The Verse Networked 3D Graphics Platform

Emil Brink
The Emotional and Intellectual Interfaces
Studio, Interactive Institute
P.O. Box 24081
SE104 50 Stockholm, Sweden
+46-(0)8-7832470
emil.brink@interactiveinstitute.se

ABSTRACT

This paper introduces the Verse platform for networked multiuser 3D graphics applications. The heavily client/server-oriented nature of the network architecture is described and motivated, as is the very heterogenuous client concept. The data storage model is introduced, as are several features of the object representation. A single-primitive approach to high-quality graphics based on a hybrid of Catmull-Clark and Loop subdivision surfaces is described, together with a high-level material description system. Verse is briefly compared to several existing projects.

Keywords

Virtual reality, networked, subdivision surfaces, subscription, client/server, methods.

1. INTRODUCTION

The Verse project was started at the Interactive Institute in June of 1999, in an attempt to make the threshhold for working with networked 3D graphics applications lower. It was felt that existing tools and architectures either were not open enough, lacking in modern features, or both.

Verse is, at its core, a custom network protocol optimized to describe 3D graphics data according to a certain data format. The data format, in turn, is optimized for flexibility and dynamism. Around these two components, we are trying to build an entire platform

By basing the platform on an openly available communications protocol, we try to encourage and simplify independent development and interoperability.

1.1 General Philosophy

The general idea with Verse is to create a platform that can support various kinds of applications involving networked 3D graphics. We want the platform itself to be have a fairly “low profile,” so that it leaves as many design and policy issues as possible to the application developer. Data describing graphics should be of the highest possible quality, and should describe the objects without regard to how they are rendered by clients. Verse is not intended to be a research prototype, but should be good enough for real-world use.

2. ARCHITECTURE

The bulk of this paper will describe various aspects of Verse’s architecture, including network and data storage issues.

2.1 Client/Server

Verse as a whole is designed as a client/server system. There is a single central server, to which multiple clients connect. The server stores the data describing the world, and clients can then connect to the server and read and write the data it stores. The server "knows" which clients are reading which data, and distributes any changes accordingly.

We favor a pure client/server approach rather than a peer-to-peer or hybrid network architecture for several reasons. One is that by making all the data that describe a world reside in a single process, administration and ownership of the world becomes easier to understand. Also, we feel that access security is easier to achieve if there is a single point through which all accesses must go. Persistance also comes naturally in a client/server system: as long as the server is kept running, the world persists.

2.1.1 A Lightweight Server

The most important characteristic of the server in the Verse architecture is that it is small, both theoretically (it has few responsibilities) and in practice (as a computer program it is not very big[1]). Conceptually, all the server does is store data, provide an interface through which that data can be accessed, keep track of which client is accessing what data, and forward changes.

2.1.2 Heterogenuous Clients and Servlets

The word "client" is heavily used when discussing Verse. This is because since the server does so little, much of the work other systems might put in the server is delegated into clients. Programs that are "server-like" in their nature, but technically Verse clients, are often called "servlets". We assume that the administrator of a server will also chose a set of desired additional services and run the required servlets on (or near) the machine that hosts the server itself. A future version of the server might incorporate support for running clients inside its own memory space, thus preserving the strict separation while minimizing communication overhead.

2.2 Data Organization

From one perspective, Verse can be seen as a network-accessible special-purpose database. This section describes how data are stored and managed in Verse.

---

[1] Currently, the server is roughly 100 KB in binary executable form (stripped, on Linux x86).

2.2.1 *Nodes*

Data stored by the Verse server are divided into a set of discrete entities called nodes. Several distinct types of nodes exist, each one is specialized to store a subset of the data required to describe a world. Currently, the types object, geometry, material, bitmap, code and emitter have been defined.

Object nodes are used to represent entities that should be visible in the world. An object is given a "look" by linking it to geometry and material nodes. Material nodes in turn link to bitmap nodes for textures and other image-like data. An object can be given a method interface, in which case it might use a code node to store the implementation. The emitter node is supposed[2] to express various emissive effects, such as sound and particles.

