In Proceedings of ACM Symposium on Virtual Reality Software and Technology, pp. 189-196, Sept. 1997.

A Framework for Performance Evaluation of Real-time Rendering Algorithms in Virtual Reality

Ping Yuan Mark Green
Department of Computing Science
University of Alberta
Edmonton, AB. Canada
{ping,mark}@cs.ualberta.ca

Rynson W. H. Lau
Department of Computing
The Hong Kong Polytechnic University
Hong Kong
cswhlau@comp.polyu.edu.hk

Abstract

We describe a framework for a performance evaluation system for real-time rendering algorithms in Virtual Reality. The system embeds a numeric method to evaluate image quality and approaches to construct different types of VR tasks to conduct fair tests of real-time rendering algorithms. We test a real-time rendering algorithm with this framework. Experimental results show that the system supports broad tests of real-time rendering algorithms, and can form the basis for comparing their performance.

Keywords: real-time rendering algorithms, virtual reality, performance evaluation

1 Introduction

Given enough CPU-time, current computer graphics technologies can render very complex scenes and produce nearly photo-realistic pictures. However, for interactive computer graphics, such as virtual reality applications, data management and programming decisions have to be made in trading off rendering quality for interactive update rates. This has been called the real-time rendering problem. In recent years, many algorithms have been proposed to solve this problem. However, the performance of these algorithms have not been evaluated thoroughly. Most of these algorithms were only tested in limited conditions, such as testing the algorithm while doing a fixed path navigation in a simple virtual environment. In addition, there have been very few comparisons between the algorithms. This makes it difficult to determine which algorithm performs the best and under what conditions its performance is optimal. When a new algorithm is proposed, it is often difficult to determine if it is an improvement over existing algorithms, and where it fits into the range of existing algorithms. A performance benchmark for real-time rendering algorithms will solve these problems.

The reason for this limited evaluation is that evaluating real-time rendering algorithms in virtual reality is much more difficult than static rendering algorithms. There are many different hardware configurations and support software configurations; complex virtual environment models are tedious to build; and performance targets are hard to define, complicating the evaluation process. However, a fair and thorough test is crucial to determine how well an algorithm works and how much it indeed helps in solving the real-time rendering problem. A good performance evaluation system can also help identify the direction that real-time rendering researchers should go in the future. To our knowledge, there is no performance evaluation systems that try to conduct a fair and thorough test to these algorithms and compare them.

In this paper, we describe a framework for a performance evaluation system which conducts a broad range of tests on current real-time rendering algorithms, and reports a fair benchmark. We define the performance targets for real-time rending algorithms in VR, which include a metric of image quality and a metric of framerate. We also define and classify typical VR tasks and behaviors, under which real-time rendering algorithms are evaluated. At last, we evaluate a real-time rendering algorithms within this framework and give its performance figure.

Our evaluation system will be used to evaluate a number of the existing real-time rendering algorithms, resulting in a relative ranking of their performance. New algorithms can easily be integrated into the evaluation system, so their performance can easily be compared to existing algorithms. Our software will be available to other researchers, so they can easily evaluate and compare their algorithms.

Our approach solves the problems outlined above in the following ways. By providing a standard software

system for evaluating real-time rendering algorithms we sidestep the hardware and software configuration problem. The complete set of benchmarks can be run on a new hardware and software platform to produce timing information for that platform and allow new algorithms to be ranked there. Since a standard evaluation framework is used, the development of new virtual environments for testing can be amortized over many uses. That is, once a new virtual environment has been produced, it can be used to evaluate all the existing algorithms and distributed to other researchers. Since the evaluation system uses the same performance measures for all algorithms over a range of tasks, the performance rankings produced are a reliable comparison of the algorithms. All the algorithms are evaluated in the same way, so there is no need to translate different performance measures from one hardware platform to another.

The next section of this paper describes a survey of previous works. The performance targets of the real-time rendering algorithms are presented in Section 3. Section 4 introduces the design of typical VR behaviors. Section 5 describes the performance evaluation system design. The experiments and results are reported in Section 6. Section 7 closes with conclusions and future work.

