A Report on Master's Project entitled

RemoteVIS: A Remote Rendering System

by

Soumyajit Deb

200207002, MS by Research

International Institute of Information Technology

Gachibowli, A.P., India. 500 019.

sdeb@gdit.iiit.net

Advisors

Dr. P. J. Narayanan (pjn@iiit.net)

Nov 30, 2003.

Contents

List of Figures

Abstract

Large graphics models require a large amount of memory to store and high end graphics capabilities to render. It is useful to store such virtual models on a server and stream it to a remote client on demand for visualization. In many dynamic environments, this is the only option available. We present the design and implementation of `RemoteVIS`*, a system to perform . The goal of our system is to provide the best visualization quality given the capabilities and connection parameters of the client. The system is designed to adapt to a wide range of clients and connection speeds. The system uses a suitable streaming representation based on the situation. This could range from geometry-based representation combining levels of detail and visibility culling to an image-based representation for clients with low capability. The client's graphics and storage capabilities, the speed of viewer motion, and the network bandwidth and latency are taken into account for this. The client acquires model data sufficient for an interval in the future using path prediction for freeze and jerk free rendering. In a remote rendered walk through, the virtual environment including all models, textures and other data are stored on the server side. The client only receives the required parts of the virtual environment that the client is viewing without having to download the entire virtual environment. One of the major bottlenecks of the system is network bandwidth between the server and the client. The system optimizes the rendering quality and mesh detail of the world based upon this bandwidth. The geometry based renderer uses Object Based Visible Space Model Representation algorithms for selecting visible geometry to be transmitted to the client. This automatically culls out all hidden and invisible objects in the scene. On the client side, View Frustum Culling is used to render only the visible polygons. The client side code includes path based prediction of motion for a smooth jerk-free walk through. The image based renderer is useful in cases of extremely low bandwidth and/or when the client does not possess a hardware graphics accelerator. Our current implementation uses Visible Space Models as the geometry-based streaming format and trifocal tensors as the image-based streaming format. We present results of the study conducted on a representative range of the relevant parameters.*

Chapter 1

Introduction

The last decade saw a tremendous growth in the popularity of 3D graphics applications on the personal computer. Graphics acceleration capabilities of PCs improved drastically in recent years. The size and quality of the graphics models used also improved steadily. Large models are bulky to store and require significant graphics capabilities to render. We've witnessed huge advances in improvement of detail of graphics models rendered by visualization and entertainment applications for the personal computer. Even a consumer level graphics board is capable of a throughput of tens of millions There are many instances when it is useful to store the large virtual environments on a server and provide access to clients as needed. This could be because the model can change often or because access to it needs to be controlled.

1.0.1 Streaming of 3D Virtual Environments

Inspite of these massive advances in graphics technology, the basic model of 3D content delivery has remained the same. Nearly all 3D applications are rendered using content stored on the local system. The client-server model of computing used elsewhere has not really been popular for 3D applications. The main reason for this is the fact that the improvement in the speed and latency of computer networks hasn't been as impressive as that in graphics technology. Hence, there are, no effective means for streaming virtual environments over the network. Even though the advent of broadband networks has made significant bandwidth available, interactive 3D rendering over the net is still far from optimal. The current standards of 3D over the web - VRML and Java3D do not properly support 3D over the web at interactive rates over low bandwidth/high latency connections. They also do not support optimization of data based upon client characteristics. There does not exist a single standard for 3D content delivery

over the web. This is mainly because transmission of large models is difficult even after including geometry compression. Latency for transmission is undesirable. The need is for a streaming system that automatically adjusts to the varying network conditions between the client and server and adapts the world to be transmitted based upon the bandwidth between itself and the client. Streaming of 3D data is more complicated than audio/video data since unlike videos, it is not possible to explicitly break up 3D data into discrete frames. With increase in computing power, the size of virtual environments is becoming extremely large. Hence transmission of the entire virtual environment is impractical. If a streaming approach is utilized, the virtual environment may also change with time without affecting the walkthrough experience of the viewer. Hence streaming allows dynamic virtual worlds to be displayed in virtual reality applications. Effective streaming should provide interactive experience for a variety of clients and connection speeds. The streaming mechanism, thus, should adapt to the client's capabilities and the varying network conditions of the connection between the server and the client. A streaming system will find wide applications in distributed virtual reality applications and tele-immersion.

1.0.2 Why Stream 3D Virtual Environments?

The benefits of 3D content streaming are numerous. Having a central repository for all the models and assets enables the content provider to consolidate all his content at the same place. The provider can then stream the content to the client based upon the capabilities and connection speed of the client. Any device ranging from a professional graphics workstation to a smartphone will be able to utilize the same assets and models.The rendering technique and detail levels will depend upon the type of client. The server must optimize the dataset based upon the type of client.Having a central repository is also useful in case of dynamic data. The data needs to be updated only at a single server. The client will then receive the most current data available. This is very useful in cases where data is received and needs to be transmitted in real-time like in meteorological data such as cloud and weather patterns. It is also useful in computer games which feature dynamic worlds. The new maps and assets need to be present only on the server. Whenever the client connects, it receives only the newest state of the persistent world.

Generally the graphical datasets are very large. Hence some techniques are needed to reduce the data to be streamed to the client. The usual technique is to send a visibility limited model of the virtual environment based upon the position of the viewer. The success of the streaming technique is very dependent upon the how effectively the visibility culler is able to remove invisible portions of the virtual environment.

1.1 Related Work

For interactive navigation in a virtual environment, the frame rate achieved must be as high as possible. However, the interactive frame rates must not be at the expense of image quality. Image quality is of paramount importance and should be as high as possible. There are quite a few methods to accelerate the rendering of virtual environments that are based upon both geometry based and image based rendering techniques.

1.1.1 Methods to Accelerate Local Rendering

Over the years, there have been different methods for accelerating local rendering. Most of these bandwidth saving methods rely on removing non-visible portions of the viewing environment. Hence the invisible regions of the VE are not rendered leading to massive speed boosts. These algorithms are collectively called visibility culling methods. ([1],[15], [40]) Another approach is to use Level of Detail (LOD) models of the objects in the scene or image based impostors.([2], [25],[32])

Accelerated rendering of complex scenes can also be achieved using pure Image Based Rendering. In IBR, the time required to render a scene is independent of the geometric complexity of the scene. Pure IBR such as Plenoptic Modelling by McMillan and Bishop [28], View Interpolation by Chen and Williams[5] ,Lightfield rendering (Levoy and Hanrahan) [22] and Lumigraph methods (Gortler and others) [14] in which the scene is represented as a collection of images without any geometric model have been successfully used. Shashua [3] utilizes Trifocal Tensors for Novel View Synthesis.This method provides a general warping function from the reference images to the novel synthesized images based the parameters of the viewing camera.

1.1.2 Remote Rendering Systems

There have been quite a few web based visualization systems over the past few years. VRML has been developed for remote interaction. It has been used successfully with compression of models in Djurcilov[9] and Earnshaw[10]. They develop an integrated visualization system, where the basic selection of data is done by the user. This is accomplished by integrating the visualization system with a database containing the additional information. The highlights of the system are its integration with a database front end that enables data to be queried and retrieved using a user-driven, web interface; the ability to specify both the location and time parameters of the data requested. The system supports query of realtime data as soon as its available.

However even after revisions of VRML which included geometry compression (Li et al.)[23], it has still not been used significantly. Many web based visualization approaches have used Java to generate VRML files dynamically including Trapp [37], based upon user input. In this system, a WWW server accepts data from a user in order to produce a visualization of the data. The user can control the visualization algorithm over the network using a JAVA-based user-interface. The visualization is transferred back to the client as a VRML world file and is pushed to the VRML Viewer. This system solves the problem of a platform independent access to visualization methods. However since these files are generated in real-time, they tend to be low in complexity with low detail textures and geometry.

Remote rendering algorithms have also been applied to volume visualization systems. [11] Elvins describes a system which accelerates volume visualization in collaborative environments. They develop a system that has the advantages of a turnkey volume visualization system but offers the performance of a massively parallel system rendering environments at interactive frame-rates.

Karonis,Papka etc [30] develop a system that utilizes distributed resources connected by a network Grid. They develop a prototype system based upon the Globus Toolkit and the Access Grid. These are based upon the MPI (Message Passing Interface) rendering system. Commercial products for remote visualization includes the The Silicon Graphics VisServer software which allows rendering of any OpenGL application on remote clients. It utilizes image compression to transmit individual frames(images) of the virtual environment. Holbrook, Singhal and Cheriton [17] develop a multicast protocol for transmission of terrain and environmental updates in an interactive simulation. They describe a system based upon Log based Receiver reliable Multicast communication(LBRM). They evaluate a LBRM optimizations that provide an efficient, scalable protocol for high performance simulation applications. Humphreys, Buck etc [19] describe a novel distributed graphics system that renders to a large tiled display. The system called WireGL, utilizes off-the-shelf PCs components. They present an algorithm to minimize the network traffic for scalable displays. Singhal and Cheriton [33] describe a method called projection aggregation, a technique for grouping identities by both their organization and location. Remote Hosts use projections to control which entities are represented locally and to what level of detail. They describe how to utilize projection aggregations to reduce network bandwidth and computational requirements. Funkhouser [12],[13] describes a system based upon client server architecture for multi-user virtual environments. In this system, multiple servers coordinate execution, managing communication, offload processing, and provide persistent storage for their clients. Cohen [7] presents a streaming technique for synthetic texture intensive 3D animation sequences. As the animation is played, the remainder of the data is streamed online seamlessly to the client. Macedonia [24] explores issues involved in designing

and developing network software architectures for large scale virtual environments.

1.1.3 Image based Rendering

Image based representations have been used more often than geometry based descriptions for remote rendering. Levoy [21] uses a high end graphics system to render high quality as well as low quality images. This acts as a server and transmits only the residue between the high quality and low quality images to the client. This approach assumes that the client already possesses a low quality model of the virtual environment. Cohen[6] and Mann[26] describe a system in which the client is able to generate several frames of information without transmission of residual images. The client receives visible portions of 3D data (without textures) and an initial image. The images are rendered into the Z-Buffer and back-projected to the available image to fetch the texture color. Yoon [38] utilize ray casting and epipolar constraints to exploit the spatial and temporal coherence between images and generate a residual image to be transmitted based upon user parameters. Biermann et al. [4] use Novel View Synthesis using trilinear tensors for rendering views on the client.

1.1.4 Geometry based Methods

Among geometry based approaches, Hesina [16] employs continuous Level of Detail algorithms on the server side. The server attempts to transmit to the client all objects within a circular region of interest around the current viewpoint. Schneider [31] describes a framework which adapts to the client characteristics including network bandwidth and the client's graphics capabilities. However this system concentrates on transmission of individual models rather than complete virtual environments. Teler[35] describes a remote rendering system utilizing path prediction and bandwidth based level of detail reduction. The exact representation to be sent is dependent upon an online optimization algorithm, which is based upon a benefit measure dependent upon the path taken by the viewer. The system transmits only parts of the scene and lower quality representations of the objects, based upon the user's viewing parameters and available connection bandwidth. This system allows for navigation even when bandwidth available is really low. [39] extends this method to include a pre-fetching algorithm to improve performance in case of very large geometric models which utilize out-of-core rendering. Pre-fetching is critical in cases where the rendering capabilities of the client and memory available are constraints as well. Martin [27] describes a framework - ARTE, an Adaptive Rendering and Transmission Environment that facilitates the delivery of 3D models in heterogeneous environments by monitoring the resources

available and by selecting appropriate transmission modalities.