Each node is identified by a 32-bit unsigned integer identification number, assigned by the server upon creation. Node IDs are globally unique on a server.

2.2.2 *Node Commands*

Each node type defines its own set of commands that operate on the data stored in the node. Node commands are the only thing sent by the Verse network protocol; all communication between a Verse server and its connected clients is done using node commands. Commands are symmetrical, meaning that the same command is often used both as a request and as a reply. Communication in Verse is mainly client-driven; the server does not send unrequested data to clients.

2.2.3 *Subscriptions*

The concept of subscription forms the basis of Verse's data distribution design. Clients need to actively tell the server which nodes they are interested in, by subscribing to them. Nodes typically contain smaller parts which are in turn subscribable. A client learns about the subscribable parts of a node by subscribing to something at the next higher level, beginning by subscribing to the node itself.

2.2.4 *Dynamic Data*

One of the most important aspects of Verse, that differentiates it from many existing systems, is that *all* data stored and handled by it are fully dynamic.

Things such as an object’s geometric representation, or a color bitmap for texturing, are transmitted and handled so that very small parts of these data structures are always addressable and thereby changeable at any time. For example, any vertex in a Verse geometry node can be moved at any time, polygons can be created and destroyed at will, bitmaps can be repainted ”on the fly”, and so on. This dynamic nature makes Verse well suited for realtime networked cooperative applications.

The data storage formats and the node commands that express them have been designed so that, from a client’s perspective, there really is no difference between the ”original” and ”changed” versions of e.g. a vertex position. They both look exactly the same, which makes it easier to write clients to support this level of dynamism.

[2] The emitter node is currently being redesigned, and might go away or be renamed soon.

2.3 Network Layer

Networking is of course important in a system such as Verse. We use a custom-built asynchronous protocol designed to send node commands, layered on top of unicast UDP.

2.3.1 *UDP*

Verse uses UDP, a low-level datagram transport protocol that is part of the TCP/IP standard suite of protocols. UDP, unlike TCP, does not guarantee that datagrams sent using it actually reach their destination; they can be dropped at any point in the network. This means that Verse must handle dropped datagrams itself, by doing resends. Verse uses UDP rather than TCP because it is inherently better suited for realtime applications, and also because it gives us more control over the flow of datagrams.

2.3.2 *Unicast*

Verse datagrams are sent using ”classic” unicast semantics, i.e. each datagram has only one recipient. This is in contrast to the use of multicast [7], where each datagram is sent to an entire group of recipients. There are many reasons why we do not use multicast in Verse. One is that the basic service provided by it, efficient one-to-many data distribution, does not fit well with a general 3D world. All clients do not want all data; each client only wants the data is is subscribed to. An alternative might be to use one multicast group per node, but that is not very appealing either. First, IPv4 reserves no more than 24 bits (0.39%) of its address space for multicast groups, while Verse’s node ID space is a full 32 bits. Second, separating transmissions into distinct multicast groups means more datagrams in total, which increases the cost of per-datagram overhead. Third, Internet-wide support for multicast is still rather limited.

2.3.3 *Asynchronous*

The network layer is asynchronous; commands are collected into datagrams which are then emitted. The recipient decodes the datagram and generates a stream of command envocations, which is handled either by the server or delivered to the client (by the VLL helper library, see below). Each datagram is handled separately, without regard for the datagrams that preceeded it, or the ones that will succeed it. The datagrams are given a sequential number, so that lost (dropped) ones can be detected. When this happens, a resend is eventually done.

When commands are collected into datagrams to be sent, the network layer uses knowledge it has about each command's content to overwrite duplicates when possible. This is a form of event compression, and conserves bandwidth by not sending redundant data over the network.

2.3.4 *VLL*

Client programmers do not need to know much about how Verse looks "on the wire". Instead, they use our application programming interface (API) VLL, which is delivered as a C link library. By calling functions in VLL, a client program can establish a connection to a server, and also exchange data with it. For the most part, functions in VLL map 1:1 to node commands, which are sent to the server. When a reply comes back, the client program is notified through a callback function, which is called with the command’s parameters (if any) as its arguments.

