Optimizing Communication in Distributed Virtual Environments by Specialized Protocols

Dieter Schmalstieg, Michael Gervautz, Peter Stieglecker
Institute of Computer Graphics, Vienna University of Technology
schmalstieg|gervautz|stieglecker@cg.tuwien.ac.at - http://www.cg.tuwien.ac.at/

Abstract

A successful implementation of a distributed virtual environment should be built on a strong network layer. The network as a constrained resource must be used efficiently, and also the structure of communication should allow to select those features that are needed without having to support multiple complicated protocols. Therefore we designed a set of specialized protocols tailored for dedicated tasks of communication in virtual environments. The combination of these protocols yields the desired communication functions without introducing much overhead. In particular, it is possible for participants with varying degrees of capability to use the virtual environment to communicate with each other.

1. Introduction

The restrictions that are most hard to overcome in distributed virtual environments are the need for consistency, and constrained network bandwidth. It is because of these restrictions that virtual environments either focus on rich interaction [Carl93, Snow94, Bric94, Brol95] or on large-scale distribution [Mace94], but not both.

Our goal is to develop a distributed virtual environment in which users can participate and contribute content. We favor a client-server based approach, that lets users run client software and connect to servers over a network [Schm95]. Such a scheme will separate the participants from the providers of the VE infrastructure. Users can use the VE with inexpensive desktop machines, and do not have to be responsible for setting up the VE infrastructure. This is important if a large, loosely coupled user community is to be supported. In particular, the simulation of the environment is independent of the presence of users - the VE exists even if no user is currently present. The server provides consistency, concurrency control and persistence, that are otherwise hard to accomplish. Scalability is achieved by localizing the simulation: every server is responsible for a region in the virtual universe, and maintains a loosely coupled connection to its neighbors. Inside the server's region, the influence of objects is also localized to a relatively small area of interest.

Simulation kernel. Our virtual environment consists of actors. An object-oriented hierarchy allows diverse actor types representing different levels of "intelligence". Pure *static actors* serve as "decoration" of the scene (walls, trees etc.) and have no built-in behavior. Such actors can be extended to include key-framed, deterministic animation (e.g. a clock with moving hands). A more sophisticated class of actors exhibits *behavior* that is formulated in an interpreted scripting language. Behaviors are triggered by messages that are exchanged between actors. As messages are exchanged, the simulation progresses. Modifications to the internal state of the actors, in particular their visual representation, are reflected in the virtual environment. The most powerful actor type is controlled by an *external application*.

Multiple levels of participation. The separation of server and clients allows multiple levels of participation, dependent on the type of client that is used:

An *observer* can only explore the VE, but cannot interact with any objects. The client may limit its display to a pure walk-through (only considering static geometry), or also request the dynamic changes of the visual artifacts, that are created by the simulation. Nothing an observer does affects the VE.

A *participant* may introduce his own avatar and use it to fully interact with the simulation, its autonomous agents and other avatars. His actions modify the dynamic state of the VE; they are distributed to other clients and stored persistently in the database.

An *author* may contribute his own content to the virtual environment. He may create and destroy objects, and even more importantly create new types of objects with their own behavior, that can continue to exist as autonomous agents without the user's aid.

This hierarchy of participation is similar to the one found in conventional MUDs (text-based multi-user games) that can in many respect be seen as VEs without a visual component.

To maximize flexibility, we also allow *applications* to function as clients. While most behaviors of objects in the VE can be formulated using our scripting engine, the most interesting behaviors are too complex or computationally demanding for scripts. Therefore an interface is provided to allow external applications to "remote control" objects in the VE. Because the only restriction for an application client is that it complies to the network protocols we use, any application can be made to cooperate with the VE.

Overview of the protocols. The requirements we have for our system are diverse and demanding. A single unified framework for communication that incorporates every form of information exchange into a single protocol is not sufficient. Instead, we define a set of highly specialized protocols that complement each other and can be tailored for the task. This approach allows us to exploit domain-specific properties for efficiency, assign network access to the protocols for an optimal bandwidth usage, and combine protocols as needed by the different levels of participation.

