**RC22161 (W0108-055) August 31, 2001**
**Computer Science**

# IBM Research Report

## An architecture for QoS Data Replication in Network Virtual Environments

**George Popescu, Christopher F. Codella**
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

**Research Division**
**Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich**

# An Architecture for QoS Data Replication in Network Virtual Environments

George V. Popescu and Christopher F. Codella
IBM T.J. Watson Research Center

## 1. Introduction

Network Virtual Environment architectures have been successfully deployed over local area networks (see [10] for a review of networked Virtual Environments). With the advance of graphics capabilities on client platforms (PCs, game consoles) and availability of broadband Internet access (DSL, Cable Modem, etc.) networked Virtual Environment (netVE) technology is ready for deployment over the Internet. However the Internet is a much more heterogeneous environment, requiring new architectural solutions. A central problem in netVE design is geometry data replication. Replicating the whole database is not needed and may be prohibitive in case of very large Virtual Worlds.

Therefore the replication model proposed here assume the data is downloaded on demand. This replication model can be used in several network applications such as online 3D communities, network games, and 3D web browsing. The model is in contrast with currently available 3D web plugins which download the whole scene content before allowing user interaction. Browsing 3D content on the Internet uses VRML [12] files and 3D-enabled browsing clients. User navigates the Web space by pointing to URL's and waiting for content to download. This model, designed for browsing HTML pages, is not suited for interactive 3D content. Navigation in 3D environments involves controlling the viewpoint. New content is typically explored through continuous movement of the viewpoint. For large Virtual Worlds, data visibility is also limited in order to achieve interactive frame rates. While navigating these Worlds, the browser will have to download the 3D content just in time for the user to view it. This can be achieved by tracking the viewpoint and prefetching geometry data in Client's cache.

## 2. Related work

Reducing Web latencies motivated the early research on predictive prefetching. Padmanaban and Mogul [8] showed that significant reduction of average access time of WWW documents can be achieved at the expense of an increase in network traffic by a similar fraction. In [6] prefetching was used to reduce latencies over a low-bandwith connection between client and a caching proxy. A significant latency reduction was achieved without generating additional Internet traffic in the backbone.

Prefetching was used in distributed virtual environments for geometry streaming [9] [4]. Multiresolution geometries are stored at the server and send to the client just in time for being rendered. This allows better resource utilization for very large geometry databases with complex models. Chim et al. [4] also proposes a multiresolution caching mechanism optimized for Virtual Environment navigation.

A general framework for task-specific asset prioritization in distributed virtual environments has been proposed in [3]. The framework performs resource management based on quality, importance and cost (QUICK) ratings of each component of the Virtual Environment. The approach attaches to the scenegraph a set of QUICK annotations used by resource managers to decide how to optimize the scene at run-time. The framework has been implemented in a prototype display and a distributed cache management system.

These previous prefetch techniques can be characterized as “best effort”, as they do not attempt to provide Quality of Service for the prefetching mechanism. The prefetch policies use display optimization techniques, ignoring network related aspects. QUICK method consider the cost of data download among other display optimization variables, but does not provide an adaptive QoS method for data replication. Performance measures of previous methods were obtained on LANs, which exhibit a stable behavior.

## 3. Client Server Interaction for netVE navigation and download

Client server interaction in network VEs is influenced by user navigation. Two popular navigation models are continous navigation and portal navigation. In continous naviagation the viewpoint moves incrementally between successive frames, controlled by an input device or by an animation sequence. Portal-based navigation consist of switching to another 3D universe. The C/S system uses a controller to manage data download based on viewpoint motion. A controller for VE navigation processes user input differently than a Web browser. The controller model of a HTML browser can be described by the following sequence:

*<User action>* → *<Client request information from HTTP Server >* → *<Client receive content>*

A 3D navigation controller uses two asynchronous loops - one for user input, the other for data download control:

*<User action>* → *<Change viewpoint>* &

*<Check data visibility conditions>* → *<New data request>* → *<Get new data in Client cache>*