2.4 Object Nodes

Nodes of type object are arguably the most important nodes used in Verse. For something to be visible in a world, it has to be associated in some way with an object node. Each connected client is given an object node that represents that client in the world hosted by the server. This object is known as the client's *avatar*, but there is nothing special about it. It is just like all other object nodes.

2.4.1 Transform

Verse currently uses a fairly simple transform system to express object movements. It is based on a clock, which measures time in milliseconds. The clock is synchronized between client and server when the client connects, using a simplistic measurement of the network latency that separates the two. An object's transform consists of the three quantities position, rotation and scale. Changes to each of these three quantities can be done at the zero'th, first or second derivative.

As an example, an object's position is controlled by the following equation, familiar from simple Newtonian physics:

$$r(t+\Delta t) = r(t) + r'(t)\Delta t + \frac{r''(t)\Delta t^2}{2}$$

Where r(t) is the last known position at time t, r' and r'' denote the first and second temporal derivatives of the position respectively, and Ä*t* is the time step being simulated.

Because all commands that handle transforms using this system include a timestamp for when they should take effect, it is possible to build up a queue of events in advance, if the events are known beforehand. This decouples the network traffic that describes a series of transform changes from the changes themselves, which is sometimes useful.

Setting a transform quantity's value directly (at derivative level 0) is akin to ”teleporting” the object, and will likely be restricted in worlds that try to appear realistic. In that case, movement must be done by setting either a velocity or an acceleration.

The current transform system handles only the entire object as a rigid whole. This will be extended in the future to support internal transformations, for skeletal animation and object embedding.

2.4.2 Tags

Verse object nodes contain a system for storing application-managed named values, called tags. Each tag stores one value, which can be of either integer, floating-point, textual or unformatted binary type. Tags are arranged into tag groups, each of which is a subscribable entity. The Verse server does no interpretation of the content of tags; tag data is viewed as being totally owned and controlled by the applications that use the objects.

Tags are intended to be used by applications that need to extend objects with domain-specific data. For example, a game might need to associate a health value with each player. Tags provide a flexible and straight-forward way of doing this.

2.4.3 Methods

In addition to the tag system, which stores application-defined "passive" data in objects, there is also support for storing something called methods [1]. A method is simply a description of a program entry point associated with the object. The description does not include any code or other implementation of the method; this is left to dedicated clients. Methods can accept parameters of various types, much like functions in C. However, object methods do not have return values, mainly because the underlying network layer is inherently asynchronous.

Methods are collected into method groups, which are named and subscribable. The intended use is to allow various kinds of specialized behavior and/or "intelligence" to be associated with objects. In the full-fledged version of method use, the code that implements each method is stored on the server in a code node, and interpreted by a general virtual machine client. It is, however, entierly possible to write dedicated clients that only deal with handling calls to methods in a given group, bypassing the code node and code interpretation completely.

As an example of how methods could be used, consider a general 3D world where clients connect and navigate around through the use of avatars. Navigation might be done through the use of keyboard and mouse input, as is common on the PC platform. Without methods, the browsing client would directly translate keypresses into transform commands sent to the object node representing the avatar. If the avatar is given a set of methods for movement, the client would instead issue calls to these methods when corresponding keyboard keys are pressed. Somewhere, perhaps on the same machine as the server, another client would receive the method calls and translate them into object transform commands. Using the method system in this way allows important properties of an object, such as its control interface, to be abstracted out in a way we find very flexible.

2.4.4 Linking

Object nodes do not contain data about things such as geometry or material properties directly. Instead, such data is stored and managed by instances of dedicated node types, and the object node simply links to these instances as needed.

Other nodes can also contain node pointers, for example material nodes often need to refer to bitmap nodes that act as data sources for texture mapping and filtering.

Data stored in a node can be shared between multiple users by simply letting each user link to the desired node. For example, two objects with the same geometry will link to the same geometry node. They can still have different materials, by linking to distinct material nodes.

