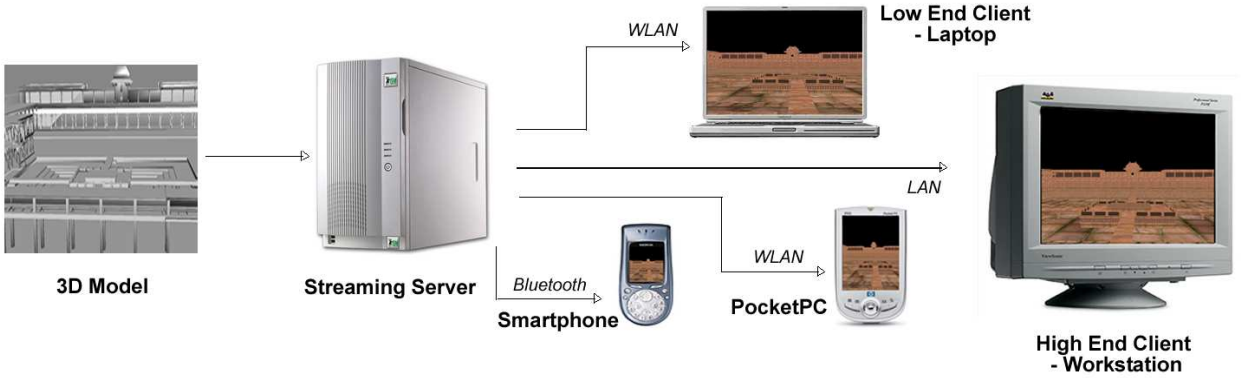
Design of a Geometry Streaming System

Soumyajit Deb and P. J. Narayanan
Center for Visual Information Technology
International Institute of Information Technology
Hyderabad 500019



Abstract

The size and detail of graphics environments in everyday use has gone up considerably recently. Still, most applications use locally resident geometric content for rendering. In this paper, we present a system to stream large graphics environments from a central server to multiple number of clients. The streaming is transparent to the user who can treat remote models just like local ones. The streaming system automatically adapts to the rendering capabilities, network bandwidth and latency of the client and transmits an optimized model. We present the design of the streaming system and give results of streaming a large model 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, visualization of large scientific data sets, etc. Graphics acceleration capabilities of PCs improved drastically in recent years. However, nearly all 3D applications are rendered using content stored on the local system. The client-server model of computing has not really been popular for 3D applications. The reason could be the slower improvement in the speed and latency of computer networks compared to that of the graphics technology. Hence, there are no effective means for streaming bulky virtual environments over the network.

Rendering streamed geometry remotely could be preferable to local rendering in many situations. (1) Bulky models need not be completely downloaded in order for them to viewed. Only the relevant portions can be streamed to the client for remote rendering. (2) Having a central repository for all the models and assets enables the content provider to consolidate and manage all content at the same place. The provider can then stream the content to the client based on the capabilities and connection speed of the client. The rendering technique and detail levels will depend on the type of client. Remote rendering is helpful in cases when the content on the server needs to be protected. A recent work by Koller et al. [8] describe a remote rendering system intended for this. (3) A central control of the environment is also useful in case of dynamic data that changes in response to external inputs. The data can be kept consistent as it needs to be updated only at a single server. The client automatically receives 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. (4) Multiplayer games with dynamic worlds and large extent will also benefit from streaming.

1.1. Related Work

Web-based visualization systems have received some attention recently. VRML was developed for remote interaction. It has been used for streaming with compression of models earlier by Djurcilov and Earnshaw [4, 5]. This however works only for static models. Li uses Geometry compression [10] and level of detail to reduce the size of the data to be transmitted. Most remote rendering and geometry streaming efforts have focussed only on geometry compression. Excellent work has been reported on both lossy and lossless compression of geometric data that in-

clude progressive meshes by Hoppe [6] and Edgebreaker by Rossignac et al. [7], etc. While compression is necessary for streaming geometry, other system design issues need to be addressed as well. Rusinkiewicz and Levoy [9] use a multiresolution splat for streaming geometry. Splats use only points as features and need no polygon connectivity information. They are also faster to render than polygons, but are riddled with resolution issues. A similar method by Bischoff [2] uses ellipsoids to approximate the geometry of the model. These approaches require rigorous preprocessing and is unsuitable for dynamic data. The remote rendering system from Stanford mentioned above gives a crude model to the client which can be used for navigation [8]. The viewing parameters are sent to the server which generates the actual view and sends the image to the client. The Silicon Graphics VisServer follows a similar philosophy and streams rendered images to clients.

