Remote View-Dependent Rendering

Jihad El-Sana

Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel
el-sana@cs.bgu.ac.il

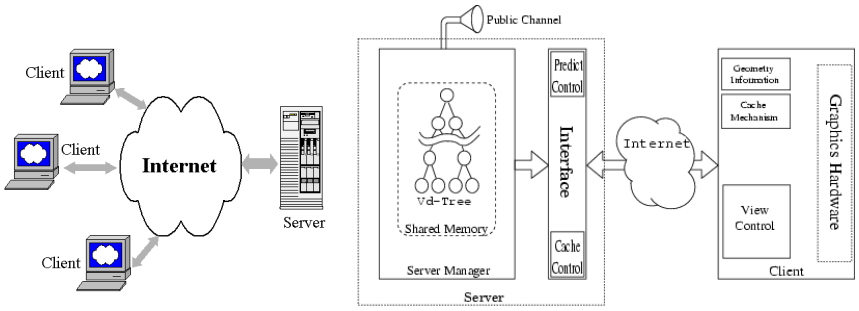
Abstract. In this paper we are presenting a novel approach that enables rendering large-shared datasets at interactive rates on a remote inexpensive workstations. Our algorithm is based on view-dependent rendering and client-server technologies. In our approach, *servers* host large datasets and manage the selection of the various levels of detail, while *clients* receive blocks of update operations which are used to generate the appropriate level of detail in an incremental fashion. We assume that servers are capable machines in term of storage capacity and computational power while clients are inexpensive workstation that have limited 3D rendering capabilities. To avoid network latency we have introduced two powerful mechanisms that cache the adapt operation blocks on the clients' side and predict the future view-parameters of clients based on their recent behavior history. Our approach dramatically reduces the amount of memory needed by each client and the entire computing system since the dataset is stored only once in the local memory of the server. In addition, it decreases the load on the network as a result of the incremental update contributed by view-dependent rendering.

1 Introduction

Recent advances in computer and communication technologies have dramatically increased the number of computers connected to the Internet, thus extending usage of the Internet as a media to exchange and share knowledge. The increased need for more information and the limited bandwidth of the shared network has been calling for smart application and algorithms to bridge the increasing gap between hardware capabilities and the large amount of data that resides in various servers all over the world. In addition, advances in three-dimensional shape acquisition, simulation, and design technologies have led to a generation of large polygonal models and virtual environments that are beyond the local storage of the Internet clients and cannot be downloaded in a reasonable amount of time.

The structure of the Internet as an enormous number of computers connect via global network has led to development of various distributed application. These applications include multimedia, three-dimensional graphics, and virtual environments that require a transmission of large data over short periods of time. Distributed three-dimensional games, virtual museum, and virtual walk-through are good examples of such applications. These distributed applications usually manage very large datasets that clients can not download either because they

are too large to fit into their local machines' memory or because these datasets are shared by other clients. On one hand, most of these applications are required to run at interactive or semi-interactive rates. On the other hand, the time it takes one machine to send a request to another machine and receive an answer may be too long for interactivity or even unpredicted. In the network community they refer to this time interval as *network latency*. Level-of-detail rendering could be used to reduce the size of the shipped three-dimensional datasets. El-Sana [5] has shown that view-dependent rendering dramatically reduces the traffic load over a local network by using a client-server technology. View-dependent rendering has been introduced to enable seamless and adaptive level-of-detail representations on real-time for polygonal datasets. The level-of-detail selection is based on view parameters such as view location and illumination. However, view-dependent rendering may not be enough to overcome the latency problem, which is the major obstacle in extending view-dependent rendering to work over the Internet. In addition, traditional view-dependent rendering schemes usually increase the size of the dataset, require the existence of the entire dataset in main memory, and do not enable different users to share the dataset currently in memory. To overcome the memory size drawback El-Sana and Chiang [2] have developed an external memory view-dependent rendering. However, their approach provides only a single-user solution and requires an additional level of preprocessing.



