Module 8

Visualization

The visualization of data using GUI applications should be distinguished from other computer-graphics concerns. In visualization, we are concerned with *exploration* of data, with its attendant concerns of encoding strategies and so on. In computer-graphics, we may be more concerned with rendering techniques.

Before exploring the implementation of a 3D visualization, we look at some aspects of the context within which the visualization is to be used.

8.1 The use of 3D

A successful visual metaphor has some analog with *real-world* physics. Some studies suggest that a 10-fold improvement in item density can be achieved in using three dimensional displays.

The **etherman** display has proven effective in observation of networks with 50 or less nodes, but becomes cluttered and unusable with more nodes on-screen. By extending the display into the third dimension, it immediately becomes clearer. Our familiarity with spatial location allows us to understand that objects further away will be smaller, and this reduces the visual clutter. However if a *far away* object increases in size, we immediately notice, and can rotate the display to observe more closely. We notice even if the *far away* object is still smaller (in screen *real-estate* terms) than closer objects. This human cognitive behaviour becomes apparent as soon as sufficient visual cues have been given to persuade the observer that the display is in three dimensions.

For example: Figure 8.1 shows the output of an original program[1] to display tasks active on a UNIX machine. The size of the spheres indicate the amount of memory used by each process, the colour represents the owner. This display can be rotated and used to examine activity in a way unattainable using standard system process viewing tools. The display has over 100 visible nodes, but it is still easy to identify and investigate individual nodes.

[1] http://opo.usp.ac.fj/~hugh/Public/Viz/ThesisWork/processes1

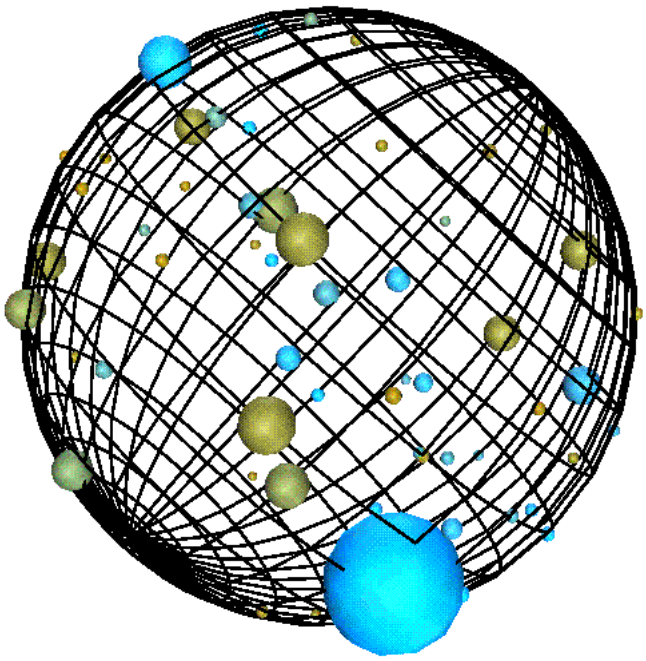


Figure 8.1: Display of tasks in a multi-tasking environment.

8.2 OpenGL

OpenGL was originally the SGI in-house graphics system, but now is the most widely accepted graphics standard, with chip, API and OS support for all platforms. It is possible to code directly using the OpenGL API, but more normal to use a toolkit which encapsulates some abstraction, built on top of OpenGL calls.

Open GL is standard on all UNIXes and all versions of Windows since Win95. The API supports functions for rendering, buffering, anti-aliasing, shading, colouring, texture-mapping, a display list, Z-buffering and so on. To give the flavour of raw OpenGL programming, here is a small application:

