REMOTEVIS: REMOTE VISUALIZATION OF MASSIVE VIRTUAL ENVIRONMENTS

Soumyajit Deb *P. J. Narayanan*

Centre for Visual Information Technology
International Institute of Information Technology
Gachibowli, Hyderabad 500019
{sdeb@students,pjn@}iiit.net

ABSTRACT

Graphics models for virtual environments are increasingly getting massive and require a large amount of memory to store and high end graphics capabilities to render. It may not be possible or desirable to store the entire model in the station where it will be rendered, especially for environments that change. Such virtual models could be stored on a server and streamed to remote clients on demand. In this paper, we present the design and implementation of a client-server system to render large virtual environments. Our system can provide the best visualization quality for a wide range in connection bandwidth and latency and the rendering capacity of the client. The system uses a visibility-based geometry representation so that little invisible geometry is sent to the client. Incremental updating and client-side prediction of user motion optimize the communication requiremnts for a given client. We present results of the study conducted on a representative range of the relevant parameters in this paper.

1. INTRODUCTION

The last decade has seen a huge growth in the popularity of 3D graphics applications. There have been many attempts at creating collaborative virtual environments over the web. However most of them have been unsuccesful. Large models are bulky to store and require significant graphics capabilities to render. There are, however, no effective means for streaming virtual environments over the network. Transmission of the entire virtual environments 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. 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.

A few web-based visualization systems have been built earlier. VRML was developed for remote interaction. It has been used for streaming successfully with compression of models earlier [1, 2]. Geometry compression [5] can reduce the size of the data to be transmitted. Some web-based visualization approaches use Java to generate VRML files dynamically [9] based on user input. The Silicon Graphics VisServer is an attempt at rendering an OpenGL application on remote clients. Continuous level of detail algorithms have been tried on the server side [3]. Schneider and Martin describe a framework which adapts to the client characteristics including network bandwidth and the client's graphics capabilities [7]. They concentrate on the transmission of individual models rather than complete virtual environments. Teler [8] describes a remote rendering system utilizing path prediction and bandwidth based level of detail reduction.

2. OVERVIEW

A graphics streaming system must provide the best quality of visualization that the capabilities of the client and the available network bandwidth can afford at a consistent frame-rate. The models streamed must improve and match the capability of the client. 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.

2.1. Requirements

The RemoteVIS system architecture envisages a server connected to multiple clients. The client and server module are joined together in a common philosophy of optimizing the navigation experience in the virtual environment at the client side. We enumerate the requirements a remote rendering system must satisfy for a good user experience.

- *High Quality Rendering at Interactive Frame Rates:* The data streamed to the client must match with the client's graphics capabilities and network bandwidth.

- *Freeze-Free Rendering:* 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 and data may be requested ahead of time.

- *Latency Immunity:* 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.

- *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.

2.2. The Client and Server Modules

The server must serve each client without delay. It should also serve as many clients as possible simultaneously. Each client request should be translated into an optimized representation for the current client state based on the parameters supplied to it which is then streamed to the client. The server must also keep track of the data being sent to the client to avoid duplicating any of it in the future. Different clients may use different policies internally and the server should be able to support them.

The clients may have varying capacities with respect to available memory, graphics acceleration, and speed of the CPU. The network bandwidth and latency between the server module and the client may also vary. 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.

2.3. Establishing the Client Parameters

The client first indicates its network speed, rendering capabilities, etc. to the server at the start of the connection. The client and server agree on a range of algorithms and parameters for rendering. The server then starts streaming data in the appropriate format optimized for the particular client. The client parameters can be updated at arbitrary intervals by the client provided the server acknowledges the change.

2.4. Implementation of the System

The RemoteVIS system keeps the above factors in mind and uses a visibility culling algorithm to reduce the amount of data streamed to the client by a large amount. Some important features of the current system are given below.

- The clients are classified based upon the available network bandwidth and graphics capabilities.