2 Survey of Previous Works

2.1 Graphics Workstations Performance Evaluation Survey

Work has been done on graphics workstation performance measurement. In the 1980's, the Graphics Performance Characterization (GPC) group of the National Computer Graphics Association (NCGA) proposed four levels of performance measurement for graphics workstations. These levels are low-level primitives(points, lines and polygons per second), pictures, systems(input and action response times), and applications. Many attempts have been made at measuring the four levels of graphics workstations performance [23] [22].

Our work is different from the work on graphics workstation performance evaluation. The graphics workstation performance work concentrates on measuring graphics hardware performance over a range of graphics tasks. These tasks include 2D graphics, 3D graphics, windows, visualization tasks and so on. This makes graphics workstation performance measurement a large and complex problem. Our work, however, is on measuring algorithm performance for a particular task, which is real-time rendering in virtual reality. This is a single, well defined task, thus removing the problem of establishing a set of measures that cover a diverse set of graphics tasks. We reduce the effects of different graphics platforms by evaluating the algorithms on the same platform. Our research goal is to evaluate how well a real-time rendering algorithm performs, how much it assists in solving the real-time rendering problem and under what condition its performance is optimal. We are using this framework to assist researchers in improving existing real-time rendering algorithms, and discovering new and better solutions.

2.2 Real-time Rendering Algorithms Survey

Considerable research has been done in the real-time rendering area. This work can be classified into the following categories: Level of Detail Modeling, Geometry Compilation, Visibility Culling, Image-based Rendering, and Hierarchical Image Cache Rendering.

2.2.1 Level of Detail Modeling

The Level of Detail Modeling approach relies on a hierarchical representation of a virtual environment in which objects are described at multiple levels of detail. The level of detail of the representation is selected according to the current time budget. When the time budget is tight, a coarser representation of the objects is rendered. In this way, the target frame rate is guarrenteed and the "best" possible image is produced. [8] [18] [10] [1] [19] are typical real-time rendering algorithms based on Level of Detail Modeling . Funkhouser and Sequin [8] put their efforts on time management, while they generate the multiple levels of detail manually. Their algorithm yields good results on maintaining consistent frame-rate. However image quality is not evaluated and analyzed adequately in their work, as in almost all the other Level of Detail Modeling methods. The overhead of generating multiple levels of detail of the model and storing the huge amount of data also should be considered to evaluate these algorithms.

2.2.2 Geometry Compilation

The idea of Geometry Compilation [9] is similar to that of Level of Detail Modeling in that it simplifies the geometric object's graphics representation to gain speed. The difference is that geometry compilation uses a geometry compiler to generate a proper tessellation of an geometric object at run time. It avoids the preprocessing work of generating multi-resolution models. The performance of this method also needs to be investigated.

2.2.3 Visibility Culling

Visibility Culling algorithms accelerate rendering by avoiding the rendering of objects that are not visible in the image. The object hierarchy can be used to cull surfaces that are beyond the viewing frustum [6]. Spatial subdivisions of scenes is also a good way to cull invisible geometry [17] [21] [11] [7]. The hierarchical Z-buffer [14] is another approach to fast visibility culling. Almost all the visibility culling algorithms need a long preprocessing procedure to construct some object hierarchy or scene hierarchy of the virtual environment. Animated objects in a virtual environment are not suitable for hierarchi-

cal preprocess, which is crucial for the visibility culling approach and therefore is a limitation of this approach.

2.2.4 Image-based Rendering

In recent years, Image-based Rendering has been introduced in the computer graphics community. In image-based rendering systems, the scene is composed by a set of images from photometric observation. Geometric primitives, such as polygons are only used for representing the boundary of the images. Rendering is accomplished by image morphing [3], view interpolation [5], or plenoptic modeling method [13]. Imaged-based rendering technologies can take constant time to render regardless of the geometric complexity of the scene. It also takes advantage of avoiding the difficult problem of the acquisition of accurate and realistic models of the real world. However, this method has the restriction of relying on path coherence, and limits the interaction with particular objects. The performance of this method is proved to be good if the application is restricted to a "fixed path" navigating task.