CODE LISTING **teapot.c**

```
#include <GL/glut.h>

void
Teapot (long grid)
{
    /* ... code to construct drawlist of teapot here. */
}

static void
Init (void)
{
    glEnable (GL_DEPTH_TEST);
    glLightModelfv (GL_LIGHT_MODEL_LOCAL_VIEWER, local_view);
    /* Lighting model, materials... */
}

static void
SpecialKey (int key, int x, int y)
{
    switch (key) {
     case GLUT_KEY_UP:
        rotX -= 20.0;
        glutPostRedisplay ();
        break;
        /* Move in other directions */
    }
}

static void
Draw (void)
{
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix ();
    /* ... translations ... */
    glCallList (teaList);
    glPopMatrix ();
    glutSwapBuffers ();
}

int
main (int argc, char **argv)
{
    glutInit (&argc, argv);
    type = GLUT_RGB | GLUT_DEPTH;
    type |= (doubleBuffer) ? GLUT_DOUBLE : GLUT_SINGLE;
    glutInitDisplayMode (type);
    glutInitWindowSize (300, 300);
    glutCreateWindow ("TeaPot");
    Init ();
    glutReshapeFunc (Reshape);
    glutKeyboardFunc (Key);
    glutSpecialFunc (SpecialKey);
    glutDisplayFunc (Draw);
    glutMainLoop ();
}
```

8.3 Java 3D, VTK - toolkits for 3D

These systems are 3D OO toolkits embedded in Java and C++ respectively. The Java 3D application programming interface (API) provides a set of object-oriented interfaces that support a simple, high-level programming model.

The Visualization ToolKit (VTK) is an open source OO software system for 3D consisting of a C++ class library, and several interface layers for Tcl/Tk, Java, and Python. VTK has a wide variety of visualization and graphical functions, and has been installed and tested on both UNIX and Windows.

8.4 Case study - network traffic application

A user-requirement specification for a network traffic application may begin with something like:

---

*This visualization is to assist network managers in planning and monitoring their networks. It allows interactive exploration of network datalink traffic, and is intended for use both for visualization of immediate-mode (*real-time*) data, and for visualization of historical data. (The visualization is the same in each case, except that* time only travels forward *in the immediate mode.)*

The visualization will help answer questions such as the following:

- *Which segments carry the most traffic?*
- *Which sections of the network are down?*
- *At what times, and where do traffic bottlenecks occur?*
- *What is the line utilization for different lines at different times?*
- *What types of traffic are used most?*
- *Would routing or switching be effective here?*

For this *network* traffic application, the following elements are represented:

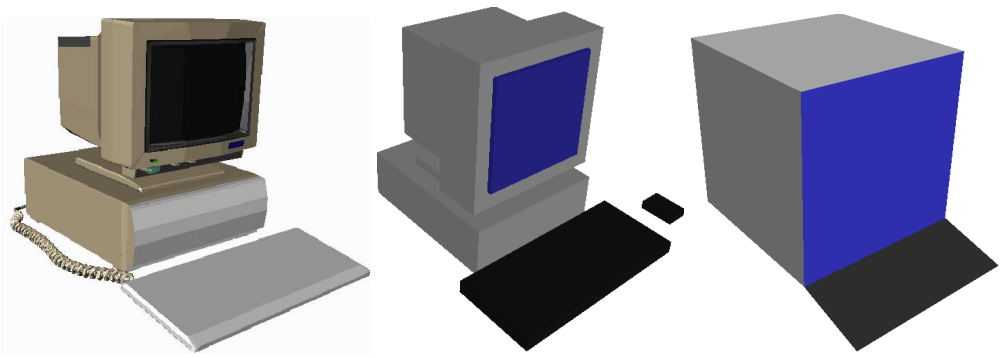
- Background: - to convince the viewer that the display is *three dimensional*...
- Nodes: - a computer, a network device...
- Traffic: - the amount of traffic flow...
- Protocol: - the *type* of traffic...
- Errors: - errors in traffic could be further traffic *protocols*...
- Trends: - for changes over time...
- Association: - for network insights...

8.4.1 Node representation

In our chosen context, the nodes represent computers or network components such as hubs, routers, bridges or switches. In locational or representational displays we may want to differentiate between the type of node, but in the more *abstract* displays, there may be no need to do this.

In Figure 8.2 we see a range of possible options for more concrete representations of nodes.

The computer represented in Figure 8.2(a) has about 2000 flat triangular surfaces (some of them hidden). If we were visualizing a campus with (say) 500 computers using this representation,



(a) 2000 polygons (b) 84 polygons (c) 14 polygons

Figure 8.2: Concrete node representations.