The first loop updates the viewpoint position according to user input. The second one performs data download transparent to the user. There are two choices for controlling the data download:

1) *Pull model*: data flow is controlled by the client. This requires that the client has acquired a description of the Virtual Environment before the navigation process starts. A practical implementation of this model uses a scenegraph with "NULL" geometry nodes. These nodes contain data indicating the URL of the geometry node and the size of the geometry file. The client application switches the NULL nodes with geometries as required by the visibility mechanism provided that they are available in the client cache.
2) *Push model*: the server keeps track of client viewpoint movement through frequent updates and pushes new content into the client disk cache as needed. This model does not require any additional annotations in the scenegraph, but requires a constant client/server data flow and additional server CPU load for each client.

Client controlled data replication is well suited for browsing 3D Web content while dynamic netVEs (like multi-player 3D games) could be implemented more efficiently with server control. We analize here the client controlled data replication.

VE data is modeled using a scenegraph; the viewpoint motion is controled with input devices or by animation. The scenegraph contains several nodes which describes 3D objects geometries, transformation, behaviours, animations, etc. The client (viewer) import the scenegraph description and control data replication based on the interest model of client viewer.
The 3D web navigation model uses a three-level data accessibility model including: *visible data*, *area of interest (AOI) data* and *unknown data*. Limiting the visibility of data is necessary for very large Virtual Environments in order to achieve interactive frame rates. The shape of the visible region was chosen circular for simplicity. In general it can be an arbitrary shape, obtained at runtime through intersection of several geometries (e.g. intersecting room plan with a circular visibility region). Visible data changes continuously as the viewpoint navigates the Virtual Environment. As a consequence new data is constantly loaded in the main memory at run-time from the local cache. Every time a scenegraph node become visible a new thread is dispatched to load the geometry data which replaces the null geometry node. The node is now labeled *visible data* for the time it stays in the memory. When the node moves out of the visible region it is labeled *AOI data*, its geometry is discarded from memory and the null geometry is switched back in.

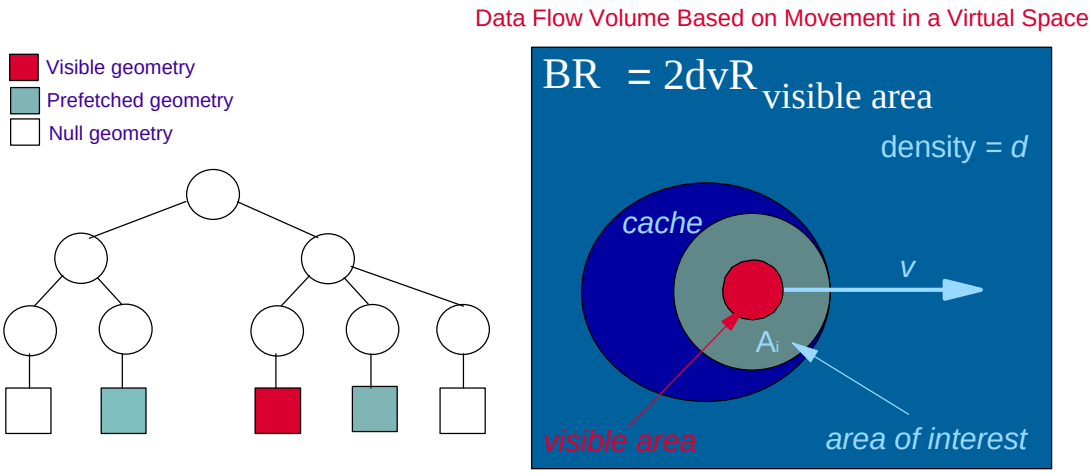


Figure 1: Geometry prefetching in netVEs

Area of interest (AOI) specifies a surveillance region used by the client for prefetch planning. AOI has a circular shape concentric with the visible data region. A collision detection mechanism signals when a null geometry node enters the area of interest. The list of null geometry nodes in the area of interest is continuously evaluated by the prefetch module. When the prefetch condition is triggered, a new thread is dispatched to download the geometry file into the cache. If the geometry is requested by a load thread before it is available in the cache, the request is saved in a waiting queue. The load module is notified when the geometry becomes available.

