Demand-Driven Geometry Transmission for Distributed Virtual Environments

Dieter Schmalstieg and Michael Gervautz

Institute of Computer Graphics, Vienna University of Technology, Austria
[schmalstieg|gervautz]@cg.tuwien.ac.at http://www.cg.tuwien.ac.at/research/vr/demand/

Abstract
We present a strategy for rendering in distributed virtual environments. A geometry database is maintained by a server, while users invoke individual clients to interact with the environment. Instead of downloading a complete copy of the geometry data, the data is distributed on demand, thus gaining significant savings in network bandwidth. Our strategy combines several techniques, including levels of detail, progressive refinement and graceful degradation to deliver the data "just in time" over the network to the rendering process. The method allows operate on a tight resource budget, which important if attempting to use low cost systems for virtual reality applications.

Keywords: virtual reality, distributed graphics, multi-user applications, levels of detail

1. Introduction

Networked multi-user virtual environments require that users share a common scene over a network [Mace95, Ghee94, Gerv95]. Examples include networked walkthroughs of large information spaces (buildings, databases, ultimately a 3-D Internet?) and interactive applications such as immersive cinema, networked games and computer supported cooperative work.

Given today's typical hardware setup with high-speed CPUs, fast system buses and comparatively slow network transmission, it is very reasonable to assume that the network is the most constrained resource of the whole system. We therefore have to develop new strategies for the visualization of distributed geometry databases, with the overall goal of minimizing bandwidth consumption on the network, and we can afford to devote substantial computational resources to the task.

Three distinct models for distributed graphics are in use today:

1. Image-based: the pixel-based images are sent over the net (e.g. digital TV, X pixmaps)
2. Immediate-mode drawing: used by drawing APIs, e.g. distributed GL [SGI93], PEX in immediate mode [Rost89] (some drawings APIs – like PEX – also allow display lists, that support reuse of transmitted data).
3. Geometry replication: A copy of the geometric database is stored locally for access by the rendering process. The database can either be available before application start (kept on local harddisk, such as seen in computer games like DOOM [ID94] and networked simulations such as NPSNET [Mace94]), or downloaded just before usage, as current VRML browsers do [Hard95].

Variations of geometry replications are now commonly used for networked VR applications. However, several severe problems constrain the usability of the method: Low network throughput and large database sizes are responsible for long download times. As the data has to be shipped to the user at some point, this problem is always present. Making the user wait for more than a couple of seconds destroys immersion and makes many interactive applications completely useless. Furthermore, extended waiting periods mean that a download process cannot be invoked frequently, so exploratory behavior of 3-D dataspaces becomes impossible.

Geometry replications also means that the whole scene must be stored locally. In order to display it, the data must fit into memory. This prohibits the exploration of large, continuous data spaces.

Commonly, multiple representations of one object with decreasing resolution (levels of detail, LODs) are used to reduce rendering cost for distant or otherwise less important objects. If the whole scene is stored locally, it must contain all levels of detail for all objects, even if the bulk of the high-resolution data is never actually used, for we cannot foresee what will be needed by the rendering process in the future.

From these arguments one can see that the transmission of geometry data for a scene as a whole has several drawbacks. We therefore aim at the development of a method for more fine-grained network transmission, that works incrementally. If the required data is delivered over the network „just in time" for the rendering process, both intractable setup times and elevated storage requirements can be significantly reduced. As only one level of detail of a given object can be displayed at any time, we can further improve performance by considering single levels of detail as the unit of transmission instead of complete objects.

2. Related work

While the usage of networked graphical applications is becoming more and more widespread, little work has been done on methods for efficient distribution of graphical data over a network. As pointed out in the introduction, most applications assume that the data is distributed prior to display in an off-line fashion.

A closely related problem that has been worked on is the management of large graphical databases that do not fit into memory and have to be paged in from disk. Access to the secondary storage is slow, fetching the data from the disk is the difficult part of the process. Funkhouser et al. [Funk92] have reported on such a system for interactive walkthrough of buildings, using potentially visible sets for data reduction. Terrain data for vehicle simulation is managed in a similar manner [Falb93].

Aside from the raw geometry, complex distributed simulations require the distribution of ongoing events and user actions (e.g. movements). As this communication quickly exceeds network capacity even for moderately sized network groups, several approached have been developed to reduce the need for simulation-related communication in distributed virtual environments. Proposed solution include explicit registration of interest in particular message types such as in DVS [Ghee94], utilization of IP-multicast [Mace95], dead reckoning [Mace94], a history-based protocol [Sing95], and the use of visibility information [Funk95].

Furthermore, compression of geometry data can be used to save bandwidth, such as shown for X [Dans94], polygonal data [Deer95], or a combination of geometry and image data [Levo95].

3. Data management

We attempt to optimize transmission of geometry data in a client-server system [Figure 1]. The server stores the data for a virtual environment, composed of objects that are arranged spatially. This database can become very large, potentially infinite [Brec95]. The client allows the user to display and navigate this VE database. For this purpose, the client needs only those data items, that are actually being displayed. Consequently, there is no need to transmit the whole database from the server and store it at the client. It is sufficient if the client has the data for those objects available that are contained in its *area of interest* (AOI). We have decided to use spherical AOIs rather than the viewport itself, because rapid head movement as possible with head-mounted displays cannot change the set of objects in the AOI so rapidly.