(a) Client-Server network (b) Client-Server View-dependent rendering

Fig. 1. Systems Configuration

In this paper we have developed a remote view-dependent rendering for large datasets over a wide area network. Our approach is based on the view-dependence tree data-structure [7] and a client-server technology. The view-dependence trees are stored in a remote and capable machine in terms of memory size and computation power. We shall refer to this machine as *server* which are accessed by different remote inexpensive machines that we will refer to as *clients*. To overcome the latency problem in wide area networks we have introduced two powerful mechanisms that provide caching of adapt operations and prediction of future view-parameters of the associated client. Our architecture enables inexpensive workstations to take advantage of view-dependent rendering to visualize

large remote 3D datasets. In addition, it reduces the memory needed by each client machine, improves the rendering frame rates, and dramatically reduces the communication load.

2 Previous Work

Our system is based on two main ideas: view-dependent rendering and a set of users who share the visualized dataset. Next, we will briefly discuss some of the previous work in these two fields.

2.1 View-Dependent Rendering

Recent advances in level-of-detail rendering have taken advantage of temporal coherence to adaptively refine or simplify the polygonal environment from one frame to the next in a view-dependent manner. In particular, adaptive levels of detail have been used in terrains by Gross *et al* [8] and Lindstrom *et al* [14]. Hoppe has developed the concept of progressive meshes [10] which are based upon two fundamental operators – edge collapse and its dual, the vertex split.

Merge trees have been introduced by Xia *et al* [22] as a data-structure built upon progressive meshes to enable real-time view-dependent rendering of an object. Hoppe [11] has developed a view-dependent simplification algorithm that works with progressive meshes. This algorithm proceeds to construct a vertex hierarchy over a progressive mesh in a top-down fashion by minimizing an energy function. Luebke and Erikson [15] define a *tight octree* over the vertices of the given model to generate hierarchical view-dependent simplifications. De Floriani *et al.* [4] have introduced multi-triangulation(MT). Decimation and refinement in MT are achieved through a set of local operators that affect fragments of the mesh. Klein *et al* [13] have developed an illumination-dependent refinement algorithm for multiresolution meshes. El-Sana *et al* [6] have developed Skip Strip: a data-structure that efficiently maintains triangle strips during view-dependent rendering.

View-dependence tree [7] is a compact multiresolution hierarchical data-structure that supports view-dependent rendering. In fact, for a given input dataset the view-dependence tree construction often leads to a forest (set of trees) since not all the nodes can be merged together to form one tree. View-dependence trees are constructed bottom-up by recursively applying the vertex-pair collapse operation (see figure 2). The order of the collapses is determined by a predefined simplification metric.

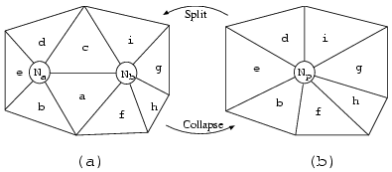


Fig. 2. Vertex-pair collapse and vertex split operations.

View-dependence trees are able to adapt to various levels of detail. Coarse details are associated with nodes that are close to the top of the tree (roots) and high details are associated with nodes that are close to the bottom of the tree (leaves). To be able to handle non-manifold cases, a more general scheme is used so that when a vertex split occurs more than two new adjacent triangles can be added. The reconstruction of a real-time adaptive mesh requires the determination of the list of vertices of this adaptive mesh and the list of triangles that connect these vertices. Following [7], we refer to these lists as the list of *active nodes* and the list of *active triangles*. At each frame the *active nodes* list is traversed to adapt to the appropriate level-of-detail representation of the given scene.

2.2 Distributed Interactive Systems

Experimental distributed systems have also been developed for real-time iteration in shared virtual environments such as Maverik [12], WAVE system[19], Shastra[1], NPSNET[16], and Cspary[18], and RING system[21]. Man and Cohen-Or [17] have developed a selective pixel transmission for navigating in remote virtual environments [17]. Hesina and Schmalstieg [9] have suggested a network architecture for remote rendering. Chim *et al* [3] have suggested algorithms on caching and prefetching of virtual objects in distributed virtual environments. Schmalstieg and Gervautz [20] have developed a demand-driven geometry transmission for distributed virtual environments.

Recently, El-Sana [5] has introduced a multi-user view-dependent rendering approach that allows multiple clients to visualize in a view-dependent manner large datasets that resides in a local server. Even though this approach shows a good improvement over external memory view-dependent rendering however, it has failed to handle the wide area network since it lack any prediction or caching mechanism.

3 Our Approach

Our approach provides a novel solution that enables Internet users to take advantage of view-dependent rendering. The technique of view-dependent rendering provides a mechanism to accelerate the rendering process as well as the transmission process.