The rest of the scenegraph, containing null geometries, is classified as unknown data. The data accessibility model is presented in Fig. 1. We also show the cached area in order to contrast it with AOI area. Part of data in the AOI is in the local cache if the prefetch manager downloaded it. Please note that some geometry data might still be in the cache while not in the area of interest.

An important measure is the average data density for a virtual world. This is calculated as the size of the data contained in the virtual world over the extent of the virtual world. The download bit rate when viewpoint move in the VE can be expressed as:

***BR = 2 *data density * speed * Radius(visible area)*** (eq 1)

The bit rate is the main QoS parameter for the data download. This is the minimum bit rate that the client will accept during the QoS negotiation protocol.

## 4. Scene graph extension to support partial data replication

In order to support QoS prefetch, the scenegraph is extended with a data size field. This additional field is added to any online node, such as geometry, texture, etc. The requires either extension of the scenegraph modeling language (VRML) to accommodate the data size labeling, or querying at the run-time to obtain the data size. Labeling is different for static and dynamic content. Static content can be pre labeled and doesn't add run-time load. Dynamic content requires run-time data labeling to be done at the server. For each file containing dynamic objects the server has to check and insert the correct data label.

The virtual world parameters necesarry for QoS download are: center (Cx, Cy, Cz) extents (Ex, Ey, Ez) and data weigth (W[kB]) (see Fig. 2). Given the extended scenegraph, these parameters are obtained using a graph traversal algorithm which compute the CEW parameters for each node in the scenegraph. The CEW of root are used to calculate the average data density virtual world: *data_density = W(root) / Ex(root)*Ez(root)* - for the planar viewpoint motion. A volumetric data density can be calculated for 3D viewpoint motion: *data_density = W(root) / Ex(root)*Ez(root)* Ey(root).* The CEW of each node in the hierarchy can be similarly used to determine local average densities. The data density affect the AOI size as shown in section 5.3.

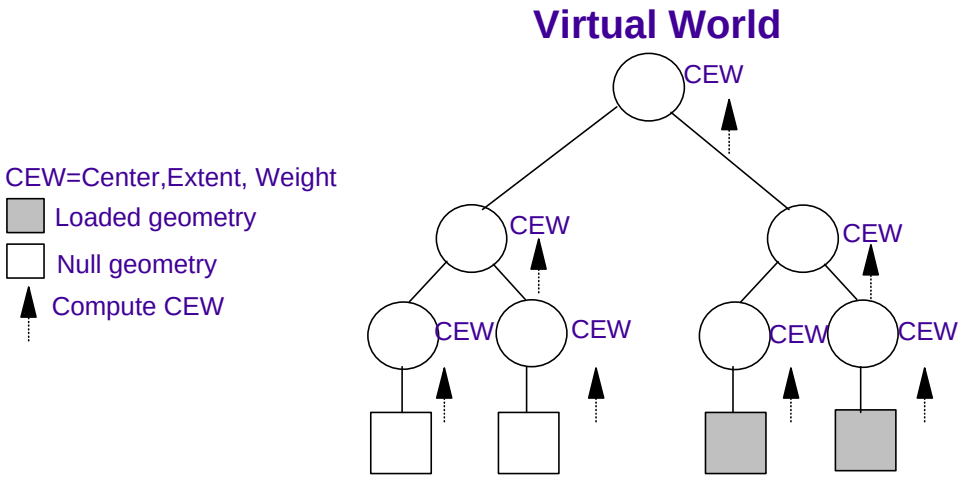


Figure 2: Scenegraph extensions

The scenegraph of large virtual worlds is tipically segmented in smaller local worlds. Such local worlds are represented as portals to be explored at run-time. Assuming a run-time replication model, the data behind these portals can be similarly prefetch just-in-time. This can

be achieved by labeling the hyperlink to a new of the new virtual world with the CEW (center, extent and data weigth) parameters (Fig. 3). These parameters need to be calculated at run time using the graph traversal algorithm described above.