These approaches do not address the overall system issues of network speed and lag and the client capabilities for rendering. Biermann et al. [1] describe a method of remote rendering utilizing prediction and image based rendering. This method renders frames on the server and sends the views in compressed format to the client which utilizes IBR. The main problem with IBR algorithms for remote rendering is that they are not robust enough to handle rapid view changes. Schneider and Martin describe a framework which adapts to the client characteristics including network bandwidth and the client's graphics capabilities [13]. They limit themselves to individual 3D models and not to entire virtual environments. Teler [14] describes a remote rendering system utilizing path prediction and bandwidth based level of detail reduction. This system assumes a powerful client and ignores its characteristics. It also does not use visibility to remove unnecessary geometry from the scene.

We present an adaptive system for streaming of graphics environments in this paper. Our focus is on the overall system aspects of geometry streaming; geometry compression; culling etc are important and can improve the performance of our system also. A preliminary system with a fixed client and static environments was described in an earlier paper [3]. The architecture supports 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. The data streaming approach ideally is handled in a completely transparent manner for the user. User programs do not distinguish between local models and streamed remote models; there can be a free intermingling of remote and local objects. The system adapts to the client's rendering capabilities, network bandwidth and latency, and the user motion and automatically scales down content to match the capabilities of the situation so that the user experience of navigating the virtual environment is as fluid and jitter free as a local walkthrough.

Section 2 describes the objectives of a transparent geometry streaming system. Section 3 presents our system in some detail. Summary of results indicating the performance of the system is given in 4 and some concluding remarks in the last section.

2. Transparent Geometry Streaming

The streaming system consists of multiple independent modules that exchange data between them. The schematic diagram of the system is presented in Fig 1 and 2 The main components of the system are:

- User Program: The user program on the client-side actually renders the virtual walkthrough. Any graphics program could be the user program; no assumptions are made about it. It uses the client API to request models from the remote server.
- Client Module: The client module receives geometric data from the server and interfaces with the user program. It also utilizes caching and user path prediction to improve the user experience.
- Server Module: The server module generates the optimum representation to be streamed based on the client parameters. It interfaces with the client module and keeps track of relevant aspects of the client and the user. The server also keeps track of the dynamic objects and notifies the clients if they change.
- Server Input and Pre-Processor: The input program uses the server API to register a model with the server. The model is converted to the internal representation by the pre-processing module and is then ready for streaming.

2.1. User View

Under transparent streaming, the user needs to make no distinction between a local model and one that is streamed from a server. The user interfaces with the client module and loads remote models using the API like other models. The user can render, reason about, and manipulate these models like any other. The user can also change aspects of the remote model such as replacing a texture with a local one. Conceptually, a subtree of the user's graphics scene graph sits on the remote server. The user program is responsible for the interaction with the user and the navigation control in the virtual environment. The user program passes the motion parameters to the client module on user movement in the virtual environment.

2.2. Client View

The client library interfaces with the user program, performs caching and prediction needed for better performance, and interfaces with the server. The client API provides the interface hooks for the user program. It has access

