# A Network-Enhanced Volume Renderer

Jeff LaPorte

University of British Columbia
Vancouver, British Columbia
Canada

August, 1997

# Abstract

Volume rendering is superior in many respects to conventional methods of medical visualization. Extending this ability into the realm of telemedicine provides the opportunity for health professionals to offer expertise not normally available to smaller communities, via computer networking of health centers. This report describes such a software system for collaboration, a network-enhanced version of an existing program called Volren. The methods used to provide network functionality in Volren can serve as a prototype for future networked multiuser applications.

# Contents



# List of Figures

# 1 Introduction

The emerging availability of inexpensive high-speed networks is creating interest in a number of new types of software applications, particularly in the areas of science and medicine. Volume rendering, the visualization of blocks of 3D data on screen, is one type of application finding use in a number of fields. In particular, volume rendering has been adapted successfully for use in the visualization of abdominal aortic aneurisms[1]. Given the usefulness of this visualization technique, and new widespread networking capability, the next logical step is to provide a volume rendering application with network capabilities.

Volren is a volume rendering application that takes advantage of the hardware texture-memory capabilities of certain Silicon Graphics computers, allowing it to render 3D volumes in real-time at several frames per second. Volren was modified to implement full networking of all features in the non-networked version. This report describes the new features of Volren, the methods used to network it, and the problems encountered during the development of this new version of Volren.

# 2 New Usage

New command-line parameters were introduced for control of network features:

```
-nets                          // startup as server
-netc <server hostname>        // startup as client
-port <port number>            // Override default port
-dofilter                      // Override disabling of
                                    mouse motion filter
```

Here are some example usages:

```
correct:
      host1% volren -nets                          (server)
      host2% volren -netc host1                    (client)

incorrect:
      host1% volren -nets -port 1500               (server)
      host2% volren -netc host1 -port 2500         (client)
```

The `dofilter` option is a leftover from the debugging process. Its significance will be explained later.
When Volren is loaded with networking enabled, the “‘” key is used for control switching between users. Users may either give control with this key, or take control for themselves. The user not in control may not manipulate the volume or select menu items until a control switch occurs. Permission is not required to take control.

The current control status is indicated in the lower right-hand corner of the Volren window, using pixmaps. On startup, the software looks for two files: `incontrolpm.xpm` and `notincontrolpm.xpm`. If these files are not present in the path, Volren will use a text message fallback to indicate control.

The files named above may be replaced with alternate pixmaps, but the colors will still be green (in control) and red (not in control). Most importantly, they *must* both have the same size! If two pixmaps with different sizes are used, the window may resize during each control switch. This causes the windows on either end of the connection to have different sizes, and volume manipulations will be transferred incorrectly. Loss of synchronization between terminals results.

It is important that the windows have identical sizes on either end of the connection for this reason. If both windows are resized by the user to the same dimensions, no problems will occur. However, resizing of the windows is strongly discouraged until a feature to force identical resizing is introduced (not currently implemented). Position of the windows does not matter, they may be moved around on the screen safely.

## 3 Event-Based Networking

Networking of a windowed application can be divided into two general methods:

**Event-based networking**

- Install an event handler to capture mouse and keyboard events from the transmitting host's application
- Transfer the events across the network, possibly with some supplementary information
- Reproduce the events within the receiving host's application

**Action-based networking**

- Code retrofitted to window callbacks prepares a message containing information about user actions
- Transfer message across the network using a compact, custom message code
- Interpret message, and call the appropriate function(s) to produce user action in receiving host's application

The route taken in the design of this project evolved out of issues specific to X that arose throughout the development period, borrowing from both methods. However, the overall implementation is based mostly on the event-based networking method.

The event-based networking method has several advantages over the action-based networking method. Most importantly, much less code is modified under the first method. An action-based implementation would entail the modification of nearly every user interface callback in the application. Also, in the case of Volren, the software has very complex internal state. As such, any action-based implementation would need to know *how* to modify this state... a very difficult thing to do.

## 3.1 Message Transfer

Data is transferred inside a struct which is written and read from the port. This simplifies the packet transfer because each packet is a standard size. The struct has the following definition:

```
struct NetMessage {
        int mesgType;            // 4 bytes
        int widgetnum;           // 4 bytes
        XEvent event;            // 32 bytes
};
```

Peak event transfer has been measured to be approximately 50 events per second, during constant, and unrealistically rapid, mouse movement. The NetMessage struct contains two ints (8 bytes) and one 32 byte XEvent member[2]. This implies a peak bandwidth usage of approximately 2KByte/s, corresponding to under half the available bandwidth of a single-ISDN line running at 56Kbit/s. This low bandwidth usage provides the software with zero network latency on Ethernet or faster networking hardware. All testing of the software was carried out on 10Mbit/s Ethernet connections. It is useful to note that this leaves the bulk of a mid to high-bandwidth system available for simultaneous video conferencing between users of the software.