1.1.5 Geometry Compression

To reduce the data transferred over the network, there are various geometry compression schemes that may be utilized. These methods effective compress the vertex data to very low levels.Deering [8], Taubin and Rossignac [34] and Touma [36] describe methods to reduce network bandwidth to as little as 10 bits per vertex (the coordinates and connectivity). Some method of compression on the vertices helps in reducing the bandwidth required for transmission dramatically. Another approach for reducing the amount of geometry transmitted is the use of progressive meshs [18]. Progressive meshes refine the object by continual transmission of data. A rough shape can be approximated by transmitting a very small amount of data which can then be improved in quality by further transmission of data.

Chapter 2

Requirements of The System

2.1 Basic Requirements

he RemoteVIS system architecture envisages a server connected to multiple clients. Each client has a viewer who is interactively navigating through a large graphics model. The server module and the client module provide these functions. They are joined together in a common philosophy of optimizing the navigation experience in the virtual environment at the client side. The client must render at the highest possible quality without the motion becoming jerky or slow. The server module must support the client in achieving this goal and stream the required data to it. We enumerate the requirements a remote rendering system must satisfy for a good user experience in this chapter.

A remote rendering system for virtual environments must adhere to certain basic requirements. The system must provide the best quality of visualization that can be afforded by the capabilities of the client and the available network bandwidth. The server also needs to adapt to a wide range of clients with varying graphical capabilities and network connection speeds. The system must dynamically and gracefully degrade in performance based upon client capabilities. The models streamed must improve and match the capability of the client over a long period of time. The client must not stall or halt due to connection latency. It must be able to predict and fetch data from the server in advance so that the client side motion is smooth. Lastly it must be able to support clients with limited resources - it must take into account memory and CPU constraints on the client side and stream data accordingly. The typical clients that may connect to the server may be the following:

- A system with good Graphics capability and very good network bandwidth. E.g.: An SGI Octane Graphics Workstation or a High End PC with a good graphics accelerator card such as GeForceFX

connected to the server on a low latency T1/T3 line.

- A system with good Graphics capability and poor network ban-width. E.g.: The above described systems connected to the server over a modem.
- A system with poor Graphics capability and good network bandwidth. E.g.: A PC with no add-in graphics card or a game console connected to the server over a broadband network.
- A system with poor Graphics capability and poor network bandwidth. E.g.: A PocketPC or a Smartphone (Sony Ericsson P800) receiving data from the server over a GPRS connection.

In interactive applications, maintaining a good frame rate is very important. Very often, a slightly lower quality model is more tolerable than lower frame rate. The system must dynamically and gracefully degrade the performance based on the current parameters. The models streamed must improve and match the capability of the client when the viewer is slow or stationary. The client must not freeze due to connection latency. This necessitates predicting the viewer motion and fetching data from the server in advance so that the client side motion is smooth. Lastly the server must be able to support clients with limited resources in terms of graphics capability, memory resources and the CPU power. The main considerations into designing a good remote rendering system would be the following:

2.1.1 Interactive Frame Rates

The data streamed to the client must match with the client capabilities. If the client possesses excellent graphics acceleration capabilities and very good network bandwidth, then highly detailed models and texture may be streamed to the client. Lower detailed models may be used if the client has inadequate hardware power to render the highest quality models, even if the network bandwidth between the server and client is high. Image based rendering or image based impostors may be utilized for extremely low-end clients. The idea is to provide the client with data that exactly matches its capabilities to provide the best user experience.

2.1.2 High Quality Rendering

The rendering quality achieved by the system must be the best possible for any given client and network bandwidth. When clients have lower available bandwidths, the models streamed initially may be of low quality but could be refined progressively with time. If the viewer is traversing the environment

at high speeds, full quality models need not be transmitted as the viewer won't be able to distinguish intricate detail in the models. When the viewer slows down in speed, low resolution models may be replaced with higher detailed ones.

2.1.3 Freeze-Free Rendering with Latency Immunity

To avoid pauses and jerks in motion, the client must request sufficient data to cover not only the current view but also the possible views in the immediate future. The viewers' path of motion must be predicted based on the past. Data may be requested from the server ahead of time based on the predicted viewer motion. The latency between the client and the server is of paramount importance. Even if the available bandwidth between the client and server is high, a poor latency can result in delays and freezes, reducing the interactive viewer experience. The solution to this problem is to predict for a higher period of time in the future, graceful degradation of initial quality of models and progressive improvement of the streamed models.

2.1.4 Independence of Server and Client Module

The system must allow clients to use different algorithms for rendering based on its declared capabilities and motion prediction. These parameters are fixed initially and may be changed by the client at anytime. The server must update the client information accordingly. After the initial negotiation, the server starts streaming data based upon these parameters. This is useful as the client can then use any algorithm internally for prediction. In case of heterogeneous clients such as a PDA and a graphics workstation, the prediction algorithms will be drastically different.

Chapter 3

Architecture and Design

It is mandatory for a remote rendering system to employ an algorithm to effectively limit the virtual environment to a much smaller subset of the original large model. Any algorithm such as those based on Visible Surface Determination or Level of Detail may be utilized. A completely complementary approach is one in which the system uses image based representations instead of geometry based approaches.

The ideal case of remote rendering assumes that the client is able to render any model transmitted to it by the server at interactive frame-rates - i.e. it is not bottlenecked by the graphical subsystem. But there may be clients with a large network bandwidth that are possess poor graphical capabilities. In such cases, the server must send data commensurate with the client's capabilities.

For a smooth and jerk-free motion, prediction of motion on the client side is essential. If the client requests data in advance, the viewer will not experience jerks and breaks in motion as a result of lag.

3.1 The Server Module : Objectives

The server module has multiple objectives of serving each client without delay and also serving the maximum number of clients possible at a time. The basic objectives of the server include the handling of the client requests as promptly as possible and optimizing the data streamed to the client based upon the parameters supplied to it by the client. The server must also keep track of data being sent to the client to avoid resending of already streamed data. The clients might have different policies internally and the server should be able to support these. The server module must be also be able to respond to different levels of service and dynamically change the level of service from one to another.

3.2 The Client Module : Objectives

The client module must optimize the rendering performance on the client-side based upon its capabilities. The clients may have varying characteristics with respect to available memory, graphics acceleration capabilities and speed of the host CPU. The network connection between the server module and the client may vary in terms of available bandwidth and overall network latency. These are the crucial factors in deciding the size and quality of the models to be transmitted to the client. The navigation speed of the viewer is another factor that may be used to decide the overall rendering quality.

3.3 Establishing the Client Parameters

The client and server modules agree on a common set of parameters to be used such as the data format and algorithm to be utilized (e.g. Image based rendering or Geometry based rendering). The client then indicates its network speeds, rendering capabilities to the server by the client. The server then starts streaming data in the appropriate format optimized for the particular client. As the viewer moves through the virtual environment, the client requests the server for more data by supplying the current viewing position of the user in the virtual environment. The server must also take into consideration, how much load it is currently handling. In case of too much load, the server may reduce the detail levels appropriately. It may also improve the detail levels when the load drops again. The client parameters can be updated at arbitrary intervals by the client provided the server acknowledges the change.

3.4 Design Requirements of the System

The system is a server client architecture with some of the modules resident on the server and the rest on the client. The server and client are connected to each other by their respective network modules. The server stores all the scene information on its local storage and streams the data to the client upon request. The important modules on the server and client side are described below. An effective remote rendering system will be tightly knit together to allow maximum efficiency while rendering and transmitting data, but must also be flexible enough to allow different algorithms and paradigms to be utilized for rendering without changing the structure of the system. Keeping these ideas in mind, the design of the system is formulated as a combination of different modules linked under the common thread of providing the best user experience possible. The block diagram of the system is outlined in 3.4

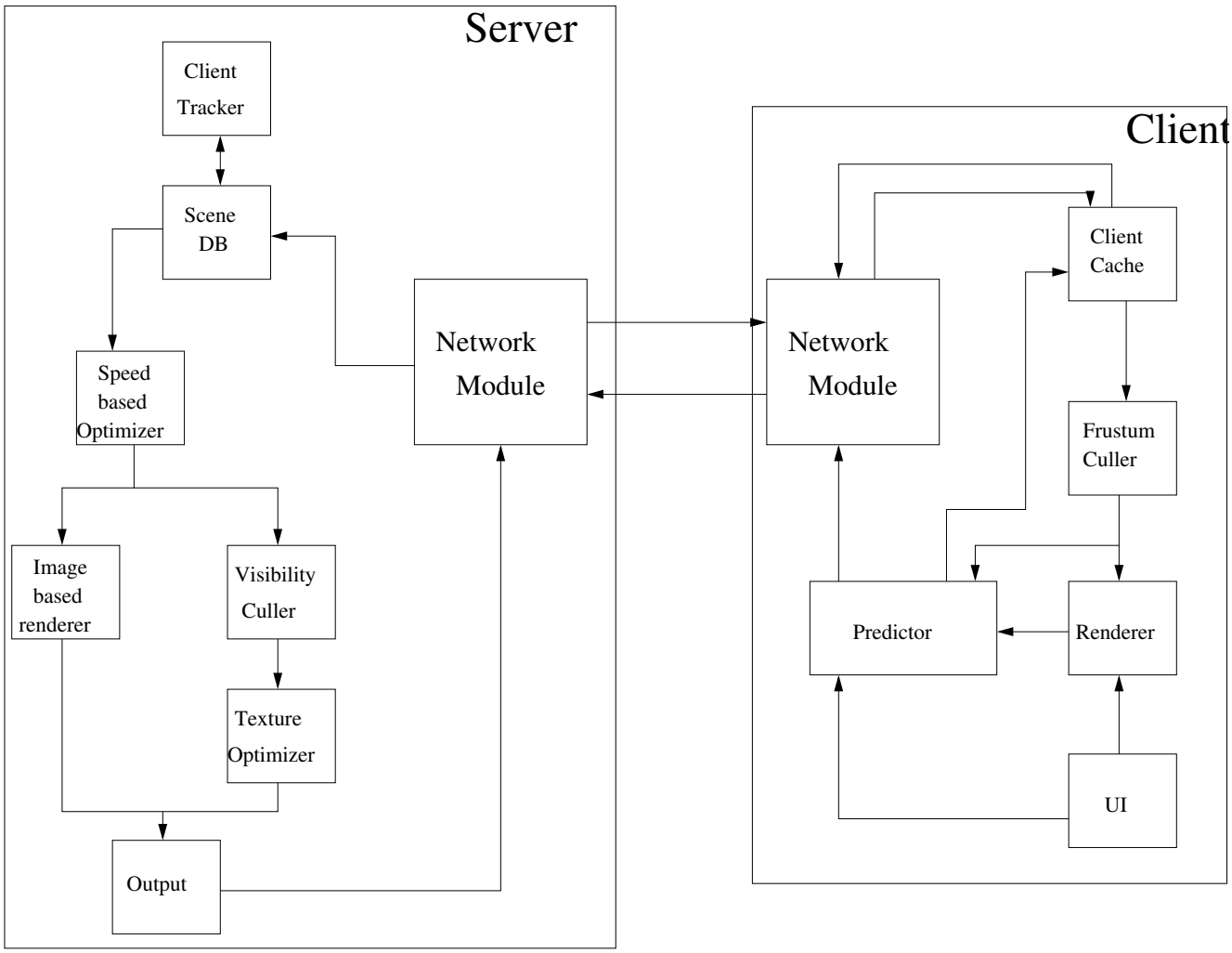

Figure 3.1. Architecture of the Remote Rendering System

3.4.1 The Network Module

The network module on the server and client-side are the most important components of the system. They are responsible for all the communication that occurs in between the server and client. The server modules and client modules are linked to each other by their respective network modules. The network modules decide the connection parameters of the client based upon user input and create a seamless environment for transmission of large volumes of data. The initial negotiation between network modules also decides the algorithms to be utilized by the systems (Geometry based or Image Based) for the entire duration of the walkthrough. The client and connection parameters may be changed at any time during the walkthrough based upon mutual agreement between the server and the client.