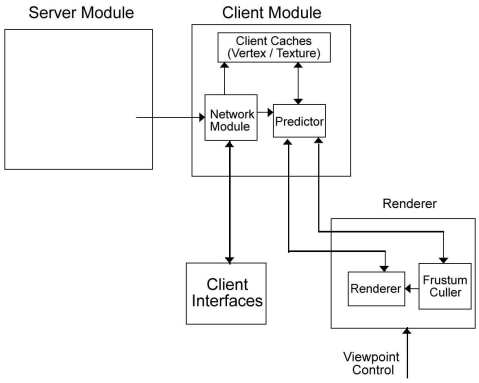


Figure 1: Client Side Block Diagram

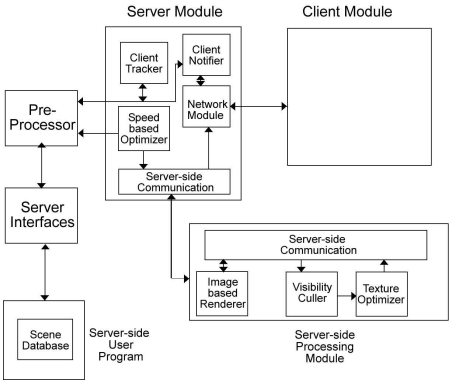


Figure 2: Server Side Block Diagram

to certain aspects of the remote model after it is streamed and loaded into the user's scene graph. This is needed for caching parts of the model. The client enables the user to browse and select from available environments for streaming.

The client also interfaces with the server module. The server sends relevant sections of the spatially partitioned model in response to a client request. The data could be sent in compressed form or otherwise, depending on the available network bandwidth. The model is cached at the client even after it is not needed anticipating a retracing of the user's path. Subsets of the scene graph of the remote environment model will be present at the client, based on the path visited by the user. The client returns a handle to the potentially visible subtree to the user which is used for rendering. Client is also notified by the server when an object has changed. The client then decides if the changed model should be streamed in immediately or later. The client module uses path prediction in addition to caching so that models required in the future are available before they are needed. This is a handle to fight the network latency of the system so that freeze-free rendering is possible.

2.3. Server View

The server module has the total model of the environment in the prescribed internal representation format. Server receives requests for streaming from clients. In response, it generates and transmits a representation of the model suitable for the client. The server has the following functions.

(1) The server assesses the client capabilities, bandwidth, and latency and selects a level of detail suitable for it. This can change with time. For instance, when the bandwidth is low, a low quality model is sent to the client initially. When spare bandwidth is available, a higher detail model could be sent if the client has the capacity to render it. The server keeps track of the essential state of the client including the list objects in its cache and the local assets used by the client (if any) in addition to its rendering capability and the user speed.

(2) The server is notified when an object changes its position or appearance by whoever is in charge of it. The server pro-actively notifies every client who holds the object in its cache about its change. The changed model is sent to the clients only on demand.

(3) The server has an API for registering environment models. These are converted to the internal representation before being made available for streaming. This conversion may involve computing LODs, texture manipulation etc., and could be a time consuming process. This offline task is performed by the pre-processing module of the system.

(4) The server itself could have a distributed architecture. The server module could manage the client interface and could coordinate the activities of other processors that actually perform the selection and transmission of the model to the client.

2.4. The Internal Representation

The internal representation is the in-core format of the complete environment to which all registered environments are converted. Multiple external representations may be supported by the system. The characteristics of our internal representation are described below.

Only potentially visible portions of the environment is streamed to the client, selected on the fly by the server. Thus, the internal representation should facilitate visibility determination using rendering on a per object basis. Parts of this representation will be transmitted to the client and will be used for rendering there. Thus, the internal representation should be hierarchically organized and spatially partitioned for maximum utility.

A tree with leaf nodes containing geometry and the internal nodes containing additional information such as transformations is well suited for this. The scene graph structure of tools like Performer and Open Inventor are organized this way. We use the Performer scene graph as it is a general data structure that provides all graphics features. As mentioned earlier, a subset of this scene graph will be present at the client, depending on the portion of the environment

seen by its user. Spatial ordering of objects will help in this; when an object is sent to the client, the entire branch from the root to the leaf node is transmitted. Multiple levels of detail of each object may be present at the leaf node. Only the relevant levels of detail will be sent to the client.

3. The Streaming System

The primary objective of the streaming system is to optimize the walkthrough experience of the user given the client parameters. The following are the requirements of an ideal remote rendering system.