Machine	Rendering speed	Computer (a)	Computer (b)	Computer (c)
Graphics Workstation	485,000 Δ/sec	0.485 frames/sec	11.5 frames/sec	69 frames/sec
PC1	30,000 Δ/sec	0.03 frames/sec	0.71 frames/sec	4.3 frames/sec
PC2	11,000 Δ/sec	0.011 frames/sec	0.26 frames/sec	1.6 frames/sec

Table 8.1: Workstation redraw speed.

then our rastering engine has to recalculate the positions and shading of 1,000,000 polygons each time it redraws the screen. This will happen even if the item is so *far away* that it only takes up a single pixel on the screen.

A typical modern hardware rastering engine can calculate 485,000 shaded Δ/sec (triangles per second), and hence our screen refresh rate would be about half a frame per second, giving a *jerky* look. By contrast, the computer shown in Figure 8.2(c) has only 14 flat triangular surfaces, giving a frame rate in excess of 70 frames per second.

Standard PCs often come with graphics cards that support pixel movement on screen, but their overall performance in shaded Δ/sec is normally considerably below 485,000 polygons per second. Table 8.1 gives the resultant frame rates for displaying onscreen 500 of the node representations in Figure 8.2.

It is clear from this table that if we wish our visualizations to be viewed on a range of platforms, we must choose our node representations carefully to minimize rendering time.

Some representation methods for three dimensional objects allow different *levels of detail*. In the VRML specification, a single object may be represented in different ways depending on how much screen *real estate* it uses up. If the object is near you, it could be represented in detail, but if it is a long way away, the representation could be as simple as a coloured square.

The following VRML code represents a cone in two ways using an LOD (Level Of Detail) node. If the distance from the user to the object is smaller than the first range value specified, then the first version is drawn. If the distance is greater than the last range specified, the last version is drawn.

```
#VRML V2.0 utf8
LOD {
    range [20]
    level [
       #full detail 16 sided cone
       Shape{
          appearance Appearance { material Material { diffuseColor 1.0 1.0 1.0}}
          geometry Extrusion{
             crossSection [ -1 0, 0 0, -1 -2 -1 0]
             spine [1 0 0 , 0.866 0 0.5, 0.5 0 0.866, 0 0 1 , -0.5 0 0.866, -
0.866 0 0.5,
                   -1 0 0, -0.866 0 -0.5, -0.5 0 -0.866, 0 0 -1 ,0.5 0 -0.866,
                    0.866 0 -0.5, 1 0 0]
          }
       }
       #low detail 4 sided cone, actually a pyramid
       Shape{
          appearance Appearance { material Material { diffuseColor 1.0 1.0 1.0}}
          geometry Extrusion{
             crossSection [ -1 0, 0 0, -1 -2 -1 0]
             spine [1 0 0 , 0 0 1, -1 0 0, 0 0 -1 , 1 0 0]
          }
       }
    ]
}
```

8.4.2 Traffic and protocol representation

A simple immediate way to represent traffic between two nodes is to just draw a line between them. The nature of network communication on a typical Local Area Network (LAN) is such that the resultant lattice is likely to be relatively sparse.

For example: at the datalink layer, on average, a workstation at any one time may only be communicating with six or seven other datalink addresses - two broadcast addresses, (say) two file servers, a WINS or DNS server. and a proxy. So - rather than having a lattice with $\frac{n^2-n}{2}$ interconnections, we have $k(n-1)$ interconnections, where k is some small integer. Even so, a lattice with 500 nodes may have 3,000 interconnections and may look jumbled.

A line indicates source and destination, but not the *amount* of traffic. Three systems for this purpose have been examined:

1. Colour coding (black through red to white for maximum traffic),
2. Line width, and
3. The length of partial lines, as discussed in Eick's papers.

Using a linear increase in the line width appears most effective, although it does increase clutter. It also leaves the colour information free for use in some other encoding. The linear scale needs a sensible maximum, and experimentation has shown that a maximum width should be equivalent to the size of the node.

A simple line or cylinder also does not tell which way the traffic is flowing. We evaluated the following cues by modeling them in ***geomview***, a geometrical modelling package.

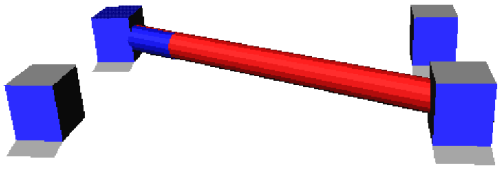


Figure 8.3: Partial length representation of bi-directional traffic.