3.4.2 The Scene Database

The scene database stores all the models present in the scene along with all its assets including texture maps, light maps, bump maps etc. The scene database must be stored in a convenient format which allows for quick retrieval and loading on the server side. Ideally the format must also support some form of compression so that it allows us to save storage space on the server. This is useful since the data sets may be extremely large (in gigabytes). But the most important feature is to allow retrieving the data quickly and efficiently when required. The decompression of the model should not take much time.

3.4.3 Client Tracker

The client tracker module keeps track of the position of the client in the virtual environment. This is essential to allow the server to maintain the a record of the data streamed to the client so that data that has already been streamed need not be retransmitted again. Its absolutely essential to minimize the amount of data transferred to the client. So the scene objects transferred must be logged and only be retransmitted if a higher detail mesh of the same model is requested.

3.4.4 Speed Based Optimizer

The speed based optimizer selects the correct level of detail mesh from the Scene Database depending upon the speed at which the viewer is moving in the virtual environment. The Speed based optimizer selects the correct mesh detail and then sends this mesh to the IBR/Visibility Culler module. The Speed based optimizer also provides the correct image size and formats to the Image Based Renderer.

3.4.5 Visibility Culler

The visibility culling module is utilized only when the system utilizes geometry based approaches for rendering on the client-side. In that case, the Visibility Culler module utilizes one of many visibility culling methods to decide which parts of the virtual environment need to be streamed to the client. The other portions of the VE are ignored straightaway and not transmitted to the client. The algorithm for selecting the visible portions of the VE may be different for every individual case, based upon the type of model and also the rendering parameters. An efficient visibility culler will remove most of the non-visible portions of the virtual environment and potentially reduce the amount of data to be transmitted by a very large amount. The main point to be noted is that there must be some margin for error on the client side, so that there need not be retransmission of data due to local motion on the client. Motion in the absolute vicinity of the viewpoint must be handled properly.

3.4.6 Texture Optimizer

The texture optimizer is a simplistic module that scales and optimizes textures in the model based upon the connection parameters and the speed of motion of the client. This module must not only optimize textures but all other assets that are required to be transmitted for proper visual fidelity on the client side. The detail levels of the assets must be properly optimized based upon the client capabilities as well. If the client is low on capabilities, then there is no point of sending high detail assets.

3.4.7 Image Based Renderer

The Image Based Rendering module of the system is essential for very low-end or low-bandwidth clients. Low-end clients possessing no hardware acceleration must utilize some form of image based rendering on the client to display the virtual environment. The image based renderer on the server-side generates the seed views and the set of correspondences. The actual views must be generated on the client-side. The size and quality of the seed views will again depend upon the connection parameters of the client.

3.4.8 Client Cache

The client cache must store the data that has been received from the server during the course of a walkthrough. Caching ensures that the server need not send the data it has once sent on account of

viewer retracing a particular path. This saves bandwidth which can be utilized in improving the quality of output. Ideally a cache infinitely big cache to store all incoming data is desired. However due to limited client resources, only a small finite cache of data can be stored on the client side. Hence an algorithm for cache replacement is necessary. Another point to be noted is that we must never replace a high detail model in the cache with a lower detail model. The vice-versa is permissible though.

3.4.9 Frustum Culler

The frustum culler on the client side is a very simple module whose job is to cull out the non visible parts of the virtual environment during the walkthrough. The algorithm utilized is dependent upon the Visibility Culling algorithm utilized on the client side. This module is active only when the Geometry Based Renderer is in use. The IBR system doesn't require this module.

3.4.10 Predictor

The system must utilize path prediction for smooth motion. The path prediction algorithm must request for data along the predicted path of the viewer in the virtual environment based upon the current position and speed of the viewer. This is necessary to avoid the ill effects of latency and also to provide a smooth jerk-free experience to the user. The data requested by the predictor module will depend upon the expected path and also the latency between the server and the client. The penalty for failed prediction due to random user motion must also be handled properly.

3.4.11 Renderer

The rendering system utilized at the client is totally independent of the other modules. The system may utilize any library/ rendering system based upon the requirements and also the hardware specifications of the client. A client with good hardware acceleration capabilities may utilize a library like OpenGL. Clients with a high end CPU but poor graphics acceleration may instead utilize a software renderer. It is completely dependent upon the specific client system.

3.4.12 User Interface

The user interface is tied to the choice of the rendering system. The UI must provide controls for changing client parameters, the rendering method and also allow for rapid processing of the user input

while the user is navigating in the virtual world. The the user interface will also depend upon the operating system used.

3.5 Comments on the Design

The splitting of the entire system into modules allows us to have maximum flexibility. We may easily replace a module with a better or advanced version without breaking the entire system. Also since the modules are independent of each other, it allows a particular module to stay completely ignorant of certain other modules which do not affect it directly. For example, the Prediction module can utilize any algorithm for prediction without the server actually having to change any of its parameters to adapt to it. This flexibility is extremely useful when we need to test different algorithms for visibility detection, prediction etc without affecting the complete system.

3.6 Image and Geometry based rendering techniques

Both Geometry based and Image based rendering techniques have their advantages and disadvantages. Geometry based methods benefit a lot from hardware acceleration. Even with modest acceleration, a geometry based method will outperform an image based rendering system in case of moderately complex scenes. GBR systems are affected less by changes in latency and local motion in the vicinity of the surroundings. The overall visual fidelity of the scene is generally better than IBR methods. The Geometry based methods fail to provide adequate performance in cases where the system doesn't possess hardware acceleration or in cases where the available network bandwidth is very low. The time taken to render a frame in case of IBR is independent of scene complexity and hence a constant frame-rate is maintained throughout the walkthrough. The amount of data transferred in case of IBR is generally lower than GBR based methods of the same fidelity. It depends upon the situation

3.7 Comparison of Various Visibility Culling Techniques for Geometry based Rendering

There are various methods of visibility culling that could be employed in a remote rendering system. The usual techniques are BSP Trees, Octrees, Portal Rendering and Visible Space Models. Out of these, Visible Space Models offers the best in terms of flexibility and rendering speed. One of the primary concerns is the dynamically create the best possible view based partial model of the original model. It

should also allow for local motion of the viewer in the immediate vicinity. The method should also allow easy incorporation of level of detail methods into it. It should also allow the model to change dynamically without much of a performance loss. BSP Trees cannot be used as they need preprocessing to create an optimal tree. Hence the choice remains between Octrees, Portal Rendering and Visible Space Models. Portal Rendering as well requires model preprocessing and cannot be applied to a real-time situation like this one. Out of VSMs and Octrees, VSMs definitely offer better performance in a wide variety of complex scenes. Octrees excel in subdivision of terrain data but in the general case, they do not provide much of a speed up. Hence the system utilizes a combination of object based VSMs and varying Levels of Detail to accelerate rendering.

3.8 Image Based Rendering

The Image Based Rendering system is utilized for clients lacking hardware geometry acceleration. In such cases, Image Based Rendering offers a faster alternative to geometry based methods. The system utilizes the IBR based methods for very low-end clients with low client capabilities and also for clients which possess low bandwidth. The major advantage of Image Based Rendering is the fact that the time taken to render a scene on the client-side is independent of the complexity of the scene geometry. Once the pre-processing is performed, rendering the scene is a linear-time operation. Hence the client can have a constant frame-rate throughout the walkthrough.

The IBR algorithm utilized by our system is based upon Novel View Synthesis using Cascaded Trilinear Tensors [3]. This method provides a general warping function from the reference images to the novel synthesized images based the parameters of the viewing camera. The view synthesis approach is based on fact that three views of a scene satisfy certain matching constraints represented by a tensor. Thus given two views in correspondence and a tensor, the corresponding third view can be generated uniquely by means of the warping function. The system utilizes two reference views to compute the dense correspondence and for recovering the tensor on the server side. On the client-side, given the tensor and the camera parameters, the corresponding tensor is calculated and this tensor is used for rendering the novel view. Generally the correspondences are calculated using the complete geometry of the model. However in our case of a virtual environment, calculating the correspondences at the server is much easier as we know the correspondences at object level. The seed views are chosen based upon the position of the viewer in the virtual environment.

Image Based Rendering using Trifocal Tensors has its own advantages and disadvantages when com-

pared to geometry streaming. The biggest advantage is the simplicity in transmission of an image based representation. We only need to send two seed views, the tensor and camera parameters to generate a novel view. These images can easily be JPEG compressed for efficient transmission. The major disadvantage of this method is its inability to handle occlusions and sudden changes in viewing direction in a proper manner. One must note that this system will have lower performance in terms of frame rate than a system utilizing hardware geometry acceleration.

3.8.1 Trifocal Tensor based IBR algorithm

The IBR system uses novel view synthesis for rendering based upon the following algorithm.
At the Server, given the viewpoint P the following procedure is adopted.

- Generate source images about the point X about all the six axes. We will have three pairs of images $P + \delta(x)$, $P - \delta(x)$ where $\delta(x)$ is a deviation from the viewpoint about X axis. Similar source images are generated for Y and Z axes.
- For each pair of images, calculate the dense correspondence and recover the tensor $< 1, 2, 2 >$. Transmit these three sets of images and tensors to the client.
- At the client, when the viewer moves to a viewpoint V, calculate the new camera parameters (D, t). Find the pair which has a reference image closest to the position V and use its received tensor from the server to calculate the new tensor $< 1, 2, \psi >$
- Render the novel view using the two corresponding source images.

3.9 Visibility Culling Methods

For reducing the amount of data to be streamed, it is necessary to apply an algorithm that effectively culls out parts of the virtual environment that is invisible to the user. An effective visibility culling algorithm for remote rendering is one which culls out most of the invisible geometry but must also be able to handle local movement without choppiness or visual anomalies. A Level of Detail algorithm may also be employed to reduce the detail of objects that are far away from the viewer. In such a case, distant objects may be replaced by low polygon representations or image based impostors.

3.9.1 Visible Space Models (VSM)

Our system employs Visible Space Models due to P.J.Narayanan [29]for visibility detection. A visibility limited, partial model is taken as the basic unit of the virtual environment representation. A VSM has the following features.

- The model has an origin in the 3D global virtual space, a preferred direction of orientation, and a field of view.
- It contains a description of the environment visible from its origin in the given direction, limited by the field of view.

Formally, we define a Visible Space Model of a virtual environment as

$$VSM(E, \vec{C}, \vec{D}, \alpha) = \bigcup_{O \in V(E, \vec{C}, \vec{D}, \alpha)} R(O)$$

where E is the virtual environment, $\vec{C}$ is the VSM origin, $\vec{D}$ is the orientation and α the field of view. R is the representation of the object O in the coordinate frame of the VSM, which could be in terms of surfaces or volumes. The fundamental geometric primitives are described either in the VSM's coordinate frame or as a hierarchy of transformations which when composed, convert them to the VSM's reference frame. R itself is visibility limited and does not need to contain a part or subpart of O that is not visible from $\vec{C}$. The visibility constraints may be imposed per primitive or per object represented using a collection of surfaces. V(E,$\vec{C}$,$\vec{D}$,α) is a visibility function that gives the set of objects or primitives of E visible from $\vec{C}$ in the direction $\vec{D}$ subject to the field of view.