- The client requests models that cover the current position as well as its neighborhood

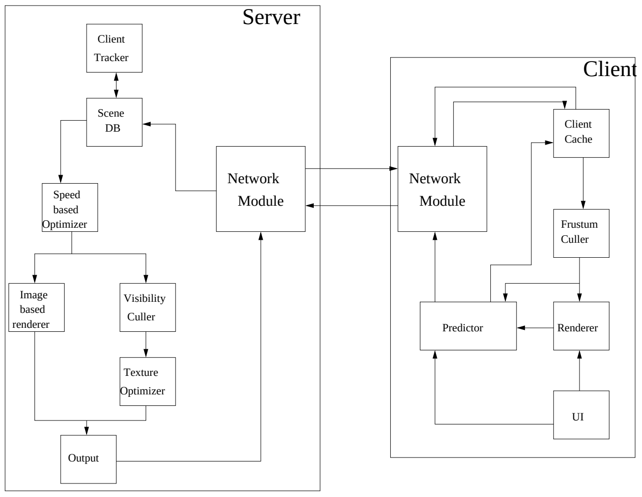

Figure 1: Architecture of the Remote Rendering System

on all the four sides of it.

- The client keeps predicted data which depends on the latency of the connection between the client and server.
- The system possesses three network bandwidth levels which correspond to typical broadband, ISDN and modem data rates. There are three levels of client capabilities which correspond to high-end, midrange and low-end clients.
- The server uses an object based Visible Space Model (VSM) representation for geometry based representation.
- The system dynamically scales textures in based on the client parameters.
- The system uses a simple compression scheme for the model data to reduce the network traffic.

3. VISIBILITY BASED REPRESENTATION

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 are invisible to the user. It must also be able to handle local movement without visual anomalies. A level of detail algorithm may also be employed in addition to visibility culling.

3.1. Visible Space Models (VSM)

Our system uses Visible Space Models as the geometry based streaming representation [6]. A visibility limited, partial model is taken as the basic unit of the virtual environment representation. It 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.

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 visible portions of the virtual environment may be determined either using a polygon based approach or using an object-based approach. The system utilizes object based an object based VSM approach for visibility based culling.

3.2. Rendering Using VSMs

In general, multiple overlapping VSMs are required to achieve hole free rendering [6]. However as a special case, for object based VSMs, only one is sufficient. In addition an object-based representation ensures that each object is sent only once by caching it on the client, Client side caching can hold on to the objects easily in case they are needed, perhaps due to the viewer retracing the path. The system also utilizes multiple levels of detail of the visible objects based upon the distance of the viewer from the object and also the client capabilities. The system also reduces the LOD in case the viewer is moving with high speed. LOD can be integrated seamlessly into a system utilizing object based VSM representations.

3.3. 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. The object-based VSMs, therefore, contains all objects visible from a range of positions instead of one. Since there is considerable overlap between the views from proximate locations, additional data transmitted is generally small as duplication of objects is avoided.

3.4. Texture Optimizer

All textures must be handled in real-time to send the optimal quality textures to the client based upon its connection profile. The size and quality of textures must be optimized. The system utilizes JPEG compression of textures to reduce the bandwidth required for transmission and scales the textures based upon the connection class of the client.

3.5. 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 vertex and polygon data are compressed using ZLIB. The textures need not be compressed as they are already in JPEG format. There are various schemes for vertex data compression based on entropy encoding. However such encoding schemes are expensive to perform and unsuitable for real-time computation.

4. CLIENT SIDE PREDICTION OF MOTION

For smooth, jerk-less motion in the virtual environment and a seamless, hole-free display, prediction of viewer motion is extremely important. A couple of factors 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. Prediction in Virtual Environments

Prediction of motion involves guessing the position of the viewer and requesting the server for data that will be needed in the future. The system currently makes a few simplifying assumptions. The motion of the viewer is also assumed to be decomposed into components that are either rotational motion about a point or translational in a fixed direction. Using these assumptions and the position of the viewer at earlier instances of time, the motion of the viewer is predicted using standard equations of motion.