2.5 Graphics

Since Verse is ultimately a system for building networked 3D graphics applications, it naturally contains quite a bit of mechanism for handling the actual graphics. Specifically, it has a flexible geometric primitive and a material description system.

2.5.1 Creased Subdivision Surfaces

Unlike many other systems and file formats for virtual reality and networked graphics, Verse has a single primitive used to represent graphics. We use a hybrid of Catmull-Clarke [4] and Loop [14] subdivision surfaces, and support meshes that mix triangles and quadrilaterals freely. Also, these surfaces have been extended with crease data for vertices and edges, allowing sharp features to be expressed accurately and simply.

We feel that subdivision surfaces have many features that make them a good choice for a networked graphics system. They are a form of higher-order surface representations, which means that arbitrary geometric detail can be generated from a small initial description, thus conserving network bandwidth and ensuring client-side quality scalability. They support smooth, curved surfaces, thus making them useful for organic models such as human-looking avatars. With the addition of creases, well-defined hard edges are simple to express as well. Further, since our subdivision scheme works on arbitrary meshes of triangles and quads, they are very easy to work with from a modeling perspective, lacking the constraints that make working with e.g. NURBS surfaces occasionally troublesome.

Further, subdivision surfaces lend themselves very well for displacement mapping [13], thereby giving us a way to represent highly detailed surfaces in a compact manner.

By chosing just a single geometric primitive and sticking with it, we hope to reduce the burden on rendering implementors while also making content creation easier and more rational. Please note that subdivision surfaces are "backwards-compatible" with plain polygonal meshes; by making all vertex and edge creases as sharp as possible, the entire subdivision scheme is essentially short-circuited and the base control mesh survives unchanged.

2.5.2 Material System

The material node is used to express how the surface of an object should look. It does this by defining a set of operands and operations, and then letting them be connected to form a graphical data flow description of a material.

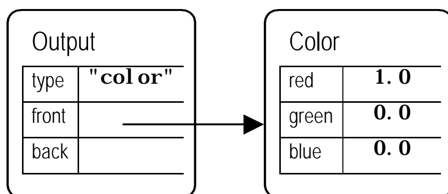

Figure 1. A simple material.

As an example, consider Figure 1. This shows a very simple material, containing just two fragments; an output and a color. The output fragment is always at the root of a particular description. The type field in the fragment selects what aspect of the surface's material is described by the tree, with "color" being typical. The color fragment simply defines a constant (R,G,B) triplet. Linking the front field of the output to the color fragment expresses the idea that front-facing polygons in objects having this material should be colored by the linked-to color. The back field is left unconnected, which means that back-facing polygons can be culled away (they have no material, so they are invisible). Note that this simple material assigns the exact same constant color to all polygons; no shading or lighting is done.

Through the use of other more advanced fragments, the material system is capable of expressing a wide range of properties. Recursive texture mapping, refraction, BRDFs and micro-geometry, for instance, are some of the more advanced effects that are easily expressed by the material node. Figure 2 shows a slightly more advanced material, namely the default way to do a single lit color texture map. The blender fragment multiplies the light with the texture, and passes the result to the output fragment. The figure itself is a cropped screenshot from Adamant, a realtime online graphical material editor for Verse.

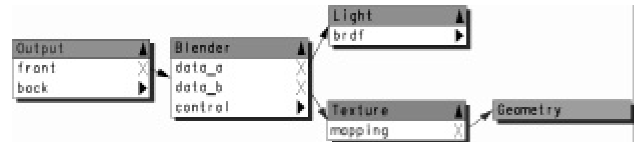

Figure 2. Simple lit texture map material.

In the general case, an abstract description of a material is much easier to come up with than a program to render an object using that material. The material system is easily capable of expressing materials that today are far from possible to render in realtime, or even at all. The idea here is that the description of an object stored on the server should be as "perfect" as possible, leaving it up to (rendering) clients to do a "best effort" attempt to approach it in practice. This provides a way of achieving some form of future-proofness in the system, since as rendering technology develops, clients can be adapted and old objects start to look better.