2.2.5 Hierarchical Image Cache

The Hierarchical Image Cache method is one of the most recent real-time rendering approaches. For this approach, the virtual environment is generated by geometry-based modeling in a static preprocess phase. Then the scene is hierarchically stored in image caches. During the interactive rendering period, instead of rendering the geometric model, the images in the image caches are reused as textures mapped to the polygons which represent its boundary. This idea was first investigated by Regan and Pose [15]. It is called Multiple Frame Buffer Rendering. It relies on hardware with multiple frame buffers. Objects are organized into different buffers based on their distance from the viewer. These frame buffers are updated at different rates. The frame buffers which contain close objects are updated frequently, while distant parts are updated at a much slower rate. The reason is that distant objects do not change or move as much as close ones. They do not need to be updated in every frame. Thus only a small portion of the environment needs to be updated frequently. The final image is created by overlapping the frame buffers form front to back. Recently, some software solutions for hierarchical image cache, have been proposed [12] [7] [20] [2]. They put more effort on the hierarchical organization of the scene, preservation of the image quality or time management, and thus make this approach more attractive. However, a fair performance evaluation of this approach is still required.

All the above algorithms tried to solve the real-time rendering problem by trading off between real-time and realism. Without a fair and thorough test, it is not clear how well these algorithms work. To our knowledge, there is no performance evaluation system that tries to conduct a fair and thorough test of these algorithms and compare them. In the following sections, we describe our performance evaluation system.

We rule out the image-based rendering method at this point, because the current image-based rendering approach is only for VR navigation and is restricted to particular paths while moving through the virtual environment. Once this approach is extended to general purpose tasks, we will port it to our performance evaluation system.

3 Performance Targets of Real-time Rendering Algorithms

Real-time rendering algorithms are for interactive computer graphics applications. Maintaining a constant user-specified frame rate and preserving image quality are the key performance targets. In addition, we certainly do not want to bring in huge amount of extra data which could be several times larger than the original model. It not only increases the workload of generating the data, but also involves memory management problems if the data set is too large. The model preprocessing work is also an overhead for a good real-time rendering algorithm. Extra hardware requirements, such as multiprocessors is also a restriction for general purpose real-time rendering algorithms. When we define the performance targets, all of these factors are taken into consideration.

3.1 Major Performance Targets

3.1.1 Frame Rate

In order to maximize user performance and comfort, any VR system must satisfy the real-time requirement, which means maintaining a constant frame rate that is above a certain threshold. This threshold can be specified by users according to different VR application requirements. Many real-time rendering algorithms try to achieve real-time performance by rendering as fast as possible. The ability of accelerating the frame rate is certainly one criteria for evaluating real-time rendering algorithms. Maintaining a constant frame rate is also very important. Especially when the mean frame-time is high, fluctuations in frame-rate can influence the performance of VR tasks. In Watson's report [4], frame time variation on VR task performance is carefully studied. High and constant frame rates are both important. VR users may feel sick, lose immersive feeling and lose hand-eye coordination during performing a VR task without satisfying either of these two requirements. Therefore, both fast frame-rate and constant frame time management should be considered for any real-time rendering algorithm.

When we evaluate frame rate performance, we test both of the above targets, fast and consistent. We use two methods to evaluate a algorithm. First we specify a frame rate and test the algorithm to see if it can always keep the frame rate above this threshold. The second way is to run a real-time rendering algorithm and record each

frame time. We plot the the frame time and calculate how large the fluctuation is.

For frame rate performance, the following baseline is used. We render a virtual environment with a non-time-critical rendering algorithm, e.g. without involving any real-time rendering algorithm. The frame rate produced by this regular algorithm is the baseline for frame rate performance. Any real-time rendering algorithm should achieve better performance than this baseline, otherwise it should be ruled out as a real-time rendering solution.

