Working with Remote VRML Scenes through Low-Bandwidth Connections

Alberto Barbosa Raposo[1], Léo Pini Magalhães[1,2], Ivan Luiz Marques Ricarte[1]

[1] State University of Campinas (UNICAMP)
School of Electrical and Computer Engineering (FEEC)
Dept. of Computer Engineering and Industrial Automation (DCA)
CP 6101 — 13083-970 — Campinas, SP, Brazil
Phone: +55-19-239-8385 — Fax: +55-19-239-1395
alberto, leopini, ricarte@dca.fee.unicamp.br
[2] University of Waterloo — Computer Science Dept. — Computer Graphics Lab.
200 University Ave. W Waterloo, On N2L 3G1, Canada
Phone: +1-519-885-1211 ext. 2041 — Fax: +1-519-885-1208
lpini@cgl.uwaterloo.ca

Abstract. We have developed a Web-based application to accelerate the visualization of VRML scenes located in a remote server. This application enables the user to extract only the parts of a scene that are of actual interest. The extracted parts represent one or more sub-trees of the hierarchical structure of the VRML scene, and only these parts will be rendered and visualized in the local computer. By reducing the complexity (size) of the remote scene, less data are transmitted from the remote server and the rendering process becomes faster in the local computer. The application is written in Java and is executed as an applet embedded in an HTML page.

1 Introduction

The World-Wide Web (WWW) [1] has integrated and expanded the services offered by the Internet and other networks, with an easy-to-use user interface, responsible for the impressive growth of the global information space. Furthermore, the number of WWW users is increasing rapidly and the information they need is becoming more and more complex, requiring new ways of communication and interaction with these networks. The challenge is to reach the Global Information Infrastructure (GII), considered as a step beyond the WWW, which will allow more *efficient interaction with the information* and will be *globally accessible* [2, 6].

Regarding the *interaction with the information*, computer graphics plays a crucial role, since it can provide efficient ways to visualize and interact with a large amount of data. Particularly concerning the GII, the Virtual Reality Modeling Language (VRML) [15] has emerged as a promising tool. It describes interactive 3D objects and worlds in the WWW just as HTML describes hypertext pages, allowing hyperlinks to other VRML worlds or media, such as sound, animation, and hypertext. Furthermore, VRML is deeply integrated with Java, probably one of the dominant languages for GII applications.

Regarding the *global accessibility*, mobile computing and wireless networks are key technologies, since only they can effectively provide access to anyone, anywhere and at anytime. However, these technologies face two major problems, namely the low bandwidth of wireless networks and the limited processing power of mobile devices.

Our paper presents an application whose main goal is to reduce the size of VRML files while preserving most of the relevant information. By reducing the file size, the visualization of VRML scenes is accelerated due to two effects: lower transmission time and faster rendering for a simpler scene. It is a further small step towards the GII. On the other hand, the application has also the interesting capability of mixing elements of different VRML worlds. In this way, the user can read several worlds, selecting and combining elements from each of them.

In the next section we introduce the VRML and its features. In Section 3 we describe our application and in Section 4 we present analytical results that illustrate the benefits that can be obtained with this application. Finally, in Section 5, we present the conclusions and future works.

2 VRML

VRML [9, 15] is a file format to describe highly interactive 3D graphic environments, allowing the user to define static and animated worlds, and to interact with them.

The current version of VRML supports interaction by the definition of sensors and detectors, such as time sensors, touch sensors, and collision detectors. Animation resources in this version includes keyframe and scripting facilities that support custom protocols for many scripting languages, specially Java [3] and JavaScript [10].

The work in conjunction with these programming languages opens many possibilities for VRML. It can be used, for instance, to query databases and display results, to interact with users via pop up menus, and to work in multiuser environments [7].

In order to view a VRML scene a specific browser is necessary. Such browser may be an independent program, a Java applet, or a plug-in to a conventional Web browser. Some available browsers are Silicon Graphic's Cosmo Player [13], Sony's Community Place [14], and Liquid Reality [5].

The structure of the objects in a VRML file is highly hierarchical. The transformations applied to an object are applied to all its descendants. Our application follows this approach, enabling the user to select or to remove whole sub-trees as a single object.