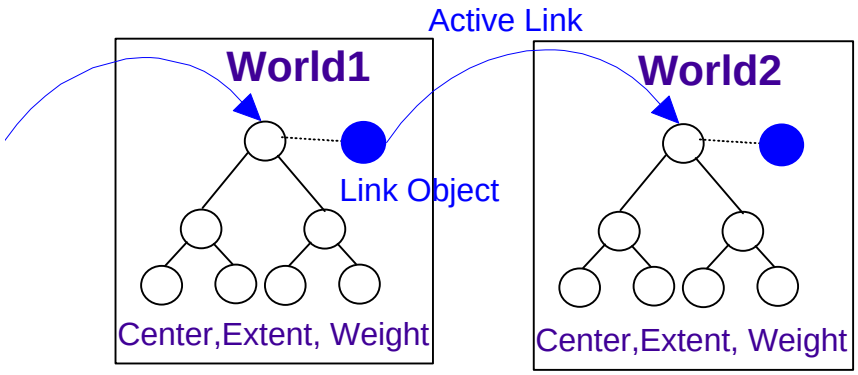


Figure 3: Prefetching segmented virtual worlds

## 5 . QoS data replication

### 5.1 The QoS protocol

QoS data replication involve client/server negotiation of download bandwidth and adaptation to varying network condition. The negotiation of download bandwidth is intended to allow the server to control the network resources allocated per client. Server congestion is avoided by limiting the number of connections accepted. The client uses negotiation to select the server that satisfy its quality of service requirements. By using negotiation, application requirements are shaped according to available network resources; as the clients and servers are aware of these resources, they adjust the communication parameters according to availability.

The **QoS tuple** has the format **<Bw_download, T_response, Download Class>**. The first parameter represent the average download bandwidth available for the connection. The second term is the maximum allowable sever response time for a client request. The last term indicate the class of the client (e.g. a Web 3D portal has a lower download class than a subscription-based online network game).

The protocol start with a preliminary step when the client is probing the network resources. This include latency measurements and bandwidth availability on the routing path to the server. A ping-like utility is used to determine latencies. Bandwidth estimation measurements are performed next (TCP congestion window, rtt; network level: loss rate, ...). After collecting these data, the QoS-tuple corresponding to client application profile is adjusted with network measurements. Then the client starts the QoS negotiation process. The protocol consist of three steps:

1. Request: QoS client requirements are announced to the server. The requested bit rate should be larger than the bit rate calculated with (eq. 1).

2. Adjustment: The server adjust the QoS-tuple based on its QoS serving policy and available resources. The parameters can be accepted without modification, be degraded or rejected. The QoS-tuple is returned to the Client.

3. Confirmation: The client accept the adjusted parameters and set the download model parameters.

**5.2 Prediction and Prefetching:**

The prefetching model uses the client negotiated download bandwidth for just-in-time data replication. The prefetch manager keeps a table with time penalties, **T_download**, (calculated for the estimated Client's network bandwidth) for all NULL nodes in the AOI. The size of the AOI is selected so that the largest virtual object can be downloaded just in time. The time when the geometry becomes visible – **T_visible** - is also estimated. Whenever the time to get to a node is less than the download penalty time plus buffer time (to account for variation in connection bandwidth), a download thread is dispatched to get the geometry into the local cache:

***T_download*** *(obj_i) = Data_Size(obj_i) / Bw_download ;*
***T_buffer*** *= T_response+T_parameter;*
***T_visible*** *= Distance (visible boundary, C(obj_i)) / Estimated_speed ;*
***T_visible*** *<* ***T_download*** *+* ***T_buffer*** *→ start downloading the node*
*Radius (AOI)=max_speed*(T_buffer+T_download(maxsize obj))*

Estimated speed parameter depends on the model used for viewpoint motion. A simple choice is using a maximum speed value. A predictor with a good behaviour for rapid changes in viewpoint dynamics is: $speed\_est[i] = \alpha * speed[i-1] + (1-\alpha) * speed\_est[i-1]$ with $\alpha$=3/4. In cae the virtual object is moving, the estimated relative speed is used to calculate ***T_visible***. The buffering time depend on the response time of the server and the ratio between the average bit rate calculated for the virtual world and the negotiated download bandwidth.