3.1.2 Image Quality

In computer graphics community, when we talk about the image quality, we always use 'realistic' as a criteria. However realistic is a subject of much scholarly debate. It involves how accurate the geometric models and fine textures resemble real objects, and how well it captures many of the effects of light interacting with objects. In our performance evaluation system, we are mainly concerned with how well the real-time rendering algorithms preserve image quality. The original image may be a very realistic one, which is generated by accurate geometry, fine textures and very good lighting and shading algorithms. It can also be a rough model. In either case, a good real-time rendering algorithm should preserve the original image as well as possible. For a comparison of the original image and the degraded one, we use the RGB space to compare two images pixel by pixel. Three criteria are used in the comparison. First, average RGBA error between the original image and the degraded one. Second, maximum RGBA error between the two images. Third, number of pixels reflecting the shape distortion of the two images in 2D space. Even though perception-base criteria can give better results, it does not change the relative correspondence of the numerical values of degradation error. Therefore real-time rendering algorithms that produce small errors in RGB space are expected to perform as well as in Lu^*v^* or HSV spaces.

Thus, the problem of image quality measurement can be described as follows:

An image I is an array of N pixels (x, y), Let $Ip_{(x,y)}$ be a 4-dimensional vector, which represents the RGBA value of each image pixel. I_i is the image snapped from the ith frame of virtual environment model rendered by a non-time-critical algorithm. RI_i is the image snapped from the same frame, i.e.the ith frame of the same model rendered by a real-time rendering algorithm. The average error between the two images is defined as :

$$AE_{(I_i,RI_i)} = \frac{1}{N}\sum_{(x,y)\in I_i} \| I_i p_{(x,y)} - RI_i p_{(x,y)} \| \quad (1)$$

where $\| * \|$ is the Euclidean L_2-norm.

The maximum error between the two images is defined as :

$$ME_{(I_i,RI_i)} = \max_{(x,y)\in I_i} \| I_i p_{(x,y)} - RI_i p_{(x,y)} \| \quad (2)$$

where max is the maximum value.

The number of pixels reflecting the shape distortion of the two images is defined as follows:

$$NP_{(I_i,RI_i)} = N_{Ii} - N_{RI_i} \quad (3)$$

where N_{Ii} is the number of non-background pixels in image Ii, and N_{RI_i} is the number of non-background pixels in image RI_i.

The overall error of the two images is defined as the weighted sum of the above three factors, which can be represented by dot product of two vectors :

$$ImagePer_{(I_i,RI_i)} = \begin{pmatrix} Iw1 & Iw2 & Iw3 \end{pmatrix} \bullet \begin{pmatrix} AE_{(I_i,RI_i)} \\ ME_{(I_i,RI_i)} \\ NP_{(I_i,RI_i)} \end{pmatrix} \quad (4)$$

where $ImagePer_{(I_i,RI_i)}$ is the image quality performance of the RI algorithm; $Iw1$ is the weight for average error ; $Iw2$ is the weight of maximum error; $Iw3$ is the weight for number of pixels reflecting shape distortion; $AE_{(I_i,RI_i)}$ is the the average RGBA error between the image from the ith frame generated by non-time-critical rendering algorithm I and the image from the ith frame generated by real-time rendering algorithm RI; $ME_{(I_i,RI_i)}$ is the max RGBA error between the two images; $NP_{(I_i,RI_i)}$ is the number of pixels reflecting the shape distortion in 2D space between the two images.

The combination of the average error, max error, and number of pixels reflecting the shape distortion in 2D space, $ImagePer_{(I_i,RI_i)}$ can give a reasonable estimate of a perceived image difference. However, it is not a perfect one. We will continue to investigate better metrics from the perception domain.

The baseline for image quality performance is designed in the following way: In order to keep the frame rate above a threshold, we randomly delete triangles and texture, or randomly change to less expensive lighting and shading algorithms to render a virtual environment. The image produced by this method is compared with the original image. The error generated by this algorithm is the baseline for image quality performance. The error produced by a real-time rendering algorithm should be less than this baseline, otherwise this real-time rendering algorithm should not be considered as a solution to the real-time rendering problem.

3.2 Other Performance Targets

Besides the above two major performance targets, there are other factors which can also be measured when we evaluate a real-time rendering algorithm. They include the expense of the algorithm's preprocessing procedure and resource requirements.

Our performance evaluation system produces a performance benchmark of a real-time rendering algorithm by

calculating the weighted sum of all the above performance targets.