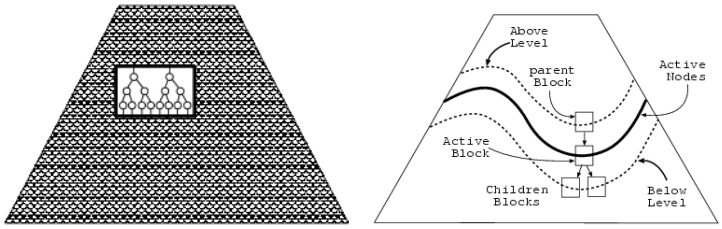
On one hand the Internet becomes "the place" of shared-large media including large 3D models. On the other hand, Internet users' machines usually lack the memory size and the rendering capability to interact with large dataset by themselves. In fact the vast majority of the Internet clients use low-end PC machines.

Our algorithm is based on a *view-dependent server* and a set of *view-dependent clients*. In our current implementation, we assume that the large dataset resides in a *view-dependent server* which has enough memory and computation power to store and handle the entire multiresolution representation. The *view-dependent clients* visualize in a view-dependent fashion the dataset that currently resides in the server through the Internet using inexpensive machines. Each client receives

only the level of detail required to visualize a dataset with respect to its current view-parameters and its machine computation capability. It is important to note that clients interact with the large dataset without downloading the entire dataset into their local machines' memory because of memory size, consistency, or security issues. In addition, our approach aims to reduce the load on the wide area network and avoid undesirable slowdown.

4 Adopting View-Dependence Trees

View-dependence tree [7] is a compact multiresolution hierarchical data-structure that supports view-dependent rendering. To use view-dependence trees in remote shared multi-users environments we have added new functionality and performed several changes.



(a) A View-dependence block (b) Above and below cache levels.

Fig. 3. View-Dependence Tree

We have divided the view-dependence trees into blocks based on parenthood and sibling relation in the view-dependence trees, as depicted in Figure 3(a). This blocking scheme is similar to the one developed by El-Sana and Change [2] to support external memory view-dependent rendering. Our blocking scheme improves the transmission time, make it possible to take advantage of compression, and paves the way for a successful caching scheme. We shall refer to each block as *view-dependence block* and to blocks that contains at least one active node as *active blocks*. Instead of using active-nodes list as in traditional view-dependence tree we use active-blocks as a list of pointer to the active blocks in the blocked-view-dependence trees.

To allow multiple users to access the same view-dependence trees concurrently we have separated the active-blocks list from the view-dependence trees. Recall that implicit dependencies [7] have been developed to prevent foldover at runtime by carefully assigning *id*-es to the nodes of the view-dependence tree. In addition, we have found that the implicit dependencies enable the separation of the active-nodes from the view-dependence tree. This separation enables multiple interfaces to have different active-blocks while sharing the same view-dependence tree.

The size of the transmitted data determines partially the transmission time, therefore we compacted the view-dependence tree by making the geometry of a parent node identical to one of its children's geometry. Such a scheme of the view-dependence trees stores geometry information such as vertex position, normal,

and color on the leaves of the tree. Each Internal node only keeps a pointer to the appropriate address where its geometry information is stored. These changes reduce the overhead of storing a view-dependence tree to only 2/3 times more than the 3D representation of the original mesh.

5 Our System Configuration

In this section we shall review the configuration of our system which is based on a view-dependent server and clients that access this server via the Internet. Next we shall overview our view-dependent server and client architecture.

5.1 View-Dependent Server

The view-dependent server is responsible for delivering three-dimensional objects in view-dependent fashion over the Internet. Since servers on the Internet should provide services for multiple clients, the view-dependent server should also send different appropriate level-of-detail to the seeking-service clients.

The server loads a view-dependence trees [7], then based on the view-parameters of each client, it delivers an appropriate level of detail for the loaded dataset. We shall refer to the main process that sets the public communication channel, initializes the shared memory arena, and loads the view-dependence tree into memory as the *server manager*. The *server manager* also listens, on a public channel, for requests for connection by new clients. For each calling client, the server creates a new thread, which is responsible for the interaction with the calling client. We will refer to this thread as an *interface*.

Each interface stores the characteristic parameters, maintains the active-blocks list, and handles the view-parameters changes of its associate client. On each change of the client's view-parameters its interface updates the active-blocks list to adapt to a better representation of the viewed scene. The interface is also responsible for packing and sending the view-dependence blocks with the proper additional data to its associate client.