Some protocols (such as the one that builds a network connection to the VE) must obviously be spoken by all types of clients, while others can be used or neglected at the client's disposal. They will, however, determine the client's capabilities. Table 1 gives an overview of the protocols we use and their scope:

Protocol	**Responsible for**
Connection management:	login, logout, protocol negotiation, user migration
Avatar control:	navigating the user's representation through the VE
Geometry:	transmitting geometric description of the objects in the VE to the client
Animation:	transmitting changes in the visual database of the VE to the client
Simulation:	exchanging messages concerning the ongoing simulation among actors and between actor and client
Interaction:	letting the user interact with objects in the VE and other users
Authoring:	managing modifications to the structure of the VE, such as object instantiation and deletion, creation of new object types and configuration of external applications

Table 1: Overview and characterization of the protocol framework

2. Connection management protocol

Connection management provides fundamental networking functions on which the distributed environment is built. It handles the login and logout process of user and authentication. It also allows the user to migrate from server to server, in which case the network connection is transiently passed on. Client and server software negotiate the range of protocols used for communication.

3. Avatar control protocol

The avatar control protocol is concerned with the control of the virtual representation of the user. The user may upload his own avatar representation, or else decide to use a default representation. He may also transiently switch to an alternate avatar representation, if the situation or application demands it (e.g. for participating in a game).

The avatar control protocol's task is to determine where the user is and what he can see. The protocol is extremely simple because it only needs to be able to communicate transformation matrices. The avatar control information is sent from the participating client to the server, and then distributed to all relevant clients. High performance regarding this protocol is of extreme importance for high fidelity interaction among a large group of users. We therefore chose to isolate the task in a separate protocol to have room for optimizations, that can be based on proximity [Benf93], visibility [Funk95], dead reckoning [Mace94] or fidelity channels [Sing95].

Other control commands concerning the avatar (object manipulation etc.) are part of a more general protocol, the interaction protocol.

4. Geometry protocol

Detailed models consume a lot of network bandwidth, so usually network transmission is either avoided [Zyda92], or image generation stalls until transmission is complete (such as current VRML browsers do [Hard95]). Our approach differs in trying deliver the model data “just in time” for display.

The overall geometric database held on the server is much larger than the client’s area of interest (AOI, comparable to the “aura” [Benf95]). Models contained in the AOI are held in a geometry cache (local memory) for immediate display [Figure 1]. If an object is no longer in the AOI, it will eventually drop out of the cache. Prefetching of objects that approach the AOI compensates for network delays.

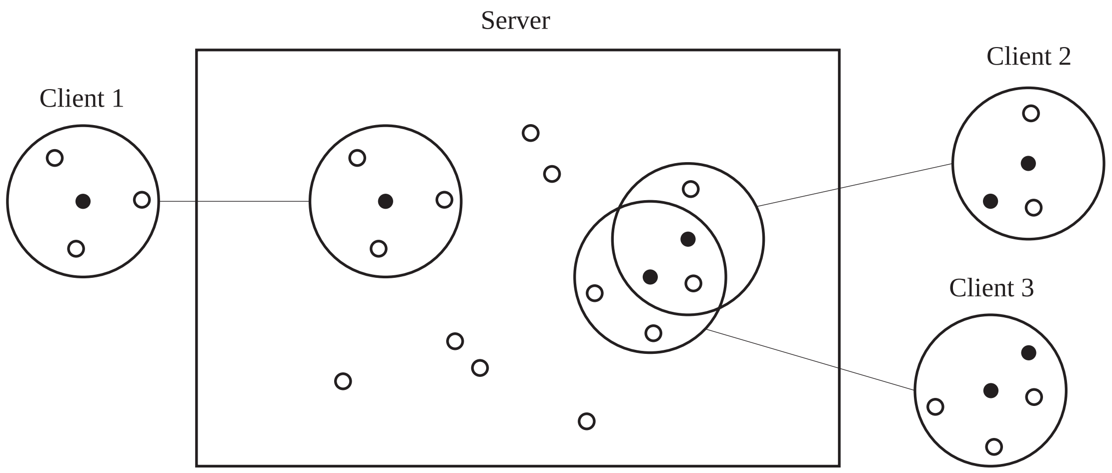

Figure 1: A server maintains a geometry database of objects (small white circles), and also represents clients (small black circles) and their corresponding areas of interest (large circles)