1. Separate arrows
2. Partial lengths

In Figure 8.3 we see the traffic between two computers, the size of the cylinder between the machines indicating the total amount of network traffic, and the two colours indicating the relative amounts of traffic going each way. The nodes and cylinders themselves are coloured according to the dominant protocol type.

8.4.3 Trend representation

Trends are sometimes difficult to find in large sets of data such as found in our application. Once an examination of a visualization has indicated that a trend may be possible, it is normally easy to frame the questions needed to verify the trend.

- *"It looks like HTTP usage on these segments is increasing..."*
  (⇒Plot HTTP usage for the segment machines versus time).
- *"It looks like HTTP usage is increasing when FTP usage is decreasing..."*
  (⇒Plot HTTP and (1-FTP) versus time).

Graphing continues to be the pre-eminent way of representing trends and the role of visualization is to assist in finding the trends.

The four-dimensional visualization methods outlined and demonstrated by Olaf Holt and Nils McCarthy in NDdemo (the fourth dimension being explicit time) could perhaps be used in trend analysis, but the visualization is a little hard to use.

A final method is to attempt to encode previous visualizations *on-top-of* the current one (but perhaps semi-transparent) - the idea here is one of *visual* echoes. In only some circumstances can this be successful. There are two options:

1. Echoes are fixed on the screen, and we can move the visualization away from them, leaving a trail like this:
   In the worst case though, we have just ended up using one of our three display dimensions for *"time"*.

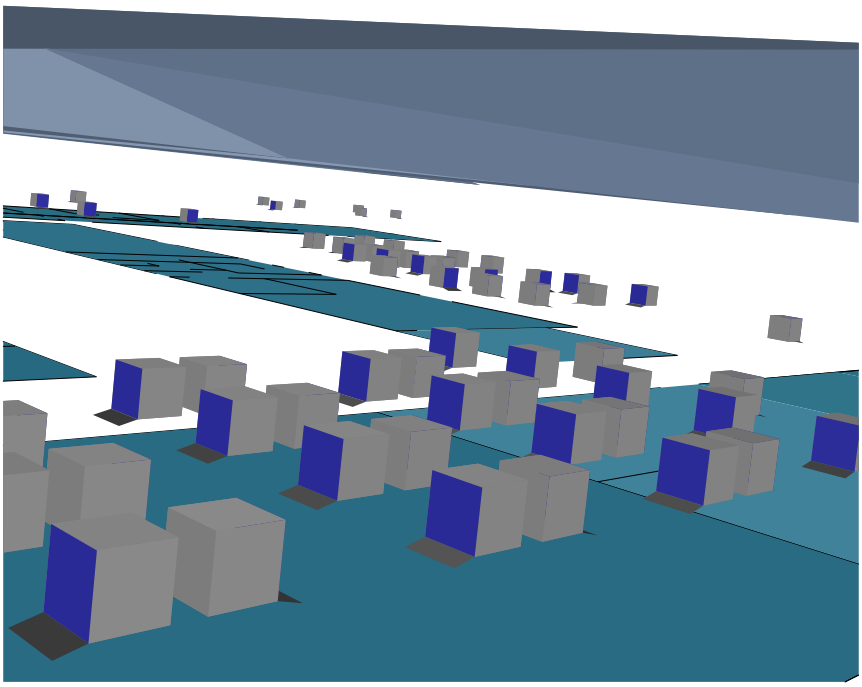


Figure 8.4: Locational view

2. Semi-transparent echoes are co-located with the visualization. In this case, we can only show some of the history. We can show a reducing item, but not an increasing one.

8.4.4 Display

In Figure 8.4, we see an early locational view showing the fixed components of the visualization, and modeled using **geomview**. It shows nodes for the computers, floor plans for the buildings, and a transparent roof. The display uses the normal 3D navigation tools for adjustment. From the display it is easy to identify the location of machines, and the display should be efficient enough to support display and manipulation of the entire network (with 500 machines as a suitable goal), and - yes - the computers are floating in mid air. (Since we are concerned with efficiency we choose the simplest understandable visualization, and tables just become extra polygons to draw).