The active-blocks list should match the active-nodes list that depends on the view parameters such as camera position, view direction, and illumination. Instead of watching for changes on the view parameters the interface receives only a *switch-up* and *switch-down* operations which reflect the changes on the view-parameters. On receiving the switch operations, the server updates its active-blocks list and transmits the added blocks to its associate client.

5.2 View-Dependents Clients

Clients should receive and maintain the minimum amount of information needed to maximize the visualize appearance of the inspected 3D dataset. In our implementation, clients maintain a small fraction of the view-dependence trees in a form of blocks; an active-nodes list as a list of pointers into nodes in the active nodes list; and active-triangles list which is sent to the graphics hardware at each frame. Practically, the active-nodes and triangles lists represent the appropriate level of detail of the visualized dataset with respect to current view parameters of the client.

Periodically, the client scans its active-nodes list and tries to converge to the best visual appearance of the currently visualized dataset by increasing or decreasing the resolution in various regions. For split operations that produce nodes that are in the lower quarter of the active-block, the client sends a *switch-down* message to its interface. And for merge operations that activate nodes in the top quarter of the active block the client sends *switch-down* message to its interface.

5.3 Messages and Session Phases

Our current implementation relies on the data-network layer hence, we assume that the connection is sequential and reliable. Nevertheless, our system maintains a timeout on the data channel to check for critical simple-detected errors such as communication breakdown.

Clients take advantage of the memory and computation power of the server to acquire the ability to visualize large datasets over the Internet. Because our system is distributed over a wide area network, it involves several different phases of interaction. In the setup phase the server loads the view-dependence trees and sets the public communication channel at which it listens for clients' requests for connection. On receiving a *request-for-connection* message the server creates an interface to handle the connection with the calling client. The interface, which has access permission to the shared memory, then establishes a private data channel shared with the calling client.

As soon as the client and the interface have established a connection the interface initializes its active-blocks, active-nodes, and active-triangles lists using the root-nodes list of the view-dependence trees. Then it sends the created lists to its associate client. On receiving the data, the client initializes its data-structures and buffers and both the client and its interface enter the *adapt dialog* phase. The client-server dialog allows the client to converge to the best possible visual appearance of the inspected scene with respect to its computation power, local memory, and rendering capabilities. During this phase the client updates its active nodes-list and sends switch messages based on its view parameters. Based on the received switch messages the interface sends blocks of view-dependence nodes.

To terminate the connection session the client sends to the server a *disconnect* message. On receiving this message, the associate interface releases its allocated resources, returns an acknowledge message, and terminate the session.

6 Overcoming Network Latency

Distributed applications over the Internet take advantage of sharing computation power and storage space, but they suffer from the disadvantage of the Internet latency. Latency of a network link is the time delay between a transmitted packet and its associated acknowledge. The disadvantage of latency becomes very severe for interactive applications.

We have chosen to tackle the problem of latency by caching some information on the client side and predicting the future view-parameters. We have also developed a new adapt-operation form that is used for performing a split and its dual

merge without sending additional data. Caching adapt operation on the client's local storage not only reduces the amount of transmitted data but it can also avoid transmission of a complete group of adapt operations. The prediction of future view-parameters allows the server to send more data to the client without waiting for its reply.

6.1 Caching the Adapt Operations

By using view-dependent rendering we manage to reduce the transmitted data by orders of magnitude, but to avoid the latency problem we should try to avoid transmitting data when possible. We have achieved that by caching previously sent blocks and reuse them instead of re-transmitting.

Our cache algorithm is based on the blocking scheme of the view-dependence tree we have introduced in section 4. The client caches every block it receives from the server as long as it has enough memory space (note that a cache could also be on an external media). We assume that clients keep at least the active-blocks list in its local memory. In order to be able to converge to the appropriate visual appearance of the visualized scene we cache one level of view-dependence blocks above the current active-blocks list and one level of view-dependence blocks below the active-blocks level as shown in Figure 3(b). We maintain these addition levels in a lazy manner by taking advantage of the cached blocks and by transmitting blocks that have a better chance to be used in the near future. In such a scheme the same block could be transmitted multiple times from a server to the client during one session. We could often avoid such cases by caching every block transmitted either in local memory or on an external media as long as we do not reach the cache buffers' limit.