3 Extracting Elements of a VRML Scene

The target application is mainly directed to a scenario where the users are connected to the Internet via a low-bandwidth communication channel (e.g., dialup or wireless connection), with typical transmission rates about 20Kbps. In this scenario, the user requests the visualization of a VRML scene located somewhere in the WWW. The description of the requested scene should be retrieved, rendered, and displayed on the user's computer (*client*). However, due to the low bandwidth of the connection and also due to the restricted power of the client computer (a laptop or a Network Computer [4]), we need intelligent strategies for the interactive handling of the scenes. Simply stated, the straightforward approach of retrieving the scene description and rendering on the client is not an adequate solution, specially if we think of more complex scenes.

In order to deal with these problems, we have developed an application using the strategy of reducing the amount of data to be transmitted and rendering on the client only the parts of the scene that are really of interest. In this way, less transmission is required (adequate to the low-bandwidth connection) and the rendering process is simplified (adequate to the limited resources of the client).

The application defines three main entities (Figure 1): client, application server and information server.

The *client* is the user's computer, that downloads an HTML page containing the application's interface (a Java applet).

The *application server* is the application's host, where all the application is executed (except the user interface). The application server has two main goals:

- to execute most part of the data reduction process, reserving for the client only the user interface, and

- to overcome security restrictions of some Web browsers (e.g., Netscape), allowing the user to work with VRML scenes located in other servers besides that of the application (the application server acts as a proxy server [8], a service to grab information from the WWW and send to whoever has requested it).

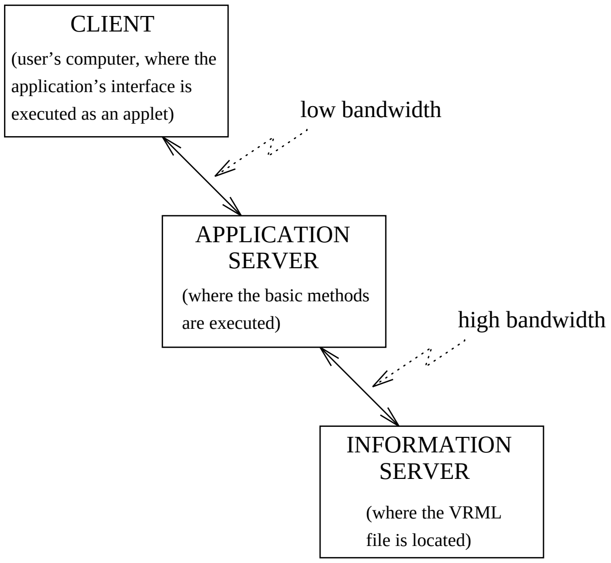

CLIENT
(user's computer, where the application's interface is executed as an applet)
low bandwidth
APPLICATION SERVER
(where the basic methods are executed)
high bandwidth
INFORMATION SERVER
(where the VRML file is located)

Figure 1: The entities of the application.