2.5.3 Particles and Effects

Much work is currently being done in designing and testing a general system for expressing and simulating a wide variety of particle system effects. Such effects are commonly used to mimic natural-world phenomena like smoke, fire, liquids and hair.

2.6 System Overview

Running an application using Verse typically involves many processes, all working with the data held and distributed by the server.

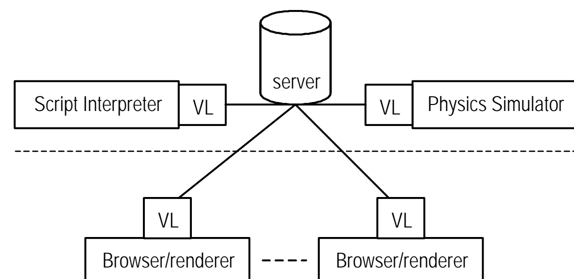

Figure 3. Example Verse system configuration, schematic overview.

Figure 3 shows a schematic picture of how Verse looks when used for some kind of simple application. At the top center of the figure is the server itself, surrounded by a pair of servlets. Below the dotted line is the "true client" side of the network, showing a couple of connected browser clients. Note how both servlets and end-user clients communicate with the server through the same VLL library. Technically, there is no difference at all between a servlet and a client; the term servlet only implies that that particular client is best suited to run close to the server, for reduced latency.

2.7 Client Freedom

One important thing in Verse is that the client has a large amount of freedom in deciding what to actually do with the data made accessible by the server.Rendering, for example, is not in any way a built-in, intrinsic part of the system itself. A rendering (or

browsing) client is just another client, that happens to present the data describing geometry and materials graphically.

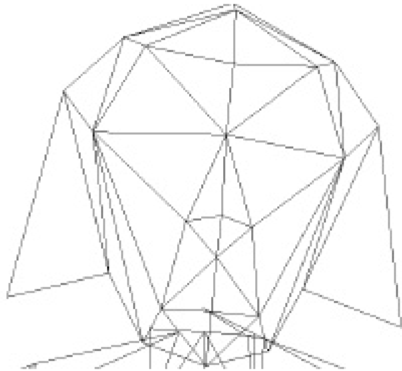

Figure 4. Base mesh (simple rendering client).

Consider the (touched-up) screenshots of Figures 4 and 5. Figure 4 is from a very simple prototype-quality renderer, while Figure 5 depicts Enough, a much more ambitious renderer. Both clients are connected to the same server, so the data they base their displays on is identical. The renderer in Figure 5 uses the full subdivision-surface scheme, while the one in Figure 4 simply renders the control mesh as-is.

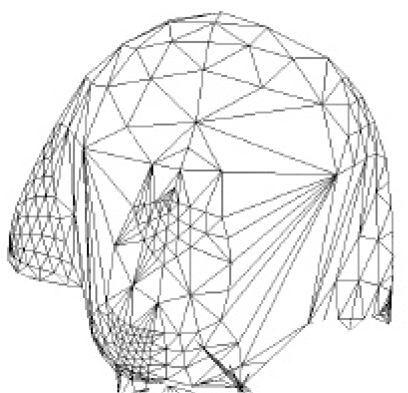

Figure 5. Subdivided mesh (high-quality renderer).

Since display is decoupled from data management, client programmers have lot of freedom in how they chose to interpret and operate upon the data stored on a server. This means that Verse is not inherently tied to polygonal rendering. A client could be written that visualizes the subdivision surfaces using raytracing, or maybe by exporting the scene to some off-line rendering package.

3. VERSE VS. EXISTING SYSTEMS

This section will briefly compare Verse to some other existing systems and technologies. The area of multi-user 3D-graphical application frameworks is growing very rapidly, so this section is far from complete.

3.1 Active Worlds

The Active Worlds system, by Activeworlds.com Inc. [1], is a commercial system for building interactive 3D worlds. Users can build new structures in the worlds, and also move existing objects around. However, creation is done by chosing from a library of ready-made objects; users cannot build new objects from scratch.