4.2. Fighting Latency using Prediction

Once the position of the viewer is predicted, the client must request data from the server for future instances of time. Our system requires that the client possess data for the next l seconds, where l depends on the bandwidth and 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 at least $2t$ 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. Our system utilizes only client side prediction. The client may then utilize any algorithm it wishes for prediction. 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.

5. CLIENT CACHING

When a client retraces a point in the VE, the data once transmitted need not be retransmitted if it can be cached on the client side. The server needs to retransmit only in cases when higher quality data is required. The server maintains a list of objects that have been already transmitted to the client and their level of detail.

5.1. 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. All objects that are within a fixed threshold from the current viewpoint are never deleted from the buffer. 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. 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.

6. 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 and server machines were connected on an 100BaseT LAN. The lower bandwidth conditions were simulated over this network. The model 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. A fixed walkthrough path was created and this same path was used for measuring data and frame rates for homogeneity.

6.1. Performance over varying Data Rates

he amount of data transferred during a walkthrough which lasts for approximately forty seconds is shown in Fig. 2. Other than the case of maximum bandwidth, there is progressive refinement of the models and the bandwidth is utilized even when the viewer is idle. 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. In case of a high latency connection,a large amount of data will arrive at the client after substantial intervals.

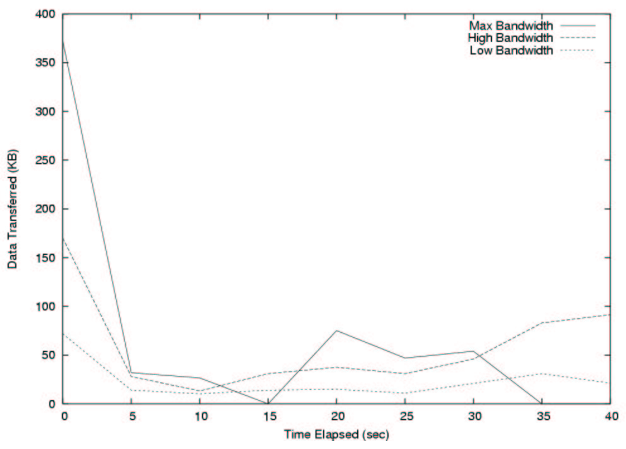

Figure 2: Data Transfer during Walkthrough

6.2. 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 average frame rates are shown in Fig. 3. For comparison, the frame rates achieved by normal brute force rendering methods (View Frustum Culling only) are outlined in Fig. **??**. It is interesting to note that the reduction in primitive count does not affect the frame rate. 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.

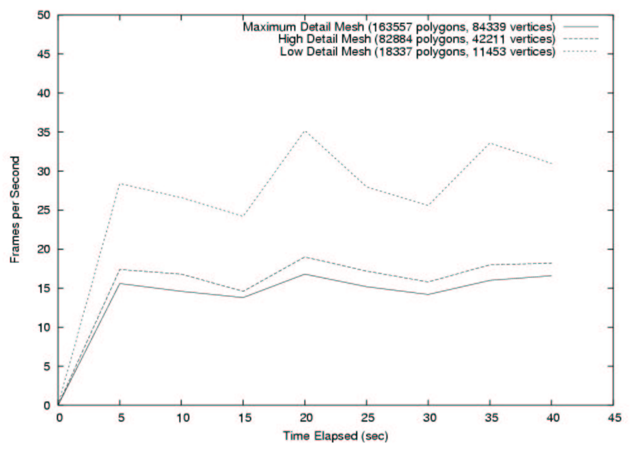

Figure 3: Walkthrough Frame-Rates

7. CONCLUSIONS AND FUTURE WORK

We presented a remote rendering system that adapts to the client characteristics and provides the best possible walkthrough experience to the client. We presented the requirements and design of a generic 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. The results were presented on a large model under different conditions.

REFERENCES

[1] 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.

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

[3] 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.

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

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

[6] 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.

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

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

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