- *Highest Quality Rendering at Interactive Frame Rates:* The data streamed to the client must match with the client's graphics capabilities and network bandwidth. A lower detail model only should be sent if the client has low capabilities. The model quality could be lowered further if the size of the model will cause the frame to freeze at the available bandwidth.
- *Freeze-Free Rendering:* The client must request sufficient data to cover the current view as well as its immediate neighbourhood to avoid pauses in rendering. This assumes that the user moves continuously in space. The models of individual objects received should also be cached. This avoids the need to get them again if the user retraces the path, which is quite likely.
- *Latency Immunity:* Poor latency can result in delays and freezes, reducing the interactive viewer experience. Predicting the user motion can alleviate this problem. The client module predicts the user's future locations assuming a parametric motion model based on the past positions. Geometry data for possible future locations is fetched from the server in advance. A good rule is to request data that may be needed in the next t seconds, where t depends on the network latency.

The server must serve multiple clients simultaneously without delay. Each client request should be translated into a model optimized for its parameters. The server should be able to support different types of clients simultaneously. The parameters of the client that affects the model sent to it are the following. (a) Client capability: The geometry transmitted must match with client's hardware. Better quality models need to be transmitted in case the client has good graphics acceleration. It is fruitless to send a high quality model to low-end client machine. (b) Network bandwidth: The quality of the models to be transmitted is directly proportional to the available network bandwidth as higher quality models contain more bytes. A low quality model at the client could be replaced by a higher quality one when the request for new models is modest. (c) Connection latency: The higher the latency, the slower the response of the system to user motion. (d) User speed: The model detail can be optimized based on the user's navigation speed. Lot of data is required when moving fast as more new geometry will be visible. However, low quality models will suffice when the user is moving fast. When the user slows down, better quality models can be sent to the client.

3.1. Salient Features of the System

The important features of the system are summarized in this section. Many of the details relating to prediction and caching algorithms used are similar to those used in [3]. The five most important aspects of the current system are given below.

Visibility Based Representation: Only the potentially visible portion of the scene is sent to the client by the server. Thus, invisible parts of the environment are removed initially. We use Object-based Visible Space Models as the basic representation [12]. In this scheme, an object is transmitted to the client if any portion of it is visible. Visibility is checked by rendering and reading back. Each object could have multiple levels of detail. The right level of detail is transmitted based on the client capabilities and navigation speed of the viewer.

Handling Local Motion: Visibility is not limited to a single point in space. Visibility from a small region surrounding the user's actual camera location is used so that small local motion can be supported without needing new data. A single transmitted block of data handles local motion around a particular VSM viewpoint.

Compression of Transmitted Data/Textures: The representation we use supports a number of geometry compression schemes. Thus, the model could be represented as progressive mesh [6] or as an advancing fan front [11]. In addition, the data to be transmitted is compressed using a library like `zlib` if bandwidth demands it. Textures are compressed using JPEG and reduced in size based on available bandwidth.

Client Side Prediction of Motion: Prediction of viewer motion is used to fight the network latency. Our system currently supports looking around, linear motion, and quadratic motion. The user's future path is predicted based on the past and geometry is requested from the server. In our system, prediction is transparent to the server; the client can use any algorithm it wishes for prediction. Currently, the prediction window used is 2 seconds by default. This can go up or down based on the actual latency.

Client Caching and Dynamic Scenes: Object models are cached on the client side to avoid retransmission. The server keeps track of objects present in the client cache for consistency. An object needed by the client is not sent to it by the server if it is present in the cache. If an object changes position or appearance, the server informs the client of the same. The client first sets a dirty bit in its cache for the object. The client then is free to either request the object

immediately or adopt a lazy strategy, requesting for it only when the object comes into view. An LRU algorithm is used if objects are to be removed from the cache. Whenever an object is removed, the server is notified of the change.

4. Results

We implemented a prototype geometry streaming system. This includes the client and server modules and the user API. The preprocessing module is currently decoupled from the system and run separately. The system performance figures and the rendering quality figures is not affected by this, however.