This strategy can still fail if too many and too complex objects are in the user’s AOI. We therefore incorporate a level-of-detail (LOD) method: Objects are modeled at multiple resolutions, and the appropriate resolution is selected at runtime based on heuristics [Funk93]. We extend this method to work with our distributed protocol along the lines of [Funk92] by transmitting single LODs instead of objects. This has several advantages: If the object can be represented by a coarse approximation (normally if seen at a distance), only the coarse model has to be transmitted, which saves transmission time and client memory. Assuming hierarchically modeled objects, transmission can be incremental (a finer LOD is based on the next coarser LOD), which also saves bandwidth. Finally, if timing constraints cannot be met (i.e. the desired LOD is not delivered in time), a coarser LOD can be used instead, to keep image generation from stalling (at the expense of degraded image fidelity).

Because the client requests the geometry at his disposal, the sophistication of the LOD strategy is up to the client. The server only needs to inform the client about any activity in his AOI. We use the VRML format for our protocol, but are developing a custom compressed variant to reduce the sizes of transmitted models.

5. Animation protocol

The animation protocol communicates dynamic changes in the scene. The nodes of the hierarchical scene graph we use consist of attributes, dependent on the node type. If an attribute is modified, the modification can be encoded by specifying the object ID, name of the attribute, and the new value.

To make the transmission efficient, updates for a particular client are collected and sent at regular intervals. For a particular modified attribute, only the most recent value is sent. Dependencies in the scene graph can be exploited by specifying attributes as functions of one or more parameters. Many attributes can depend on the same parameter, which provides a small and powerful interface for simulation updates, and also drastically reduces the number of updates that must be transmitted. Parameters can also depend on time, so that self-contained key-framed animations can be constructed. The principle has been known in computer animation for quite a long time [Magn83]. Parameters are built as an extension of Inventor engines [Stra92].

6. Simulation protocol

A uniform mechanism is needed to allow external programs (either clients or external applications) to talk to an actor living on the server. The protocol can automatically be built from the actor's method list. Method invocations with appropriate parameters are used as remote procedure call stubs. We use this protocol for three purposes:

1. to let actors exchange messages within the server,
2. to let external applications or users send control messages to actors, and
3. to let remote-controlled actors pass on all received messages to an external application.

While these three ways of usage are semantically very different, the protocol syntax is identical.

7. Interaction protocol

The interaction protocol addresses several requirements:

- For interaction of actors and humans, the simulation protocol alone is not sufficient. We also need a description of the interaction style.
- We do not want to require everybody to own the same type of I/O hardware (e.g. a position tracker or data glove). Instead, we aim at an abstract workplace characterization such as known from PHIGS [ISO89]. The user interface should be dynamically created from an abstract characterization.
- A client-server system suffers from the lag introduced by the relatively long round-trip an interaction message takes from client to server and back. We want to support local interaction for simple interaction tasks, that do not require the server's simulation capabilities (e.g. positioning in 3-D).

To describe interactions, we use *interaction rules* similar to the dialogue manager presented in [Appi92]. Interaction rules specify how to map input events generated by the user to output, generating feedback both locally (tightly coupled) and globally (distributed via the server). An interaction rule consists of input and output specification:

Input is characterized by the quality of interaction (e.g. positioning, selection, action-trigger), input dimension, input mode (discrete or continuous, absolute or relative), default and range values, importance (priority) and preconditions (to logically link multiple inputs together).

Output is defined by (1) one or more simulation protocol stubs of the simulation protocol to call with the parameters from the input, and (2) direct feedback to the geometry of the actor (replicated at the client, so we use the animation protocol locally). Note that there is no one-to-one mapping between interaction rules and simulation protocol stubs.

With a combination of simulation protocol stubs and interaction rules, we can decouple simple interactions from the server and run them locally with high fidelity, while interaction with the server's simulation is not restricted in any way.

8. Authoring protocol

A virtual environment should allow dynamic modification of all its components. While the modification of existing actors is handled by the simulation protocol, for creation and deletion of actors and actor types we introduce an authoring protocol.

