The Verse Networked 3D Graphics Platform

Emil Brink*
Royal Institute of Technology

Eskil Steenberg†
Royal Institute of Technology

Gert Svensson‡
Royal Institute of Technology

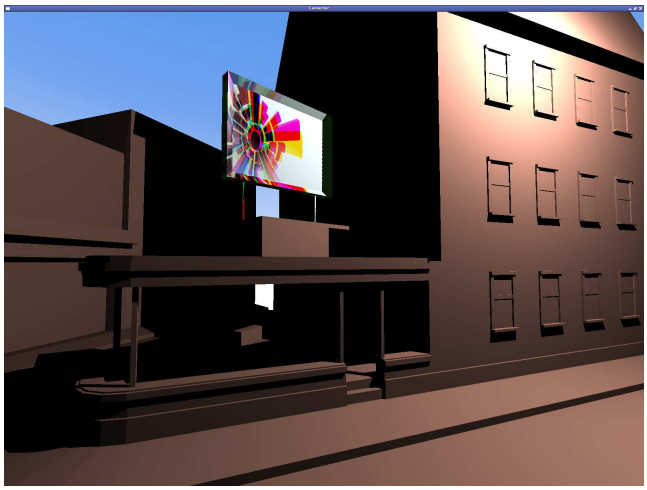

Figure 1: Screenshot rendered by Quel Solaar, showing multiple materials, texture-mapping, shadows and per-pixel lighting.

Abstract

This paper introduces the Verse platform for networked multiuser 3D graphics applications. The client/server-oriented use of the network architecture is described, as is the highly heterogeneous 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.

Keywords: network, 3D, graphics, subdivision, open, platform, cooperation, distributed

1 Introduction

The Verse[1] [Brink and Steenberg 2006] project was started at the Interactive Institute in June of 1999, in an attempt to make the threshold 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. The first version of the protocol was developed by Eskil Steenberg and Emil Brink.

*e-mail: ebr@kth.se
†e-mail:eskil@obsession.se
‡e-mail:gert@pdc.kth.se

[1] See http://verse.blender.org/

In 2001 Eskil Steenberg continued the Verse development by writing a second version of Verse independent of the Institute. In 2002 researchers from the Royal Institute of Technology and the Interactive Institute met to formulate a proposal for a European Commission three-year project formed by seven European research institutes and companies. The project was granted funding and was launched in February 2003 under the name Uni-Verse[2]. The idea with this project is mainly to develop and evaluate 3D tools based on the Verse protocol.

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 designed to be flexible and dynamic. 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 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 just 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] See http://www.uni-verse.org/

2.1 Client/Server

The Verse protocol is designed to support any network architecture but is often used as a client/server system. Normally 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 client/server approach rather than a peer-to-peer or hybrid network architecture for several reasons. One is that by making all the data describing 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. Persistence 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[3]). 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 Heterogeneous 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 also will chose a set of desired additional services and run the required servlets on (or near) the machine that hosts the server itself.

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.

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, text, curve and audio 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. Text nodes are used to store text, for whatever purpose. Curve nodes store multi-dimensional interpolation curves, for animation. Audio nodes store and stream sampled audio.

[3]Currently, the server is roughly 169 KB in binary executable form (stripped, Linux x86)

Each node instance is identified by a 32-bit unsigned integer number, assigned by the server upon creation. Node IDs are globally unique in each connection to a server. All nodes also support human-readable textual names. Furthermore, each node supports tags, which are simply named values of various types. Tags are collected into named groups.

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 [4] 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 objects 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 interactive 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 the dynamic properties of the data model.

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 standard 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 actually reach their destination;

[4]Such as tag groups, layers, buffers, streams etc.

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 interactive applications, and also because it gives us more control over the network traffic.

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[Deering 1989], 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 it is subscribing 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. This means that messages are generally sent one way only, and the sender does not wait for a response before continuing and sending the next message. Messages generally consist of *commands*, which are either system-level or sent to a specific node instance.

Commands are collected into datagrams which are then emitted. The layer decodes the datagram and generates a stream of command invocations, which is delivered to the application-level software. The application might be either the server or a client. Each datagram is handled separately, without regard for the datagrams that preceded 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 The Verse API

Programmers who wish to develop for Verse do not need to know much about how Verse looks "on the wire". Instead, they use our application programming interface (API), which is delivered as a C link library[5]. By calling functions in the Verse API, a client program can establish a connection to a server, and also exchange data with it. For the most part, functions in the API 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. The server is also implemented on top of the same API.

[5] Called "libverse.a"

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 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.

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 this case, movement must be done by setting either a velocity or an acceleration. The object-level transform system handles only the entire object as a rigid whole. The animation system supports defining a hierarchical skeleton of bones, with weighted influences on the vertices of the object. Objects can have child objects, to build hierarchical transformations at the object-level, too.

2.4.2 Methods

In addition to the tag system, which stores application-defined "passive" data in objects, there is also support for storing something called methods[Brink 2000]. 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 text node, and interpreted by a general virtual machine client. It is, however, entirely possible to write dedicated clients that only deal with handling calls to methods in a given group, bypassing the text node and online 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 move-

ment, 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.

2.4.3 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-Clark[Catmull and Clark 1978] and Loop[Loop 1987] 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 simply and accurately.

2.5.2 Material System

Verse features a very powerful and flexible material system. Rather than supporting a set of pre-defined complete shading models (such as a Blinn shader, a Phong shader, and so on), Verse lets the user define the material from scratch. This is done by building a shading tree, consisting of various fragments. Each fragment represents a simple operation, such as the application of a constant color, the light hitting the surface, blending two colors, and so on. Currently, there are 16 such fragments defined.

