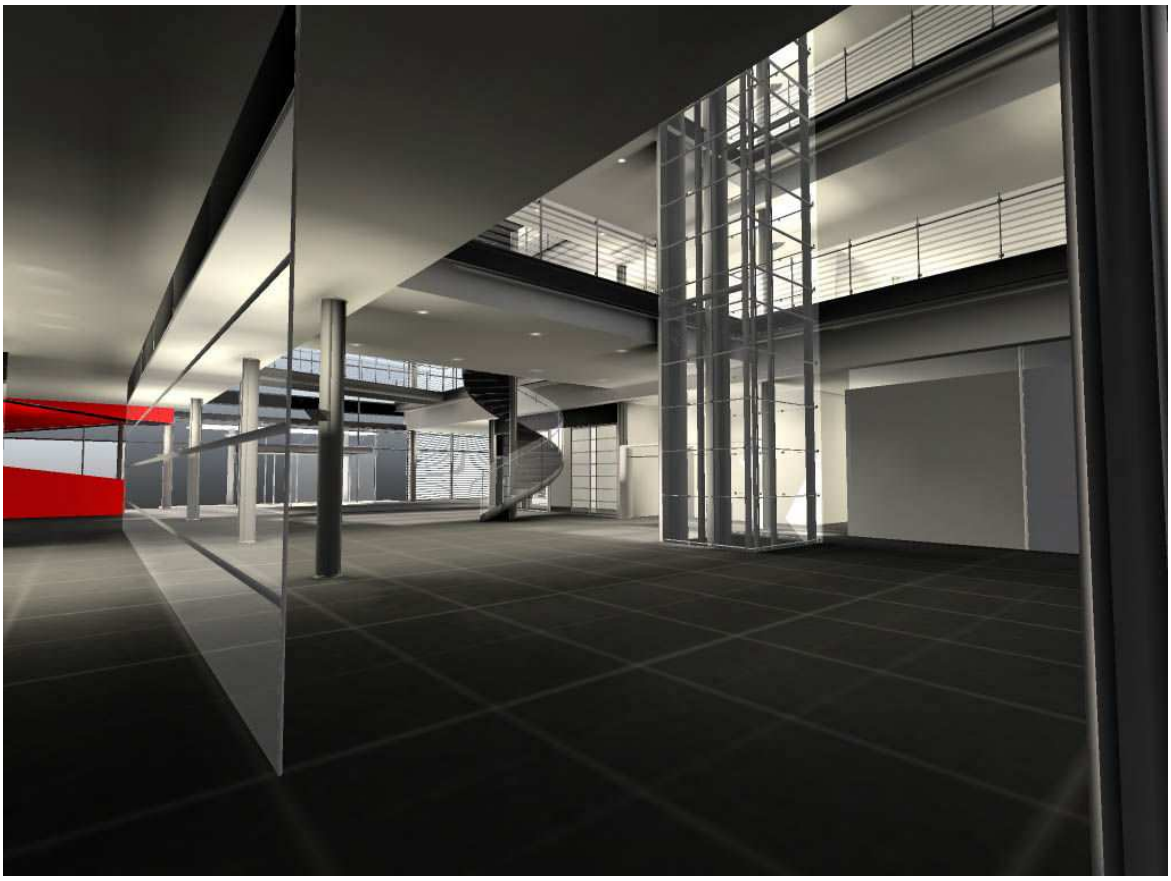


Figure 29 Martela model rendered by Cipher 3D engine

5 Final Results of the Production Work

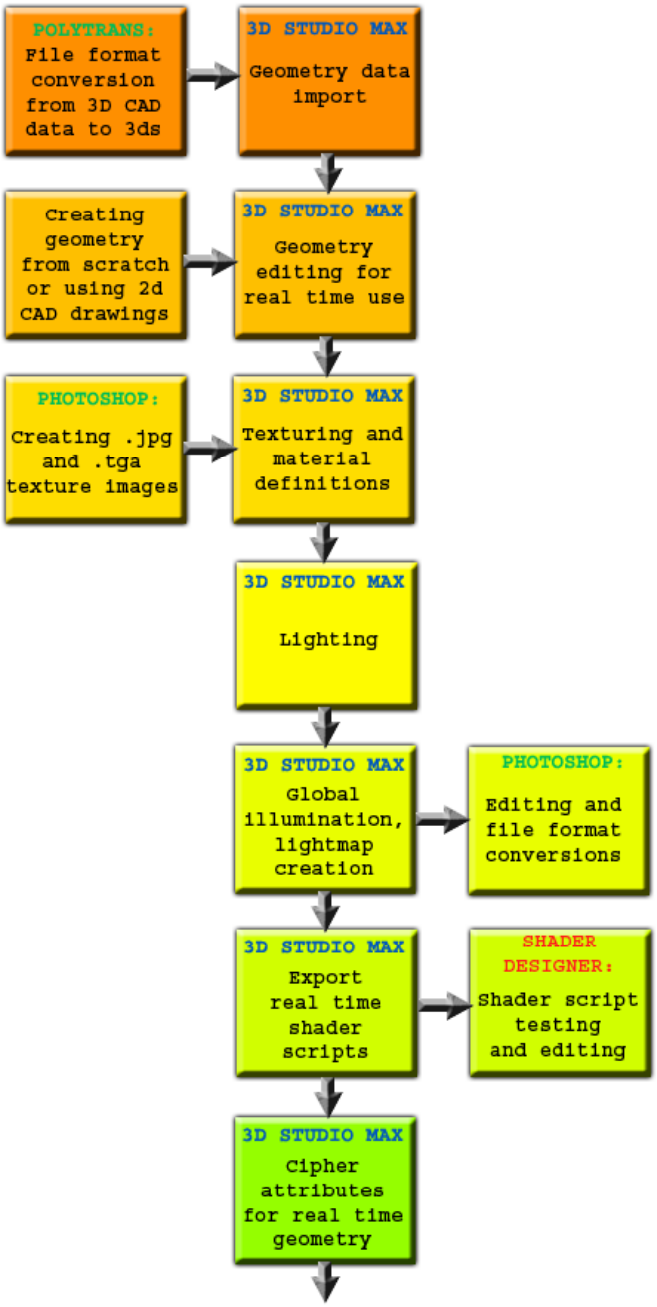
During the production work I did with the Lumeportti system I worked on two major tasks, one was developing the content creation work flow and another was the implementation of virtual environment for pilot application demonstrating the use of the Lumeportti system for architectural visualization. In previous chapter I explained the key solutions found for the problems and challenges I faced during the production work. In this chapter I try to give more complete overall description of what was the final outcomes of the production work. First I describe the final content creation work flow of the Lumeportti system and then I'll say few words about the final pilot case virtual environment.

5.1 Overview of the Final Content Creation Work Flow of Lumeportti System

Overall the final content creation work flow is build on the top of the Cipher engine's content creation work flow, and it's more likely just a modification to the original content creation work flow of the Cipher engine rather than Lumeportti system's completely own content creation work flow. However by this modification it has been possible to add the content creation of the physics engine to the same work flow and that has been far from trivial solution. Also the increase of image quality achieved by merging 3ds Max advanced lighting techniques to the work flow is a key factor for being able to use Cipher engine for architectural visualization.

