RepVis: A Remote Visualization System for Large Environments

Soumyajit Deb and P. J. Narayanan
Centre for Visual Information Technology
International Institute of Information Technology, Hyderabad
{sdeb@students,pjn@}iiit.net

Abstract

Large virtual environments arise in many applications and may be used by many users simultaneously for walkthroughs. It is not feasible to store a copy of the whole environment at each of the client locations due to their size. The dynamic nature of the environment also makes it more convenient for a central server to ensure consistency across different users. In this paper, we present a client-server based system for visualizing large environments from remote users. The client gets only the portion of the geometry necessary for navigating around the current user location. The system can adapt to different client parameters such as graphics capability, network bandwidth, communication latency, etc. We present the design and implementation of the system in this paper and produce relevant results using it.

1. Introduction

The last decade has seen a huge growth in the popularity of 3D graphics applications. These include games, walkthrough of large architectural spaces or large outdoor scenes, and visualization of large scientific data sets. Large models are bulky to store and require significant graphics capabilities to render. Large models are also likely to be useful to a number of users simultaneously, possibly sitting far from each other. Transmission of the entire virtual environments is impractical due to its bulk. When the environments change frequently, consistency also demands that the model be stored and manipulated by a central server, which ensures that all clients using it see updations in a structured manner. Streaming of models must be utilized for this. There are no effective means for streaming virtual environments over the network.

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. Lastly the server must be able to support clients with limited resources in terms of graphics capability, memory resources and the CPU power.

A few web-based visualization systems have been built earlier. VRML was developed for remote interaction. It has been used for streaming with compression of models earlier by Djurcilov and Earnshaw [1, 2]. It, however, works only for static models. Geometry compression [3] and level of detail can reduce the size of the data to be transmitted. The Silicon Graphics VisServer is an attempt at rendering an OpenGL application on remote clients. Schneider and Martin describe a framework which adapts to the client characteristics including network bandwidth and the client's graphics capabilities [5]. Teler [6] describes a remote rendering system utilizing path prediction and bandwidth based level of detail reduction. None of these explicitly address the problems of a dynamic environment effectively.

1.1. Streaming System: Requirements

The RepVis system architecture envisages a server connected to multiple clients that 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 the current view as well as the possible views in the immediate future
- *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.

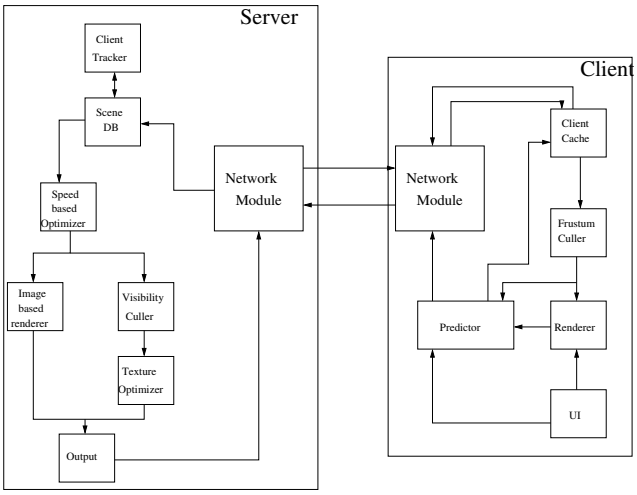


Figure 1: Architecture of the Remote Rendering System

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 model representation. 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 rendering hardware, network bandwidth and 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.The client and server agree on a range of algorithms and parameters for rendering which can be changed at will. The server then starts streaming data in the appropriate format optimized for the particular client.

2. Design and Implementation

The RepVis system utilizes a visibility culling algorithm based upon Object based Visible Space Model(VSM) representation to dramatically reduce the amount of data streamed to the client. The client utilizes path prediction and requests models that cover the immediate vicinity and also the models on the predicted path. The amount of prediction depends upon latency between the systems. The system possesses varying levels of service corresponding to high-end, midrange and low-end clients and scales the models and assets accordingly. To reduce network traffic, the system utilizes a simple compression scheme.

2.1. 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. Our system uses Visible Space Models as the representation [4]. 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 frustum but also the objects inside that are occluded by closer objects.The visible portions of the virtual environment may be determined either using a polygon based approach or an object-based approach.

2.2. Rendering Using VSMs

In general, multiple overlapping VSMs are required to achieve hole free rendering [4]. However as a special case, for object based VSMs, only one is sufficient.

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. Conversely there is gradual improvement in LOD when the viewer is stationary.

2.3. 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.

2.4. 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.

2.5. 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. We refrain from using entropy based schemes as they are unsuitable for real-time computation.

2.6. 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. Prediction of motion involves guessing the position of the viewer and requesting the server for data that will be needed in the future. Using the position of the viewer at earlier instances of time, the motion of the viewer is predicted using standard equations of motion. 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 so that latency would not hurt the performance of the renderer on the client side.

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.

2.7. 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. Since the client only has a finite cache, the system utilizes a modified Least Recently Used(LRU) algorithm based approach to expunge objects. All objects within a fixed threshold from the current viewpoint are never expunged. Whenever an object is expunged, the server is notified of the change.