4 Virtual Reality Tasks and Behaviors

Our performance evaluation system measures the performance of real-time rendering algorithms on a well-defined set of virtual reality tasks. Currently, most VR tasks are the simulation of human behaviors in the real world. They include navigation tasks which concentrate on moving through 3D virtual environments, and operational tasks which concentrate on interaction with one or more objects in a virtual environment. Typical navigation tasks are walking in a building, flying in an outdoor scene or even wandering or jumping in any imagined virtual world. These behaviors can be described as the behaviors of a user (human) holding the camera in a virtual environment. Examples of operational tasks are grabbing a moving fish in a virtual pool, or playing a virtual hand ball game in a virtual hand ball court. No matter what categories a VR task belongs to, or how complex it is, we can always use some behavior modeling technique to analyze it and and simplify it. We use Roehl's [16] hierarchy of behavior to describe VR behaviors. Roehl defines a hierarchy of behaviors containing four levels:

- Level 0: direct modification of an object's attributes, eg. "set the camera location to $< 123.4, 23.4, 567.8 >$"
- level 1: change in an object's attributes over time, e.g. "moving in the orientation of $< 1, 1, 1 >$ at 30cm/sec"
- level 2: series of calls to level 1 behaviors to perform some task e.g. "find a crystal ball in the virtual world"
- level 3: top level decision making, e.g. "decide whether to turn left of right at a corner"

The highest level, level3 behaviors are beyond our research scope at this point. We only consider how the level1 and level2 behaviors are composed in typical VR tasks.

In our system, we provide ways to construct both level 1 behaviors and level 2 behaviors so that fair VR tests can be produced. The level 2 behaviors are based on 6DOF trackers. A user holding a tracker can navigate a virtual environment in whatever way s/he wants, such as any combination of looking up, down and around, walking, running or jumping in 3D space. 6DOF trackers can also simulate any operational task, such as playing hand ball. The position and orientation data from the trackers are recorded. Then, the data are used to test all real-time rendering algorithms. The level 1 behaviors are constructed by defining the mathematical representation of a path and speed, such as walking in 30cm/sec along a B-spline path.

The level 2 behaviors are for testing how the rendering algorithms behave for typical VR tasks. The level 1 behaviors are used to evaluate the algorithm for some particular task, such moving along a path with high path coherence, or with little path coherence. They are also good for some special tests, such as how the algorithm works while speed changes from 10cm/sec to 50 cm/sec.

5 Performance Evaluation System Design and Benchmark

The purpose of this performance evaluation system is to test if real-time rendering algorithms can produce uniform user specified frame rates and preserve image quality as much as possible while performing typical VR tasks in different types of virtual environments. The system consists of several parts, including various virtual environments, VR behaviors designer and recorder, performance evaluator, which include image quality evaluator, frame-rate evaluator, and a interface between the system and real-time rendering algorithms. Currently the system is implemented on Silicon Graphics Crimson/RE with 150Mhz MIPS R4400 processor and 150Mhz MIPS R4010 FP processors, 64MB of memory, 128MB swap and Polhemus Isotrak Trackers. The software configuration consists of IRIX6.2 OS, C++ compiler, OpenGL, Motif, and MR toolkit.

The benchmark is given by the weighted sum of all the performance metrics from section 3 , which include frame-rate performance, image quality performance and other performance measures. It can be represented by a dot product of two vectors.

$$B_{(R_i)} = \begin{pmatrix} w1 & w2 & w3 & w4 \end{pmatrix} \bullet \begin{pmatrix} FP_{(R_i)} \\ IP_{(R_i)} \\ PP_{(R_i)} \\ RP_{(R_i)} \end{pmatrix} \quad (5)$$

where $B_{(R_i)}$ is the benchmark of real-time rendering algorithm i; $w1$ is the weight for frame-rate performance; $w2$ is the weight of image quality performance; $w3$ is the weight for preprocess performance; $w4$ is the weight for resource performance; $FP_{(R_i)}$ is the frame-rate performance; $IP_{(R_i)}$ is the image quality performance; $PP_{(R_i)}$ is the preprocess performance; $RP_{(R_i)}$ is the resource performance. We can also add other performance elements and weight elements to the two vectors.