When a packet is read from the input socket, it is cast as a NetMessage struct. The client examines the mesgType member, which has the following possible defined values.

```
MESG_NONE
MESG_CONTROL
MESG_EVENT
```

The possible values for the mesgType member have gone through several revisions during development, mostly because of the multiple methods attempted to network the menuing system. The `MESG_NONE` value is used only for error handling and has no function in practice. The `mesg_Type` member was left as an int for ease of addition of future abilities. The `MESG_CONTROL` value denotes a control switch message, and the `MESG_EVENT` value denotes a mouse or keyboard event message.

Upon receival of a `MESG_NONE` message, the client will perform no action. Receival of a `MESG_CONTROL` will cause the client to perform a control switch, and trade roles with the server. When receiving a `MESG_EVENT` message, more processing is involved.

### 3.2 Event Postprocessing

First, the client will examine the `type` member of the XEvent. After finding the true identity of the XEvent (ie: XKeyPressEvent, XKeyReleaseEvent, XButtonPressEvent, etc.), the XEvent is cast as the appropriate variety and certain details of the event are modified. The details of the modifications are almost the same for different XEvent varieties, so I will present only an XMotionEvent as an example. Here is the definition of the XMotionEvent struct[3]:

```
typedef struct {
   int type;                /* type of event (MotionNotify) */
   unsigned long serial;    /* # of last request processed by server */
   Bool send_event;         /* True if this came from a SendEvent request */
   Display *display;        /* Display the event was read from */
   Window window;            /* ``event'' window reported relative to */
   Window root;             /* root window that the event occurred on */
```

```
    Window subwindow;        /* child window */
    Time time;               /* milliseconds */
    int x, y;                /* pointer x, y coordinates in event window */
    int x_root, y_root;      /* coordinates relative to root */
    unsigned int state;      /* key or button mask */
    char is_hint;            /* detail */
    Bool same_screen;        /* same screen flag */
} XMotionEvent;
```

Since the event was captured directly from the server, many of the members in the struct refer to locations in memory that are valid for the server, but not valid for the client. Before the event can be used by the client, these members must be changed to reflect the new environment in which the event will be used.

In this example, the `display` and `root` members would be changed to the display and root window corresponding to the X server[1] on the client machine.

At this point the role of the `widgetnum` member of the `NetMessage` struct becomes apparent. In order to properly modify the `window` member of the event to the window on the client corresponding to the window on the server, both the client and the server must have a common way to refer to widget windows. To accomplish this, each machine creates an array of widget addresses at startup. When sending an event, the server sets the `widgetnum` member of the `NetMessage` struct to the array index corresponding to the widget that produced the event. The client is now able to replace the `window` member of the XMotionEvent with the correct widget window by retrieving the widget from the array and calling `XtWindow()` with the appropriate widget as an argument.

The client can now send the modified event to the correct window, using the `XSendEvent()` function. The widget recieving the event then responds as if the event had been generated by a real device attached to the client machine.

## 4 Menu Networking Difficulties

The most problematic portion of the Volren software to network was the menuing system. Volren is very much a mixed-model software application, a

[1] Note that "X server" does *not* refer to the instance of Volren acting as a network server; it refers to the portion of the X Window System that manages access to screen hardware, keyboards, and mice.

situation arising more often as the X Windowing system continues to evolve. Volren makes use of the X Toolkit and Xlib, OSF/Motif, and ViewKit. This presents problems because Motif and ViewKit are each intended in their own way to be object oriented systems for encapsulation of menu constructs. Motif exists in part to encapsulate X Toolkit menuing in a pre-C++, loosely object oriented API. ViewKit makes this class encapsulation stronger by moving to C++ (although there is a C interface available), and encapsulating code constructs at a higher level than Motif.

### 4.1 Problems with ViewKit's OOPness, and Motif's Pointer Grabs

This mixed-model design of Volren presents several difficulties. Firstly, the event-based networking technique used requires that events are sent to individual widgets in the program. Since ViewKit encapsulates these widgets at a high level, it is a violation of the spirit or intent of the ViewKit package (and OO principles in general) to communicate directly with widgets belonging to ViewKit classes. However, this was the route initially followed to network the menuing system. When implementing event-level code alongside ViewKit, there is just no alternative to bypassing the ViewKit API. The ViewKit API does not implement the necessary functionality. The next section comments more on this topic.