We manage the cache by maintaining two level buffers: local and external memory. In addition, we maintain the booking of the received blocks in a hash table based on the unique *id* of each view-dependence block. Before asking for any block, the client checks its local-memory and external-memory buffers respectively to verify that the needed view-dependent block does not exist.

Since we have used limit cache buffers we had to come up with a scheme that manage removing items form the cache buffers and switch blocks between the local and external memory. With each block we store its level on the view-dependence trees, then we give priority to blocks according to the distance between their level and the current active level. We keep blocks with higher priority on the local memory and the ones with lower priority on the external memory.

6.2 Prediction Mechanism

Sine the user is the human factor in the real-time interaction, idle time is almost inevitable. In these intervals the client and server often do not exchange any data over the communication channel. We utilize these intervals by performing operations that reduce the need for data transmission in the future. We shall refer to the performed operations as *prediction*. The server's interface utilizes its idle time by predicting the future view-parameters and sends view-dependence blocks needed for the predicted view-parameters. For better prediction, the server's interface tracks the speed, velocity, and tangent of the client's motion. Then a

client chooses whether to cache the predicted adapt-operation and notify the server or discard them.

6.3 Client Server Synchronization

Assume that a server has an idle time and it sends future predictions to a client. Concurrently, the client's camera moves and sends switch messages to the server. Both the client and the server cannot determine which action took place first and each one thinks its action happened first. Choosing the incorrect order for the two events order may cause the to consider the arriving prediction for its new view-parameters. To avoid such a problem we synchronize the server and client to ensure the consistency of the view-parameters they both consider.

Clients consider predications that are either accurate or lead to transmission of uncached blocks. In our scheme, it is not easy for clients or servers to measure the "goodness" of prediction because the server is not aware of the exact view-parameters and both – client and server – do not keep a history of the switch messages. Practically, we do not need to determine the relation between the sent switch messages and the received view-dependence blocks. It is sufficient to determine where in the partial view-dependence tree in the client side the received block will fit. We use the unique *id* of each block to determine its position on the partially cached view-dependence trees. In such a scheme, a server may send blocks that is already cached in the client side. In such case, the client discards these blocks as soon as they arrive.

7 Implementation and Results

Similar to [5] the server allocates shared memory buffers to store the view-dependence tree. The server interfaces have a read-only access to the shared memory to maintain their active-blocks list. Active-blocks list keeps a record for each active block that includes a pointer to the actual node of the view-dependence tree. View-dependent rendering algorithms usually store in the local memory the multiresolution hierarchy on the rendering machine. Instead, our approach keeps a set of active blocks that include the selected level of detail in the rendering machine (the client machine). Such a technique usually provides more memory space which enables the selection of more detailed representation for large datasets. In addition, the display process and the adaptation process run completely in parallel since they live in two different machines.

We have also tested the performance of our unoptimized implementation on different hardware using various datasets and have received encouraging results. Next we are reporting some of the acquired results. For security reasons we were not able to conduct test outside the state of Israel, therefore all the reported results were carried out in two geographically distributed regions within the same country. In such a wide area network we can at the maximum, get either 2Megabit/second or 10 Megabit/second (depends on the client position) but in our experiments we have never exceeded 1 Megabit/second. For each experiment session, we periodically check the latency of the network.

One of the major advantages of view-dependent rendering is the ability to takes advantage of coherence between consecutive frames. Therefore, we have

Dataset	Original		Average/frame			
	Vertices	Triangles	Triangles (K)	Split/Merge (K)	Blocks Sent(KB)	Time(ms)
Dragon	151 K	300 K	72	1.6	0.5	240
Terrain	262 K	522 K	80	1.8	0.6	230
Palace	280 K	760 K	83	2.3	0.64	243
VR-City	612 K	1,250 K	91	2.6	0.68	260

Table 1. Run time over a sequence of frames for various datasets.

chosen to compare its performance over sequences of frames and not over one isolated frame. Hence, the reported results were obtained using sequences of frames.

In table 1 we report the average number of split/merge operations, data sent over the data channel, number of triangles at each frame, and the average time for a frame. These results were obtained using the configuration: the server runs on an SGI ORIGIN 200 and the client runs on an Pentium III machine with 128 MB. As can be seen from Table 1 we have achieved acceptable frame rate even for large datasets. In the same time the average amount of data transmitted at each frame is quite small.