Since Active Worlds is commercial, hosting a world requires that a server license be purchased. The cost of a server license depends on the size of the world hosted (in virtual square meters) and the maximum number of simultaneous users.

3.2 Cult3D

Cult3D [3], by Cycore [5], is a technology that allows interactive 3D objects to be embedded into ordinary documents, most notably web pages and PDF files. The objects can have interactive features, such as a button that pops open the lid of a cassette player. The objects are, however, static and can not be permanently changed by the user. The experience is also strictly one-to-one; multiple users cannot see and manipulate the same copy of an object at once.

Cult3D is primarily intended for product visualization, by extending existing ”flat” documents with embedded 3D objects that users can do simple manipulations of. This is in contrast to Verse, which is more aimed at creating fully immersive 3D worlds.

3.3 DIVE

The Distributed Interactive Virtual Environment (DIVE) [9] is a research prototype for a multi-user 3D VR system developed by the Swedish Institute for Computer Science (SICS).

DIVE is loosely coupled, using peer-to-peer networking. It uses both reliable and non-reliable communications, both running on top of IP multicast. Objects in DIVE can have dynamic behavior described by scripts written in Tcl/Tk [12]. Being a prototype, DIVE is made available for restricted use only.

Although fundamentally different in network architecture, Verse and DIVE are similar in scope, although Verse (in our opinion) takes a more minimalist approach. The design of Verse focuses on data organization and distribution, and does not include things such as user interface widgets, file formats, or input sensor support. Verse aims to be useful for all kinds of applications involving networked 3D graphics, including research.

3.4 VRML97

VRML97 [12] is an ISO standard file format for describing interactive 3D objects and worlds. The focus of VRML97 is to define a format for ordinary text files that describe 3D scenes.

VRML97 is a fairly large standard; it defines over 50 distinct node types, including at least ten different geometry nodes (such as Cone, Cylinder, PointSet, etc.).

VRML97 does not in itself deliver a platform for multiuser networked interactive 3D. There have been many developments of such worlds, but these are all outside of the VMRL97 standard itself. Verse, in contrast, lacks a standardized file format, since our focus is almost completely on how to handle a world once it is up and running on a server.

The Verse distribution includes a very simple tool that allows some VRML97 scenes to be (partially) imported into Verse.

3.5 vrtp

The virtual reality transfer protocol (vrtp) [2] is an effort to create a communication protocol to support VRML in the same way that http [6] supports HTML [16].

The vrtp architecture aims to support not only client/server and peer-to-peer network approaches, but a full spectrum in between. An important part of the architecture is improved network monitoring, i.e. various forms of automated measurements of network parameters, useful for optimizing the protocol and simplifying its use.

The vrtp developers intend to establish a dedicated network offering “moderately high bandwidth and guaranteed low latency” in order to test and optimize the protocol. Verse, in contrast, is being designed and optimized to run over the general public Internet.

4. WORKING WITH VERSE

Work with Verse focuses on developing or adapting clients. The intent is that the server, data format, and overall architecture all should be general enough not to require changes for particular applications. Instead, an “application” consists of one or more clients that are run as appropriate, to get the desired results.

End-user rendering is a large task, and again the intent is that existing renderers should be general enough to be re-used for all or most applications. Still, there’s nothing preventing anyone from developing a rendering client of their own if they feel a need or commercially viable opportunity to do so.

5. FUTURE WORK

The evolution of Verse will require numerous additional subsystems. The most important of these have been narrowed down to the following.

5.1 Force-Based Transformations

The current transformation system is based on direct assignment of position, velocity and acceleration to one of the three quantities (translation, rotation and scale). While this works well for simple movements, it is rather limited. For example, it does not easily allow friction to be modeled, since no term for friction is included in the underlying equation.

A better approach, which we will be implementing is to replace the explicit movement model with a new one based on forces. In this model, moving an object would be done by excerting a force on it. Multiple forces would add, moving the object in the resulting direction.

5.2 Animation Subsystem

Verse currently lacks support for animation, so all objects are completely rigid. This will be solved by implementing a rather comprehensive and powerful animation system, based on parameterized mesh morphing and hierarchical transformations.