Thus by restricting the geometry transmission to the data that is actually required for display, we can gain significant savings in network bandwidth and local memory requirements, allowing to handle more complex, more interesting data sets. Note that the visible data set is dependent on the viewpoint of the observer, which changes over time, and on it the visible data set. If the system is able to deliver the data just in time for display, there are no visible differences over a non-distributed virtual environment that has all data available locally.

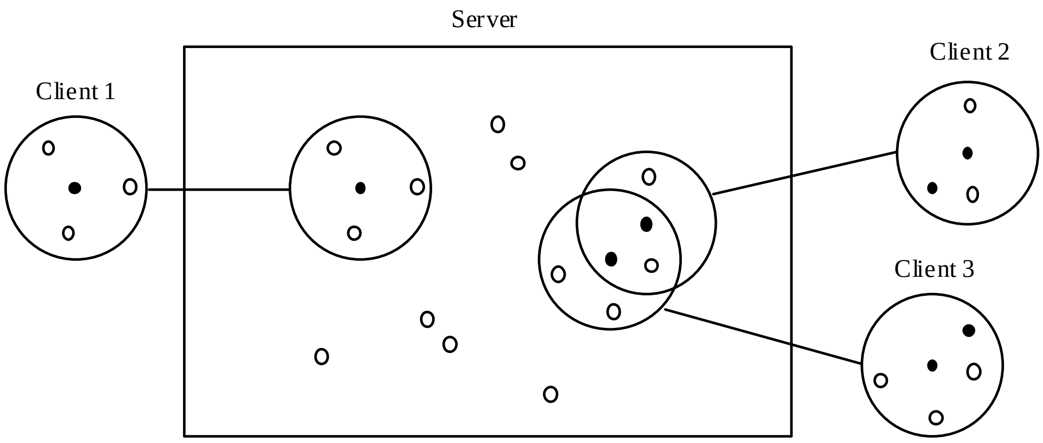


Figure 1. *A distributed geometry database: A server stores geographically dispersed objects (small white circles). Each client's view is limited to an AOI (large circles). If AOIs overlap, clients can see each other (small black circles)*

The set of objects contained in the AOI changes as either objects or the client itself move. To keep the area of interest up to date regarding the objects contained within, the client can request data from the server. The selection of the data is up to the client, so various strategies for data management are possible. To perform the task of requesting data, the client has to know about the objects that are contained within its AOI. Again, we do not want to require the client to know about all objects in the environment, for tracking the objects in the AOI is sufficient.

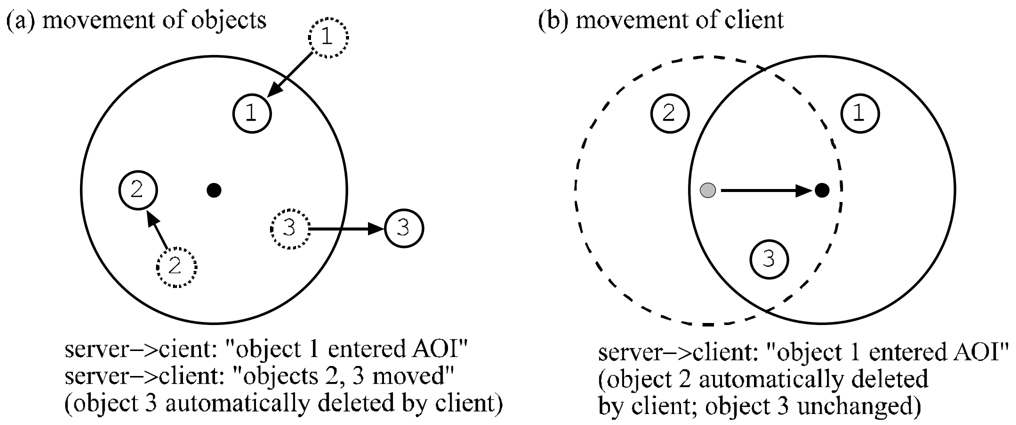

(a) movement of objects
1
1
2
2
3
3
server–>cient: "object 1 entered AOI"
server–>client: "objects 2, 3 moved"
(object 3 automatically deleted by client)
(b) movement of client
2
1
3
server–>client: "object 1 entered AOI"
(object 2 automatically deleted
by client; object 3 unchanged)

Figure 2. *The server updates the client on activity within its AOI, if objects (a) or the client (b) move. To reduce network traffic, only those messages are sent that cannot be deduced by the client independently.*

The server monitors the AOI for every connected client and periodically sends updates regarding the activity of contained objects as appropriate. If an object moves into the AOI or the user moves the AOI near an object, the object has to be newly introduced to the client (send object info), that cannot know about this object otherwise. From then on, it is sufficient to send updates if the position of the objects changes. Figure 2 shows some cases. This scheme requires the server to remember the set of object infos have been transmitted to the client. The task is not too complicated for the server because the set changes incrementally.