Actors are categorized by type. In our system Python - an object oriented interpreted language - is used to define actor types and to instanciate actors at the server. The authoring protocol allows the creation of a new actor class (actor type), the instanciation of a new actor of a specific type, and the deletion of an actor. New actor types require a description of the actor's geometry (specified in extended VRML), behavior (specified in a scripting language), and user interface (interaction rules). An actor can also be configured to cooperate with an external application that determines the actor’s behavior (remote-controlled actor).

All parts of the actor’s description can be written using simple text files, that can conveniently be transmitted between sites and edited. The authoring protocol has no time-critical requirements.

9. Tying the protocols together

While the multitude of protocols we use is certainly more complex than a simple uniform protocol, the benefits make it worthwhile:

Efficiency. Every protocol can use specific knowledge from the particular domain to tailor the protocol specifically for the task. This is important because network bandwidth is precious, and must be preserved as far as possible to allow scalability. Measures for efficiency include compact encoding of information, data compression, and the use of multiple low level networking standards (e.g. TCP vs. UPD) as needed by the protocol.

Contention management. Because multiple protocol streams execute concurrently, conflictual situations may arise when multiple communication streams are competing for limited network bandwidth. In particular, if network performance degrades significantly, it is important to prefer those protocols that have tighter timing requirements. A priority mechanism can be used to resolve the problem (for example, transmission of animation has a very tight time window, while authoring is not really time-critical).

Protocol combination. The selection of protocols allows to combine them as needed by a client (fig. 1).

An *observer* will only need to support the connection management protocol, avatar control protocol, and geometry protocol. Sending the avatar control protocol allows the observer to navigate the VE. This is sufficient for a simple walkthrough system. Optionally the observer can receive avatar control messages from other avatars, so that its environment is not static, or even the animation protocol for highly detailed animation.

A *participant* subscribes to the same protocols as an observer, plus mandatory support for the animation protocol. The difference to the observer is that the participant supports the interaction protocol.

An *author* must at least run connection management and the authoring protocol. Usually the author will also run other protocols, so that the user can see the effects of his work.

An application will run connection management and a bi-directional simulation protocol: incoming simulation messages are passed on from the actor to the application for processing, and the reactions of the application are re-inserted into the server’s simulation by also using the simulation protocol. An application may also choose to subscribe the geometry and possibly animation protocol (e.g., for collision detection).