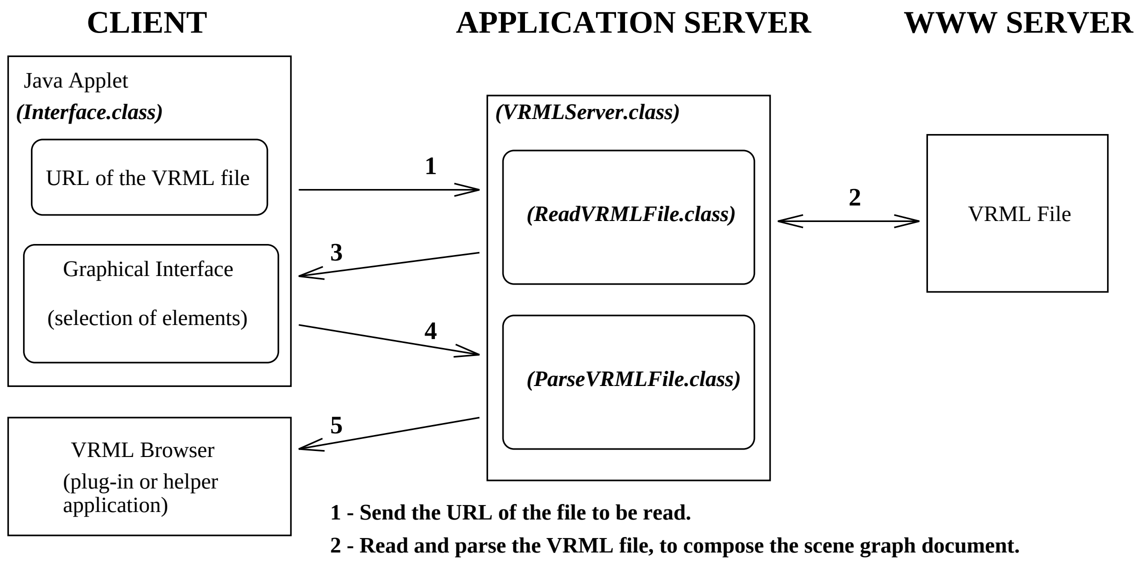
The third entity defined by our application is the *information server*, where the VRML file is located.

The client is connected to the application server via a low-bandwidth communication channel, while the application server is connected to the information server via a high-bandwidth channel (in fact, both servers may even be the same computer).

The application server is capable of requesting and receiving a VRML scene from the information server, parsing it and sending its hierarchical structure to the client. The client then provides an interface adapted to that hierarchical structure, enabling the user to select only the elements he/she wants to visualize. The selected elements are then sent to the application server, which parses the the original VRML file and extracts from it only the desired elements, sending a valid "sub-VRML" scene to the client, which can finally visualize it. This process is illustrated in Figure 2.

3.1 Tasks Sequence

The application is initialized when the user activates it using any Java-enabled WWW browser. At this moment, a connection is established between the client and the application server. After that, the user can specify the URL of the desired VRML scene to the application server (this



1 - Send the URL of the file to be read.
2 - Read and parse the VRML file, to compose the scene graph document.
3 - Send the scene graph document (with the elements' hierarchy).
4 - After the selection, send the document with the desired elements of the scene.
5 - Send the "sub-VRML".

Figure 2: Sequence of tasks in the use of the application.

is represented by arrow labelled 1 in Figure 2). The application server then uses Java resources to connect with the information server, reads the VRML file and parses it (arrow 2 in Figure 2), composing the document representing the hierarchy of the elements of the VRML scene (we call this document "scene graph document"). This document is then sent to the client (arrow 3).

Based on the scene graph document, the client's applet creates an interface that allows the user to select the desired elements of the scene. After this selection, a new document is sent to the application server, describing the selected elements (arrow 4 in Figure 2). Using this new document, the application server creates a new VRML file from the original one, by extracting only the desired parts of it. This final sub-VRML scene is then sent to the client (arrow 5), that visualizes the results using any VRML browser connected to the Web browser (as a plug-in or helper application).

3.2 Java Classes

The application, as shown in Figure 2, is composed of four main Java classes. The application server is divided into three main classes (*VRMLServer.class*, *ReadVRMLFile.class*, and *ParseVRMLFile.class*) and the client has one class (*Interface.class*).

The *VRMLServer.class* represents the application server's main program. It has an infinite loop that waits for a connection on a specified port. When this connection occurs, it creates a new thread[1], which deals with the client communication protocol, reading the VRML file, translating it into the scene graph document, and doing the final parsing of the VRML (i.e., removing the undesired parts of the original file), as requested by the client. The creation of a new thread when the client connects to the server enables multiple simultaneous client connections, each one treated separately.

The *ReadVRMLFile.class* is the program that effectively reads the remote VRML file and translates its structure into the scene graph document to be sent to the client.

The *ParseVRMLFile.class* is used in the final steps of the process, after the user has selected the desired objects of the scene. This class receives the information about the selected objects from the client, parses the original VRML file (stored in a global variable since the moment it was read by *ReadVRMLFile.class*), and composes the sub-VRML to be visualized in the client. This sub-VRML file is stored in the application server having the name `subVRML`*ddmmyyhhiiss*`.wrl`, where *ddmmyy* is the current date (day, month, year) and *hhiiss* is the time (hour, minutes, seconds) when it was created.