Figure 4 illustrate the just-in-time download mechanism. The prefetch is optimized in order to reduce the wasted bandwidth and increase the saved requests. The saved requests will be particularly impacted for small caches. Minimizing wasted resources is especially important for the server, as the prefetch mechanism will require significant sever bandwidth and CPU load.

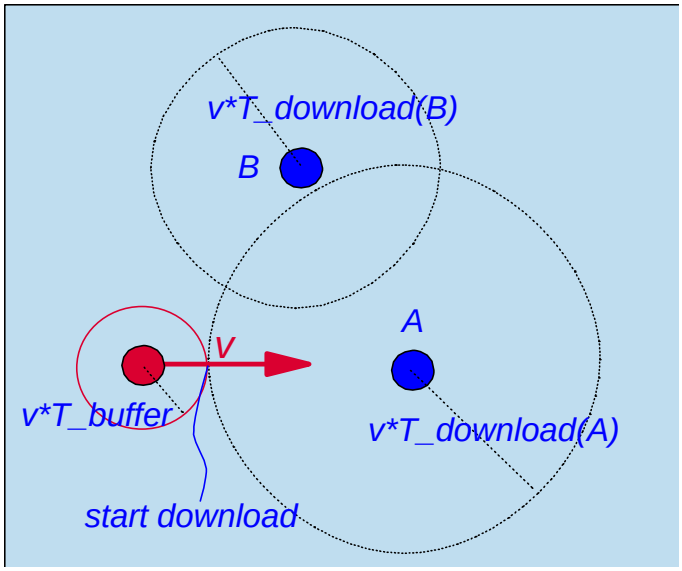


Figure 4: Prediction for just-in-time data download

### 5.3 QoS Adaptation

QoS adaptation is needed to perform the just-in-time download under varying network conditions. The adaptation involve monitoring the bandwidth available for the data replication application. Network statistic is collected periodically by the application. The statistic is obtained through application level (download thread) measurements. The instant bandwidth is calculated with the formula:

$Bw\_i = (\frac{1}{time\_w}) \Sigma(Bytes(i))$ *for i=1 ... # of threads*, where *time_w* is the time window used for bandwidth calculation (in the order of tens of seconds). The formula for conection bandwidth estimation is:

$Bw\_est[i] = \alpha Bw\_i + (1-\alpha)Bw\_est[i-1]$ [Brian D. Noble, Li Li, Atul Prakash, 1999, umich.edu/~llzz/papers/hotos99.ps] where *Bw_est* is the estimated bandwidth, *Bw_i* is the measured (instant) bandwidth.

The estimated bandwith is plugged in the prediction formula of section 5.2. Note that some regions of the virtual world may have higher densities and require a bit rate higher than the one estimated in (eq. 1). In case the required bit rate is higher than the one available for the connection, degradation of the content quality (e.g. lossy geometry compression, lowering level of detail) is required.

## 6. Cache management

Cache management for continuous 3D navigation is driven by viewpoint motion. The goal is to keep the nodes closer to the viewpoint in the cache as long as possible. That requires continuous evaluation of distances from the viewpoint of the nodes in the cache. Using a standard Last Recently Used algorithm will not give good performance unless viewpoint motion is unidirectional. Since viewpoint motion includes some randomness, a different cache management policy needs to be considered. The policy proposed here uses the same time labels as in the prefetching mechanism. Each node in the cache is labeled with:

*T_replace = T_visible - T_download.*

According to this formula, the nodes with the biggest score are the ones further away from the viewpoint and which incur a smaller penalty to be downloaded again. Therefore the expiry policy replaces the nodes in the decreasing order of their *T_replace* label. These labels are evaluated every time a new data request requires more space than is available in the cache. When removed from the cache, such a node is again labeled *unknown data* and requires a new prefetch if becomes of interest.