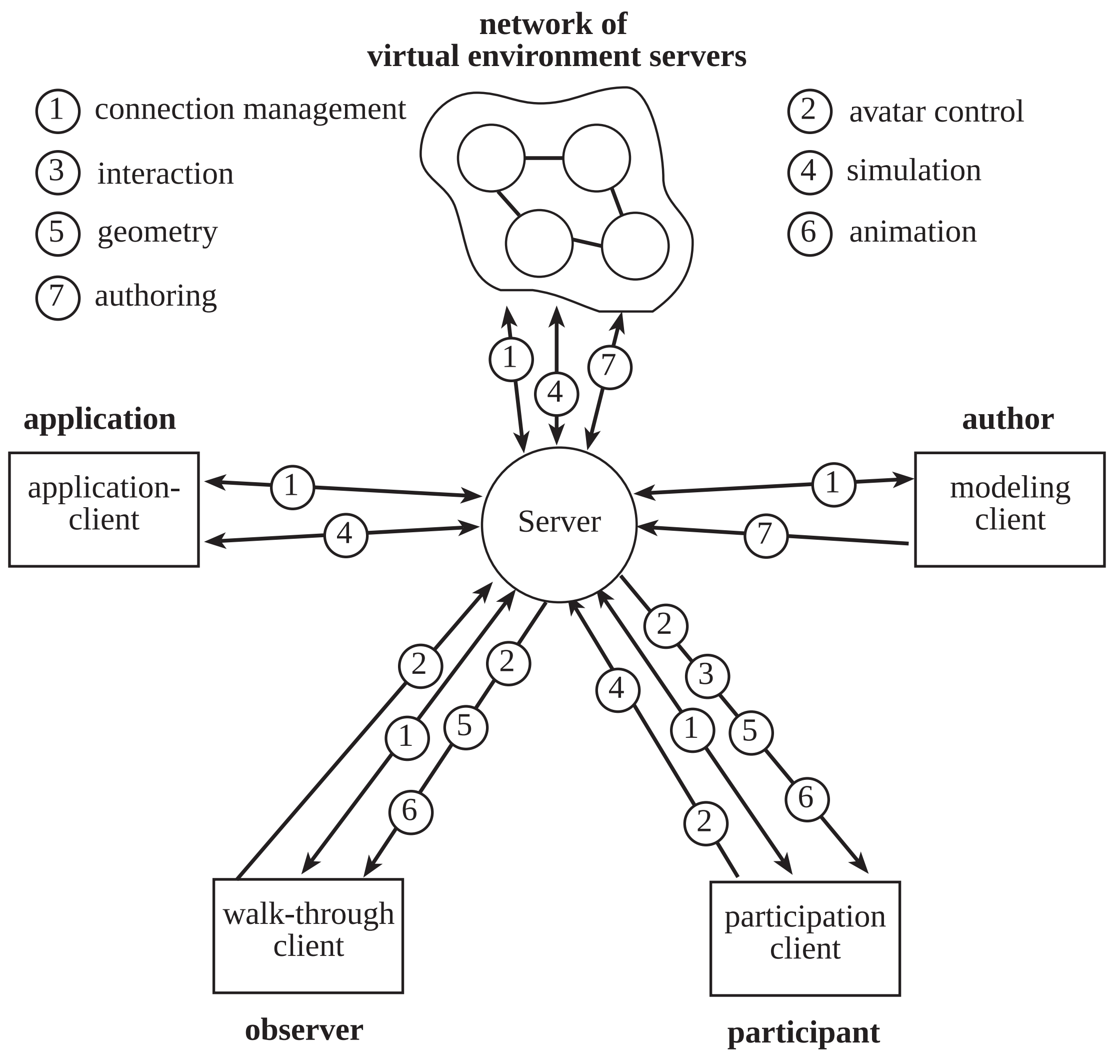

Fig. 2: Different kind of protocols are needed for different kind of clients.

Communication between servers naturally differs from communication between client and server, but basically re-uses the protocols already described. Beside connection management, server-server communication involves actor migration (sending a package containing actor and actor type, mostly the same as the authoring protocol), and the usage of the simulation protocol should actors wish to communicate over server boundaries.

10. Implementation

The possibility to combine protocols allows us to start with a subset of the full architecture, and extend it as needed. We have currently implemented a distributed system supporting communication for what is characterized above as an observer client. Such a systems supports navigation in a large virtual environment composed of static geometry (walkthrough) for multiple users that can also see each other. Of particular interest is the management of geometry data in the very large environment (details can be found in [Schm96]). Client and server software support three protocols: Connection management, avatar control, and geometry management.

Connection management basically allows a client to connect (init_connection) and disconnect (kill_connection) while the environment is running. Upon log-in, the client receives a unique client ID, states his initial position and orientation and the size of his AOI, and uploads the user’s geometric description (avatar) to be seen by other users [Table 2] (column labeled „Dir.“ indicates direction of message - from client to server or vice versa).

Message	Dir.	Parameters
init_connection	c→s	client_id, position, orientation, AOI_data, avatar_data
kill_connection	c→s	client_id

Table 2: Connection management protocol units

Adding avatar control requires two additional protocol units [Table 3]: With update_client_position the client tells the server about its new position after if the user has moved. The server uses update_object_position to inform the client about movement activity in the client's current area of interest. Movement of both animated objects (if the server simulates objects' behavior) and of other users is transmitted to the client.

Message	Dir.	Parameters
update_client_pos	c→s	position, orientation
update_object_pos	s→c	object_id, position, orientation

Table 3: Avatar control protocol units

Geometry management requires communication in two directions: The client decides it needs a particular piece of geometry and issues a request to the server (request_geometry). The unit of transmission is a single level-of-detail of a particular object. The server packages and sends the requested geometry data (transmit_geometry). Additionally, the client must know details about the objects in his AOI, so it can compute its needs and issue requests. This information is continually kept up to date by the server as the environment changes (transmit_object_info). Other necessary information includes: update of the client if an object is deleted (e.g. blown up; kill_object), and update of the server if the client decides to change the size of its AOI (e.g. if running out of memory; update_AOI). Table 4 shows the protocol units.