The final content creation work flow was used with small adjustments on implementation of a virtual environment for pilot application on the area of architectural visualization. After this first complete virtual environment implementation the same process of content creation has been reproduced in other project where similar environment was implemented. So it can be said that the functionality of the work flow has also been proven.

The following diagram shows the different stages and sub tasks of the final content creation work flow of the Lumeportti system. On the diagram there are short descriptions of each stage and programs used in it.



continues on next page

continues from previous page

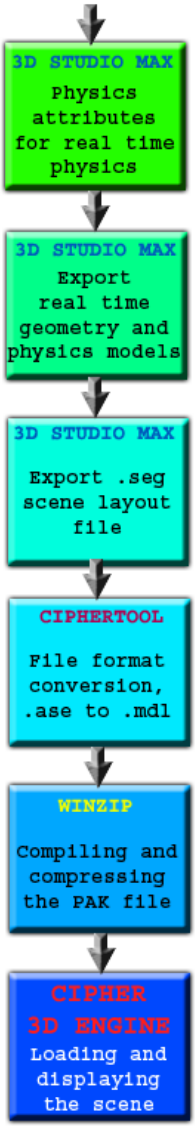


Figure 30 Lumeportti system's content creation work flow

As can be seen from the diagram the content creation work flow has many different steps and it's still far from easy process, but now the whole process has been centered in to the 3ds Max software where all the different elements of the scene can be edited concurrently in one environment with consistent measures and coordinate system.

Shortly described the content creation work flow starts with the geometry creation in the 3ds Max. The geometry can be imported from different sources, but as I explained earlier, there is no general methods for importing CAD data. If user desires to use CAD data on 3ds Max they have to first find workable solution for converting the CAD data at hand to the format which can be imported to the 3ds Max. When I was trying to find some solution for the conversion I had best results with two applications; Polytrans [Ref. 29] and Rhino [Ref. 30]. Polytrans is commercial software developed purely for data format conversions and it supports wide range of data formats. Rhino is a NURBS (Non-Uniform Rational B-Splines) modeling software

which supports many of the common CAD formats. In some cases I had quite good results in the data conversion by using these applications.

