# Interactive Visualization over the WWW

ALEXANDRE DONIZETI ALVES
MARIA CRISTINA FERREIRA DE OLIVEIRA
ROSANE MINGHIM
LUIS GUSTAVO NONATO

Departamento de Ciências de Computação e Estatística
ICMC/USP - São Carlos
Caixa Postal 668
13560-970, São Carlos - SP, Brazil
{adalves, cristina, rminghim, gnonato}@icmc.sc.usp.br

**Abstract.** In this paper we discuss several approaches for providing visualization solutions over the WWW exploiting available technology for programming Web-based applications. An implementation that supports interactive visualization of user data through a set of JAVA applets that interface with the *Visualization Toolkit* is presented, and its advantages and limitations are discussed. Such an implementation provides a general framework for providing high-quality visualization resources on the WWW.

## 1 Introduction

The WWW has revolutionized the way people access information and established itself as a media for cooperation among spatially distributed researchers. It certainly has the potential to change the manner by which visualization applications may be developed, distributed and used, and provides a rich and flexible media to make the connection amongst application, data, information and users [9]. For example, data sets may be kept by those persons that generate them (the editors) while other people may conveniently access them over the Internet using a standard Web browser. Such a working model ensures that receivers will always get access to up to date data (as long as the producers keep them so) [13], and fosters collaboration amongst spatially distributed collaborating groups.

Despite its potential, this media has not yet been extensively exploited due to a number of practical reasons. Many of them are related to the limitations of the available technology that make it difficult to attend typical demands of visualization users. These include, for example, interactive rates for visualizing large data sets and complex models, simple access, ease of use, portability, acceptable transmission rates over a wide area network.

In this paper we discuss how current technology for programming Web applications may be employed for providing interactive visualization resources on the Web. We review some approaches described in the literature and some practical solutions that may be found on the Web itself. We also describe an implementation based on one of the solutions presented, which ensures interactive remote access to some classical visualization algorithms. A discussion on the advantages and limitations of the solution provided is also presented. Finally, we present some final remarks and perspectives for further research on distributed Web environments for visualization.

## 2 Approaches for Visualization over the WWW

In this section we discuss how available technologies may be exploited to produce Web-based visualization tools. We describe possible scenarios for creating visualization resources, and illustrate some approaches described in the literature to produce Web-based visualizations, pointing out to their strengths and limitations.

A reference model for Scientific Visualization has been proposed by Upson *et al* [12]. It treats the visualization process as a pipeline in which a data source is fed, filtered, mapped and rendered to produce a final image, as illustrated in Figure 1. This model is useful to analyze possible scenarios for visualization on the Web, as described by Wood et al. [13]: data sets are produced by someone (the editor), images are presented to observers, but execution of the intermediate processes may be placed either with the observer at the client or on the server side, with the editor.

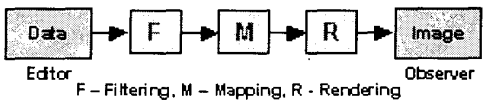


**Figure 1** Visualization pipeline.



0-7695-0878-2/00 $10.00 © 2000 IEEE

A common situation is to create the visualization as an image or animation displayed in a standard graphical format. This solution allows no user interaction with the visualization models. All the intermediate steps (F, M and R) are executed at the server, and the observer is limited to seeing the result. HTML documents containing images and animations exemplify such a scenario. CGI forms embedded in HTML pages may be used to give the observer some control over the process of generating an animation sequence at the server. As soon as the client submits the form, the server calls its CGI script with the parameters defined. However, this is a limited approach for data visualization, as images/animations are slow to download and the observer has no chance of manipulating them. Even if some control is provided to the observer for defining visualization parameters, he cannot interact with the visualization models created. This solution may also overload the server if too many requests are sent.

Alternatively, the editor may create the visualization as a 3D model that can be manipulated at the client, which is possible using VRML, for example. In this case, stages F and M are still executed by the editor, but R is placed at the client. The observer still cannot manipulate the parameters used to produce the visualization models, but can interact with them using an appropriate browser/plug-in. A limited set of exploration strategies and geometric transformations may be applied to the model, and interaction occurs at reasonable rates, as rendering is done locally. This solution has been adopted by Brodlie et al. [3] in their implementation of a virtual environment for surgical training. A different scenario is to place on the observer the responsibility for generating the visualizations from given data. That means executing all stages F, M and R using visualization software at the client. Data sets may have to undergo some pre-processing, as there are no established standards.