The visualizing tool supports rotation and translation of the display, so that the observer can easily focus on regions of interest. Suitable systems are found in the CosmoPlayer VRML viewer, and in **geomview**. Note that this visualization is not dependant on the navigation or implementation method.

Cables and network infrastructure are not marked on the display, but the display does support an aggregation-by-rule construct. This aggregation can be used to associate machines all on the same segment, or all used by the same department.

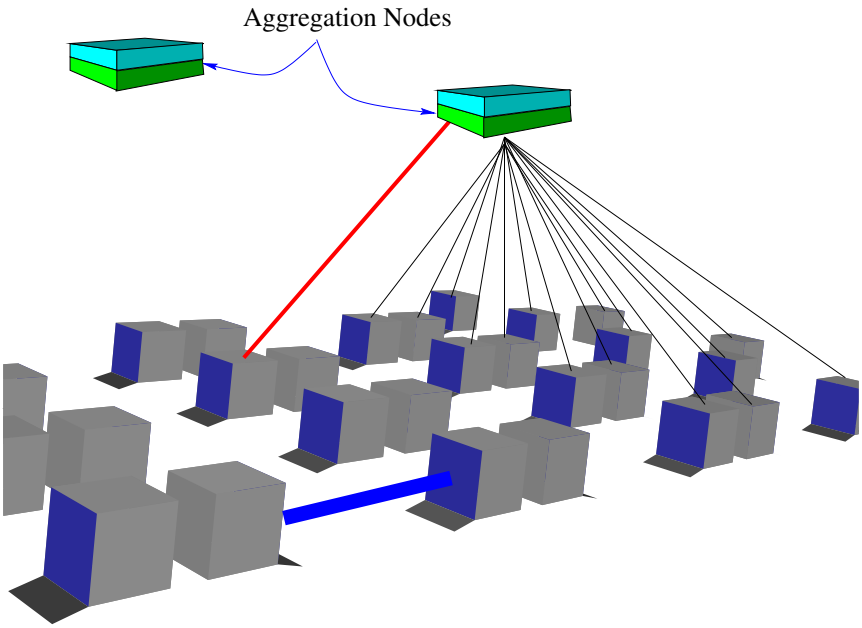


Figure 8.5: Aggregation nodes

The components are chosen to be the best minimum complexity representation consistent with fast updates. The frame update speed for the most complex display is better than 2 frames per second.

Each node or aggregation in the overview display is clickable to turn it off or on. If a node is turned off, its traffic no longer is displayed (either directly or as part of some aggregated traffic). If an aggregation is turned off, any existing traffic displays to its nodes are removed. This facility is used to allow fast reduction of visual clutter.

Aggregation nodes also have an aggregation *switch*, which allows them to combine all traffic for subsidiary nodes. When this switch is on, lines connect the aggregation to its subsidiary nodes.

The aggregation node floats above its associated nodes. In Figure 8.5, we see two aggregation nodes, with the one on the right aggregating traffic to and from all its subsidiary nodes. All traffic to or from these nodes is displayed going to the aggregation node. The other nodes are not aggregated, and display traffic directly.

Each node or aggregation in the overview display is clickable to identify specific information about that node. This information does not replace the display, but appears in a separate window. Initially this information may just be textual information such as the name of the node along with traffic totals, but eventually, it is expected that the drill-down display will be the metaphor display specified elsewhere, showing only the selected node in 3D, along with any associated nodes.

Operating System	Web Browser	VRML 2.0 Plugin
IRIX	Navigator 3.01S	CosmoPlayer 1.0.2b3 or later
	Communicator 4.04	CosmoPlayer 1.0.2b3 or later
	Communicator 4.07	CosmoPlayer 2.1 beta
Macintosh	Communicator 4.04	CosmoPlayer 2.1 or later
WIN32	Navigator 3.01	CosmoPlayer 1.0 beta 3 or Intervista WorldView 2.0 or later
	Communicator 4.04	CosmoPlayer 1.0 beta 3 or Intervista WorldView 2.0 or later
	MSIE 3.0	CosmoPlayer 1.0 beta 3 or Intervista WorldView 2.0 or later
	MSIE 4.0	CosmoPlayer 1.0 beta 3 or Intervista WorldView 2.0 or later

Table 8.2: Systems which support the EAI.