If data is imported to the 3ds Max by converting CAD data the next step is checking the geometry and fixing the problem that it might have. Alternative is to build the geometry for the virtual environment from a scratch completely inside 3ds Max. Especially the geometry of the objects which are going to be used for generating dynamic shadows must be checked carefully. Cipher engine's dynamic shadow generation algorithm requires that object's geometry used for shadow generation must be completely closed without any holes in it.

Next step in the content creation is the texturing of the objects in the scene. This is done with basic tools the 3ds Max offers for texturing and material definitions. Here the designer must take in to the consideration that if the base textures used here are also going to be used as base textures in rendering with Cipher engine these textures must comply with the texture rules of Cipher engine; pixel size resolution in power of two and correct file formats.

After the geometry of the scene is modeled and textured the next step is the lighting of the scene. Here it must be decided which elements of the scene are going to be dynamic i.e. moving in some manner and which parts of the scene are static. The static elements of the scene can use pre-rendered radiosity lighting and dynamic objects must be lit by real time lighting. For lighting of the static objects designer can use any of the light source models available in 3ds Max and place them freely in the scene. If static lighting is used it is good idea to use radiosity lighting method to create more natural and coherent light for the whole scene. The lighting of the dynamic objects is done by the 3D engine in real time and here only the places of lights used for this purpose can be defined. Here the designer must remember that dynamic shadows are rendered according to these real time lights and therefore they should be placed compatibly with the lighting used with static objects. Of course the designer can choose not to use pre-rendered static lighting at all, but the image quality of real time rendering with only real time lighting is not very realistic or good.

When lighting is done the lightmaps or baked textures should be rendered for the static elements of the scene. This is done by using 3ds Max's render to texture feature. Here it must be checked that the render to texture parameter is set to produce textures with pixel resolution in powers of two and if using targa format for saving images that the file compression is set off. After rendering the lightmaps or baked textures the shader script file should be exported for the scene. The shader script file is for the 3D engine to know how the objects should be rendered. Shader scripts are exported with macro function implemented to the 3ds Max for the purpose. The macro function also resets the object materials so that they can be exported for Cipher engine.

At latest in this point the designer should model the physics objects that are linked with graphics objects and used for calculating physics simulations. It's good idea to use as much as possible basic primitives; box, sphere or cylinder, since physics engine performs simulation with them much faster than with custom shaped physics models. When all previous stages have been done the scene is ready to be prepared for export. Cipher attributes modifier must be added to all of the graphics objects in the scene. With Cipher attributes modifier designer can define whether object should produce real time dynamic shadows or not. Otherwise this modifier is just a tag for the level exporter indicating that the object is graphics object and should be treated as one. Physics objects are tagged similarly with VTT physics attributes modifier. With VTT physic attributes modifier designer must define all the necessary physics

parameters for physics objects so that it is processed correctly by the physics engine. Similar modifiers could be used in this stage for defining different functional elements to the final Lumeportti scene. At the moment modifiers for different physics joints between objects are implemented.

Next step is that all of the objects in the scene are exported to different file formats according to their types. Graphics objects are exported as .ase files and non-primitive physic objects are exported as .obj files. All of these files should be placed in their own directories which should all be located in one root directory. The same root directory should also have directories for textures, shaders and .seg files.

Last thing done in 3ds Max is the export of .seg layout file of the scene. This file is used by the Lumeportti system as an instruction of how the scene is constructed, where single objects are placed and what physics objects are linked to them.

Finally Cipher engines file conversion tools are used to convert .ase formatted files which were exported from 3ds Max to a .mdl format. This file conversion is needed because here the objects are optimized for the Cipher engine and also for the objects which are set to cast dynamic shadows, the extra data needed for shadow generation is generated for the model. After this all the graphics elements are packed with zip archive application to a one package. By default these packages build with zip archive application have a .zip file extension which must be changed to .pak by just simple renaming the file.

After these operations the final .pak formatted package and other files created during the process are moved in to the correct folders in the main engine of the Lumeportti system and .pak files are distributed to the rendering modules of the Lumeportti system. Then the scene can be executed on the system.

5.2 Pilot Application on the Area of Architectural Visualization

The final pilot application environment I built during my production work was first large scale environment built for the Lumeportti system and first environment built with the new content creation work flow. The environment has been used in several Lumeportti system demonstrations and received general acceptance and positive feedback.