The distributed nature of the rendering process should be transparent to the user. In particular, the image presented to the user should be smoothly animated and updated at a sufficiently high and constant frame rate. This goal is defeated by variations in scene complexity (many objects, complex objects) and in network throughput. Rendering complexity is managed by a rendering engine capable of displaying LODs. With the help of the LOD datastructure, a strategy was developed that also compensates for the shortcomings of network transmission.

The fundamental idea is to consider these LODs instead of complete object as the unit of network transmission. Objects can be displayed even if not all of their (LOD-)data is available. Only those LODs that are needed for rendering must be available locally at the client. As the user moves his viewpoint, or simulated objects (e.g. vehicles) change their position, the selection of LODs changes.

Transmission of even a coarse LOD takes time, and this delay must be compensated for, or the data will not be available when needed. Therefore *prefetching* is adopted. When the LOD selection algorithm decides to switch to a certain LOD, the prefetching module requests the next finer level.

In cases when prefetching fails, an available coarser LOD can be displayed instead (*graceful degradation*), trading a continued constant frame rate for a decrease in image fidelity. Such a degraded image can be progressively improved in phases of reduced activity by downloading the missing LODs (*progressive refinement*).

The time-critical part of acquiring the data is most important, but storage requirements may not be neglected. To keep the size of the client's database *cache* (i.e. memory available for geometry data) from overflow, we must dispose unneeded objects. If an object leaves the AOI, it is deleted and memory is freed.

For a particular client, the representations of other users (avatars) appear just like ordinary objects. If a client changes position, it has to transmit the new position to the server, which not only has to know the client's current position to monitor the client's AOI, but also has to update the other clients on the new position.

A more detailed discussion of the client's strategy is given in chapter 5.

4. Geometry Data Structure

In order to make use of the hardware support for interactive rendering, we must model the virtual environment object database as a collection of polygonal datasets. VR applications have two additional important requirements:

- Interactive rendering of large scenes in real time requires that the objects of which the scene is composed are modeled with levels of detail (LOD). At runtime, the fidelity of each object can be chosen independently from the available LODs, so that the polygon budget for a frame is not exceeded [Funk93]. In the next section it will be shown that the LODs can also be used to optimize network usage.
- A flat datastructure (e.g., a simple array of triangles) is not sufficient. For efficient manipulation and high-level animation of a dynamic environments, a directed acyclic graph (DAG) is well suited. Its hierarchical structure allows flexible manipulation of the data as needed by VR applications [Stra92].

Most level-of-detail capable renderers use a single LOD per object: the LOD selection node is placed at the root of the geometry graph. However, allowing multiple levels of detail yields additional flexibility.

We divide this datastructure into two parts: the subgraphs below a level-of-detail node (LODs) and the trunk (i.e. the graph from the root down to and including all LOD nodes, but not below). Figure 3 shows an example. For network transmission of such geometry graphs, the separation allows to transmit the trunk before the LODs, and then "mount" the LODs on the trunk. Note that not all LODs have to be available in order to display the object. The trunk plus a single LOD is sufficient to display the object, although not in every desired fidelity. Note that in most cases, the trunk will only be a very small data structure composed predominantly of "organizing" nodes such as groups and transformations, while the LODs contain the bulk of the data (polygons, color, ...).

The database of the server is a flat collection of objects. Each object is composed of a geometry representation (trunk plus LODs), and a matrix defining the object's position and orientation. In the next section we will show how a "view" (subset) of this database is kept at the client.