## 7. Implementation and Results

The prototype implementation uses Java (Client/Server interaction) and Java3D [11] (3D Client). The client ran on an Pentium III 866 MHz PC with a GEForce2 graphics card. The HTTP server was placed in a WAN; the average bandwidth of the connection was estimated at 600 Kbps. The VE consists of 1000 static 3D objects [1] arranged randomly on a terrain (a textured surface). The terrain map is 2000 by 2000 units. Each object has a distinct URL. Data size of geometry objects varies between 181 KB and 1085 KB; the corresponding size of the

geometry database is between 181 MB and 1 GB. The local cache size is fixed at 10 MB. Doom style keyboard navigation is used for viewpoint motion. The viewpoint moves with 1 unit (normal speed) or 2 units (fast speed) per frame. A screen capture of the Virtual Environment is shown in Fig. 5.

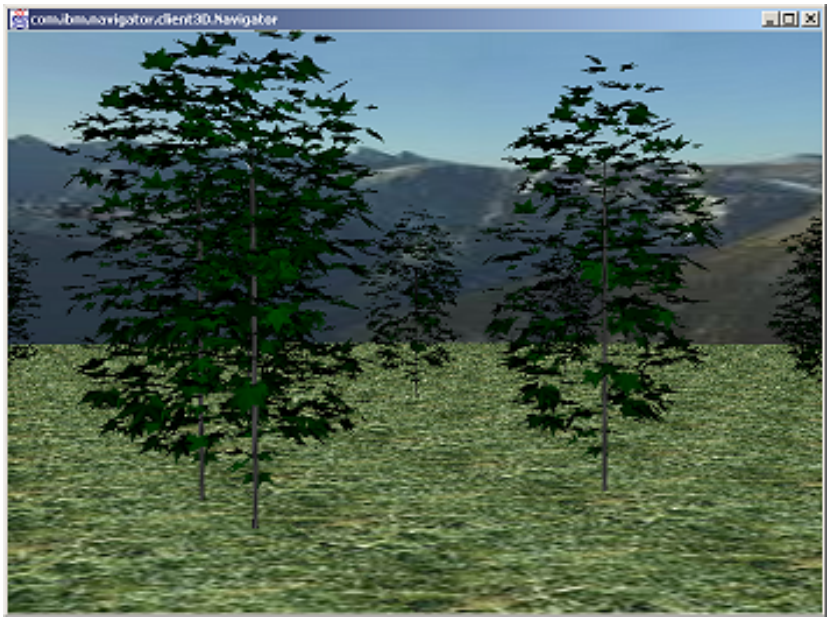

Figure 5: Screen capture of 3D browser

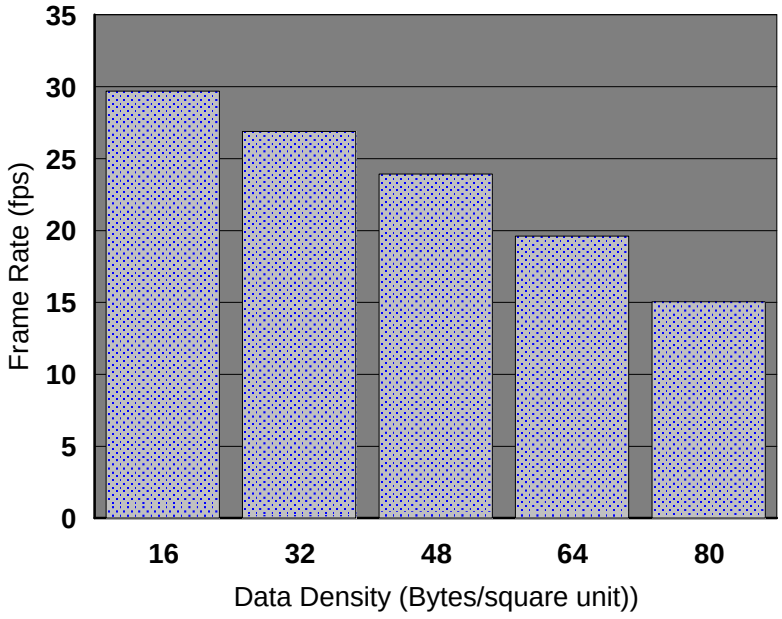


Figure 6: Frame rate vs. data density

In the initialization stage, the objects within visual range are loaded in the memory and the objects in the area of interest are downloaded in the local cache. Once the viewpoint starts moving, the browser functions as described in section 5, downloading data from the HTTP server. Interactive frame rates were achieved for geometry data densities between 16 to 80 B/(square unit) as shown in Fig. 6. This experiment didn't include QoS adaptation or viewpoint speed estimation.

We designed a separate set of experiments to measure caching and prefetching performance. The performance metrics used in the experiments are:

1) latency reduction for application startup - time from the client application request a new 3D environment until the user can control the viewpoint ;

2) saved requests - the number of times a client request hits the local cache as a percentage of total number of user requests; this is an equivalent cache hit ratio; however, since we are assuming the data is prefetched before being requested by the application, the saved requests refer only to the already cached objects.

3) wasted bandwidth - the sum of bytes prefetched but not used at the client.