One may think of this representation as describing the portions of the virtual environment visible to a camera placed at the origin and pointing in the preferred direction with the given field of view. A VSM based representation conventional representation completely eliminates portions of the virtual environment that are not visible from a given viewpoint/direction. The visibility function eliminates not only the objects lying outside the viewing cone (or pyramid/frustum) but also the objects inside the viewing cone that are occluded by closer objects. The representational complexity is therefore constant in any direction.

3.9.2 Object based VSMs

According to the above definition, the visible portions of the virtual environment may be determined either using a polygon based approach in which only the visible polygons of the virtual environment

are stored or using an object based approach in which the visible objects are stored. The second representation may still contain some redundant information as partially obscured objects will be taken to be completely visible. However this approach is beneficial as the object hierarchy of the scene is maintained. Once an object has been streamed to the client, then it need not be streamed again. In a polygon based representation, it is difficult to implement client side caching. A view frustum culler exists on the client side so that only the requisite geometry is rendered.

3.9.3 Rendering Using VSMs

While rendering using VSMs, normally (polygon or depth map based) we require atleast three overlapping VSMs to achieve hole free rendering. However if we use object based VSMs, a single VSM should suffice in most cases, since the traditional cause of holes - missing polygons in objects is no longer there. This is another advantage of using object based VSMs. Otherwise, we might have to transmit atleast three VSMs to achieve hole-freeness. This would translate to a lot more polygonal data being transmitted. When using multiple VSMs for hole-free rendering, it is assumed that the client possesses hardware that is capable of real-time stencil buffering. With a single object based VSM, this is no longer one of the requirements as stencil buffering is not required for hole filling. Hence object based VSM rendering offers an elegant method for visibility culling on the server side to reduce the amount of data transferred to the client.

3.9.4 Level of Detail

Another method of reduction in model detail to be transmitted is based upon Level of Detail(LOD) algorithms. Objects that are farther away from the viewer are of lower detail than objects that are near the viewer. This method is easy to implement but does not take into account the occlusions and hidden surfaces in the model. Hence invisible areas of the model are streamed as well, leading to wasted bandwidth. This problem will be acute in cases of multiple occlusions of objects or in cases where a highly detailed object is occluded by a low detail object. The VSM representation is superior to using LOD only representations. VSM representations cull out invisible geometry automatically. Hence no redundant data is sent to the client. In addition, when using object based VSM representations, we are free to choose the level of detail of the visible objects. Hence LOD can be integrated seamlessly into a system utilizing object based VSM representations.

Chapter 4

Implentation Details

The geometry based system utilizes a Visible Space Model representation for culling out invisible and occluded geometry. A VSM based representation allows us to only send the parts of the virtual environment that are actually visible to the viewer at the client's end. So wasteful transmission of hidden geometry is avoided. In addition, The server maintains multiple Level of Detail meshes of the original model and selects the most optimum mesh to be transmitted to the client based upon the client's capabilities. The image based renderer utilizes a Trifocal Tensor based system to render novel views.

The system employs client side prediction to avoid delays and lockups due to network lag. Prediction helps in maintaining a smooth and jitter-free user experience. An innovative new technique adopted by our system is that of changing the detail level based upon the speed with which the viewer is navigating through the world. At high speeds the viewer will be less sensitive to high detail in the virtual environment.

The RemoteVIS system keeps the following factors in consideration and uses either a visibility culling algorithm or an image based rendering algorithm to reduce the amount of data streamed to the client by a large amount.

- The client uses an intelligent approach to request models based upon its position. It requests for models that cover not only its current position but also those that cover its complete local vicinity.
- The client uses a prediction scheme for requesting data based upon the latency of the connection between the server and client. It compensates for higher latency by increasing the time it predicts in the future.
- The system supports different classes of network bandwidth levels. These include typical broad-

band, ISDN and modem data rates. It must also support clients of different capabilities and possess different features and optimizations for high-end, mid-range an low-end clients. Finally there are different levels of navigation speeds of the client. Our system supports three different levels of navigation speeds.

- The server uses an object based Visible Space Model (VSM) representation for geometry based representation. This scheme provides maximum flexibility as well as a very good culling mechanism for removing invisible data.
- The system supports image based rendering using a cascaded trilinear tensor based method for low end clients.
- The geometry based renderer may utilize either a hardware accelerated rendering system or a software rendering system, based upon the client capabilities.
- The system dynamically scales textures and other assets of the model based representation based on the capabilities of the client. All textures and assets must be transferred in compressed format takes the least amount of time for transmitting data.
- The system uses a simple compression scheme for the model data to reduce the network traffic.

4.1 Client Side Prediction of Motion

For a smooth and jerkless motion in the virtual environment and a seamless, hole-free display, prediction of motion is extremely important. A couple of things are crucial for this. Models available at the client side must cover the current user location and its immediate neighborhood. The system must also be able to tell where the viewer is going to be in the future.

4.1.1 Prediction in Virtual Environments

Prediction of motion implies guessing the position of the viewer in the future and requesting the server data, that will be used by the renderer in the future. To predict the motion in a constrained environment such as this one, it is assumed that the acceleration of the client is constant in all directions. Then the motion of the viewer may be decomposed into components that are either rotational motion about a point or translational motion about one of the three coordinate axes. Using these assumptions and given the

position of the viewer at two earlier instances of time, it is trivial to predict the motion of the viewer using standard equations of kinematics.

4.1.2 Fighting Latency using Prediction

Once the position of the viewer is predicted, the client requests data from the server for the future instances of time. The client must possess model data for the next n seconds. This amount of time n is dependent upon the latency between the server and client. If t is the round-trip time between the server and the client, the client must predict correctly for atleast $2t$ amount of time in the future so that latency would not hurt the performance of the renderer on the client side. This would ensure the client would have enough data to avoid jerks and jitters even if the data transferred from the server takes more than t seconds to arrive at the client.

4.1.3 Dead Reckoning

Dead Reckoning is a typical technique adopted for client side prediction in most client-server based rendering systems. This method makes a trade-off by simulating the viewer movement at both the client and server side. When a packet does not arrive on time, the movement is predicted for a fixed time frame. Once this buffer zone has elapsed, the movement freezes until a packet of data arrives as predicting further might lead to a higher gap between the views of the client and server. Additionally a client will not send any data to the server if it believes that the server can correctly predict its future position in the virtual environment.

4.1.4 Pure Client Side Prediction

Our system utilizes only client side prediction. Though this transmits marginally more data than a system that utilizes dead reckoning, using only client side prediction allows the system to be more flexible. The client may then utilize any algorithm it wishes for prediction. The prediction becomes totally independent of the server. In such a case, the client may optimize its prediction algorithm accurately to match its connection parameters. They client can then request data whenever required in an optimal manner to provide the best navigation experience.

4.1.5 Handling Local Motion

For client motion in the vicinity of a transmitted VSM, no more data must be requested from the server. A single transmitted block of data must be able to handle local motion around a particular VSM viewpoint. For this additional supporting VSMs on the view plane around the reference VSM are also transmitted by the server. The separation of the supporting VSMs from the reference VSM will be dependent upon the extents and size of the original model. Since there is considerable overlap between the views of the supporting and reference VSMs, the additional data transmitted is generally small.

4.1.6 Classification of Clients

The system distinguishes between different clients based upon their bandwidth by classifying them into distinct classes. These classes are based upon the typical rates of bandwidth offered by different types of connections.

- Broadband users with data rates between 100kB/s and 1MB/s are in one category.
- ISDN users with data rates ranging from 10kB/s to 100kB/s are in the second category.
- Modem users having typical data rates between 3kB/s-10kB/s fall in another category. Users of Smartphones and PDAs also fall in this category.
- Finally there is a category for systems on the LAN/high speed networks which are assumed to have unlimited bandwidth.

This divisions into classes is arbitrary and may be adapted according to the need. The client leaves it to the viewer to choose the bandwidth class it wishes to use.

4.2 Texture Optimizer

In addition to Vertex and Polygon Data, the model textures present a significant challenge in transmission. All textures must be handled in real-time to send the optimal quality textures to the client based upon its connection profile. The client initially transmits the supported texture formats and the maximum resolution textures it can display to the server. Henceforth, only supported formats are streamed to the client.

4.2.1 Optimizing Textures: Trade offs

There are two significant quantities which can be dynamically altered with regard to textures - their size and quality. Textures are generally square or rectangular with dimensions that are powers of two. The data to be transmitted is directly proportional to the size of the texture. Hence size of the texture is to be based upon the connection parameters of the client. If the texture is JPEG compressed, then the overall quality of the texture can also be controlled based upon the available bandwidth . We must achieve a proper balance of size and overall clarity of the texture since a smaller texture with intricate detail is in most cases better than a larger blurred texture. In addition, since most graphics cards perform atleast bilinear texture filtering, we must prefer quality to size of textures.

4.2.2 JPEG Compression of Textures

The current texture optimizing system directly scales the textures based upon the connection class of the client. Each reduction in connection class reduces the texture size by half. The quality of the JPEG compression is based upon the number of textures to be streamed to the client. The server reduces the quality of textures slightly when sending a large number of textures. The reduction in bandwidth comes from reduction in size of the textures. There is a small limitation with JPEG compression for storage of textures - if the texture has an alpha channel, it cannot be stored as a JPEG. In such cases, RLE compressed Targas are used for transmission. Targa (TGA) is the preferred format for storing textures because it is simple to read and supports both compression and alpha channels. As is the case of models, the texture quality improves with time when the user stands still at a particular position due to progressive refinement.

4.3 Speed Based Optimization

The system makes the assumption that the human vision is less sensitive to intricate detail when the surroundings are being traversed at a high speed. As the speed of motion reduces, the viewer tends to observe more of the detail around him. The system uses this assumption to reduce the mesh and texture detail by when the viewer navigates through the world at a speed higher than a specified threshold. Similarly, when the viewer is stationary at a particular point, the system progressively streams a higher detail model to improve the perceived visual quality.

4.3.1 Speed Thresholds

The threshold for speed will be dependent upon the model size and extent. It is a parameter that can be chosen by the user at runtime. This optimization also allows bandwidth which would otherwise be wasted to be utilized. If the viewer is stationary or if speed of motion is not taken into consideration, no data would be transmitted by the server. However with speed under consideration, the server may stream a higher detail version of the meshes and textures to the client utilizing the bandwidth more effectively.

4.3.2 Measuring Speeds

The limits for high speed and low speed must be set based upon the extents of the model. Any speed which is above one-fiftieth the total extent of the model per second is considered to be high. Normal detail meshes are streamed when the speed is between one-fiftieth and one-hundredth the extent of the model per second. When the viewer is travelling at a speed lower than that, it is taken to be low speed and models that are of higher detail are streamed. These numbers are arbitrary and any other limit may be imposed if desired. As the speed reduces, the model quality improves and as the speed increases, the model quality degrades gracefully. For low bandwidth connections, when the viewer is stationary the system tries to stream a high quality model. This is necessary to improve the rendering quality on low bandwidth clients as at typical rates, the rendering quality will be not be very high.

4.3.3 Cell/Portal Optimization

This algorithm for speed based optimization can be improved if the model in question possesses clear divisions into portals and cells. In that case the speed can be measured in the number of cells moved per second. This is a far better measure for speed as most cells in the model will be of equal geometric complexity. Hence a better classification of the speeds may be achieved.