In the scenarios described, the task of generating visualization models is either left entirely with the editor or entirely with the observer. The former approach limits the possibilities of interaction on the part of the visualization users, and the second places a too heavy burden on these users, as few low-cost systems are available and their use demands training and specific and usually costly hardware platforms. A compromise solution would be the share the responsibility: the editor at the server offers processing power and a basic visualization structure for the data, and the observer at the client has a set of options for controlling the visualization process by defining relevant parameters for producing the visualization models and for interacting with them. The R stage runs at the client, and the F and M stages are determined at the server, but may be controlled from the client.

Jern [5] defines the terms 'thin' client and 'fat' client in the context of visualization on the Web: the former requires minimum local software and acts just as an interface for a visualization application accessible from a Web server. Typical web browsers do not support visual interaction executed at the client side, and therefore, user interaction with the application is highly dependent of network transmission rates. A more effective solution for highly interactive applications such as data visualization would be to split the visualization process amongst client and server. A 'fat' client is one that supports local execution of graphical interfaces at the client by means of local software.

One solution for distributed visualization that fits into the 'compromise' scenario is described by Ang et al. [1]. They developed a volume visualization tool targeted at general-purpose hardware platforms, and integrated them into the Mosaic browser as a visualization service. The tool, named VIS, uses a pool of graphical workstations to generate 3D models from volume data, distributing the task amongst the available hardware. To allow distribution of information to remote clients they extended Mosaic to support volume data and defined a communication protocol amongst VIS and Mosaic. This allows the embedding of interactive volume visualization operations into HTML documents. From the browser, remote clients may request the execution of visualization tasks that are forwarded to the VIS environment, and the result is returned at the HTML document.

Another work that fits into the same scenario, although it is not targeted at the Web, is by Liu *et al* [6], who implemented and interactive distributed visualization system named Discover (*Distributed Interactive Scientific Computing and Visualization Environment*). Their system adopts a client-server architecture, and comprises a virtual host that includes one or more PCs, and a pool of processors that provide this host with the required computing power. The system offers functions for both client and servers, although the client services include only basic manipulation operations. It is a system oriented towards clinical applications that supports cooperative visualizations, with users positioned at different clients being able to interact with a 'single' visualization window.

In the current stage of the technology for programming Web applications, JAVA applets that may downloaded and executed at the client also provide a suitable solution for creating applications that fit into the 'compromise' scenario. JAVA applets may overcome some of the current limitations associated to interactive

applications on the Web using 'thin' clients, although they still show dependency of network rates. The JAVA language has been conceived to run at a variety of hardware platforms with no need for recompilation and with good reliability. With the upcoming of JAVA 3D API [11] improved performance for 3D graphics will also be assured by direct connection with graphics hardware.

An alternative approach for accessing graphics hardware is to use the JNI (*Java Native Interface*) mechanism, which allows direct calls to C or C++ routines. Thus, an existing visualization library such as VTK – the *Visualization Toolkit* [10] – can be accessed from within a JAVA environment (this is the approach adopted in the solution described in Section 3). JNI also allows critical functions to be implemented in C or C++ for improved performance. The major limitation of using JNI and native code is that it sacrifices portability to some extent. For example, to run JAVA applications that use VTK the client must download the native support. JAVA 3D enables *applets* with 3D graphics to be written purely in JAVA with no performance penalty. In that case, a toolkit such as VTK could be rewritten in JAVA and used for creating visualization applets.

Michaels and Bailey [8] describe a scientific visualization tool implemented as a JAVA applet that uses only the JAVA AWT – Abstract Windowing Toolkit – API [4] for 3D graphics. It has been developed with the goals of being completely platform independent, being easy to use, offering a basic set of 3D visualization functionality for users to analyze their own data sets and good interactivity. Users have to upload their data set into the server, and then can apply basic volume visualization techniques such as surface extraction, cutting planes, point clouds, and elevation plots. They can also interact with the resulting visualization models at reasonable rates. That has been made possible by careful implementation, simplifications introduced in some visualization algorithms, and a limited set of rendering options, e.g. flat shading only. This solution also fits into the scenario in which part of the work is executed at the client, although the set of functionalities available is defined at the server. In Section 5 this tool, called VizWiz, is compared to our own JAVA based solution, described in the following section.