We present several results from our system in this section. The high-end client machine consisted of a 2.0 GHz AMD Athlon 64 CPU coupled with 1GB of RAM and ATI Radeon 9800 Pro graphics. The server used was a similar machine without the graphics accelerator. The medium-end client machine was a Celeron CPU with 256MB RAM and basic video acceleration. The very low-end clients include a PocketPC connected using a wireless LAN and a Nokia 3650 cellphone connected using Bluetooth to a PC. The client and server machines were connected on an 100BaseT LAN. The lower bandwidth conditions were simulated over this network by limiting network traffic. The model used was that of Fatehpur Sikri, created by hand by NCST (http://rohini.ncst.ernet.in/fatehpur/). This model is made up of about 524,000 triangles. The uncompressed size of this model is around 140 MB and the assets add upto 15 MB. Another smaller castle model was used for certain tests, containing about 200,000 triangles.

Time	Data Received (High/Med)	FPS (High/Med)
0	2144/953 KB	85/37
5	1274/591 KB	77/28
10	1359/704 KB	69/35
15	954/487 KB	55/22
20	0/0 KB	94/34
25	688/321 KB	90/32
30	1033/597 KB	81/26
35	1563/899 KB	84/31

Table 1: Model size, data received and frame rates for high-end and medium level clients

The frame-rate and the amount of data transferred during a walkthrough which lasts for approximately thirty seconds is shown in Table 1. The data transmitted during the walkthrough is only a very small fraction of the total size of the model. The user was stationary at around 20 seconds and no new data was streamed. For clients with lower capabilities, a lower level of detail must be streamed.

4.1. Measure of Quality

We devised an empirical measure to assess the quality of the client's remote walkthrough experience, compared to a local walkthrough. The quality factor is a function of the model/texture detail and the average frame-rates achieved. There are two aspects to the overall experience: the rendering quality and the network utilization.

The asset quality a_k of the kth visible object is the ratio of its model quality at the given detail compared to the detail that the client hardware can support. At any instant of time t, let a_{kt} be the quality of the kth object. Let fpsr_t and fpsl_t be the frame rates achieved at time t using remote and local rendering of the same model respectively. We cap the frame rate at 60 fps so that lower LODs do not skew the results due to excessively high frame-rates. Let n_t be the number of objects visible at time t. Assuming the walkthrough lasts between time instants t_0 and t_1, the rendering quality factor τ_{qual} is defined as

$$\tau_{\text{qual}} = \frac{\sum_{t=t_0}^{t_1} (\frac{\text{fpsr}_t}{\text{fpsl}_t} (\sum_{k=1}^{n_t} a_{kt}))}{\sum_{t=t_0}^{t_1} n_t}$$

This measures the match of the streamed model with the client capability for rendering, subject to the client capabilities and has a maximum value of 1.0.

The factor τ_{net} measures the utilization of the network when there was demand for it. Ideally, the available bandwidth should be utilized. This depends on the available (bwa_t and the utilized bandwidth bwu_t at time t. We define τ_{net} as

$$\tau_{\text{net}} = \frac{\sum_{t=t_0}^{t_1} (1 - \text{req}_t) \frac{\text{bwu}_t}{\text{bwa}_t}}{\sum_{t=t_0}^{t_1} (1 - \text{req}_t)}.$$

where req_t instantaneous requirement, which is either 0 or 1. $\tau_{\text{net}} = 1$ can be achieved when the entire network bandwidth is used whenever there is a request.

The achieved quality factors for different types of connections are given in Table 2. Three different types of clients are used, corresponding to high-end, medium-end and low-end. We also utilize two different bandwidth levels (100KB/s and 20KB/s). These figures are averaged over the 35 second walkthrough of the virtual environment.

Processor Spec	τ_{qual} (High/Low BW)	τ_{net} (High/Low BW)
High	0.99 / 0.47	0.57 / 1.00
Med	0.97 / 0.69	0.48 / 0.96
Low	0.91 / 0.78	0.38 / 0.83

Table 2: Quality factor for different client parameters

We can infer the following from Tables 1 and 2. (1) The quality factor is the highest for the high-end client with high bandwidth as it always receives full quality models during

the course of the walkthrough. For clients with low capabilities, the quality factor reduces due to a reduced frame-rate. This happens because additional time is spent for prediction, caching, etc. (2) When the bandwidth is low, the quality factor reduces as models with lower LOD are streamed most of the time. The quality factor is affected to a lesser extent on the lower end client since the LOD of models streamed is much lower than the high-end client due to lower client capabilities. (3) The network utilization is always good for lower bandwidth connections, as it is easy to utilize the bandwidth completely. The requests may not generate sufficient data if high network bandwidth is available.