4.4 Client Tracking and Caching

If the client moves past a point, the corresponding vertex, polygon and texture data is transmitted to the client. If after a certain period of time, the viewer returns to the same position, the data once transmitted to the client need not be transmitted again if it can be cached on the client side. Hence a need for a client side caching mechanism of data arises. Since the system utilizes object based VSM representations, once the mesh and textures for an object in the scene have been transmitted, it is unnecessary to transmit the

same or lower detail representation of the same. The object will be present in the client side cache from which it may be reloaded again. However if the server intends to send a higher detail model, client can replace the model currently in its cache with the higher detail model, once it has been received. Since a lower detail model is never streamed, a model in the cache can never be replaced by a lower detail version of the same. Hence the server maintains a list of objects that have been already transmitted to the client. It also stores what level of detail of the object the client possesses so that it doesn't accidentally send a lower detail model to the client. The system maintains this information on the server side as a hash table in the Client Tracker module. Whenever a new object or a higher detail version of the object is transmitted, this hash table must be updated to reflect the change. When the client expunges an object in its cache, it informs the Client Tracker module accordingly.

4.4.1 Cache Replacement

In the ideal case, we may assume that the client has infinite cache capacity. However, if the virtual environment is extremely large, it is incorrect to assume that the client will possess a buffer large enough to store all objects in its cache. In such a case, the client must discard some of the models that are present in the cache periodically whenever the cache gets filled. We must find an algorithm to periodically replace the objects in the cache based upon some fixed methods. This problem is very similar to the page replacement algorithm found in operating systems. However only utilizing an algorithm such as Least Recently Used (LRU) on the objects in the client cache is not recommended. We must also take into consideration the distance of the object to be expunged from the current viewpoint. Another factor is the number of times the object has been visible in the client's viewport. This calls for a frequency to be attached to each object based upon the number of times it is visible. All these factors are incorporated into the algorithm currently used for object replacement.

4.4.2 Modified LRU algorithm

The system utilizes an algorithm that applies the following rules empirically on the object pool to expunge geometry. All objects that are within a fixed distance from the current viewpoint are never deleted from the buffer. The threshold for this may be set based upon the memory resources available to the client and by default it is a sphere of radius equal to one tenth of the total extent of the virtual environment. This is done so that objects in the immediate vicinity of the viewpoint, which may potentially become visible soon if the viewer rotates or moves are never expunged. The objects with the

highest frequency of display should also not be expunged. This limit may be set based upon the size of the cache. An LRU algorithm selects the objects which appeared the least recently in the viewport as the objects to be expunged from the cache. Each object is timestamped whenever it is rendered on the client. The client notifies the server that an object was removed from its cache so that the server may update its Client Tracker module appropriately.

4.5 Compression of Transmitted Data

The data transmitted between the client and the server must be compressed optimally to achieve best performance from the system. The data is compressed using a scheme that utilizes GZipping on the vertex and polygon data. The system utilizes the zlib library for this purpose. The textures need not be compressed as they are already in JPEG format which achieves maximal compression of images. There are various schemes for vertex data compression based on entropy encoding. The vertex data compression algorithm by Lee[20] achieves a ratio of 6.7 bits/vertex. However such encoding schemes are computationally expensive and unsuitable for real-time computation.

Chapter 5

Results

The prototype system developed is based Windows 2000 and utilizes the OpenGL libraries for rendering and Winsock 2 libraries for network interaction.The client machine used was an Intel Pentium III with 256MB of memory and a low-end RIVA TNT2 M64 graphics card with 32 megabytes of video memory. The server was an Intel Pentium 4 based machine with 512MB of memory and no graphics acceleration. A second high end client was used to test as well. This consisted of an Athlon XP 2 GHz processor with 512MB of main memory coupled with a GeForceFX graphics board having 256MB of video memory. The machines were connected on an 100BaseT LAN. The lower bandwidth conditions were simulated over this network.

The model (Fig 5.1) used consisted of 163557 polygons and 84339 vertices. The total uncompressed size of the model was around 7 megabytes. The textures in uncompressed form were around 2.25 megabytes. After JPEG compression optimizing quality, the total size of the textures was around 200 kB. The model was converted into two lower level of detail models which comprised of 82884 polygons with 42211 vertices and 18337 polygons with 11453 vertices. The corresponding texture data was around 100 kB and 50 kB respectively. A fixed walkthrough path was created for the quantitative results and this same path was used for measuring data and frame rates for homogeneity.

5.0.1 Performance over varying Data Rates

The amount of data transferred during a walkthrough which lasts for approximately forty seconds is shown in Fig. 5.2. Note that the periods of inactivity last only in the case where an unlimited bandwidth connection is available, when the server is always streaming the maximum detail models. In other cases, there is progressive refinement of the models and the bandwidth is utilized even when the viewer is idle.

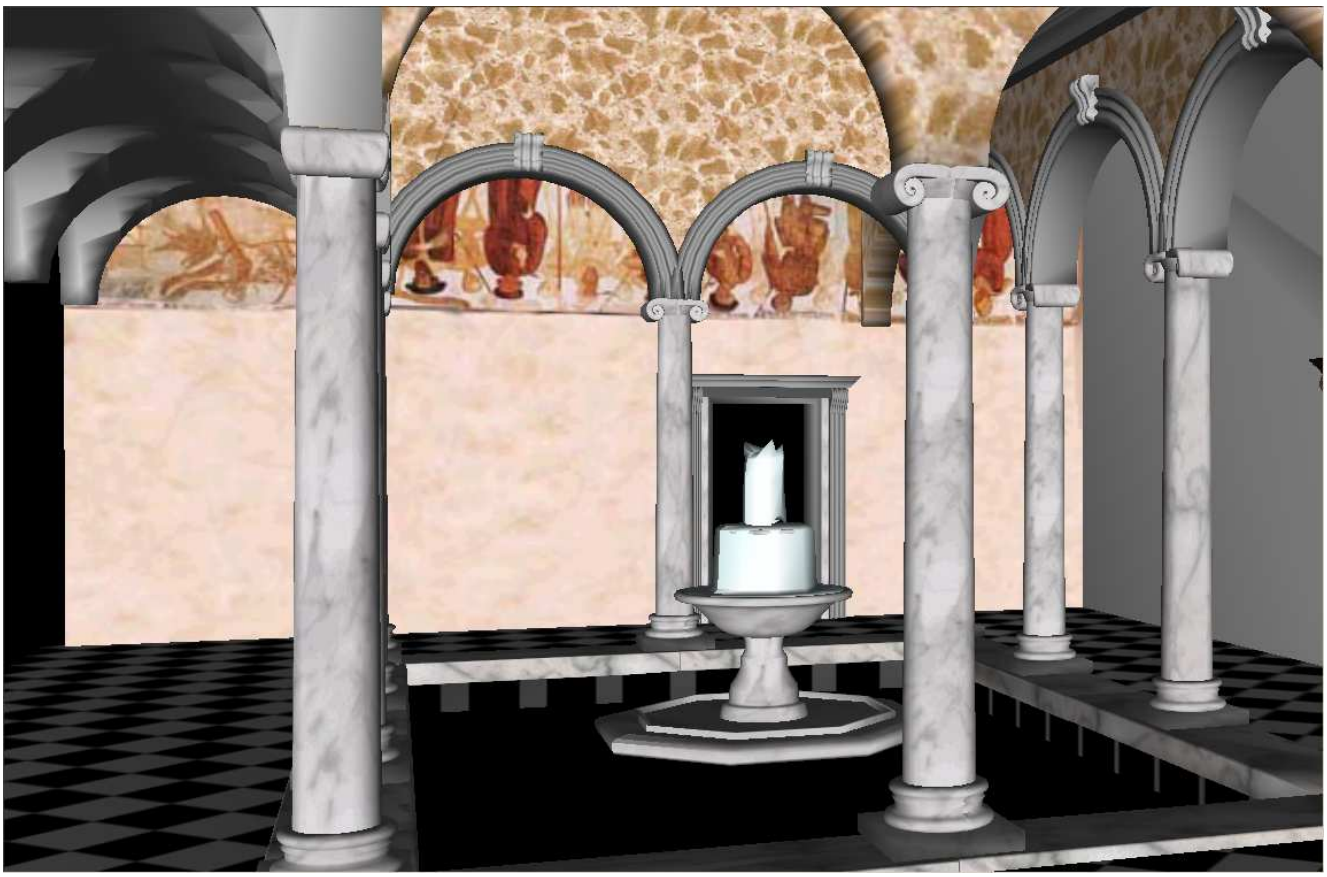

Figure 5.1. Screenshot of the Client

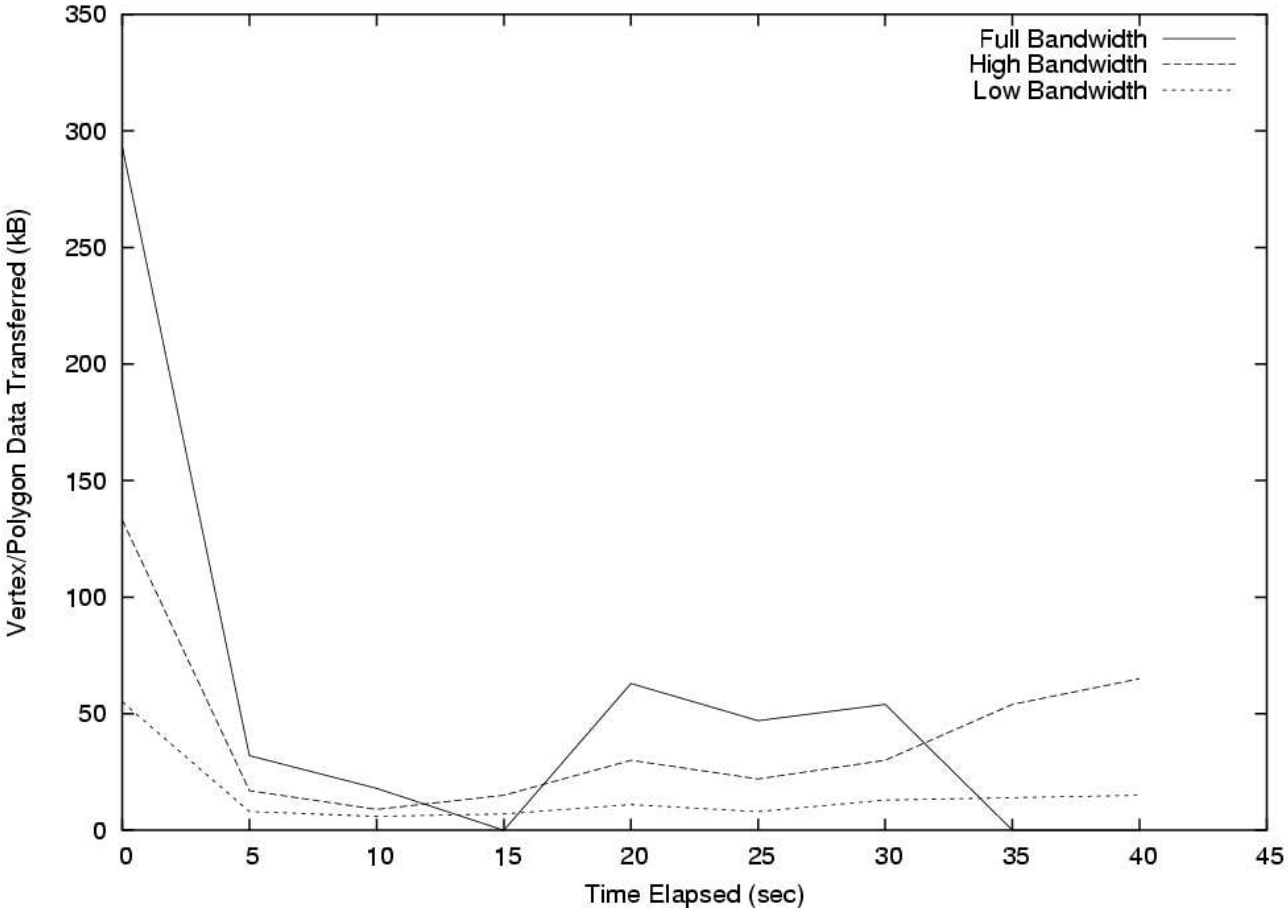

Figure 5.2. Vertex/Polygon Data Transfer Graph (PIII/TNT2)

That explains the flat graph of the full bandwidth client. After the 25th second, all textures have already been transmitted and hence the graph completely flattens out. This may not be the case if a memory constrained client having a small cache for storing textures. In the lower bandwidth clients, textures of higher quality are transmitted every time from that point of time.