## 3 An implementation

Our solution, called *VisWeb*, also uses JAVA applets to offer visualization resources on the Web and fits into the same scenario of the solutions described at the end of Section 2. A description of the environment and of some issues related to its implementation are provided in this section.

Because applets are downloaded from a remote site, several security restrictions are imposed, and one of them is that they can not read or write files into the local host. This poses a severe limitation for a visualization application that must access local data set files in order to be useful. Another restriction is that an applet is not allowed to access native code in the client machine. That posed an additional problem for the solution we had in mind, as it was our intention to use an available visualization library, written in C++, to run the visualization algorithms, as discussed in the following.

Digital signatures provide a mechanism for loosening security restrictions on JAVA applets [7]. Signatures may be created using specific tools targeted at the Netscape browser or at the Microsoft browser, and require a certificate emitted by a Certification Authority. Alternatively, in JAVA 1.2 enables programmers to establish different security levels for an applet, defining exactly what it is allowed to do. Although JAVA 1.2 is currently not supported by either Netscape or Microsoft browsers, the JAVA Plug-in from Sun allows designers to specify an external Java Virtual Machine to support applets written in this version of the language. These facilities, in addition to the availability of JAVA Swing, which extends the AWT API and incorporates it into the JFC – Java Foundation Classes –, have been decisive for our choice of adopting JAVA 1.2 as our development platform. The interface components of Swing provide better facilities for handling events and for selecting appearance and behavior than previous versions of the language. In this approach the server provides the applets, and processing is entirely allocated to the client, who also holds the data sets, as illustrated in Figure 2.

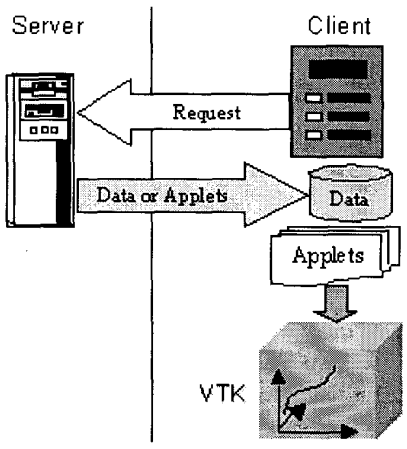


**Figure 2** Architeture of the *VisWeb* environment.

The JAVA applets making up the *VisWeb* visualization environment interface with the VTK toolkit

to implement a set of basic scalar and vector volume visualization techniques. VTK [10] is an extensible object-oriented toolkit that implements a rich set of visualization classes and data structures. It has been written in C++, and its classes may be accessed from programs in C++, TCL/TK or JAVA. VTK uses the API OpenGL for handling the 3D graphics rendering pipeline, i.e., illumination, projection, clipping and scan conversion of graphics models.

As shown in Figure 2, VTK runs on the client. Therefore, clients must have the appropriate VTK libraries for the JAVA applets of *VisWeb* to execute a visualization. Namely, files *vtkdll.dll*, *vtkjava.dll*, and *vtk.jar* must be available, assuming that the client is running the environment on a *Windows* platform. A permission file to enable the server to access the client's local file system must also be present. In our case, we provide the appropriate permission and certificate files, which may be downloaded by the clients. The certificate ensures that the client will grant local file system access only to those applets signed by this certificate.

In this solution, all the visualization functions are implemented using the VTK classes, and all the rendering work embedded in such algorithms is handled by VTK and OpenGL. The JAVA front end only provides widgets for handling events at the user interface level, and activates the suitable VTK classes using the JAVA binding of VTK. The currently available version of *VisWeb* runs on Windows platforms, and uses VTK 3.1 and Java 1.2.2.

## 4 Functionality of the Visualization Applet

In this section we shall describe the functionality of the visualization tools offered in *VisWeb*, which comprises three modules: a surface extraction module, a direct volumetric rendering module, and a vector field visualization module. Such a description is illustrated with some interface windows and visualizations obtained from the system. Both the surface extraction and direct volume rendering (DVR) modules deal with volumetric scalar data, whereas the vector visualization module handles volumetric vector fields. Data sets currently supported are structured grids with regular geometry and topology (represented by instances of the VTK class *vtkStructuredPoints*). Figure 3 depicts the main page displayed when *VisWeb* is invoked. The user is presented a window where s/he can select the idiom (English or Portuguese), and the screen resolution is automatically set up. The same window gives access to the main visualization modules of *VisWeb*.