8.5 3D VRML visualization implementation

The VRML visualizer is a relatively small Java program which must be loaded as an applet along with a VRML view of the network. A small web page is created, and may be used to view the visualization using a web browser such as Netscape along with the CosmoPlayer VRML plugin.

Unfortunately, not all combinations of web browser and VRML plugin work correctly with the EAI, but the systems in Table 8.2 are known to work. These systems were current in 1999. This year (2002) all the systems I tried at NUS appeared to work fine.

Load the default web page in the directory, and the visualization should be visible. To finish using the visualizer, you must exit the browser entirely. If not, the Java applet keeps communicating with the **collector**.

In Figure 8.6, we see an active VRML display within a browser. The computer nearby is generating a lot of traffic. In the distance we can see other nodes, and the roof and floors.

8.5.1 3DVNT VRML software

3DVNT includes software to create a default HTML web page for the VRML visualization. The current default web page is like this:

```
<html><head> <title>Sample 3DVNT Page</title> </head>
<center><H1>Sample 3DVNT Page </H1></center>
<center> <embed src="root.wrl" height="600" width="700"> </center>
<center> <applet code="View1.class" width="100" height="10" mayscript>
<PARAM name="segment" value="MACS"> <PARAM name="port" value="9876">
<PARAM name="host" value="opo.usp.ac.fj"> </applet> </center>
OK?
</html>
```