The prefetching method described in section 5.2 was compared with a simple AOI-based method (with no estimate of download time). Preliminary results show a large decrease of wasted bandwidth when "just-in-time" prefetch is used. This set of experiments is expected to be completed shortly.

**References**


[1] 3D Café. 2000. *3D models database*. Online at: http://www.3dcafe.org/

[2] Bowman D. (1999). *Interaction Techniques for Common Tasks in Immersive Virtual Environments.* PhD Thesis, Georgia Tech University.

[3] M. Capps. 2000. "The QUICK Framework for Task-Specific Asset Prioritization in Distributed Virtual Environments." *Proceedings of IEEE VR2000*, pp. 143-150.

[4] J. Chim, M. Green, R. Lau, H. V. Leong, and A. Si. "On caching and prefetching of virtual objects in distributed Virtual Environments." *In Proceedings of ACM Multimedia '98,* pp. 171-180, 1998, ACM Press.

[5] C. Greenhalgh and S. Benford. "MASSIVE: a Distributed Virtual Reality System Incorporating Spatial Trading," *Proc. 15th IEEE International Conference on Distributed Computing Systems (ICDCS'95),* Vancouver, Canada, May 30-June 2, 1995, IEEE Computer Society, pp 27-34.

[6] L. Fan, P. Cao, W. Lin, Q. Jacobson. "Web prefetching between low-bandwidth clients and proxies: potential and performance." *Proc. of ACM SIGMETRICS Conference*, pp. 178-187, 1999.

[7] M. Macedonia, M. Zyda, D. Pratt, P. Barham, and S. Zeswitz. "NPSNET: A Network Software Architecture For Large Scale Virtual Environments." In *Presence: Teleoperators and Virtual Environments*, 3(4):265-287, 1994.

[8] V. N. Padmanabhan and J. C. Mogul. "Using Predictive prefetching to improve World-Wide Web Latency." *ACM Computer Communication Review*, pp.22-36, v.27, n3, July 1996.

[9] D. Schmalstieg and M. Gervautz. "Demand-Driven Geometry Transmission for Distributed Virtual Environments." In *Proceedings of Eurographics '96*, pp. 421-432, 1996.

[10] S. Singhal, M. Zyda. *Networked Virtual Environments*, New York, ACM Press, Addison-Wesley, 1999.
[11] H. Sowizral, K. Rushforth, M. Deering. *The Java 3D API Specification.* Addison-Wesley, 1997.
[12] A. Walsh, M. Bourges-Sevenier. *Core Web3D*. Prentice-Hall. 2001.