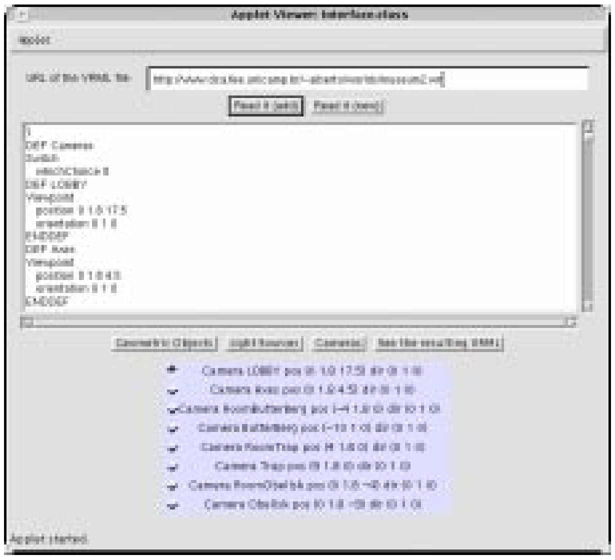
Interface.java is the program being executed at the client. It is an applet embedded in the application home page. When this applet is initialized, it opens a socket

[1]Java supports multithreaded programming, i.e., it allows programs to have many threads of execution at the same time, each thread carrying its local data and sharing global information. This is an important facility to multitask and distributed computing.

connection to the application server, that should be waiting for a connection (infinite loop of *VRMLServer.class*). At this moment, the applet also creates a simple user interface, with which the user will be able to select the VRML scene and the objects of this scene he/she wants to see.

3.3 The Applet Interface

On the top of the applet window (Figure 3) there is a text field to write the URL (Uniform Resource Locator) of the VRML file the user wants to work with. When this name is correctly written in the text field, the user can click one of the two buttons labelled *Read it* to send this URL to the application server. After that, the application server will read that VRML file, parse it, and send to the client the document with the hierarchical structure of the VRML file.



```
1
DEF Cameras
Switch
  whichChoice 0
DEF LOBBY
Viewpoint
  position 0 1.6 17.5
  orientation 0 1 0
ENDDEF
DEF Axes
Viewpoint
  position 0 1.6 4.5
  orientation 0 1 0
ENDDEF
```

Camera LOBBY pos (0 1.6 17.5) dir (0 1 0)
Camera Axes pos (0 1.6 4.5) dir (0 1 0)
Camera RoomButterberg pos (−4 1.6 0) dir (0 1 0)
Camera Butterberg pos (−10 1.6 0) dir (0 1 0)
Camera RoomTrap pos (4 1.6 0) dir (0 1 0)
Camera Trap pos (9 1.6 0) dir (0 1 0)
Camera RoomObelisk pos (0 1.6 −4) dir (0 1 0)
Camera Obelisk pos (0 1.6 −9) dir (0 1 0)

Figure 3: Interface of the developed application.

When the client receives the scene graph document, it is shown in the large text area below the two buttons. The document proposed to send the information regarding the hierarchical structure of a VRML scene is simply a text file that represents each hierarchical level by a number. Every time the word *children* is found on the VRML file, a deeper hierarchical level is defined and when the definition of the child is finished, we come back to the upper level. Besides the information about the hierarchical level, the document gives information about the objects of the scene (here, the term “objects” refers to geometric objects, light sources and cameras). For example, it contains information such as the geometry of the objects, the position and direction of light sources, being a summary of the VRML file. This information, although not directly necessary to select the sub-trees, can be used to help the user in knowing with which object he/she is dealing and to facilitate future extensions of the application — for example, the client could be able to change the characteristics of an object. To transmit this information, the scene graph document uses the same keywords as the VRML, thus ensuring consistency.

The scene graph document is shown in the applet only for demonstration purposes, since the user does not have to work on it. Actually, the user does not even have to know about the existence of this document.