Interacting directly with ViewKit component widgets may not be nice, but we aren't prevented from doing it. Unfortunately, certain details of Motif's menu handling appear to prevent the use of the event-based networking method.

When a Motif pulldown menu is selected, a passive grab is automatically initiated[4].

### 4.2 Why not use the ViewKit API?

Why not use the ViewKit API? Because its design does not allow the type of interaction with menu components that is required. The key problem with the ViewKit API is that it does not allow activation of menu items from within the software. There *are* two functions that come close[5]:

```
void VkMenuToggle::setVisualState(Boolean state)
         - selects the toggle if state is TRUE and de-selects
           the toggle if state is FALSE.

void VkMenuToggle::setStateAndNotify(Boolean state)
```

- sets the visual state of a VkMenuToggle object and activates its associated callback.

But these functions fail to suffice on two counts. First, they both require knowledge of the state to be set, rather than just acting as a "black box".

Unfortunately, there is no "`SimulateMenuSelection()`" function or some such thing. This is important because ViewKit menuing components store state. For example, consider a set of five toggle items with radio behavior (only one may be selected at one time). All state regarding the radio behavior is stored within the ViewKit VkRadioSubMenu component, not within any of its widgets.

### 4.3 A Not-So-Elegant Solution

After many false starts, a solution was arrived upon. Accelerators (keyboard shortcuts) were installed for all the menu widgets. Once again, the ViewKit API was bypassed, and the accelerators were installed by communicating directly with the menu widgets. Code was added to the menu callbacks to synthesize an XKeyEvent to be sent to the client when one of the server's menu items is selected. When the client receives a synthesized XKeyEvent, it will send the event to its top-level window. XKeyEvents received by the top-level window be passed on to all the lower level windows, which will check their accelerator tables, and respond to their installed accelerator events. Although a little bit awkward, this method provides a working method for networking the menuing system.

## 5 Future Work

The implementation presented here provides networked functionality of all features currently available in the non-networked Volren software. However, this release allows only two sites to be linked together in one session. Replacement of the client/server socket code with a more comprehensive mulit-user arbiter module would allow networking between an arbitrary number of sites simultaneously. Arbiter code could be added to the current software with changes only to the current socket code. The control code should work as currently implemented.

## 6 Conclusions

The network-enabled version of Volren presented here fills a need in the medical and science communities for a real-time networked visualization application. Moreover, it demonstrates the viability of networked multiuser software packages running under X. The methods developed in the creation of this package are applicable to the design of other networked X applications. Given the accelerating development of high-speed networks, and the growing interest in building such networks between hospitals, universities, and industrial sites, the development of such applications is an idea whose time has come.

## 7 Acknowledgments

Thanks to Roger Tam, who helped get me acquainted with the ideas behind X Event processing, and provided advice at several points during development. Thanks also to Stan Jang, who wrote the socket code used in this project, which was used in this project with few modifications. Many thanks to Peter Cahoon, who gave me the opportunity to work on this project, and guidance on design issues throughout.

## 8 References

[1] Volume Rendering of Abdominal Aortic Aneurisms
Roger C. Tam, Christopher G. Healey, Borys Flak, and Peter Cahoon

[2] X Protocol Reference Manual for X11 Version 4, Release 6
Edited by Adrian Nye

[3] Xlib Programming Manual for Version 11 of the X Window System
Edited by Adrian Nye

[4] X Toolkit Intrinsics Programming Manual
Edited by Adrian Nye and Tim O'reilly

[5] IRIS ViewKit™ Programmer's Guide
Silicon Graphics, Inc.

[6] Motif Programming Manual for OSF/Motif Release 1.2
by Dan Heller and Paula M. Ferguson