Message	Dir.	Parameters
request_geometry	c→s	object_id, lod_no
transmit_geometry	s→c	object_id, lod_no, geometry_data
transmit_object_info	s→c	new_object_id, object_info_data
kill_object	s→c	object_id
update_AOI	c→s	AOI_data

Table 4: Geometry management protocol units

Our experiments show that the geometry management as provided by the protocol bring significant savings in the amount of consumed network bandwidth. Not only can a geometry database with well-designed levels of detail yield a net traffic reduction of 2-3 times, but also the peak network load is much lower, since the transmission of single levels of detail instead of complete objects (all levels of detail) tends to distribute the network load much better.

11. Conclusion and future work

We have presented a framework of protocols designed to be used for special communication needs in client-server virtual environments. They address simulation, animation, interaction and VE authoring, and can be combined as needed for multiple levels of participation in the VE. Separating these tasks in different optimized protocols leads to more efficiency in using the given bandwidth of today's computer networks.

Our current implementation allows walkthrough and observation of large multi-user virtual environments by supporting connection management, avatar control and geometry management. Support for the complete communication framework as outlined in this paper is under development. The framework will also allow integration of new protocols, such as support for text or audio based participant communication, that we plan to include in a future project.

Acknowledgements

This work was sponsored by the Austrian Science Foundation (FWF) under contract number P11392-MAT.

References.

[Appi92] P. Appino, J. Lewis, L. Koved, D. Ling, D. Rabenhorst, C. Codella: An Architecture for Virtual Worlds. Presence, Vol. 1, No. 1, pp. 1-17 (1992)

[Benf93] S. Benford, L. Fahlen: A spatial model of interaction in large-scale virtual environments. 3rd European Conference on CSCW, pp. 109-124 (1993)

[Bric94] W. Bricken, G. Coco: The VEOS Project. Presence, Vol. 3, No. 2, pp. 111-129 (1994)

[Brol95] W. Broll: Interacting in Distributed Collaborative VE. Proc. of VRAIS'95 (1995)

[Carl93] C. Carlsson, O. Hagsand: DIVE- A platform for multi-user virtual environments. Computers & Graphics, Vol. 17, No. 6, pp. 663-669 (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. Funkhouser, C. Sequin: Adaptive Display Algorithm for Interactive Frame Rates During Visualisation of Complex Virtual Environments. Proceedings 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)

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

[ISO89] ISO: Programmer's Hierarchical Interactive Graphics System Functional Description. ISO/IEC 9592: 1 (1989)

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

[Magn83] N. Magnenat-Thalmann, D. Thalmann: The Use of High-Level 3-D Graphical Types in the Mira Animation System. Computer Graphics and Applications, pp. 9-16 (1983)

[Schm95] D. Schmalstieg, M. Gervautz: Towards a Virtual Environment for Interactive World Building. Proceedings of the GI Workshop on Modeling - Virtual Worlds - Distributed Graphics, Bonn. Also Technical Report TR-186-2-95-08, Vienna University of Technology, Austria (1995). ftp://ftp.cg.tuwien.ac.at/TR/95/TR-186-2-95-08Paper.ps.gz

[Schm96] D. Schmalstieg, M. Gervautz: Demand-driven geometry transmission for Distributed Virtual Environments. Submitted for publication. Also technical report TR-186-2-96-02, Vienna University of Technology, Austria (1996). ftp://ftp.cg.tuwien.ac.at/TR/96/TR-186-2-96-02Paper.ps.gz

[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)

[Snow94] D. Snowdon, A. West: AVIARY: Design Issues for Future Large-Scale Virtual Environments. Presence, Vol. 3, No. 4, pp. 288-308 (1994)

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

[Zyda92] M. Zyda, D. Pratt, J. Monahan, K. Wilson: NPSNET: Constructing a 3D Virtual World. SIGGRAPH Symposium on Interactive 3D Graphics, pp. 147 (1992)