Lumeportti system was able to render the final environment with very good frame rates even though the environment included some complex geometry and lot a of objects with difficult material effects without any geometry reduction. In general the final image quality of the real time rendering done by the Lumeportti system is quite high. The biggest losses on the image quality between the original building model rendered with 3ds Max and the converted model rendered with Lumeportti system are in the texture resolution which had to be lowered for the Lumeportti system and in the semi-transparent reflecting glass surfaces. The Cipher engine cannot render the lighting or ray traced reflections to the glass surfaces. However the effect done with a simple transparency and a reflection map gave at least some illusion of glass surface material. Here’s an image for easier comparison of the difference between the two renderings:

Figure 31 Comparison of image quality between 3ds Max and Cipher 3D engine

The overall experiences from this pilot case in the area of architectural visualization were very promising. My own experiences from using the Lumeportti system for navigating this environment and examining different details gave a feeling that this method was far more intuitive and gave better feeling for the space than a normal pre-rendered walk thru animation. Also the illusion of presence in this environment was quite strong but pleasant. Based on this pilot it was decided that Lumeportti system would be used for architectural visualization and design pre-examination tool during the design and architectural planning process of Technical

Research Centre of Finland's information technology unit's new office building project. On that project the actual virtual reality model was implemented by Adactive Ltd. and the content creation work flow developed during my final thesis production work was used for converting the environment to the Lumeportti system. On that project Lumeportti was actually used during the architectural planning stage and there were even some changes made to the plans based on examinations done in the Lumeportti system.

6 Conclusions and Future Work

Like in every project I've heard about there were also in this production work parts that were successful, parts that will need further work and parts where the right solutions were left unfound. However when considering entire work I did with the Lumeportti system as a whole I feel that the work has been successful. Now afterwards considering the complexity and problems associated with 3D engines and tools associated with them I feel that the resulting work flow was very successful since it was able to fulfill most of the requirements set for it. Also the end results of the production work where I build the pilot application content with the final content creation work flow were quite positive. The content creation work flow proved to be workable and final image quality of real time rendering was good.

Although during the whole production work the content creation work flow development had to be done based on requirements of the 3D engine and available tools, the construction of the final content creation work flow supports well the modular nature of the whole Lumeportti system since it could be applied to even totally different 3D engine with decent amount of modifications to the tools implemented to the 3ds Max. Also there are clear advantages for the designer working with the content creation of the Lumeportti system from the solution of concentrating many aspects of content creation under one tool, 3ds Max, since 3ds Max is one of the best known and generally mastered modeling tools and has good set of tools for various modeling tasks and styles.

Choosing Cipher 3D engine to be used for graphics rendering was a really good choice for the Lumeportti system or maybe it can be even said now afterwards that it was a lucky choice since it was not possible to know all the aspects related with it at the time when the choice was made. In general middleware type of 3D engines work much better than full 3D game engines for some specific game genres for Lumeportti type of system development. The Cipher 3D engine made it possible to use 3ds Max for the content creation and that in turn made it possible to import data from various sources to the content creation work flow and also modify the content creation work flow with reasonable effort. The experiences I had on using dedicated level editors for some specific 3D game engine indicates strongly that in general that type of level editors are not suitable for building complex or realistically accurate virtual reality environments at least not in the area of architectural visualization.

Although the final content creation work flow has proven to be working and improved during the production work I did with it, it's still far from finished. To take full advantage from the content creation work flow it should be optimized to make working with it more efficient. The whole work flow process has still too many different file format conversions and separate export phases in it. Significant portion of designer time what is spend on the content creation is spend on different data conversions, transferring files to correct locations and exporting of separate elements of the scene one by one. However optimization of the work flow would require considerable work effort and new features would have to be implemented to the 3ds Max user interface with max script, different file format conversion would have to be automated and files would have to be distributed automatically to their final locations. However in some extent it would be most likely worth the effort to do some optimization to the work flow if it's used more in the future.

In general when building such a system as Lumeportti system it would be wise to consider the different aspects of content creation tools and work flow more during the design phase of the whole system. After all that is where the real value of the whole system comes from, without

the content used in the system there is only two empty screens. If the system is going to be used regularly for example for visualizing different subjects the major bottleneck for applying the system for that use is the time and work amount it takes to prepare the content for the visualization. With the Lumeportti system even the relatively final and ready to be used materials in 3ds Max take quite a lot of time to convert to the format the Lumeportti uses. The optimization of content creation work flow is required if the system would be practically used for example architectural visualization.