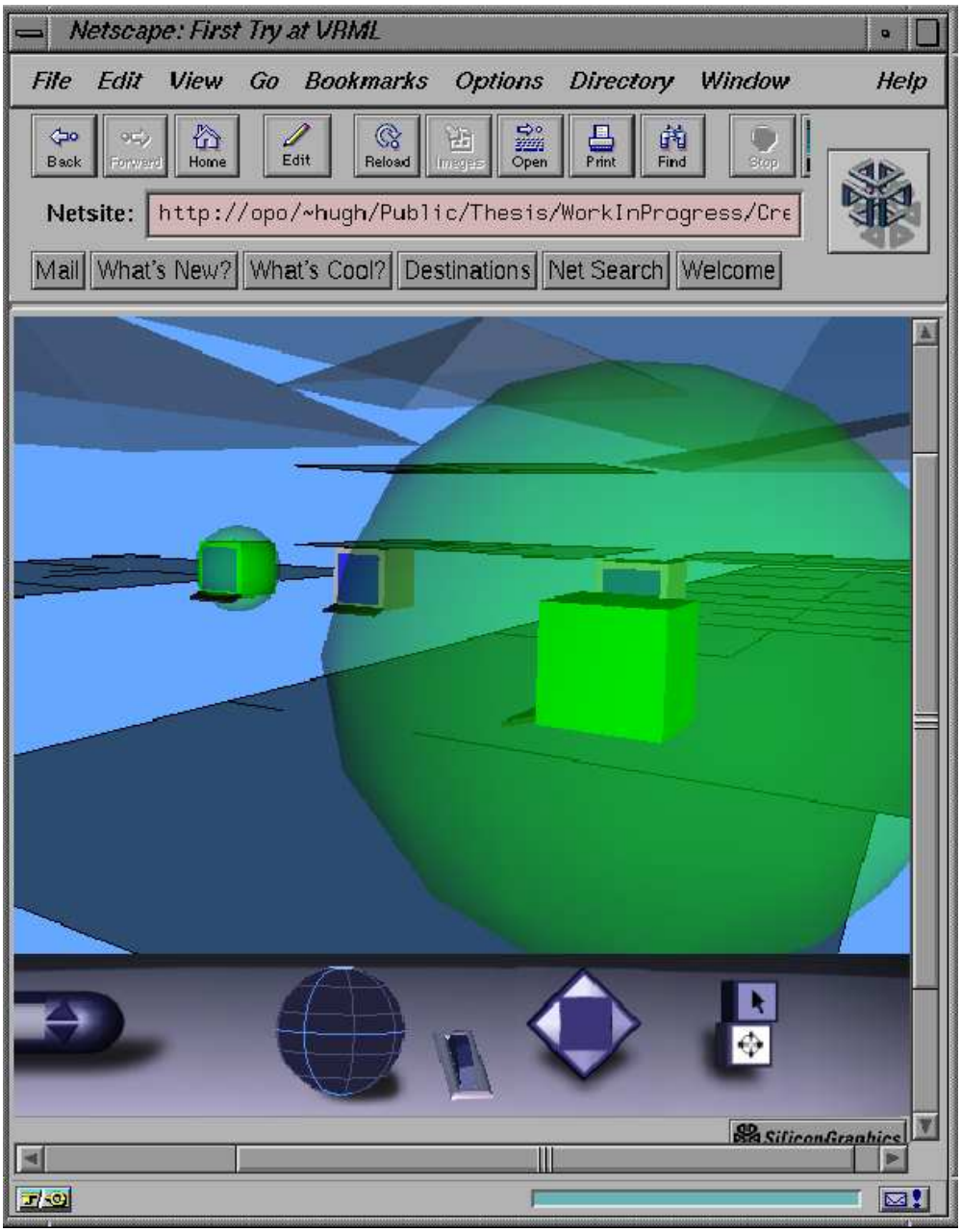


Figure 8.6: 3DVNT view within Netscape browser.

The *root.wrl* file which forms the basis of the VRML visualization is of the following format:

```
PROTO CLUSTER  [] { ... }     # Cluster definition
PROTO KEYBOARD [] { ... }     # Keyboard definition
PROTO SCREEN   [] { ... }     # Screen definition
PROTO GLOBE    [] { ... }     # Traffic sphere definition
# Some setting up declarations
Background { skyColor .4 .66 1 }
NavigationInfo { type [ "EXAMINE", "ANY" ] speed 400 }
Viewpoint { position 0 400 0 orientation 0 1 0 4 description "Camera 1" }
# Lines, floors and roofs
DEF LINES  Transform { ... }
DEF FLOORS Transform { ... }
DEF ROOFS  Transform { ... }
# and then the nodes
DEF node1  Transform { ... }
DEF node2  Transform { ... }
# ... and so on ...
```

Each node is of the following form:

```
DEF node1 Transform {
    translation 4350 150 4365
    rotation 0 1 0 4.71238
    children [
       KEYBOARD {}
       SCREEN {}
       DEF node1box Transform {
          children [
             Shape {
                appearance Appearance { material DEF node1boxcolor Material { diffuseColor 0.8 0.8 0.8 } }
                geometry Box { size 50 50 50 }
       }  ]  }
       DEF node1sphere Transform {
          scale 1 1 1
          children [
             Shape {
                appearance GLOBE {}
                geometry Sphere { radius 1 }
}  ]  }  ]  }
```

The Java visualyzer software maintains a link to a remote data collector, and uses the EAI to modify the images in the VRML view.

Printed by Hugh Anderson

Mar 05, 99 11:51 **View1.java** Page 1/3

```
// using the VRML External Interface.


import java.applet.*;
import java.awt.*;
import java.util.*;
import vrml.external.field.*;
import vrml.external.exception.*;
import vrml.external.Node;
import vrml.external.Browser;
import java.io.*;
import java.net.*;

public class View1 extends Applet {
//  public static final int DEFAULT_PORT = 9877;
  Browser browser;
  Socket s = null;
  DataInputStream in = null;
  String line;

  public void init() {
    System.out.println("Test.init()...");
  }
  void SocketStart  () throws java.io.IOException {
    String port = this.getParameter("port");
    int p = Integer.parseInt(port);
    try {
      String host = getCodeBase().getHost();
      System.out.println("Request came from: " + host);
       s = new Socket(host, p);
    }
    catch (UnknownHostException e) {
      System.out.println("No socket: " + e);
    }
  }
  public void start() {
    int count=0;
    Node node2sphere=null;
    Node appear=null;
    EventInSFVec3f[] scalein=new EventInSFVec3f[100] ;
    EventInSFColor[] appears=new EventInSFColor[100] ;
    float[] val = new float[3];
    int[] lastval  = new int[100];
    int n;
    String id,vl;

    while (count != 100) {
       scalein[count] = null;
       appears[count] = null;
       lastval[count] = 0;
       count=count+1;
    }
    try {
       SocketStart();
     }
    catch (java.io.IOException e) {
      System.out.println("No socket: " + e);
    }

    System.out.println("Test.start()...");
    browser = (Browser) vrml.external.Browser.getBrowser(this);
    System.out.println("Got the browser: " + browser);

    count = 0;
        try {
            in = new DataInputStream(s.getInputStream());
```