The user can select the desired objects of the scene using the buttons below the text area. The user should click the button *Geometric Objects* to select among the geometric objects of the scene, *Light Sources* to select among the light sources, and *Cameras* to select among the cameras. When one of these buttons is clicked, a colored panel appears, with the name of all the objects of that kind in the first hierarchical level. If there are objects of the selected kind in a deeper hierarchical level, a button *Next Level* can be used to change the panel contents to the names of the objects in the next level.

By default, all the objects are chosen; the user has to click only on the objects he/she does not want to visualize. If the user deselects (selects) an object, all its descendants will be automatically deselected (selected).

Once the user has deselected the objects he/she does not want to visualize, he/she can click the button *See the resulting VRML* to visualize the created sub-VRML scene. At this moment, the user’s work is finished; he/she must only wait for the image.

After the button *See the resulting VRML* is activated, the client rewrites the scene graph document, adding the words *true* or *false* after each object, indicating whether the object is selected or not. This new document is sent to the application server, which will create the sub-VRML scene based on it (*ParseVRMLFile.class*).

A final observation about the applet interface should be made, regarding the difference between the two buttons *Read it*. The button *Read it (new)* makes the application server read the VRML as a new file, throwing away the information about the sub-VRML scene previously created. The button *Read it (add)* makes the application server maintain that information, merging two sub-scenes. In this way, the application allows objects of different VRML scenes to be joined in another scene — the tool can be used not only as an object-extraction tool, but also as a merging tool.

A detailed description of this application is found in [12].

3.4 Instancing and Animated Scenes

In particular, two features of VRML require special attention during the extraction of objects from a VRML file.

The first one is the *instancing*, that allows an object to be named and referred again later in the file. For example, a red ball can be defined as `DEF RED_BALL <definition>` and later a similar red ball can be created by simply writing `USE RED_BALL`. If the user (using our application) wants to exclude the first ball from the final visualization and include the second one, a problem might occur, since the definition of the ball was excluded. To avoid this problem, every time a definition object (i.e., an object following the `DEF` key word) is excluded from the scene, its name and its definition are stored in a table we called "unused DEFs table". Then, every time the parser founds a referred object (i.e., an object following the `USE` key word) included in the final sub-scene, it checks that table. If the object is in the table, its definition has not been included in the sub-VRML file yet; then the reference (`USE`) is substituted by the definition (`DEF`) in the new file, and that element is removed from the unused DEFs table (meaning that its definition is now in the file, and future `USE`s of it will not have that problem). In this way, the user can exclude each object individually, not worrying about possible future instances of it.

Another aspect of VRML that should be taken into account during the selection of objects is related to animated scenes. An animated VRML scene can be defined by routing events throughout the nodes, building an event chain that drives the animation. For example, a sensor can be linked to an interpolator, that generates the movements of a geometric object. This event chain establishes that the user action on the sensor starts the interpolator, that defines the movement. If the user excludes from the final visualization a sub-tree containing the interpolator, for example, the event chain is broken and the geometric object will not move under the user action detected by the sensor.

We considered two solutions for this problem: the more rigid one establishes that an object belonging to an event chain will not be excluded from the final visualization, even if the user defines the contrary; the second solution is to simply warn the user that the object being excluded belongs to an event chain, and its exclusion may cause a "wrong behavior" in the animation (the decision of excluding it or not is left to the user). We have adopted the second solution at this time.

Another way to define an animation in VRML is by the *Script* node, that links the animation to a program, responsible for processing the events and generating results that drive the animation. If an excluded object would generate an event processed by the program, a "wrong behavior" could occur in the animation. The solutions considered for this case were the same than those for the previous case: the object is not excluded, or the user is warned about the possibility of "damage" in the animation.

4 Results

It is difficult to achieve a general numerical result for our application, since it depends on several variable factors: the bandwidth of the connections (client / application server, and application server / information server), the network traffic, the size of the VRML file to be read, and, principally, the parts of the file selected (if the user selects a small part of a large file, the application will show better results than if he/she had selected almost all the objects of the file).

In spite of that, we have developed a model based on some practical results we obtained. This model demonstrates some advantages of our approach.