6 Experiments and Results

We tested Lau's [1] real-time rendering algorithm using our performance evaluation system. While the test was conducted, our system was inactive in a multiuser environment. The main process occupied 90% of the CPU time on average. Lau's algorithm concentrates on multiresolution modeling. We add a predicative time management algorithm so that it can be ported to our test sys-

tem. The frame-time we ask the algorithm to achieve is $50msec$ (the frame-rate is 20 updates per second). Figure 1 and Figure 2 show plots of frame-time and number of triangles while performing a level1 navigation task. Figure 3 and Figure 4 show plots of frame-time and number of triangles while performing level 2 navigation task.

The preprocessing time of their algorithm is 33050 msec. All the time data recorded exclude the workload of the performance evaluation system itself. The benchmark of algorithm [1] is easily calculated by Equation 5 if the weights are given. However, fair weights values should be determined after numerous test of different algorithms.

The image quality of different frames is shown in Table 1.

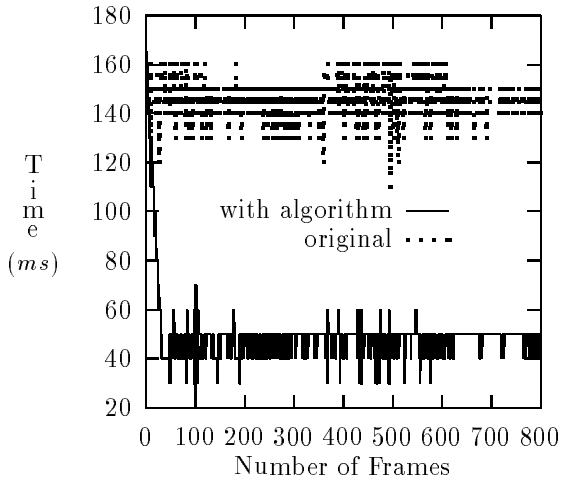

Figure 1. Plots of frame-time while performing level 1 tasks.

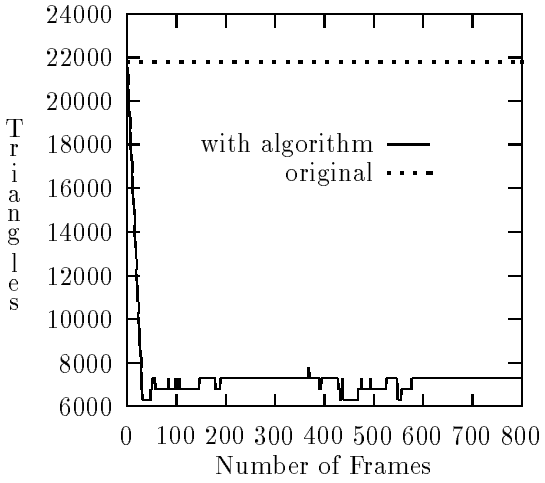

Figure 2. Plots of Number of triangles in the VE while performing level 1 tasks.

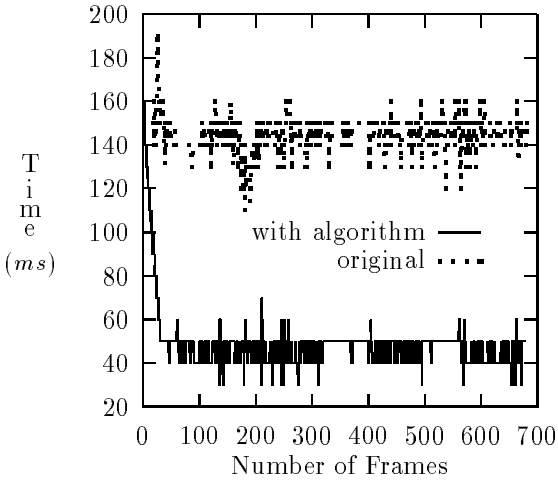

Figure 3. Plots of frame-time while performing level 2 tasks.