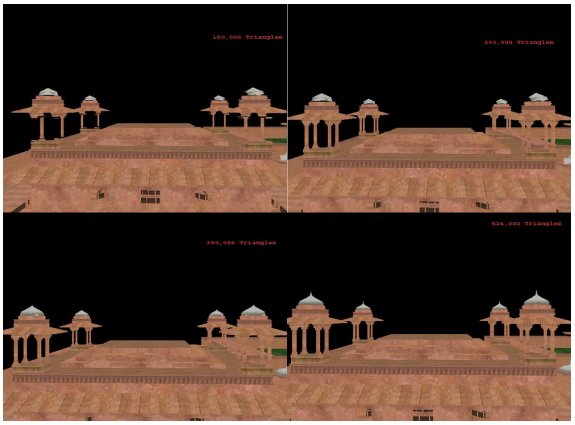


Figure 3: Walkthroughs at varying levels of quality. Top: Low (100K tris) and medium (250K). Bottom: High (390K) and Full Quality (524K). Note the difference in quality of model and texture.

4.2. Tests on Very Low-End Clients

We tested the system on a Dell Axim PDA running PocketPC OS. The memory constraints and low CPU speed make rendering geometry on it very slow. Each frame takes 5 to 8 seconds to render. The technique of streaming was modified slightly with rendering done on the server. Compressed JPEG images of appropriate size are transmitted over the wireless network and the PocketPC displayed it. Similar techniques work on a cellphone with Bluetooth connectivity. Frame rates in the range of 5fps can be got by this.

5. Conclusions and Future Work

We presented a system to stream geometry transparently from a server to the client. The users on the client machine loads remote geometry into their scene graphs normally. The quality of the streamed models adapts to the client characteristics for the best possible walkthrough experience. The system works reasonably on a wide variety of clients from high-end PCs to low-end PDAs. The current efforts are to implement a complete Performer based

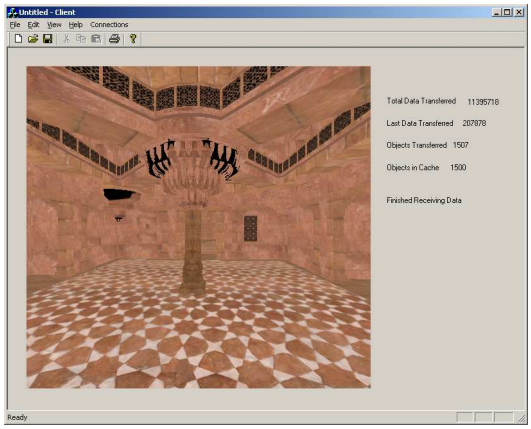


Figure 4: Screenshot of the user interface

streaming system. The user will be able to integrate any remote model with their own scene graph.

Acknowledgements: We thank the CDAC, Mumbai (formerly National Centre for Software Technology) for the permission to use the Fatehpur Sikri model.

References

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

[2] S. Bischoff and L. Kobbelt. Ellipsoid decomposition of 3d-models. *3DPVT*, 2002.

[3] S. Deb and P. Narayanan. Remotevis: Remote visualization of massive virtual environments. In *Proceedings of National Conference on Communication*, 2004.

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

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

[6] H. Hoppe. Progressive meshes. *SIGGRAPH*, 1996.

[7] D. King and J. Rossignac. Optimal bit allocation in compressed 3d models. *Computational Geometry*, 14(1-3):91–118, 1999.

[8] D. Koller, M. Turitzin, M. Levoy, M. Tarini, G. Croccia, P. Cignoni, and R. Scopigno. Protected interactive 3d graphics via remote rendering. *SIGGRAPH*, 2004.

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

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

[11] S. P. Mudur, S. V. Babji, and D. Shikhare. Advancing fanfront: An effi cient connectivity compression technique for large 3d triangle meshes. 2002.

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

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

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