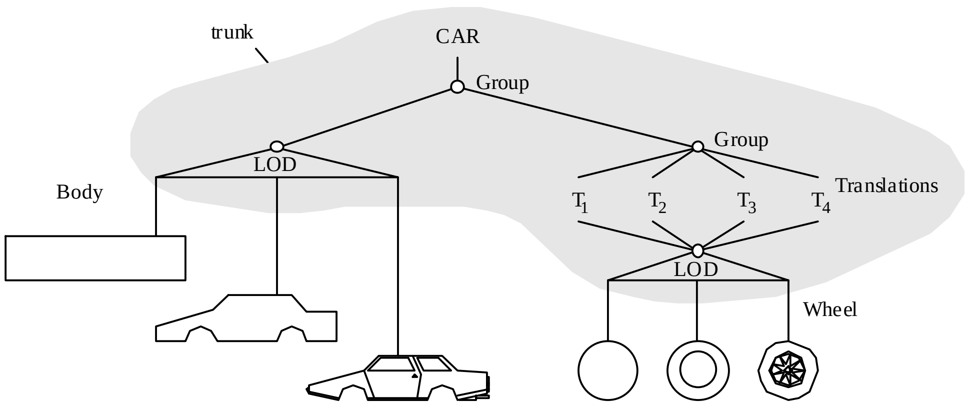


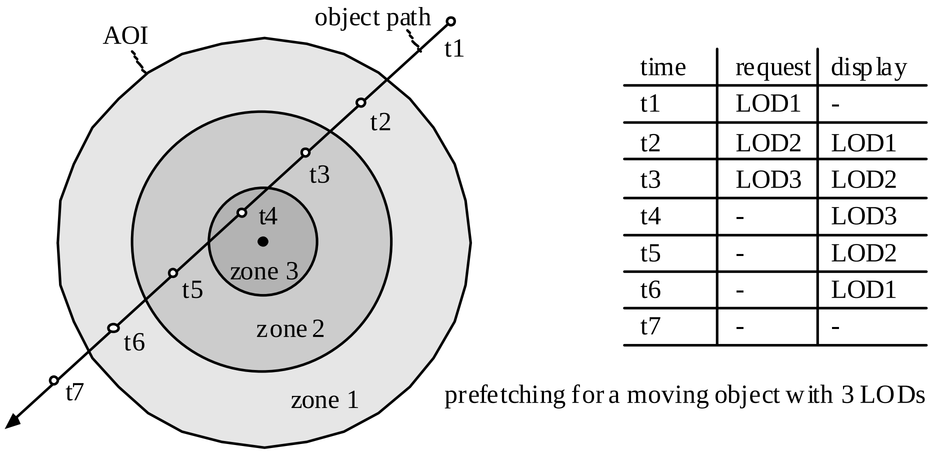
Figure 3. *Car modeled with levels of detail. The model is divided in LOD geometry and a "trunk" (shown in grey). Most of the data is hidden in the LODs, while the trunk is generally only a control structure.*

5. Strategy of the client

In this section, we give details on the strategy for management of geometry data by the client.

5.1 Prefetching

To compensate the delay introduced by the network transmission, we use prefetching to anticipate the requirements of the renderer. The already available level of detail algorithm can be used for this purpose, only with a different parameter - a finer LOD is already selected "earlier" than needed for rendering (when still relatively far away). Objects that are approached will be displayed with increasing resolution, so the next finer LOD is a good guess for prefetching [Figure 4].



time	request	display
t1	LOD1	-
t2	LOD2	LOD1
t3	LOD3	LOD2
t4	-	LOD3
t5	-	LOD2
t6	-	LOD1
t7	-	-

Figure 4. *A prefetching strategy. As an object moves through the AOI (see arrow), it traverses multiple zones indicating which LOD of the object is displayed. The next finer LOD is always requested one zone in advance to compensate for network delay.*

This scheme is not unfailable: as the strategy assumes high frame to frame coherence in the data set being displayed, violations of this assumption lead to the failure of the prefetching efforts. If either the user moves too fast, or objects move too fast, the data needed for display at the appropriate resolution cannot be made available in time.

Even worse are applications that allow sudden changes in the visible set: objects that appear out of nothing or change their representation into a more complex shape. The same applies to abrupt changes of the user’s viewport and position, including “teleport” functions and the initialization phase when no data at all has been transmitted to the client. Furthermore, if the network itself works unexpectedly slow, it may simply not allow to transmit as much data as estimated.

5.2 Graceful degradation

If timely delivery of the required LOD data cannot be achieved for one of the reasons mentioned above, the client can use a coarser version of the object instead. For immersive applications, displaying a degraded image (even bounding boxes may suffice in some cases!) is far better than stalling display update. A coarser LOD of the object under consideration should usually be available. The most frequently used metric is distance between object and observer or projected size of the object on the screen. Both metrics change slowly for typical applications, so that the display gradually switches from coarser to finer LODs, generally giving the server enough time to transmit the next LOD. If time is not sufficient to do so, the next coarser LOD can be displayed a little longer, even though it may not have the adequate resolution according to the heuristics.

Note that graceful degradation can fail if there is a total network overload or breakdown, that makes it impossible to send even the coarsest LODs in time, but this should rarely happen.

5.3 Progressive refinement

In some situations the client may be able to store and display a scene at high resolution, but network is so slow that the data cannot be transmitted in time. To deal with this problem, we make use of the fact that as the observer is approaching an object, the objects representation is updated with consecutively finer LODs. We can therefore issue requests so that the LODs of an object are always transmitted in order from coarser to finer. (If – optimally – the data from a coarser LOD can at least partially be reused in a finer LOD, the total amount of data for an object is decreased.)

If a degraded version of an object is being displayed after some coherence-destroying activity, and the situation improves (user standing still, not a lot of movement), the time can be used to complement the missing data and gradually switch to a higher resolution of LODs for the objects in question.

However, if high activity is continuing, it may be more necessary to get coarse approximations of new objects or objects with only very coarse representation first. By executing data transmission out of order (using a priority queue), the more important data items can get expedited transmission. Priority is computed from the difference between available and desired LOD. If this value is the same for multiple pending transmission, the one with the lower level of detail is selected.

5.4 Cache size and replacement

For a conventional rendering LOD algorithm, the upper bound on the number of primitives that are selected is given by the maximum number of primitives that the rendering system can process. If we combine this method with networking, another restriction has to be taken into consideration: the available memory works like a cache. Naturally, its size is limited, so the active data set must exceed neither the renderer’s processing capacity nor the cache size. As typical workstation have ample memory, the graphics processing power is usually the more restrictive factor. However, for video game consoles and set top boxes this may be different. With a different setup, it may be necessary to modify the strategy so that the memory is filled with more objects, but in coarser representation.