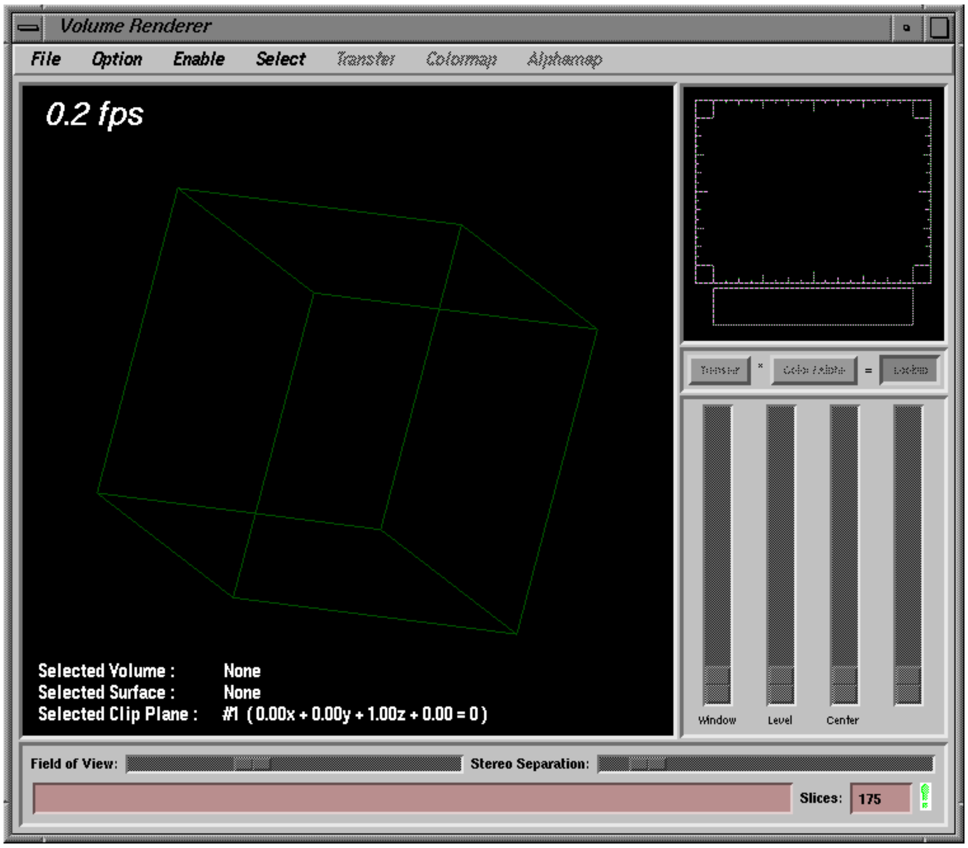


Figure 1: Volren in server mode (in control)

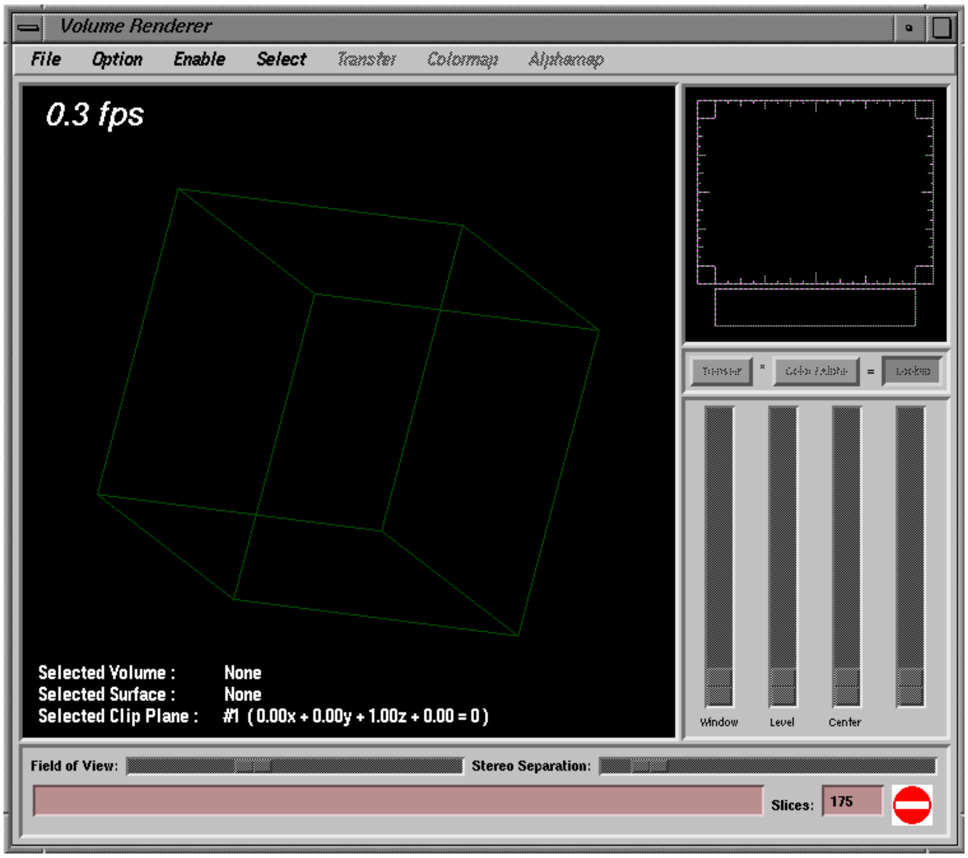


Figure 2: Volren in client mode (not in control)