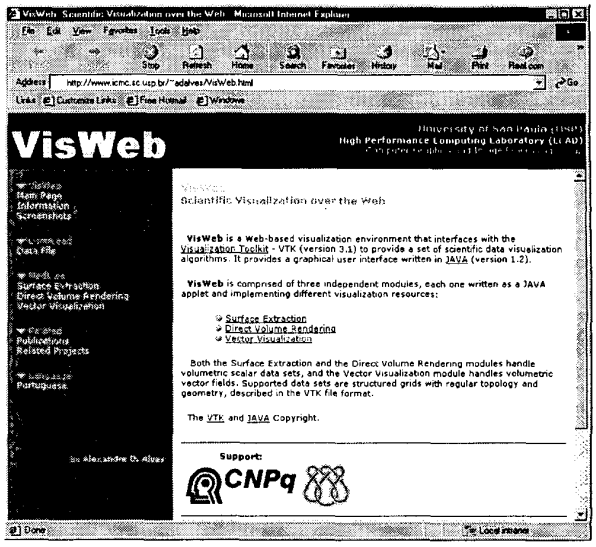


**Figure 3** Main page of *VisWeb*.

In the surface extraction module, level surfaces are extracted from the scalar data through the Marching Cubes technique as implemented in VTK in its class *vtkContourFilter*. Figure 4 (at the end of the paper) shows the main interface window for this module as seen from the Netscape browser. Multiple level surfaces can be visualized simultaneously and the interface allows the user to specify isosurface values and also to associate a color and opacity to each surface, as illustrated in Figure 5. Interactive tools such as object selection, cutting planes and animations are also supported. The bigger window shows the visualization working area, which depicts the external and internal isosurfaces obtained from a scalar data set that describes a model of a tooth.

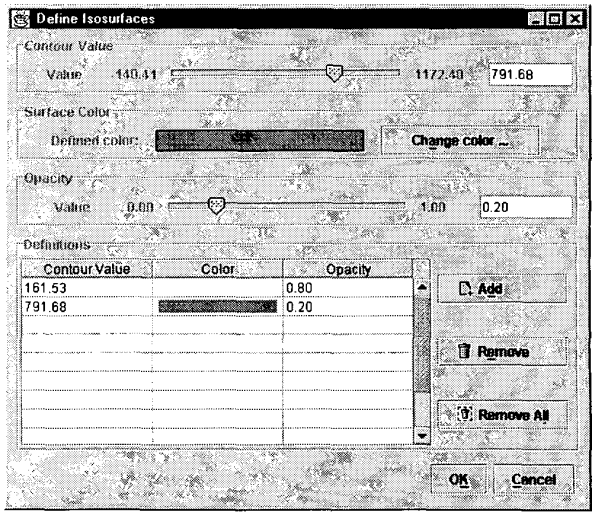


**Figure 5** Interface for isosurface definition.

The direct volumetric rendering module enables visualization of volumetric scalar data sets employing the ray casting technique, also available from the VTK toolkit. Figure 6 (at the end of the paper) shows the main interface for this module. The user can choose between visualizing the whole volume data or, alternatively, s/he can specify volume sub-regions. If suitable color and opacity transfer functions are assigned to data, i.e., data classification is appropriate, direct volume rendering makes possible to visualize details of the volume that would be difficult to observe through surface extraction. To facilitate the data classification process the user can interactively define a color mapping scheme and assign intensity and opacity ranges to data intervals.

Figure 7 depicts the interface for defining a custom transfer function for color and opacity mappings. The parameters shown are those used to define the volume visualization depicted in Figure 6. As in the surface extraction module, it is possible to define cutting planes through the volume and to animate a sequence of them. Figure 8 shows several volume images generated in this module from the same data set, which describes a tooth model.