Figures 4 and 5 show images generated by our system. Figure 4(a) show the original dragon model. Figure 4(b) shows in small window on the top-right corner the image seen by a user and the other window is the global view of the dataset. In Figure 5(a) we see the full resolution of a Terrain dataset and in Figure 5(b) we show a low resolution view of the same Terrain dataset.

8 Conclusion and Future Work

We have presented a novel approach for view-dependent rendering of large datasets on an inexpensive workstation. Our idea is based on a client-server architecture over a wide area network. We have adapted a compact-multiresolution-hierarchy data structure that supports view-dependent rendering that is known as *view-dependence trees*. The view-dependence trees reside in a remote Internet server that runs the view-dependent server. A client that may run on an inexpensive machine establishes a connection with a server and sends its view-parameters to the server which provides the client with the appropriate level-of-detail representation of the scene with respect to the sent view parameters.

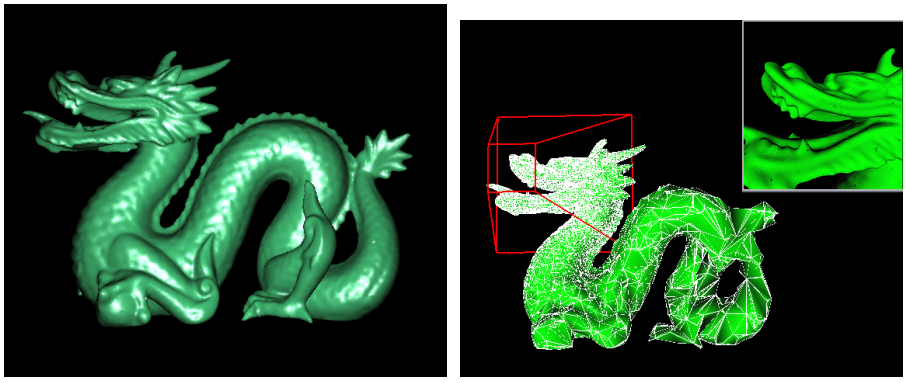
We see the scope for future work in designing virtual environment application that allows typical Internet clients to enjoy navigating through very large datasets at reasonable interactive rates. In addition, our approach could be used to promote remote touring, virtual museums, and remote shopping malls.

9 Acknowledgements

We would like to acknowledge the Weiler Family Fund, which helped in purchasing some of the equipment used in this project. We would like to thank the reviewers for their insightful comments. We also would like to thank 3DCafe and the Stanford Computer Graphics Laboratory for providing datasets.