In case of the Image Based Renderer, the data transferred consists of seed views and correspondences. The views are compressed using JPEG compression. Since the size of the images is the same each time, the amount of data transferred is nearly constant for most of the walkthrough. Hence this method is very suitable for connections that are of low bandwidth and also low latency. This is because unlike a geometry based renderer, a system based upon tensors is not very robust in handling occlusions and rapid transitions in viewer position.

The amount of prediction to be performed is directly dependent upon the latency of the system. The latency and speed of a client are independent of each other. A client may have a high speed connection but may also have a high latency. In such cases, the a large amount of data will arrive at the client after substantial intervals.

5.0.2 Walkthrough Performance and Frame Rates

Once data has been received by the client, the performance of walkthrough is solely dependent on the graphics capabilities of the client. The client system uses simple View Frustum Culling to reduce the amount of data to be rendered. The average frame rates are shown in Fig. 5.3. For comparison, the frame rates achieved by normal brute force rendering methods (View Frustum Culling only) are outlined in Fig. 5.5. It is interesting to note that on the system with a TNT graphics card, the reduction in primitive count does not affect the frame rate. The low fill rate and lack of a hardware Transform and Lighting unit is hampering performance. Only at the lowest detail, the frame-rate gets a significant boost due to a significantly lower polygon count. This is primarily because of the inherent visibility limiting nature of VSMs. Most invisible geometry is culled out ahead of time. Hence the system is able to provide a good user experience on low-end clients with average graphics capabilities. The optimal detail level for this client is the lowest detail mesh due to low graphics capabilities. The GeForceFX based system performs much better as can be witnessed from Fig 5.4. The frame rates demonstrate that the system is capable of handling much higher detailed models and textures.

The Image Based Rendering system performs much better than a system that utilizes only frustum culling, offering quality comparable to the highest level of detail of the model. The frame-rate is almost

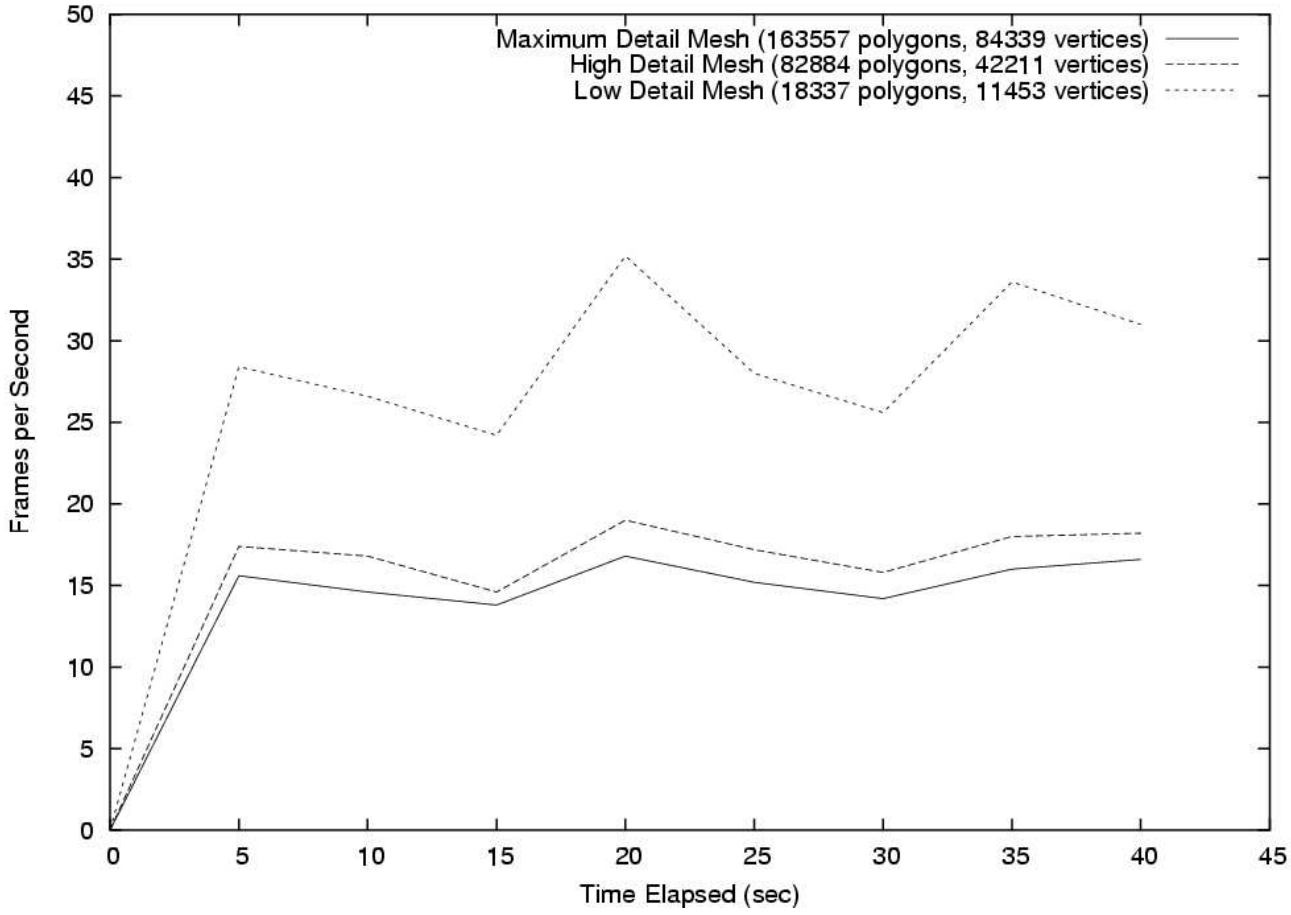

Figure 5.3. Frame-Rates (when using Object based VSMs) - PIII/TNT2

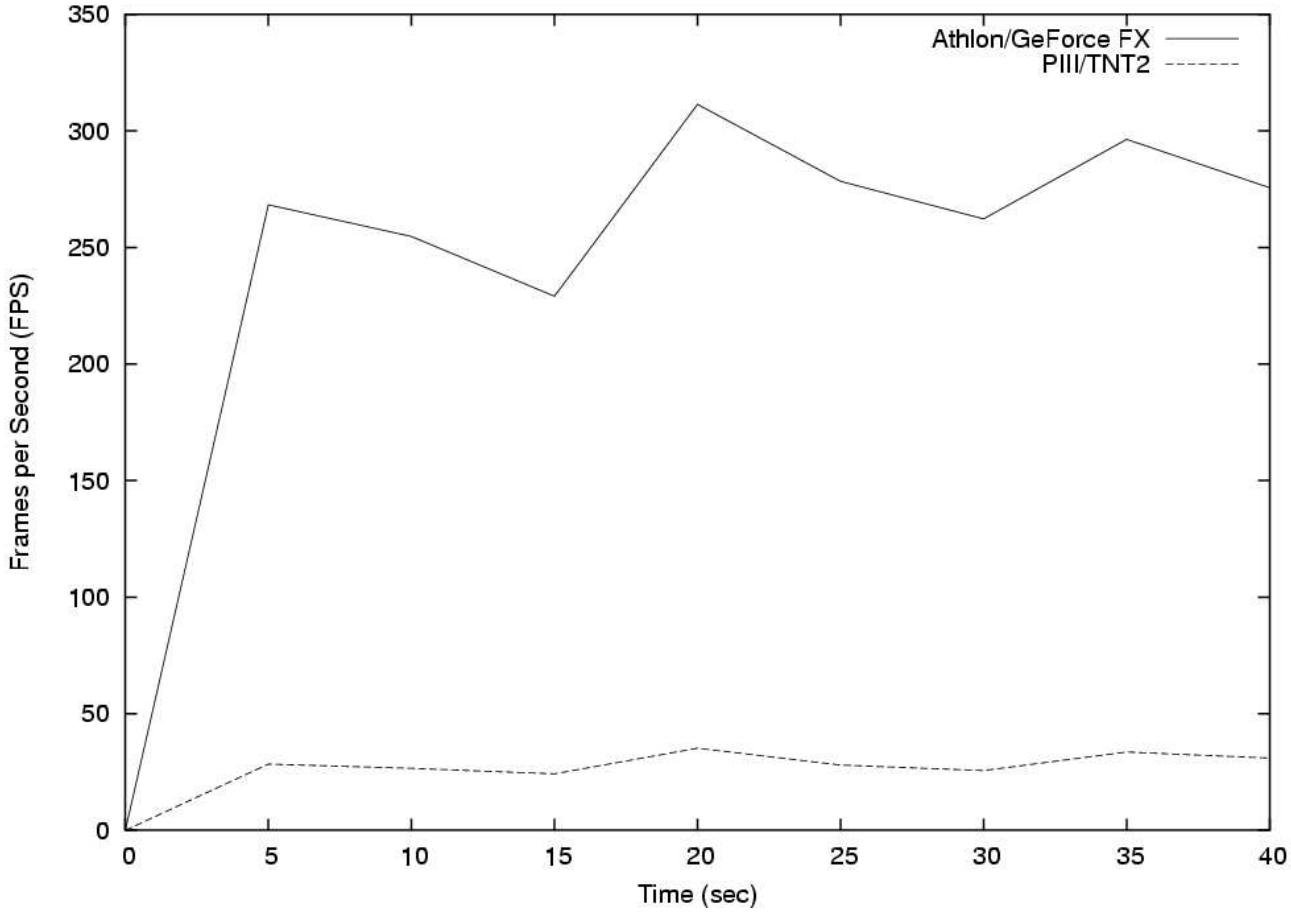

Figure 5.4. Frame-Rate Comparison - Athlon/GeForceFX vs PIII/TNT2 (Maximum Detail)