The general parametric CAD geometry conversion to polygonal geometry solution was left unfound in the scope of this production work. Some models were converted by number of file conversions but there were also models which failed to be converted. As I've already mentioned earlier this is a problem commonly known and there seems to be some work going on to solve it in more general level. Based on the experiences that were gained during the development of the Lumeportti system, Technical Research Centre of Finland is now organizing a new project called CAD-Pipe, with focus on solving the problems of CAD format incompatibly issues. Within the scope of this project would be an attempt to build a new database system for storing and archiving all the different parts of a 3D virtual environment and objects data. All different CAD and modeling applications could communicate via this database which could transform the data for appropriate format for each application. Each application could use the data they can process and the complete dataset is always stored in the database. This model could be also connected to the Lumeportti system. Lumeportti could use this same database for it's scene data storage instead of having separate .seg layout files. If this model of content distribution is developed further the whole 3ds Max would then be only one possible tools among all the other possible tools for content creation for the Lumeportti. Within the Lumeportti system's development outside the development done during my production work the use of .seg files have already been developed one step further by modifying the Lumeportti system so that instead of one .seg file the scene can be distributed to multiple .seg files and so the content can be organizing into more manageable chunks.

I feel that based on the experiences gathered during this production it should also be mentioned that in general working with 3D engines can be at times very challenging. This is much because of the fast development in the area of the real time graphics and rapidly changing financial situations of the game industry linked with the development of the 3D engines. During the roughly six months time I worked with the Lumeportti system the 3D engine used with the Lumeportti system was changed or at least tested with different engines in purpose of changing it four times. For example the Cipher 3D engine which has been the primary 3D engine of the Lumeportti system was so expensive when the development was started that only evaluation license was a reasonable acquisition within the project's budget. During the Lumeportti development the licensing prizes of Cipher engine dropped to only fraction of the original prizes before on 30th of May Synaptic Soup officially announced that it was forced to close it's doors due the financial reasons and there would be no more licenses sold for the Cipher 3D engine. At this point the development of the Lumeportti was almost finished but this new turn forced the project to a temporary halt. At this point my production work was just finished and it didn't suffer from this setback. Few months after this it was surprisingly discovered that Cipher 3D engine licenses were sold again, now under new management, and source code licenses were sold for 100$ instead of the old price which was 10.000$. This tells something about the quick turns one might be facing when working with 3D engines.

REFERENCES

[1] Michael Naimark, "VR Today", *Leonardo Electronic Almanac, Vol. 9:5, MIT Press, 2001*

[2] William R. Sherman, "Experiences with Virtual Reality Applications"

[3] Steve Bryson, "Virtual Reality: A Definition History" http://www.fourthwavegroup.com/fwg/lexicon/1725w1.htm

[4] Philip Brey, "The Ethics of Representation and Action in Virtual Reality", *Ethics and Information Technology 1: 5-14, Kluwer Academic Publishers, 1999*

[5] Sari Mokka, Pasi Välkynen, ”Taustaselvitys: Presence”, *VTT Tietotekniikka 2002*

[6] Carolina Cruz-Neira et al., "Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of The CAVE", *ACM Computer Graphics, Vol. 27, Number 2, July 1993.pp 135-142.*

[7] Karten Isakovic, Thomas Dubziak, Kai Köchy, "X-Rooms, A PC –based immersive visualization environment"

[8] Janet Murray "Hamlet on the Holodeck"

[9] Alan Watt, Mark Watt, “Advanced Animation and Rendering Techniques”, *ACM Press 1992*

[10] David H. Eberly, “3D Game Engine Design”, *Academic Press 2001*

[11] Janne Jalkanen, "Building a Spatially Immersive Display: HUTCAVE", *Licentiate Thesis, Helsinki University of Technology, 2000*

[12] www.opengl.org

[13] www.microsoft.com/windows/directx/default.asp

[14] http://ogre.sourceforge.net/

[15] http://crystal.sourceforge.net/drupal/

[16] www.remedyentertainment.fi

[17] www.idsoftware.com

[18] http://qeradiant.com/

[19] http://www.arct.cam.ac.uk/research/index.html

[20] http://news.bbc.co.uk/1/hi/education/982346.stm

[21] Douglas A. Bowman “Interaction techniques for common tasks in immersive virtual environments”

[22] http://www.fakespacesystems.com

[23] David B. Kirk “Interactive 3D Graphics for the Masses”

[24] D Hearn, M P Baker “Computer Graphics, 2nd Edition”, *Prentice Hall, Inc. 1994*

[25] Web3D Consortium, CAD 3D Working Group; http://www.web3d.org/fs_cad.htm

[26] http://www.blueberry3d.com/

[27] http://www.cm-labs.com/products/vortex/

[28] http://www.adactive.fi

[29] http://www.okino.com/conv/conv.htm

[30] http://www.rhino3d.com/