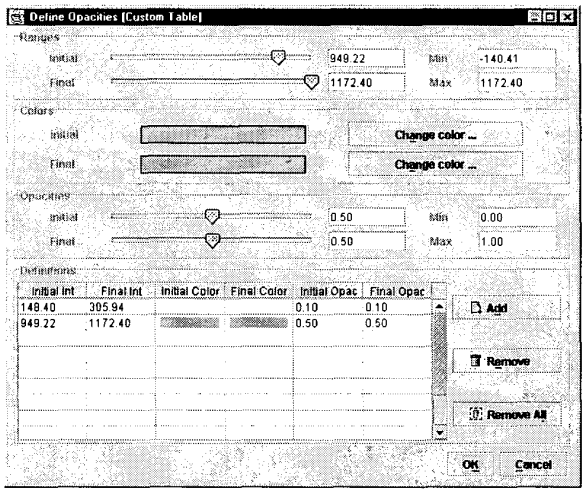


**Figure 7** Interface for data classification in the DVR module of the visualization applet.

The interaction options using the mouse supported by a conventional VTK window remain operational in the visualization area of the applet. Thus, user interaction is this window is similar to interaction in a conventional VTK window. Facilities include mouse interaction for the user to rotate, zoom and translate the models, as well as keyboard options to reset the viewpoint, using the same controls employed in VTK. For scalar visualization, the interface also enables observation of visualization models at variable levels of detail during interaction. This optimizes the amount of detail rendered in a scene while the user is continually interacting with the model (as in Figure 8d). After interaction is stopped a more detailed rendering is displayed, thus ensuring the speeding up of the interaction and overall visualization times.

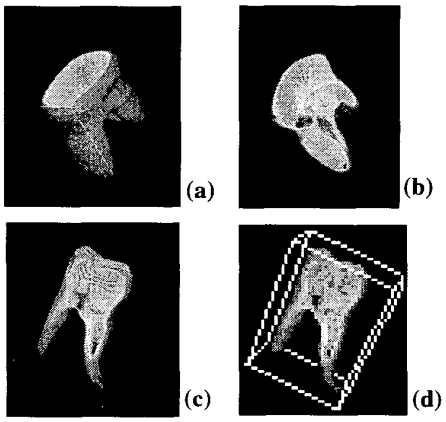


**Figure 8** DVR volume visualization of teeth data. (a) whole volume; (b) and (c) user defined sub-volumes; (d) sub-volume with coarse level of detail for rapid interaction.

The third module provided takes as input a structured data file describing a three-dimensional vector field and produces a glyph-based visualization using line segments. The main interface for this module is shown in Figure 9 (at the end of the paper). The visualization work area in the bigger window depicts a view of vector data from a magnetic field around a neuron cell model. The user can interactively specify parameters such as the vector sampling factor for controlling the number of glyphs displayed, mapping vector magnitude values using a rainbow color scale, and others (Figure 10). A more detailed description on the functionality and the implementation of the visualization applet may be obtained from Alves [2].

## 5 Discussion

The solution implemented in *VisWeb*, of providing visualization facilities embedded into a JAVA applet that accesses VTK, has both advantages and limitations. An obvious advantage is that the visualization framework provided by VTK is directly available. Our current implementation can be easily extended to incorporate additional visualization techniques, as long as a suitable interface for parameter definition is provided. Alternatively, the same framework may be employed to provide solutions targeted at specific application domains.

Some restrictions are imposed on the client, which may be taken as a limitation. The client must have some files installed to enable execution of the visualizations, in addition to the JAVA Plug-in. It must also grant (limited) access to its file system by accepting the security options defined in the permission file and certificate delivered with the system.

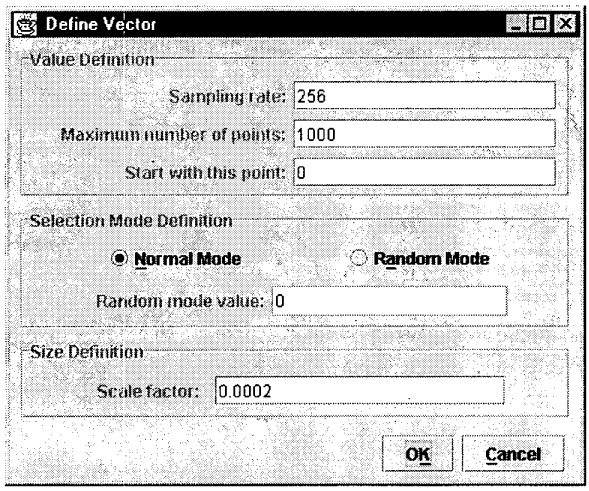


**Figure 10** Interface for setting glyph-based vector visualization parameters.

Accessing client file systems was not possible some years ago, and Michaels and Bailey [8], in their implementation of VizWiz, work around the problem by getting the clients to upload their data file at the server for access by the visualization applet. This is not a feasible solution in a production context, however, as it places a huge load on the server. Moreover, potential users will most often get stuck with network problems or unbearable delays. Additionally VizWiz has a much more restricted set of visualization techniques, as all of the available ones had to be implemented from scratch using the less than suitable AWT API.