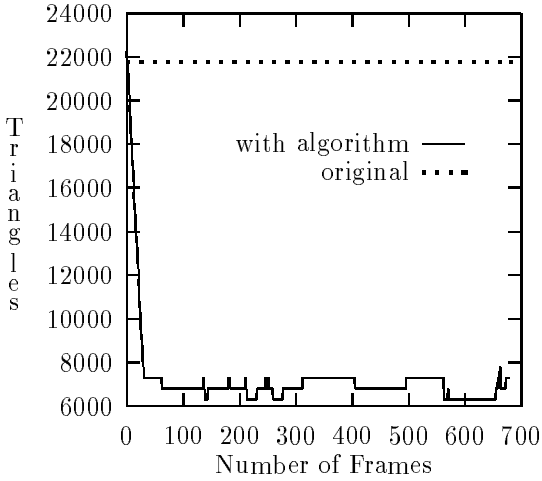

Figure 4. Plots of Number of triangles in the VE while performing level 2 tasks.

7 Conclusions and Future Work

This paper has presented a performance evaluation system for real-time rendering algorithms in Virtual Reality. The key contributions of this frame work are a metric for real-time rendering algorithms, performance targets, and methods to construct different types of VR task. The experimental results reported in section 5 show that the performance metrics are reasonable and the approach for constructing VR task is capable of producing and recording various VR tasks for tests.

Interesting topics for further study include more optimization of the image quality metrics, and obtaining fair weights for all the performance targets, control of the degree of freedom of 6OF tracker to facilitate VR task design, and extensive tests of current real-time rendering algorithms.

8 Acknowledgment

We are grateful to the Computer Graphics Group in the Department of Computing Science, University of Alberta, for many interesting discussions and support to this project.

References

[1] Daniel G. Aliaga. Visualization of Complex Models Using Dynamic Texture-based Simplification. In *IEEE Visualization'96*, pages 101–106, April 1996.

[2] T. Beier an S Neely. Feature-Based Image Metamorphosis. In *Proc. ACM SIGGRAPH '92*, pages 35–42, August 1992.

[3] Neff Walker Benjamin Watson, Victoria Spaulding and William Ribarsky. Evaluation of the Effects of Frame Time Variation on VR Task Performance . Technical report, Georgia Institute of Technology, 1996.

[4] Shenchang Eric Chen. QuickTime VR - An Image-Based Approach to Virtual Environment Navigation. In *Proc. ACM SIGGRAPH '95*, pages 29–38, August 1995.

[5] James H. Clark. Hierarchical Geometric Models for Visible Surface Algorithm. In *Comunications of the ACM*, volume 19, pages 547–554, October 1976.

[6] Jonathan Shade Dani Lischinski David H. Salesin Tony DeRose and John Snyder. Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments. In *Proc. ACM SIGGRAPH '96*, pages 75–82, August 1996.

[7] Thomas A. Funkhouser and Carlo H. Sequin. Adaptive Display Algorithm for Interactive Frame Rates During Visulization of Complex Virtual Environments. In *Proc. ACM SIGGRAPH '93*, pages 247–255, August 1993.

[8] Mark Green. A Framework for Real-Time Rendering in Virtual Reality. In *VRST'96*, pages 3–9, July 1996.

[9] P.S. Heckbert and M. Garland. Multiresolution Modeling for fast rendering. In *Graphics Interface'94*, pages 43–50, May 1994.

[10] Rynson W.H. Lau, Mark Green, Danny To, and Janis Wong. Real-Time Continuous Multi-Resolution Method for Models of Arbitrary Topology. submitted for publication.

[11] David Luebke and Chris Georges. Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets. In *Computer Graphics(1995 Symposium on Interactive 3D Graphics*, pages 105–106, April 1995.

[12] Paulo W.C. Maciel and Peter Shirtley. Visual Navigation of Large Environments Using Textured Clusters. In *Computer Graphics(1995 Symposium on Interactive 3D Graphics*, pages 95–102, April 1995.

[13] Leonard McMillan and Gary Bishop. Plenoptic Modeling : An Imaged-based rendering System. In *Proc. ACM SIGGRAPH '95*, pages 39–46, August 1995.

[14] Michael Kass Ned Greene and Gavin Miller. Hierarchical Z-Buffer Visibility. In *Proc. ACM SIGGRAPH '93*, pages 231–238, August 1993.

[15] Matthew Regan and Ronald Post. Priority Rendering with a Virtual reality Address Recalculation Pipeline. In *Proc. ACM SIGGRAPH '94*, pages 155–162, August 1994.