3 Working with Verse

3.1 Replacing File Transfers

In the Verse way of working, the 3D data is stored in a server and clients connect to that server and subscribe to the data that each client is interested in. This means that different tools can collaborate directly and without file transfers and file conversion that often are quite cumbersome for 3D data.

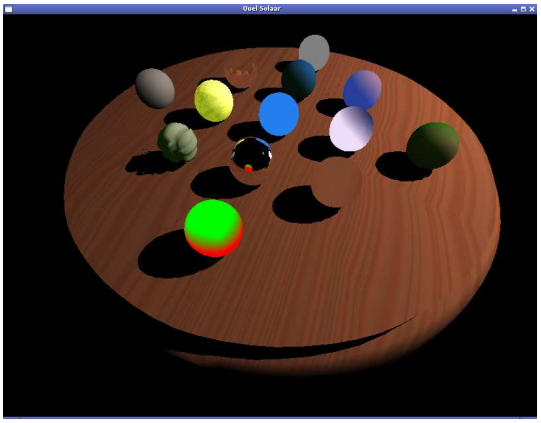

Figure 2: A number of sample materials, including transparency, displacement mapping, reflections, multi-texturing, and refraction.

3.2 Single-User Use

Verse is a big advantage also for a single user on a single computer. As a matter of fact several tools can collaborate with Verse as if they were part of a single 3D application, and this is valid on a single computer or distributed over the LAN in a company or distributed over Internet.

Traditionally, it was convenient to use a large monolithic 3D application for all steps in the 3D content creation process. With Verse-enabled tools, it is possible to choose the most appropriate tool for each task, and combine them into a virtual production pipe-line. This also means that tools can be smaller, and much simpler compared to the monolithic approach. This opens the possibility for smaller companies and open source volunteers to implement tools which are optimal for a single task.

3.3 3D Texture Painting

3D texture painting is a simple example of how Verse can speed up the process of content creation. If you are using open source tools you may use The GIMP[Kimball and Mattis 1996-2006] for texture painting and Blender[Blender Foundation 2002-2006] for the geometric modeling. Although Blender has its own set of texture painting tools built-in, being able to directly benefit from the much larger set of tools available in applications such as The GIMP would be a boon to artists. Without Verse, artists that want to use The GIMP to manipulate textures need to go through a save/reload cycle. With Verse the artists can see the result in the 3D world directly as they paint

3.4 Distributed Use

Today's companies and organizations become more and more geographically distributed, and the 3D content creation industry is no exception. Just having a customer review with the customer on a remote location and the possibility to change the model interactively

and have the customer review and directly comment on the changes opens up new ways of working.

3.5 Available Clients

In the Uni-Verse project a number of new Verse clients have been developed, and popular existing 3D tools have been adopted to communicate using Verse.

A set of tools which has been developed for Verse from the beginning consists of the modeling tool *Loq Airou*, the symbolic data manipulation and inspection tool *Connector*, the UV tool *UV Edit* and the geometry layer painter *Layer Painter*.

A tool called *Purple*[Brink 2005] for manipulation of Verse data with a data flow graph approach has been developed, hoping to illustrate how a future 3D package might be designed around Verse, and also to help get programmers "on board", by lowering the threshold to program for Verse.

Several rendering clients exist. *Quel Solaar* (see Figure 1) is high-quality, high-performance rendering client which has been designed for Verse from the beginning. This means that Quel Solaar supports changing the rendered data at any time. Quel Solaar also supports global illumination, dynamic shadows, the OpenGL 2.0 shader language, reflections, refractions and transparency as well as displacement mapping.

Furthermore, a rendering client based on OpenSG has been developed. It supports clustered rendering, where many computers cooperate to render for a single physical display.

A Verse plug-in to *Autodesk 3ds Max*[Autodesk Corporation 2006] has been developed, as well as a Verse connection for the major open source 3D package *Blender*.

Finally, an acoustic simulation client is being developed. The client makes realistic simulations of a building model in Verse, for use by architects and acoustic consultants. The model can be changed interactively and the acoustic simulation is then automatically updated. To achieve interactive speeds of the acoustic simulation the complexity of the geometric model is first automatically reduced (by a separate, dedicated client).

4 Conclusions

Verse is a network protocol for dynamic exchange of 3D data in a variety of situations. Verse clients have been developed for the standard tasks in 3D modeling and rendering.

The Verse way of connecting applications has been shown to increase productivity in the content creation process. We have the goal to establish Verse as an open industry standard for connecting 3D applications interactively.

5 Acknowledgments

The Verse project was funded by the Interactive Institute between 1999 and 2001. The Uni-Verse project runs between 2003 and 2007, supported by the European Commission's 6:th Framework Programme under the Information Society Technologies Programme.

References

AUTODESK CORPORATION, 2006. 3ds max. `http://www.autodesk.com/3dsmax`.

BLENDER FOUNDATION, 2002-2006. Blender. `http://www.blender.org/`.

BRINK, E., AND STEENBERG, E., 2006. The Verse Specification. `http://verse.blender.org/cms/fileadmin/verse/spec/`.

BRINK, E. 2000. *Dynamic Method Calls In A Networked 3D Environment (TRITA-NA-E0077)*. Master's thesis, Royal Institute of Technology.

BRINK, E., 2005. Purple. `http://purple.blender.org/`.

CATMULL, E., AND CLARK, J. 1978. Recursively generated b-spline surfaces on arbitrary topological surfaces. *Computer Aided Design 10*, 350–355.

DEERING, S. E. 1989. Host extensions for ip multicasting. RFC 1112, IETF, November.

KIMBALL, S., AND MATTIS, P., 1996-2006. The GNU Image Manipulation Program. `http://www.gimp.org/`.

LOOP, C. 1987. *Smooth subdivision surfaces based on triangles* Master's thesis, Department of Mathematics, University of Utah, Salt Lake City.