First of all, it is necessary to model the conventional situation (i.e., without the use of our application), where the client is directly connected to the VRML server via a low-bandwidth communication channel. In this situation, the time necessary to visualize a remote VRML file ($t_{conventional}$) is a function of the file's size. In our model, we divide this time into three components: the time for accessing the file in the disk (t_1), the transmission time (t_2), and the rendering time (t_3). So, we have:

$$\begin{aligned} t_{conventional} &= t_1 + t_2 + t_3 \\ t_1 &= S/B \\ t_2 &= S/W_l \\ t_3 &= S/V \end{aligned}$$

where S is the size of the VRML file to be accessed; B is the bandwidth for the disk access (a reasonable value for this constant is 8 Mbps); W_l is the bandwidth of the connection (standard modems have values of 14.4 and 28.8 kbps); and V is what we call "rendering bandwidth"; our tests showed that the medium value for this constant is about 500 kbps — this value depends on the client's computer performance, but since it is considerably larger than W_l (that means, $t_3 << t_2$), small variations on this value will not significantly affect $t_{conventional}$.

The next step is the modeling of the time necessary to visualize the remote VRML file using our application (t_{appl}). We divided this total time into five components. The first of them is the same of the conventional case (time for accessing the file in the disk). The second component is the transmission time between the VRML server and the application server; it depends on the file size (S) and on the bandwidth of the connection (W_h — high bandwidth). The third component is the processing time. In this component, we include the time necessary to read the VRML file in order to compose the scene graph document, and the time to parse it in order to compose the sub-VRML file. The fourth component is the time to transmit the sub-VRML file to the client on the low-bandwidth channel (W_l). This value depends on the size of the sub-VRML file (S'). The last component is the rendering time, also dependent of S'. For this case, we have:

$$\begin{aligned} t_{appl} &= t'_1 + t'_2 + t'_3 + t'_4 + t'_5 \\ t'_1 &= S/B \\ t'_2 &= S/W_h \\ t'_3 &= S/R \\ t'_4 &= S'/W_l \\ t'_5 &= S'/V \end{aligned}$$

where S is the VRML file size; S' is the sub-VRML file size; B and V are the same constants of the conventional case; W_h is the (high) bandwidth of the connection between the VRML and application servers; W_l is the (low) bandwidth of the connection between the client and the application server; and R is the "processing rate" (this value depends on the performance of the application server; in the current host, a Sparcstation 20, we measured an approximate value of 250 kbps for this constant).

To achieve some numerical results for this model, let us first consider a VRML file of 1 MByte ($S = 1MByte$) and a low-bandwidth connection of 14.4 kbps. Figure 4 shows the total time for the visualization (t_{appl}) for two different high-bandwidth networks. These values are also compared to the time needed in the conventional case ($t_{conventional}$ — the horizontal line). It can be inferred from the graphic that in the slower network ($W_h = 80kbps$), the use of our application is advantageous for sub-VRML smaller than 75% of the size of the original file. In the faster network ($W_h = 1Mbps$), the results are even better (our application is advantageous for sub-VRMLs with sizes up to 90% of the original file).

Similar results are presented in Figure 5 for a modem of 28.8 kbps (W_l). Our application is useful for sub-VRMLs with sizes up to 55% ($W_h = 80kbps$) and 85% ($W_h = 1Mbps$) of the original file.

Figure 6 returns to the 14.4 kbps modem, but considers a smaller VRML file ($S = 300KB$). In this case, our application has superior performance only for sub-VRML files with sizes smaller than 30% of the original file.

The following conclusions can be inferred from this model:

- The application is more efficient for higher- bandwidth connections between the VRML and the application servers (W_h).

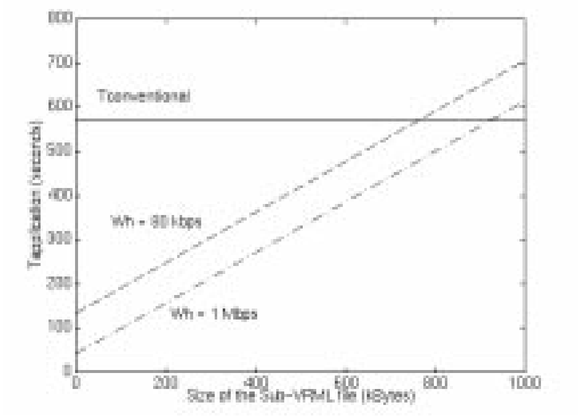


Figure 4: Total time needed for the visualization of a 1 MB remote VRML file, client using 14.4 kbps modem.