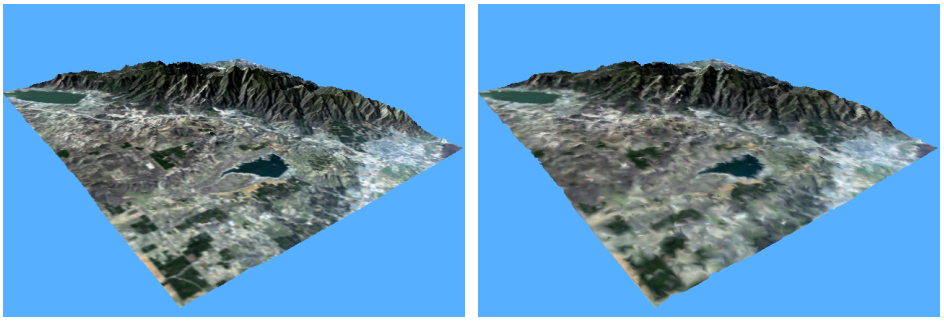
References

1. V. Anupam and C. Bajaj. Shastra: An architecture for development of collaborative applications. *Intern. Jour. of Intell. and Cooper. Info. Sys.*, 3(2):155–172, 1994.
2. J. El-Sana Y. Chiang. External memory view-dependent simplification. In *Computer Graphics Forum*, volume 19, 2000.
3. J. Chim, M. Green, R Lau, H. Leong, A. Si, and A. Si. On caching and prefetching of virtual objects in distributed virtual environments. In *ACM Multimedia*, pages 88–91, 1998.
4. L. De Floriani, P. Magillo, and E. Puppo. Efficient implementation of multi-triangulation. In *Proceedings Visualization '98*, pages 43–50, October 1998.
5. J. El-Sana. Multi-user view-dependent rendering. In *IEEE Visualization '2000 Proceedings*, pages 335–342, 2000.
6. J. El-Sana, E. Azanli, and A. Varshney. Skip strips: Maintaining triangle strips for view-dependent rendering. In *IEEE Visualization '99*, October 1999.
7. J. El-Sana and A. Varshney. Generalized view-dependent simplification. In *Computer Graphics Forum*, volume 18, pages C83–C94, 1999.
8. M. Gross, R. Gatti, and O. Staadt. Fast multiresolution surface meshing. In G. M. Nielson and D. Silver, editors, *IEEE Visualization '95*, pages 135–142, 1995.
9. G. Hesina and D. Schmalstieg. A network architecture for remote rendering. In *Inter. Workshop on Dist. Inter. Sim. and Real Time App.*, pages 88–91, 1998.
10. H. Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH '96*, pages 99 – 108. ACM SIGGRAPH, August 1996.
11. H. Hoppe. View-dependent refinement of progressive meshes. In *Proceedings of SIGGRAPH '97 (Los Angeles, CA)*, pages 189 – 197. ACM Press, August 1997.
12. R. Hubbold, D. Xiao, and S. Gibson. Maverik – the manchester virtual environment interface kernel, 1996.
13. R. Klein, A. Schilling, and W. Straßer. Illumination dependent refinement of multiresolution meshes. In *Computer Graphics Intl*, pages 680–687, June 1998.
14. P. Lindstrom, D. Koller, W. Ribarsky, L. Hughes, N. Faust, and G. Turner. Real-Time, continuous level of detail rendering of height fields. In *SIGGRAPH 96 Proceedings*, pages 109–118. ACM SIGGRAPH, 1996.
15. D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH '97*, pages 198 – 208, 1997.
16. M. Macedonia, D. Brutzmann, M. Zyda, D. Pratt, P. Barham, J. Falby, and J. Locke. Npsnet: A multi-player 3d virtual environment over the internet. In *Symposium on Interactive 3D Graphics*, pages 93–94, 1995.
17. Y. Mann and D. Cohen-Or. Selective pixel transmission for navigating in remote virtual environments. *EUROGRAPHICS 97*, 16(3):201–206, 1997.
18. A. Pang, C. Wittenbrink, and T. Goodman. Cspray: A collaborative scientific visualization application. In *Multimedia Computing and Networking*, 1995.
19. K. Rick. Making waves: On the design of architectures for low-end distributed virtual environments. In *IEEE Virtual Reality Intern. Symp.*, pages 443–449, 1993.
20. D. Schmalstieg and M. Gervautz. Demand-driven geometry transmission for distributed virtual environments. *EUROGRAPHICS 96*, 15(3):421–433, 1996.
21. F. Thomas. Ring: A clientserver system for multi-user virtual environments. In *Symposium on Interactive 3D Graphics*, pages 85–92. ACM Press, 1995.
22. J. Xia, J. El-Sana, and A. Varshney. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, pages 171 – 183, 1997.



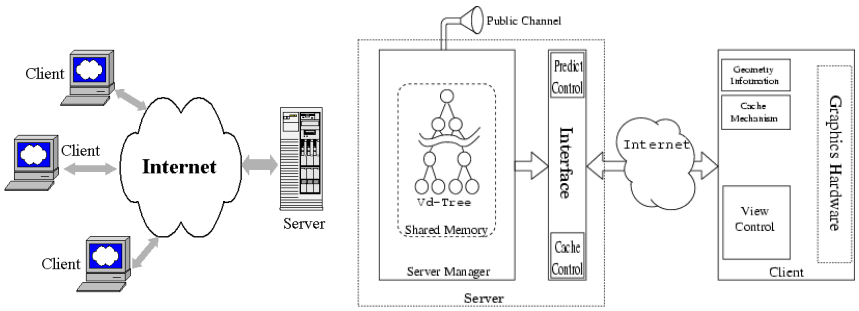
(a) Original Model:300K triangles (b) Selected View: 76K triangles

Fig. 4. View-dependent rendering of a Dragon model



(a) Original Mesh:522K triangles (b) Low resolution:32K triangles

Fig. 5. Uniform low resolution of a terrain model.



(a) Client-Server network (b) Client-Server View-dependent rendering

Fig. 1. Systems Configuration (a larger version of Figure 1).