The cache replacement strategy is governed by the LOD algorithm. It determines what items of the data set are currently needed. One LOD of a particular object is a unit data item of the cache algorithm. There are two possible units that can be selected for deletion from the cache: One LOD (of one object), or one object (with all LODs of that object that are present). While replacement of individual LODs allows a more fine-grained data management, handling objects as a whole is simpler and therefore faster. We opted for discarding objects completely if they are not needed anymore, mainly for simplicity. An alternative is to discard single LODs in the opposite order of acquisition (finest resolution – consuming most memory – first), so to keep memory footprint as small as possible.

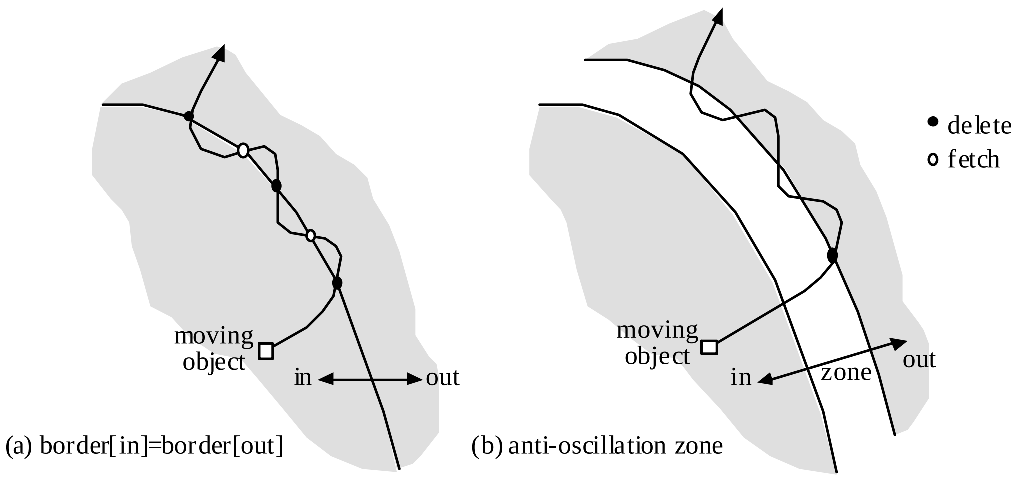


Figure 5. *Suppressing oscillation. Fetching and discarding object data can lead to unwanted oscillating activity (a) that can be suppressed by a "safety" zone (b)*

Discarding objects when they leave the AOI may lead to unwanted oscillating effects if the object continues to move near the border of the AOI. As a countermeasure, the AOI region has to be chosen sufficiently large so that geometry is not immediately discarded when the object becomes invisible. Furthermore, the AOI used for determining leaving objects is slightly larger than for entering objects, so to increase the distance an object must travel to "come back" once it has left [Figure 5].

6. Protocol design

The design of the network protocol not only determines performance of the network module, but also the capabilities and semantics of the application. Our aim was to design an application-layer protocol that interoperates with the VR application. The best way to discuss the resulting protocol is to examine the protocol units that the protocol is composed of. An overview is given in table 1. We can separate these into three groups: connection management, avatar control, and geometry transmission [Schm96]. We can further distinguish whether the message is sent by the client or the server ("origin" column in table 1).

7. Implementation

We have implemented a prototype of the system as outlined in the previous chapters to obtain experimental data on how the algorithm and protocol perform. The implementation was done on SGI workstations using C++ and OpenInventor. A discussion on some of the design decisions that have been made is given in this section.

7.1 Graph traversal

When a request for a particular LOD reaches the server, the referenced object's geometry graph is traversed and packaged [Figure 6]: The traversal is performed in preorder/depth first (multiple referenced nodes in a DAG are visited only once). When a LOD node is reached, the required child is determined. This child subgraph is linearized and added to a buffer.

Message	Or.	Parameters	Comments
Connection management			
init_connection	c→s	client_id pos, orientation AOI_data avatar_data connect_info	Building up the connection: client registers with a unique client ID; states his initial position/orientation and size of AOI; uploads user's geometric description (avatar). Other connection-management informations are not of here.
kill_connection	c→s	client_id	disconnect
Avatar control			
update_client_pos	c→s	pos, orientation	client tells its new position to server, so server can compute set of obj. in client´s AOI
update_object_pos	s→c	obj_id pos, orientation	server updates client on new position of a moving (animated) object to allow correct selection of LODs
Geometry management			
request_geometry	c→s	obj_id lod_no	client decides a specific LOD is needed, and requests it by specifying object ID and LOD
transmit_geometry	s→c	obj_id lod_no data	server answers request of client for data and sends geometry data, identified by object ID and LOD
transmit_object_info	s→c	new_obj_id data	server updates client on object set contained in client's AOI (without request!) by informing on a new object and associated info (# of LODs, size, ...)
kill_object	s→c	obj_id	delete obj. that has disappeared, e.g.destroyed
update_AOI	c→s	AOI_data	change AOI data (size), e.g.if client overloaded

Table 1. *Protocol units*

Upon completion of the traversal, the buffer contains a list of items that together make up the requested data. The content of the buffer is transmitted to the client, where it is unpacked. The client traverses the trunk of the graph (remember that the trunk is always transmitted before the first LOD), and unpacks one item of the buffer for every LOD node encountered. As the order of traversal is well-defined, the data is automatically put in the right place.