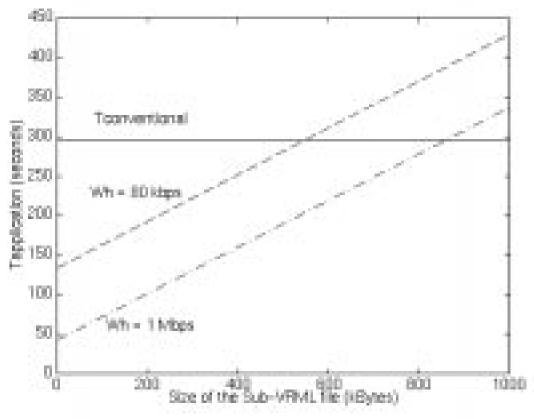


Figure 5: Total time needed for the visualization of a 1 MB remote VRML file, client using 28.8 kbps modem.

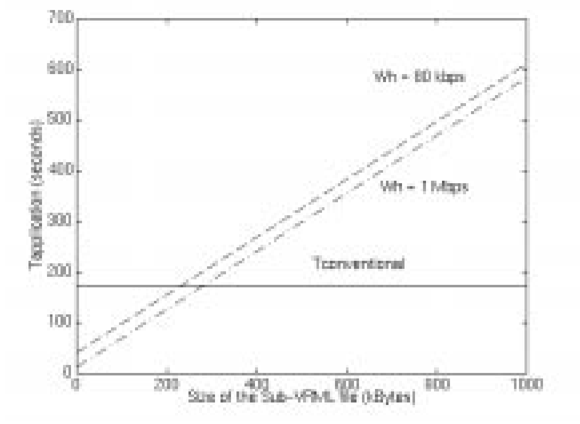


Figure 6: Total time needed for the visualization of a 300 kB remote VRML file, client using 14.4 kbps modem.

- If the connection between the client and the application has lower bandwidth (W_l), the results of our application become more significative.

- The application is more adequate for larger VRML files.

5 Conclusions and Future Work

There is a growing need for tools supporting the interactive access, manipulation, and visualization of distributed multimedia information to realize the vision of "all information at your fingertip." One great challenge is the improvement of the accessibility in the WWW using low-bandwidth connections (e.g., mobile devices and dialup).

In this paper we developed an application allowing the user to select elements of a VRML scene. This selection can reduce the amount of data transferred and the utilization of client resources. Its main advantage is the use of interactivity to achieve an efficient transmission. In order to "filter" the data or to reduce the complexity of the scene (efficient transmission) the user selection is needed (interactivity).

Our results have shown that the adopted approach is more efficient for larger VRML files and clients with slower connections.

In the current version of the application, which works with arbitrary VRML files, the interactivity still requires some knowledge of VRML by the user. In future improvements, the goal is to achieve as much transparency as possible in the selection process. One idea in this direction is to use semantic information to define the role each object plays in the environment. For example, it is possible to establish a "semantic content header" in the file that assigns levels of priority to the objects; high priority objects are essential to the scene, while low priority objects are not. This semantic content header can be included in the VRML file as special comments or using the WorldInfo node, normally used to transport general information about the VRML scene. In this way, the user could visualize only objects with priority levels above a predefined threshold level, without having to manually select them. Another interesting possibility is the automatic selection of this threshold level, based on the bandwidth of the connection. In other to achieve these goals, adequate authoring tools have to be defined.

The idea of the semantic content header is very interesting and can be used for other purposes besides objects' selection. The semantic information can be used, for example, to associate VRML objects to queries in databases or, in addition to information about system resources and user preferences, can be used to distribute the rendering tasks among various machines in a local network (details about this proposal are found in [11]).