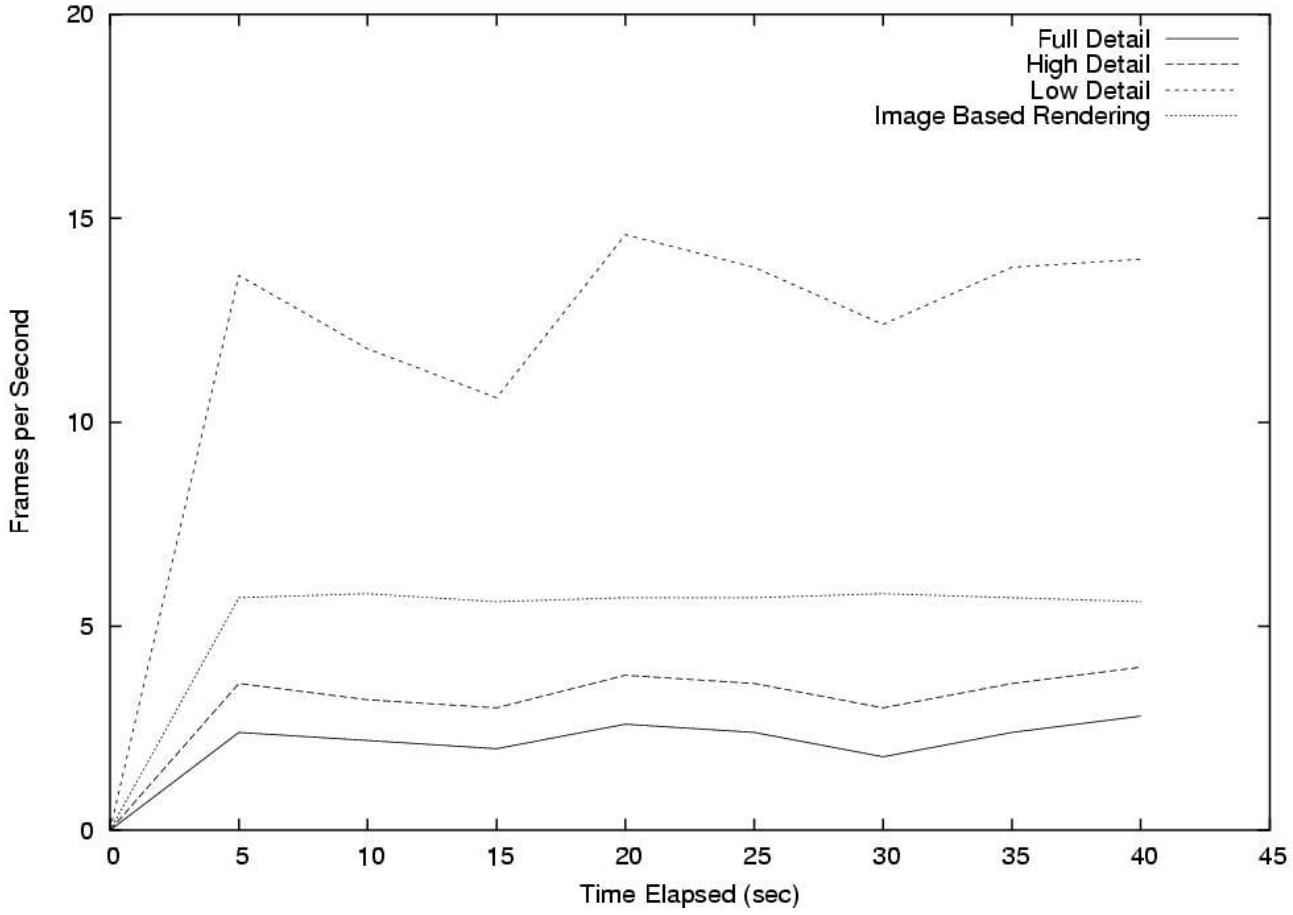

Figure 5.5. Frame-Rates when utilizing Frustum Culling and Image Based Rendering

constant at around five frames per second for the entire duration of the walkthrough and this is independent of scene complexity. This is the major advantage of IBR based methods that the frame-rate will always be at near acceptable levels and will never drop drastically. An IBR approach is best suited for very low end clients that do not possess hardware graphics acceleration.

5.0.3 Walkthrough Quality

The walkthrough quality is a taken as a ratio of the achieved quality of the remote walkthrough compared to the expected quality of a local walkthrough utilizing the same rendering techniques. The quality of the walkthrough will depend on the mesh and texture detail of the model, the average frame rates achieved by the client and an overall freeze-free motion in the virtual environment. A measure of walkthrough quality must take these issues into consideration. To measure the walkthrough quality, we use an empirical quality factor that depends on the level of detail of the models rendered on the client-side,

the quality of the textures transmitted to the client and the frame-rate achieved on the client. This ratio (β) is defined as

$$\beta = \frac{\sum_n ((\frac{1}{\text{LOD}} * \gamma) * (\frac{\text{RemoteFPS}}{\text{LocalFPS}}))}{n},$$

where β is measured after the nth equal interval of time. γ is the quality of the texture transmitted. It is taken to be 1.0 for the full size texture, 0.9 for the half sized texture, 0.8 for a texture one quarter the original size etc. The frame rates must be measured at the same level of detail. That is, the remote rendered output and the locally rendered output must have objects of the same level of detail. Otherwise, varying levels of detail may lead to frame-rate anomalies. The term LOD is taken to be as 1 for the full sized model, 2 for the next level of detail, 3 for the next and so on. LocalFPS is the average frame rate achieved by a system that utilizes the same rendering techniques but renders local models. This must be known separately. The ratio of the remote rendered frame-rate to the locally rendered frame-rates will vary with the number and duration of freezes encountered due to latency or lag. In case of multiple jerks and freezes, the remote rendered frame rate will be substantially lower than the frame rate of a locally rendered model.

For a given client with a fixed connection class, the walkthrough quality is directly dependent on the speed of the viewer and achieved frame-rate. This is because the level of detail of the transmitted model will depend on the speed of movement of the viewer in the virtual environment. Hence we observe from Fig.5.6 that the quality factor improves when the user navigates slowly in the virtual environment and degrades as the user moves rapidly. The results for high level of detail are tabulated in 5.1.

5.0.4 Effect of Speed Based Optimization

To appreciate the effect of speed based optimization, we traverse the walkthrough path at twice the speed and compare the corresponding data rates. It is observed that the data transfer graph in Fig. 5.7 is similar to the graph of the lower bandwidth class only over a period that is only half as long as the latter. This is expected as increase in speed places lowers the detail of the data transferred to the client by one connection class. However in normal cases the client will receive lower detail models only for small periods of time when its speed increases. Hence there will only be marginal variations in data rates at some points in the walkthrough. The image based renderer does not need optimization based upon speed.