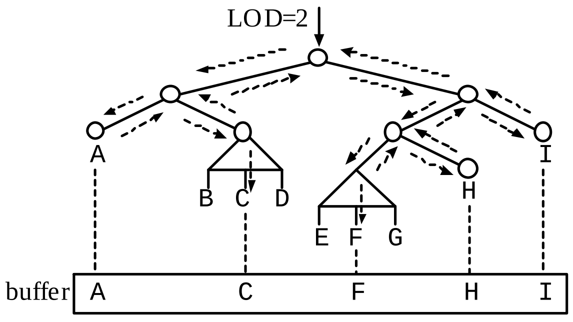


Figure 6. *Traversal and packaging of a LOD request. The graph is traversed, but for every LOD node only one child is selected. The resulting data is collected and packaged for network transmission*

7.2 Level of detail node

The datastructure for a level of detail node must provide some measure on the subjective quality of the individual LODs to select the right LOD.

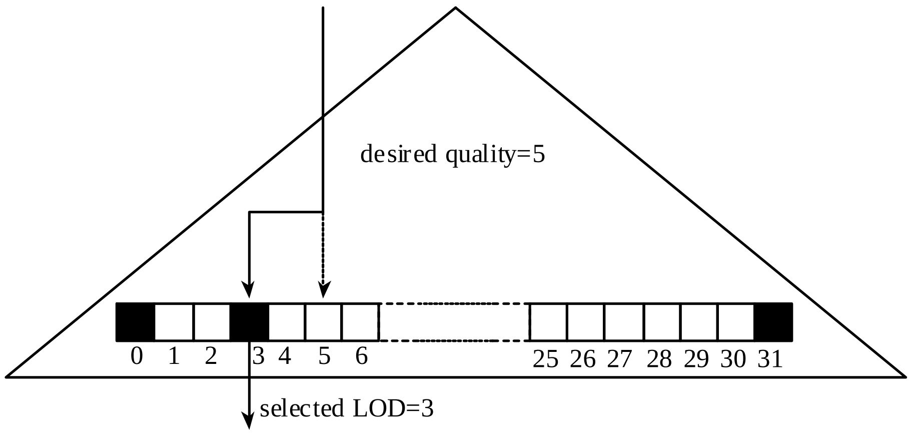


Figure 7. *LOD node traversal. If a LOD with desired quality cannot be found, the next coarser one is used.*

A simple solution is to use the index of the LOD node’s children correspond to quality. However, this requires that the quality “distance” between successive LODs is at least roughly the same.

We project the quality measure onto a finite and fixed length scale of (for example) 32 possible grades, which can be used as indices for accessing each of the children. Thus not all 32 entries have to accommodate a geometry subgraph. For example, Figure 7 shows a LOD with children 0, 3, and 31 only.

The graph must be traversed for rendering and transmission. The desired quality measure is passed to the traversal algorithm as a parameter. When the traversal reaches a LOD node, this measure is compared with the indices of the available children, and if the desired child is not available, the next available child with lower quality is chosen [Figure 7]. We are able to handle a model containing multiple LOD nodes with different numbers of children. Thus a reasonable combination of multiple LOD nodes can be found independently of the model.

7.3 Software architecture

Our software architecture is divided into server and client programs [Figure 8] The server runs two important software modules: the connection manager and the geometry database manager. Currently we run both managers within a single UNIX process, but as their relationship is well-defined, performance may be enhanced by building a decoupled system configuration in the style of MR [Shaw93], possibly running on a multi-processor machine.

The connection manager is responsible for dealing with the network. We use UDP sockets for handling the network connections between client and server, which are bi-directional. As the name suggest, this module implements connection management as shown in table 1.

The database manager stores a collection of objects forming the virtual environment. Objects can either be controlled by a simulation, or they represent a user’s avatar, and are controlled directly by the user. Our current simulation engine is rudimentary, capable of moving objects along predefined paths to create a dynamic environment, but a more powerful simulation system is under development [Schm95]. The database manager also keeps track of the client’s AOI and sends updates on objects as necessary.

The client has a network and database manager, very similar to the one of the server. However, the actions performed by the client are quite different. Before every frame, a LOD oracle is invoked to find an appropriate level of detail for rendering. The oracle is also used to find out if any LOD must be requested from the server.

For rendering, each object graph stored by the database manager is traversed and appropriate rendering actions are performed for every node. The case where rendering traverses a LOD node, but the prefetching has failed to provide the desired graph data, is automatically handled by the traversal: the next coarser available level is used instead. The users actions, in particular viewpoint changes, are passed to the database manager, that reacts appropriately.