[16] Bernie Roehl. Some Thoughts on Behavior in Virtual Reality Systems. *http://ece.uwaterloo.ca/∼broehl/behav.html*, pages 0–7, August 1995.

[17] John M. Airey John H. Rohlf and Frederick P. Brooks. Towards Image Realism with Interactive Update Rate In Complex Virtual Building Environments. In *Computer Graphics(1990 Symposium on Interactive 3D Graphics*, pages 41–50, March 1990.

[18] Jarek Rossignac and Paul Borrel. Multi-resolution 3D Approximations for Rendering Complex Scenes. Technical report, IBM, 1992.

[19] Bradford Chamberlain Tony DeRose Dani Lischineski David Salesin and John Snyder. Fast Rendering of Complex Environments Using a Spatial Hierarchy. In *Proceedings of Graphics Interface'96*, May 1996.

[20] Gernot Schaufler and Wofgang Sturzlinger. A Three Dimensional Image Cache for Virtual Reality. In *Proceedings of Eurographics'96*, pages 227–248, April 1996.

[21] Seth J. Teller. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, UC Berkeley, October 1992.

[22] Steve E. Tice, Mike Fusco, and Paul Straley. The Picture Level Benchmark. *Computer Graphics World*, pages 123–130, July 1989.

[23] Micheal J. Zyda, Mark A. Fichten, and David H. Jennings. Meaningful Graphics Workstation Performance Measurements. *Computers and Graphics*, 14(3):519–526, 1990.

Image rendered by different model	Average Error	Maximium Error	Number of pixels reflecting shape difference
Original Image (4356 triangles, 2278 verts)	0.0	0.0	0 pixels
Image 10(4346 triangles, 2273 vertices)	0.015967	10.392305	0 pixels
Image 20(4336 triangles, 2268 vertices)	0.022409	10.392305	0 pixels
Image 50(4306 triangles, 2253 vertices)	0.046020	10.392305	0 pixels
Image 60(4296 triangles, 2248 vertices)	0.052660	10.392305	0 pixels
Image 70(4286 triangles, 2243 vertices)	0.059780	10.392305	0 pixels
Image 100(4256 triangles, 2228 vertices)	0.092639	10.392305	0 pixels
Image 200(4156 triangles, 2178 vertices)	0.155883	10.392305	0 pixels
Image 300(4056 triangles, 2128 vertices)	0.259701	27.712813	0 pixels
Image 400(3956 triangles, 2078 vertices)	0.350149	27.712813	0 pixels
Image 500(3856 triangles, 2028 vertices)	0.419044	27.712813	0 pixels
Image 800(3556 triangles, 1878 vertices)	0.733365	111.664677	4 pixels
Image 1000(3356 triangles, 1778 vertices)	0.949656	111.664677	4 pixels
Image 1200(3156 triangles, 1678 vertices)	1.175150	111.664677	4 pixels
Image 1500(2855 triangles, 1527 vertices)	1.538637	114.284732	8 pixels
Image 1800(2551 triangles, 1373 vertices)	1.950621	114.284732	9 pixels
Image 2000(2346 triangles, 1268 vertices)	2.253247	114.284732	7 pixels
Image 2300(2037 triangles, 1107 vertices)	2.812113	114.284732	9 pixels
Image 2500(1831 triangles, 997 vertices)	3.162840	114.284732	15 pixels
Image 2800(1522 triangles, 832 vertices)	3.783913	136.605271	10 pixels
Image 3000(1317 triangles, 724 vertices)	4.382111	144.675499	20 pixels
Image 3300(1006 triangles, 559 vertices)	5.389454	179.323730	25 pixels
Image 3500(800 triangles, 452 vertices)	6.336828	179.323730	52 pixels
Image 3800(493 triangles, 278 vertices)	9.562351	179.323730	224 pixels
Image 4000(306 triangles, 175 vertices)	19.172414	179.323730	1481pixels
Image 4300(306 triangles, 175 vertices)	19.172414	179.323730	1481pixels

Error between the original image and the image created by simplied model

45010 out of 26144 pixels in the model.

fAngleThresh = 0.4 fVerErrLevel = 0.4

Table 1: Image Quality Performance of Lau's Algorithm