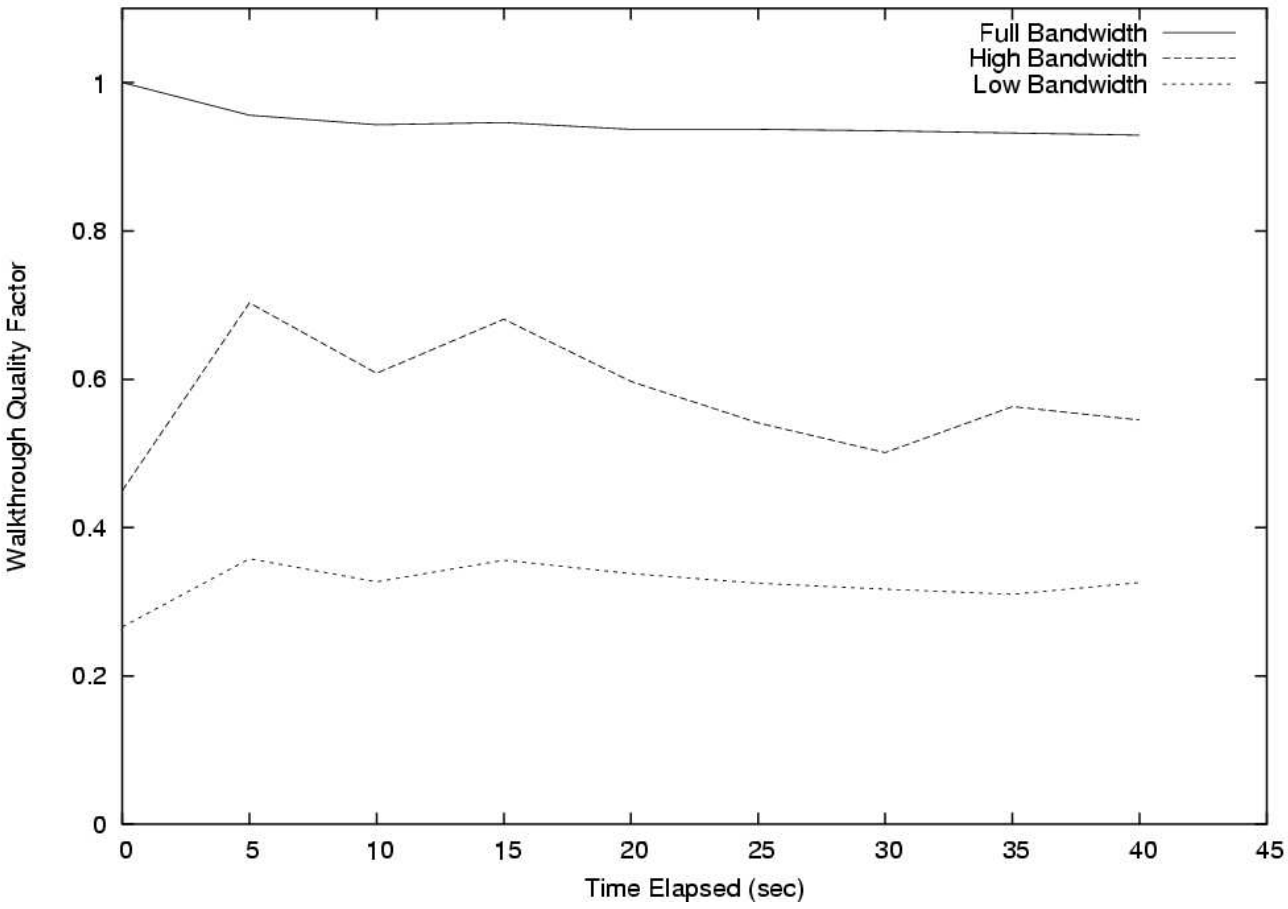

Figure 5.6. Graph of Walkthrough Quality Factor

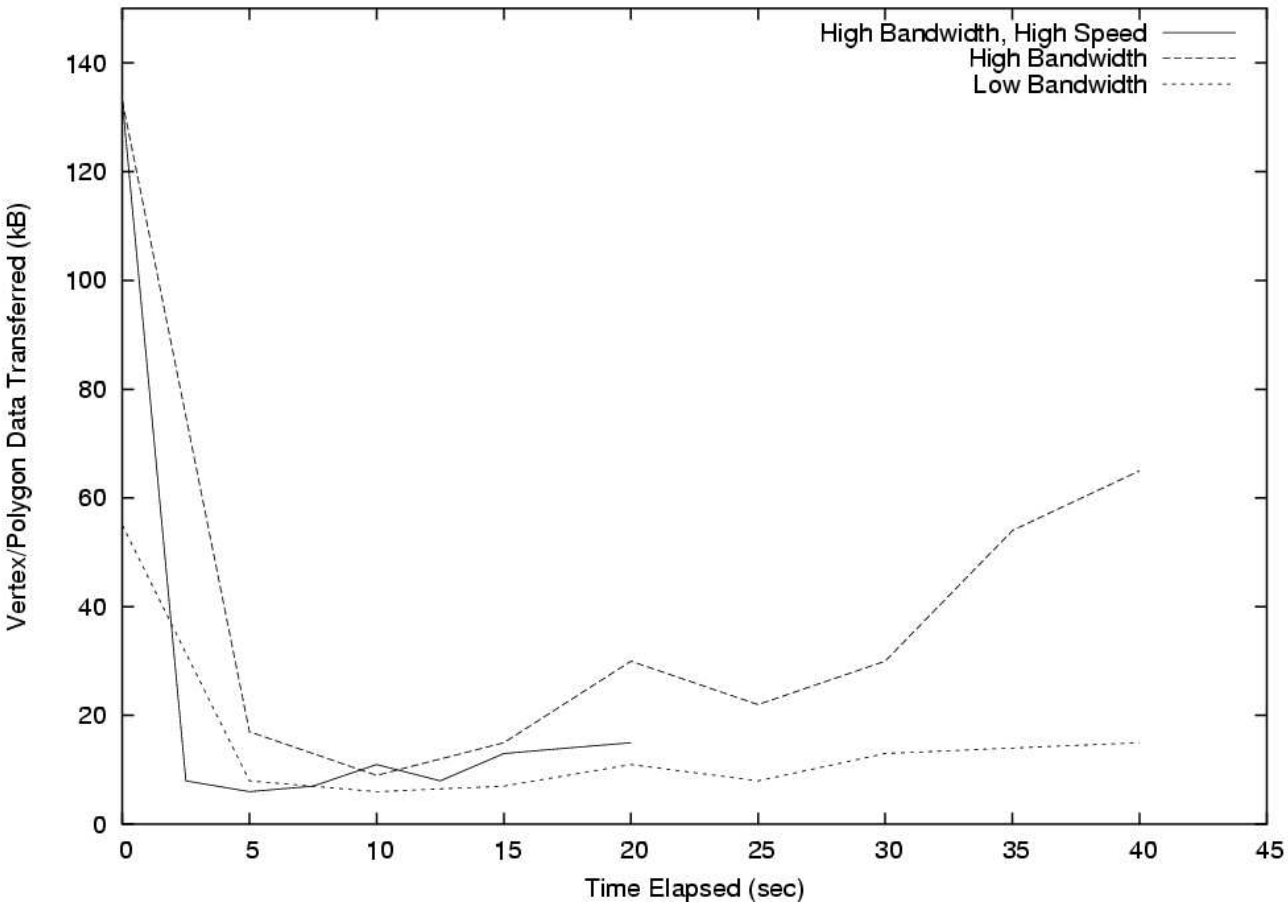

Figure 5.7. Effect of User Speed

Time	LOD	γ	Speed	RemoteFPS	LocalFPS	β
0	2	0.9	Low	0	0	.45
5	1	1	Low	17.4	18.2	.956
10	2	0.9	Med	16.8	18.1	.418
15	1	1	Low	14.6	16.2	.901
20	3	0.8	High	19	19.4	.261
25	3	0.8	High	17.2	17.6	.260
30	3	0.8	High	15.8	16.4	.257
35	2	0.9	Med	18	18.6	.437
40	1	1	Low	18.2	19.0	.958

Average β = 0.545

Table 5.1. Walkthrough Quality - High Level of Detail

5.0.5 Server Characteristics

The server system needs to be a high end system with a enough memory to hold all levels of detail of the model and a fast processor to improve throughput. The server needs to have basic OpenGL rendering capabilities but need not have a very fast or advanced graphics accelerator as it has to render only a few frames. It needs to render only three frames everything a client requests data from it. So the time taken to actually render the views is small compared to the time the system takes to transmit the data, load the model etc. This can also be verified experimentally as tabulated in the Fig. 5.0.5. The three systems used for the server were

- Pentium III 1GHz, 256MB RAM, RIVA TNT2 32MB
- Pentium IV 1.8GHz, 256MB RAM, Onboard Intel Extreme Graphics
- Athlon XP 2400+, 512MB RAM, GeForceFX 5600 256MB

The system also scales the level of detail based upon the number of clients connected. This is to avoid overloading the server with excessive data that blocks its connection completely. The system lowers the LOD by one level when three clients connect at the same time and by two when five or more clients connect.

System	Time 1(s)	Time 2(s)
1	2.109	2.045
2	1.341	1.332
3	0.894	0.634

Table 5.2. Comparison of Server Characteristics

The table shows the time taken to render the Model's View. Time 1 is the time taken with Graphics Acceleration Enabled. Time 2 is Time taken with Graphics Acceleration Disabled. As can be seen, graphics acceleration doesn't affect the time to render a single frame by much.

Chapter 6

Conclusions

We have presented a remote rendering system that adapts to the client characteristics and provides the best possible walkthrough experience to the client. We explored the requirements and design of a remote rendering system. We found that the most important factors in the design of an efficient remote renderer is the way client prediction is handled, optimization of the model data based upon client capabilities and reduction of detail based upon the speed of the viewer. We developed strategies for freeze-free rendering to avoid jerks in motion due to network lag. We implemented a prototype system utilizing Visible Space Models incorporating these principles and presented preliminary results based upon the performance of the said system. We developed an empirical quality factor to measure the effectiveness of rendering of the system.

We modified existing algorithms for visibility culling, prediction, handling local motion, client tracking and cache replacement so that they may be used efficiently for the remote rendering. We used object based VSMs for visibility culling on the source model to get a view-limited partial model that is transmitted to the client. We classify the clients into different connection classes based upon their capabilities and available bandwidth. The model's assets are optimized based upon the client's capabilities and connection speed. The client utilizes a prediction algorithm best suited to its connection speed and latency. The client requests additional data along the predicted path in advance so that motion is smooth even if the latency of the connection changes in the middle of a walkthrough. We developed a client caching policy that ensures that data once sent tot he client is never retransmitted provided that the client has not purged its cache. Based upon client capabilities,the speed of the viewer in the virtual environment and server load, the level of detail of objects is adjusted automatically by the system. Finally the data transmitted to the client is compressed using zlib.

Future work on the system could include optimization based on continuous level of detail and applications to terrain rendering. Dynamic datasets also need to be tested with the system. The scalability aspects of the system are another area for improvement. The system must also support very low end clients like a smartphone or a PDA. If that is done, the same dataset may be used for all kinds of clients. This will allow the server to function as a central repository for models that can then be streamed to all types of clients based upon their characteristics. The Image Based Rendering System needs to be developed to be on par with the Geometry Streaming System. The IBR client is still experimental and not very good in performance.

Bibliography

[1] J. M. Airey, J. H. Rohlf, and Jr. F. P. Brooks. Towards image realism with interactive update rates in complex virtual building environments. In *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, 1990.

[2] Daniel G. Aliaga and Anselmo A. Lastra. Architectural walkthroughs using portal textures. In Roni Yagel and Hans Hagen, editors, *IEEE Visualization '97*, pages 355–362, 1997.

[3] S. Avidan and A. Shashua. Novel view synthesis by cascading trilinear tensors. *IEEE Transactions on Visualization and Computer Graphics*, 4(4):293–306, /1998.

[4] H. Biermann, A. Hertzmann, J. Meyer, and K. Perlin. Stateless remote environment navigation with view compression. Technical Report TR1999-784, NYU, 22, 1999.

[5] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. *Computer Graphics*, 27(Annual Conference Series SIGGRAPH):279–288, 1993.

[6] D. Cohen-Or. Model Based View Extrapolation for Interactive VR Web Systems. *Computer Graphics International*, pages 104–112, 1997.

[7] D. Cohen-Or, Y Mann, and S Fleishman. Deep compression for streaming texture intensive animations. *SIGGRAPH*, pages 261–268, 1999.

[8] M. F. Deering. Geometry compression. *Computer Graphics Proceedings, Annual Conference Series - SIGGRAPH 98*, pages 26–34, 1998.

[9] Suzana Djurcilov and Alex Pang. Visualization products on-demand through the web. In Don Brutzman, Maureen Stone, and Mike Macedonia, editors, *VRML 98: Third Symposium on the Virtual Reality Modeling Language*, New York City, NY, 1998. ACM Press.

[10] R. Earnshaw. *The Internet in 3D Information, Images and Interaction*. Academic Press, USA, 1997.

[11] T. Todd Elvins. Volume visualization in a collaborative computing environment. *Computers and Graphics*, (2):219–222, 1996.

[12] T. A. Funkhouser. Ring: A client-server system for multi-user virtual environments. *Symposium on Interactive 3D Graphics*, pages 85–92, 1995.

[13] T. A. Funkhouser and C. H. Sequin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *SIGGRAPH*, pages 247–254, 1999.

[14] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. *Computer Graphics - SIGGRAPH 96*, 30(Annual Conference Series):43–54, 1996.

[15] N. Greene, M. Kass, and G. Miller. Hierarchical z-buffer visibility. *Computer Graphics Proceedings, Annual Conference Series SIGGRAPH-93*, pages 231–238, 1993.

[16] G. Hesina and D. Schmalstieg. A network architecture for remote rendering. *Proceedings of Second International Workshop on Distributed Interactive Simulation and Real-Time Applications,*, pages 88–91, 1998.

[17] Hugh W. Holbrook, Sandeep K. Singhal, and David R. Cheriton. Log-based receiver-reliable multicast for distributed interactive simulation. In *SIGCOMM*, pages 328–341, Cambridge, MA, 1995.

[18] Hugues Hoppe. Progressive meshes. *SIGGRAPH*, 30:99–108, 1996.

[19] Greg Humphreys, Ian Buck, Matthew Eldridge, and Pat Hanrahan. Distributed rendering for scalable displays. In *IEEE SuperComputing 2000*, 2000.

[20] Eung-Seok Lee and Hyeong-Seok Ko. Vertex data compression for triangle meshes. *Eurographics Workshop 2000*, 2000.

[21] Marc Levoy. Polygon-assisted JPEG and MPEG compression of synthetic images. *SIGGRAPH*, 29:21–28, 1995.

[22] Marc Levoy and Pat Hanrahan. Light field rendering. *Computer Graphics*, 30(Annual Conference Series):31–42, 1996.

[23] J. Li. Progressive Compression of 3D graphics. *Ph.D Dissertaion, University of Southern California*, 1998.

[24] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz. Npsnet: A network software architecture for large scale virtual environments. *Presence*, 1994.

[25] P. W. C. Maciel and P. Shirley. Visual navigation of large environments using textured clusters. *Symposium on Interactive 3D Graphics*, pages 95–102, 1995.

[26] Y. Mann and D. Cohen-Or. Selective pixel transmission for navigating in remote virtual environments. *Eurographics Workshop*, 1997.

[27] Ioana M. Martin. Arte- an adaptive rendering and transmission environment for 3d graphics. In *Proceedings of the eighth ACM international conference on Multimedia*, pages 413–415. ACM Press, 2000.

[28] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. *Computer Graphics*, 29(Annual Conference Series):39–46, 1995.

[29] P. J. Narayanan. Visible Space Models: $2\frac{1}{2}$-D Representations for Large Virtual Environments. In *International Conference on Visual Computing (ICVC99)*, pages 154–161, Feb 1999.

[30] Justin Binns etc Nicholas Karonis, Michael Papka. High resolution remote rendering of large datasets in a collaborative environment. In *International Conference on Distributed Computing Systems*, 1996.

[31] B.O. Schneider and I. M. Martin. An adaptive framework for 3D graphics over networks. *Computers and Graphics*, 1999.

[32] Jonathan Shade, Dani Lischinski, David H. Salesin, Tony DeRose, and John Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. *Computer Graphics*, 30(Annual Conference Series):75–82, 1996.

[33] Sandeep K. Singhal and David R. Cheriton. Using projection aggregations to support scalability in distributed simulation.

[34] Gabriel Taubin and Jarek Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, 1998.

[35] Eyal Teler and Dani Lischinski. Streaming of Complex 3D Scenes for Remote Walkthroughs. *EuroGraphics 2001*, 2001.

[36] C. Touma and C. Gotsman. Triangle mesh compression. *Graphics Interface*, pages 26–34, 1998.

[37] J.C. Trapp and Pagendarm. A Prototype for a WWW-based Visualization Service. *Eurographics Workshop, Visualization in Scientific Computing*, 1997.

[38] I. Yoon and U. Neumann. Web-based remote rendering with IBRAC (image-based rendering acceleration and compression). *Computer Graphics Forum*, 19(3), 2000.

[39] C. Zach and K. Karner. Prefetching policies for remote walkthroughs. *Technical report, VRVis Research Center*, 2001.

[40] Hansong Zhang, Dinesh Manocha, Thomas Hudson, and Kenneth E. Hoff III. Visibility culling using hierarchical occlusion maps. *Computer Graphics*, 31(Annual Conference Series):77–88, 1997.

[41] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes. Computer Graphics : Principles and Practice in C.

[42] Donald Hearn, M. Pauline Baker Computer Graphics with OpenGL

[43] Alan Watt. 3D Computer Graphics.