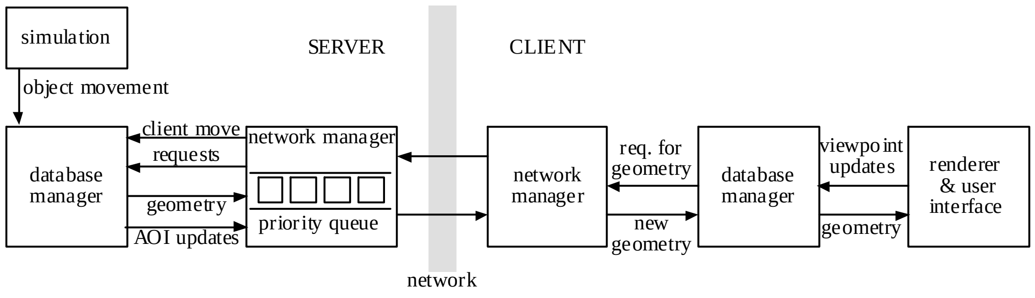


Figure 8. *Software architecture of client and server programs*

8. Results

We constructed a virtual environment for test purposes by randomly placing objects on a plane. The objects were procedurally constructed to contain multiple levels of detail with progressively less primitives. Distant objects appear smaller because of perspective projection, so we decided to reduce the number of primitives per LOD corresponding to a 1/x function where x is the LOD number. The objects have 6 LODs. LOD selection was done based on the distance, where the radius of the AOI was divided into uniform intervals. For our tests, we used a prerecorded walkthrough sequence of 500 frames.

Comparing the total size of the virtual environment database to the transmitted data is not fair, since the client's area of interest contains a roughly constant number of objects (if we assume a uniform distribution of objects), but the complete virtual environment can be made arbitrarily large. We were rather interested in the comparison of demand-driven transmission with LOD management and without (i.e. objects are always transmitted completely, including all LODs). Table 2 shows the network load of the first 100 frames of the walkthrough. The solid curve shows the network load with and the dashed line without LOD management. Note that downloading complete objects gives stronger load variations (when an object enters the AOI, all its data has to be transmitted at once), while transmitting individual LODs tends to better distribute the effort. Furthermore, high-resolution LODs of objects that never come close to the observer are not downloaded, which explains why the load for transmission with LOD management is generally lower than without.

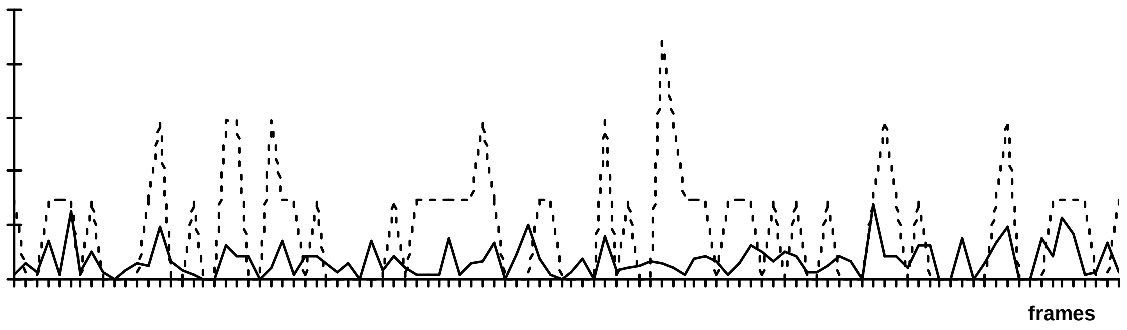


Figure 9. *Network load of demand-driven transmission with (solid) and without (dashed) LOD management*

We also wanted to see how the number of LODs would influence the performance. Therefore we measured the total number of transmitted bytes for the walkthrough sequence without LOD management, and also with LOD management for an object with 3-20 LODs. The size of the AOI was kept fixed. (Note that objects with more than 10 LODs are never used in practice.)

# LODs	3	4	6	8	10	20
load fraction(%)	53.8	48.3	39.6	36.8	34.4	27.6

Table 2. *Savings of network load of demand-driven transmission with LOD management over always transmitting complete objects. Results show remaining percentage of transmitted data using LOD management.*

We computed the total transmitted data with LOD management as a fraction of the number without LOD management. The results show that more LODs reduce the fraction of transmission, but the improvement drops significantly as more LODs are added. For a realistic setting with 6-8 LODs for an object, the total reduction in network traffic can be up to a factor of 3 compared to no LOD management [Table 2].

9. Conclusions and future work

We have presented a strategy for managing network transmission of geometry data in distributed virtual environments. In our client-server based approach, the client request geometry from the server based on individual levels of detail instead of downloading complete objects or even entire scenes. The approach is based on a limited area of interest that must be kept up to date. Load variations are handled by prefetching, graceful degradation and progressive refinement. Results show a significantly improved network performance.