3. 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. A fixed walkthrough path was created and this same path was used for measuring data and frame rates for homogeneity.

3.1. Performance over varying Data Rates

The amount of data transferred during a walkthrough which lasts for approximately forty seconds is shown in Fig. 2.

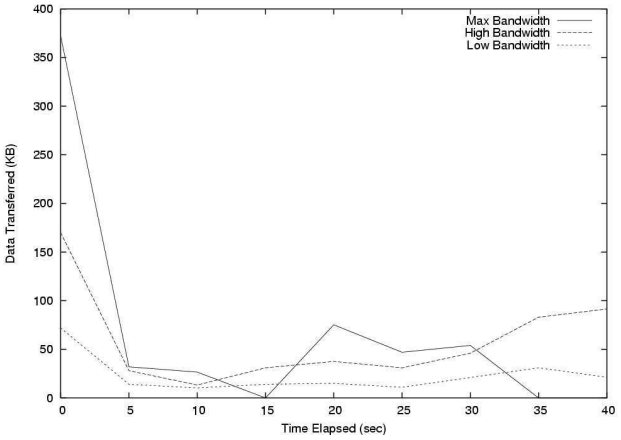


Figure 2: Data Transfer during Walkthrough

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.

3.2. Walkthrough Performance and Frame Rates

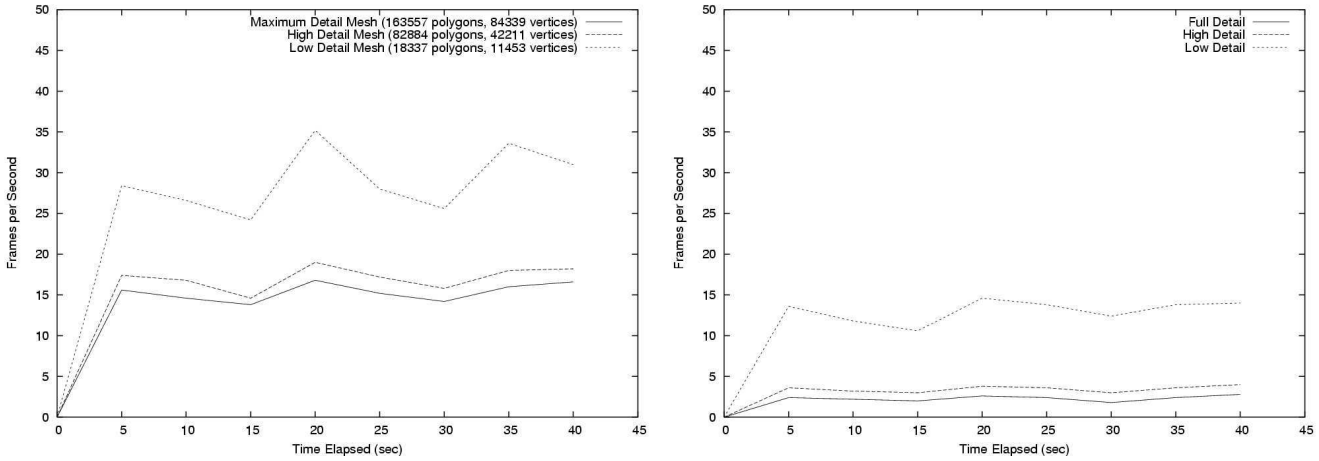
Once data has been received by the client, the performance of walkthrough is solely dependent on the graphics capabilities of the client. The average frame rates are shown in Fig. 3(a). For comparison, the frame rates achieved by normal brute force rendering methods (View Frustum Culling only) are outlined in Fig. 3(b). The frame rates achieved by VSM based methods is much higher primarily because of the inherent visibility limiting nature of VSMs, which culls out nearly all invisible objects.

3.3. 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 quality of the model and an overall freeze-free motion in the virtual environment. We use an empirical quality factor that depends on the level of detail of the models, the quality of the textures transmitted 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 proportional to the quality of the texture transmitted.



(a) (b)

Figure 3: Walkthrough Frame Rates using (a)Object based VSMs and (b)Frustum Culling

LocalFPS is the average frame rate achieved by a system that utilizing 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 reduce with increasing number and duration of freezes encountered due to latency or lag. The quality factor improves when the viewer is stationary and reduces when moving quickly due to speed based optimizations. The results are tabulated in Table 1.

Time	LOD	γ	User Speed	Remote FPS	Local FPS	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 1: Walkthrough Quality - High Level of Detail

3.4. 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 system 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. A uniform quality of service must be provided to all clients. Our system lowers the LOD by one level when five clients connect at the same time and by two when ten or more clients connect. These limits are arbitrary and may be modified based upon the available bandwidth on the server.

4. 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] S. Djurcilov and A. Pang. Visualization products on-demand through the web. In *VRML 98: Third Symposium on the Virtual Reality Modeling Language*. ACM Press, 1998.

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

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

[4] 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.

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

[6] E. Teler and D. Lischinski. Streaming of Complex 3D Scenes for Remote Walkthroughs. *EuroGraphics 2001*, 2001.