The results of this work can be useful in several different areas (actually, in all the areas that Mobile Communication makes sense), such as distance learning (for places without high-performance networks), marketing (demonstration/simulation of products running on stationary servers), and repairing of equipments in distant places.

A binary encoding format for VRML files is currently being discussed [16], emphasizing the reduction of the file size for fast transmission, and the simplification of the file structure for fast parsing. Nevertheless, this is an improvement orthogonal and complementary to the solution proposed in this work.

Despite the communication technologies improvements, it is our understanding that the relevance of this work will not be diminished. The discrepancies between mobile and static environments, mostly regarding transmission rates, will continue to exist, and the same problems shown here will be present.

Acknowledgements: This paper is a result of ProBrAl 002/94, a cooperation project between UNICAMP and the TH-Darmstadt, sponsored by CAPES (Brazil) and DAAD (Germany). We appreciate the support granted by these institutions. We would like to thank FAPESP, CNPq, and the Dept. of Computer Science of the University of Waterloo for the support of some authors, and also Luc Neumann and Dr. Rüdiger Strack for helpful pointers and clarifying discussions.

References

[1] T. Berners-Lee, R. Cailliau et al. The World-Wide Web. *Communications of the ACM*, 37(8):76–82, August 1994.

[2] M. S. Blumenthal. Unpredictable Certainty: The Internet and the Information Infrastructure. *IEEE Computer*, 30(1):50–56, January 1997.

[3] M. Campione and K. Walrath. *The Java Tutorial — Object-Oriented Programming for the Internet*. The Java Series. Addison–Wesley, 1997. `http://java.sun.com/nav/read/Tutorial/index.html`

[4] R. Comerford. The battle for the desktop. *IEEE Spectrum*, 34(5):20–28, May 1997.

[5] Dimension X. *Liquid Reality*. `http://www.microsoft.com/java/gallery/lrpro.htm`

[6] N. Gershon, J. R. Brown et al. Computer Graphics and Visualization in the Global Information Infrastructure (special report). *IEEE Computer Graphics and Applications*, 16(2):60–75, March 1996.

[7] R. Lea. Java and VRML 2.0 Part 1: Basic Theory. *VRML Site Magazine*, February 1997. `http://www.vrmlsite.com/feb97/a.cgi/spot2.html`

[8] A. Luotonen and K. Altis. World-Wide Web Proxies. In *First International World-Wide Web Conference*, Geneva, Switzerland, May 1994. `http://www1.cern.ch/PapersWWW94/luotonen.ps`

[9] L. P. Magalhães, A. B. Raposo, and F. S. Tamiosso. *VRML 2.0 — An Introductory View by Examples*, 1997. `http://www.cgl.uwaterloo.ca/~lpini/tutorial/vrml-tut.html`

[10] Netscape Communications Corporation. *JavaScript Authoring Guide*, 1996. `http://home.netscape.com/eng/mozilla/2.0/handbook/javascript/index.html`

[11] L. Neumann and A. B. Raposo. *An Approach for an Adaptive Visualization in a Mobile Environment*. IDMS'97 (European Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services), Darmstadt, Germany, September 1997.

[12] A. B. Raposo, L. Neumann, L. P. Magalhães, and I. L. M. Ricarte. Visualization in a Mobile WWW Environment. WebNet'97 — World Conference of the WWW, Internet, and Intranet Toronto, Canada, November 1997. `http://www.dca.fee.unicamp.br/projects/prosim/3djava/publications/webnet97/webnet97.html`

[13] Silicon Graphics, Inc. *Cosmo Player*, 1997. `http://vrml.sgi.com/cosmoplayer/`

[14] Sony Corporation. *Community Place*, 1997. `http://vs.spiw.com/vs/`

[15] VRML Consortium. *The Virtual Reality Modeling Language Specification ISO/IEC DIS 14772-1*, April 1997. `http://www.vrml.org/Specifications/VRML97/DIS/`

[16] VRML Consortium – Compressed Binary File Working Group. *The VRML Compressed Binary Format Specification - Draft 4*, June 1997. `http://www.research.ibm.com/vrml/binary/specification/index.html`