In any distributed system, the server must be powerful, or it will become a bottleneck. Future work will thus include to parallelize the server and execute it on multiple CPUs. Furthermore, the virtual environment can be divided over a network of servers, each of which is responsible for a region, thereby reducing the amount of objects maintained by a single server. In that case, the demand-driven geometry transmission protocol is only part of a more elaborate network communication model on which we have reported elsewhere [Schm96].

Large savings in network bandwidth can be gained by compressing the geometry data before sending it through the network. Currently, the data is not only uncompressed, but also kept in a format that was designed for readability rather than compactness (OpenInventor file format). A future version of our software will include a compression/decompression facility that uses an adapted version of Deering´s proposal [Deer95].

10. Acknowledgements

The authors would like to thank Gerd Hesina, Stefan Mantler and Peter Wonka for their work on the implementation, and for contributing many ideas to this project. Zsolt Szalávari helped with the preparation of the figures. This research was partially sponsored by an equipment grant from the TNF of the Vienna University of Technology.

11. References

[Brec95] E. Brechner (ed.): Interactive Walkthrough of Large Geometric Databases. SIGGRAPH’95 Course, No. 32 (1995)

[Dans94] J. Danskin: Higher Bandwidth X. Proceedings ACM Multimedia’94, pp. 89-96 (1994)

[Deer93] M. Deering: Data Complexity for Virtual Reality: Where do all the Triangles Go?. Proceedings of VRAIS’93, pp. 357-363 (1993)

[Deer95] M. Deering: Geometry Compression. Proceedings of SIGGRAPH’95, pp. 13-20 (1995)

[Falb93] J. Falby, M. Zyda, D. Pratt, L. Mackey: NPSNET: Hierarchical data structures for realtime 3-dimensional visual simulation. Computers & Graphics, Vol. 17, No. 1, pp. 65 (1993)

[Funk92] T. Funkhouser, C. Sequin, S. Teller: Management of Large Amounts of Data in Interactive Building Walkthroughs. SIGGRAPH Symposium on Interactive 3D Graphics, pp. 11-20 (1992)

[Funk93] T. A. Funkhouser, C. H. Sequin: Adaptive Display Algorithm for Interactive Frame Rates During Visualisation of Complex Virtual Environments. Proc. of SIGGRAPH'93, pp. 247-254 (1993)

[Funk95] T. Funkhouser: RING - A Client-Server System for Multi-User Virtual Environments. SIGGRAPH Symposium on Interactive 3D Graphics, pp. 85-92 (1995)

[Gerv95] M. Gervautz, D. Schmalstieg: Computer Animation and Visualization - Current Status and Trends. Proceedings of EUROSIM'95 (1995)

[Ghee94] S. Ghee, J. Naughton-Green: Programming Virtual Worlds. SIGGRAPH'94 Course No.17 (1994)

[Hard95] J. Hardenberg, G. Bell, M. Pesce: VRML: Using 3D to surf the Web. SIGGRAPH'95 Course, No. 12 (1995)

[ID94] ID Software: DOOM. Computer game (1994)

[Levo95] M. Levoy: Polygon-Assisted JPEG and MPEG Compression of Synthetic Images. Proceedings of SIGGRAPH'95, pp. 21-25 (1995)

[Mace94] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, S. Zeswitz: NPSNET: A Network Software Architecture for Large-Scale VEs. Presence, Vol. 3, No. 4, pp. 265-287 (1994)

[Mace95] M. Macedonia, M. Zyda, D. Pratt, D. Brutzman, P. Barham: Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments. Proc. of VRAIS'95 (1995)

[Rost89] R. Rost, J. Friedberg, P. Nishimoto: PEX: A Network-Transparent 3D Graphics System. Computer Graphics & Applications, Vol. 9, No. 4, pp. 14-26 (1989)

[Schm95] D. Schmalstieg, M. Gervautz: Towards a Virtual Environment for Interactive World Building. Proc. of the GI Workshop on Modeling, Virtual Worlds, Distributed Graphics. Bonn (Nov. 1995)

[Schm96] D. Schmalstieg, M. Gervautz, P. Stieglecker: Optimizing Communication in Distributed Virtual Environments by Specialized Protocols. To appear in: 3rd Eurographics Workshop on Virtual Environments, Monte Carlo, Feb. 1996

[SGI93] SGI: OpenGL – Programming Guide, The Official Guide to Learning OpenGL, Release 1. Addison Wesley, ISBN 0-201-63274-8 (1993)

[Shaw93] C. Shaw, M. Green, J. Liang, Y. Sun: Decoupled simulation in virtual reality with the MR toolkit. ACM Transactions on Information Systems, Vol. 11, No. 3, pp. 287-317 (1993)

[Sing95] S. Singhal, D. Cheriton: Exploiting Position History for Efficient Remote Rendering in Networked Virtual Reality. Presence, Vol. 4, No. 2, pp. 169-194 (1995)

[Stra92] P. Strauss, R. Carey: An Object Oriented 3D Graphics Toolkit. Proceedings of SIGGRAPH'92, No. 2, pp. 341 (1992)