Thursday August 26, 1999 1/3

Printed by Hugh Anderson

Mar 05, 99 11:51 **View1.java** Page 2/3

```
            while(true) {
                line = in.readLine();
                if (line == null) {
                    System.out.println("Server closed connection.");
                    break;
                }
                if (line.regionMatches(0,"n",0,1)) {
                   n = line.indexOf(32,2);
                   id = line.substring(2,n);
//                   System.out.println(">>>"+id+"<<<");
                   vl= line.substring(n+1);
//                   System.out.println("+++"+vl+"---");
                   Integer a = Integer.valueOf(id);
                   Integer b = Integer.valueOf(vl);
                   if (scalein[a.intValue()]==null) {
                      try {
                         node2sphere = browser.getNode("node"+id+"sphere");
                         System.out.println("Got the sphere node: " + node2sphere);
                      }
                         catch (InvalidNodeException e) {
                         System.out.println("PROBLEMS! node2sphere: " + e);
                      }
                      try {
                         scalein[a.intValue()] = (EventInSFVec3f) node2sphere.ge
tEventIn("scale");
                         System.out.println("Got the sphere scale node: " + appears[a.in
tValue()]);
                      }
                      catch (InvalidNodeException e) {
                         System.out.println("PROBLEMS! (scalein): " + e);
                      }

                      try {
                         appear = browser.getNode("node"+id+"boxcolor");
                         System.out.println("Got the Boxcolor node: " + appear);
                      }
                         catch (InvalidNodeException e) {
                          System.out.println("PROBLEMS! appearance: " + e);
                      }
                      try {
                         appears[a.intValue()] = (EventInSFColor) appear.getEven
tIn("set_diffuseColor");
                         System.out.println("Got the Boxcolor color node: " + appears[a.i
ntValue()]);
                      }
                         catch (InvalidNodeException e) {
                          System.out.println("PROBLEMS! appearance color: " + e);
                      }
                   }
                   if (b.intValue()==-1) {
                      val[0] = (float)1.0;
                      val[1] = (float)1.0;
                      val[2] = (float)1.0;
                   } else {
                      val[0] = (float)(b.intValue()*20)+1;
                      val[1] = (float)(b.intValue()*20)+1;
                      val[2] = (float)(b.intValue()*20)+1;
                   }
                   scalein[a.intValue()].setValue(val);

                   if ((b.intValue()==0) != (lastval[a.intValue()]==0)) {
                      if (b.intValue()==0) {
                         val[0] = (float)0.8;
                         val[1] = (float)0.8;
                         val[2] = (float)0.8;
                         appears[a.intValue()].setValue(val);
```

2/3 Thursday August 26, 1999

Printed by Hugh Anderson

Mar 05, 99 11:51 **View1.java** Page 3/3

```
                        } else {
                           if (b.intValue()==-1) {
                              val[0] = (float)0.1;
                              val[1] = (float)0.1;
                              val[2] = (float)0.1;
                              appears[a.intValue()].setValue(val);
                           } else {
                               val[0] = (float)0.0;
                               val[1] = (float)1.0;
                               val[2] = (float)0.0;
                               appears[a.intValue()].setValue(val);
                           }
                        }
                     }
                     lastval[ a.intValue()]=b.intValue();

                  }
//                  System.out.println(line);
            }
         }
         catch (IOException e) { System.out.println("Reader: " + e); }

     }


  public Browser getBrowser() {
    return browser;
  }

}
```

Thursday August 26, 1999 3/3

8.6 Summary of topics

In this module, we introduced the following topics:

- Visualization versus computer-graphics
- OpenGL
- (Briefly) Java3D, VTK
- VRML/Java/EAI

Tutorial 7 - questions for week 13 (April 3, 2002)

1. Find a minimal VRML file which constructs a solid cube.
2. Find minimal OpenGL display-list code to draw a cube.
3. Examine the Java code given for using the EAI to modify the VRML. How exactly does the code get a reference to a VRML node?
4. Examine the Java code given for using the EAI to modify the VRML. How exactly does the code modify a VRML node?

Further study

- sunsite for Java3D
- The EAI specification