5.3 Physics Client

A great number of "typical" applications for a system such as Verse will require some form of physics simulation. At the very least, collision detection and enforcement to prevent avatars from passing through objects is needed for even very simple applications. The server design currently does not include any such functionality; it is all left to clients. Therefore, a physics simulation client is a high-priority task for the future development of Verse as a platform.

6. CONCLUSIONS

After almost two years of´development by a team of only two people, Verse is showing definitive promise. Most of the time, Verse works, and it works well. We feel that the dynamic and flexible nature of the platform show much promise. More users would be a great boon, and various activities are under way to try and attract these.

7. ACKNOWLEDGEMENTS

We would like to thank the Interactive Institute for making this work possible in the first place, and also for providing such a wonderfully diverse and interesting workplace. Special thanks to Mark Ollila at the Institute for helping me put this paper together.

Verse is being developed as free software, with open sourcecode. The licenses used are the GNU General Public License [10], the GNU Lesser General Public License [11], and the BSD license [15].

Like thousands of other such projects, we are thankful of the free hosting for such projects provided by SourceForge [17]. The official homepage for Verse is at http://verse.sourceforge.net/.

8. REFERENCES

[1] Active Worlds. http://www.activeworlds.com/ (valid March 2001).

[2] Brink, E. Dynamic Method Calls In A Networked 3D Environment. Master’s thesis, Royal Institute of Technology, Department of Numerical Analysis and Computer Science. TRITA-NA-E0077 (2000).

[3] Brutzman, D., Zyda, M., Watsen, K., and Macedonia, M. virtual reality transfer protocol (vrtp) Design Rationale. Workshops on Enabling Technology: Infrastructure for Collaborative Enterprises (WET ICE): Sharing a Distributed Virtual Reality, Massachusetts Institute of Technology, Cambridge Massachusetts, June 18-20 1997.

[4] Catmull, E., and Clark, J. Recursively generated B-spline surfaces on arbitrary topological meshes. Computer Aided Design 10, 350-355 (1978).

[5] Cult3D. http://www.cult3d.com/ (valid March 2001).

[6] Cycore. http://www.cycore.com/ (valid March 2001).

[7] Deering, S. Host Extensions for IP Multicasting. Internet Engineering Task Force RFC 1112. http://www.ietf.org/rfc/rfc1112.txt (valid April 2001).

[8] Fielding, R., et al. Hypertext Transfer Protocol -- HTTP/1.1. Internet Working Group RFC 2616. http://www.w3.org/Protocols/rfc2616/rfc2616.txt (valid March 2001).

[9] Frécon, E., and Stenius, M. DIVE: A Scalable Network Architecture for Distributed Virtual Environments. Distributed Systems Engineering Journal (special issue on Distributed Virtual Environments), Vol. 5, No. 3, Sept. 1998, pp. 91-100.

[10] Free Software Foundation. GNU General Public License. http://www.gnu.org/copyleft/gpl.html (valid March 2001).

[11] Free Software Foundation. GNU Lesser General Public License. http://www.gnu.org/copyleft/lesser.html (valid March 2001).

[12] International Standard ISO/IEC 14772-1:1997. The Virtual Reality Modeling Language. http://www.vrml.org/technicalinfo/specifications/vrml97/index.htm (valid March 2001).

[13] Lee, A., Moreton, H., and Hoppe, H. Displaced Subdivision Surfaces. Proceedings of SIGGRAPH 2000, ACM Press, 85-94.

[14] Loop, C. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, Department of Mathematics, 1987.

[15] Open Source Initiative (OSI). The BSD License. http://www.opensource.org/licenses/bsd-license.html (valid March 2001).

[16] Ousterhout, J. Tcl and the Tk Toolkit, Addison-Wesley, ISBN 0-201-63337-X, 1994.

[17] SourceForge free project hosting service. http://sourceforge.net/ (valid April 2001).

[18] The World Wide Web Consortium. HTML 4.01 Specification. 1999. http://www.w3.org/TR/1999/REC-html401-19991224/ (valid March 2001).