## 6 Conclusions and Further Work

Providing visualization resources on the Web opens up a number of possibilities for cooperation, teaching and training. It ensures world-wide accessibility at a very low-cost. With our implementation, it is clear that available technology can be exploited to create sophisticated interactive visualization environments to be accessed on the Web. Such environments can go much further than manipulating VRML worlds in which realism is greatly sacrificed.

We intend to extend our current tool to provide an integrated environment targeted at the manipulation of teeth models as part of the Virtual Dentistry project under development at ICMC. The *VisWeb* system will soon be available for public remote access by potential visualization users and collaborators for further testing, including a version for Linux-based client platforms.

Additional facilities could be incorporated into the environment that go beyond just increasing its visualization functionality. For example, visualization parameters set during a visualization session could be saved for later use by the same user or by others. In the first case, such information could be saved at the client, in the latter it could be saved at the server and associated to the corresponding data file. Such a facility would provide basic support for cooperation amongst distributed visualization users, and other functionality could be added with this goal in mind. The ability to save the configuration parameters used in a visualization session could also provide valuable feedback to visualization designers about the difficulties faced by visualization users.


## Acknowledgements

We wish to acknowledge Mike Goetz and Andy Day, from the School of Information Systems, UEA, UK, for the data used to generate the teeth images; and Dr. Luciano da Fontoura Costa and members of his Cybernetic Vision Research Group, IFSC, for the neuron vector data. We also wish to acknowledge the funding of FAPESP and CNPq, Brazil.


## References


[1] C.S. Ang, D.C. Martin, M.D. Doyle, "Integrated control of distributed volume visualization through the World Wide Web", *Proc. IEEE Visualization'94*, oct. 17-21, Washington D.C., 13-20.

[2] A.D. Alves, *Visualization over the WWW: Study and Implementation of a System*, M.Sc. dissertation, ICMC-USP, August 2000 (in Portuguese).

[3] K. Brodlie, N. El-Khalili, Y. Li, "Using Web-Based Computer Graphics to Teach Surgery", *Computers & Graphics* 24(1), 2000, 157-161.

[4] J. Gosling and F. Yellin, The Java Team. "Java API documentation", Sun Microsystems,

http://java.sun.com/products/jdk/1.0.2/api/

[5] M. Jern, "Information Visualization on the Web", *Proc. IEEE 1998 Int. Conf. on Information Visualization*, jul. 29-31, London, UK, 2-7.

[6] P.W. Liu, L.S. Chen, S.C. Chen, J.P. Chen, F.Y. Lin, S.S. Hwang, "Distributed computing: new power for

scientific visualization", *IEEE Computer Graphics and Applications* 16(3), 1996, 42-51.

[7] G. McGraw and E. Felten, *Securing Java*, John Wiley & Sons, Inc., 1999.

[8] C. Michaels and M. Bailey. VizWiz: a Java applet for interactive 3D scientific visualization on the Web. *Proc. IEEE Visualization'97*, oct. 19-24, Phoenix, AZ, 261-267.

[9] R.M. Rohrer and E. Swing, "Web-Based information visualization", *IEEE Computer Graphics and Applications* 17(4), 1997, 52-59.

[10] W.J. Schröeder, K. Martin, and B Lorensen, *The Visualization Toolkit – an object-oriented approach to 3D graphics*, 2nd edition, Prentice-Hall, 1998.

[11] H. Sowizral, K. Rushforth, and M. Deering, *The Java 3D API specification*, Addison-Wesley, 1997.

[12] C. Upson, T.A. Faulhaber Jr., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, A.V. Dam, "The Application Visualization System: a computational environment for scientific visualization", *IEEE Computer Graphics and Applications* 9(4), 1989, 30-42.

[13] J. Wood, K. Brodlie, and H. Wright, H. "Visualization over the World Wide Web and its application to environmental data", *Proc. IEEE Visualization'96*, oct. 27-nov 1st, San Francisco, CA, 81-86.

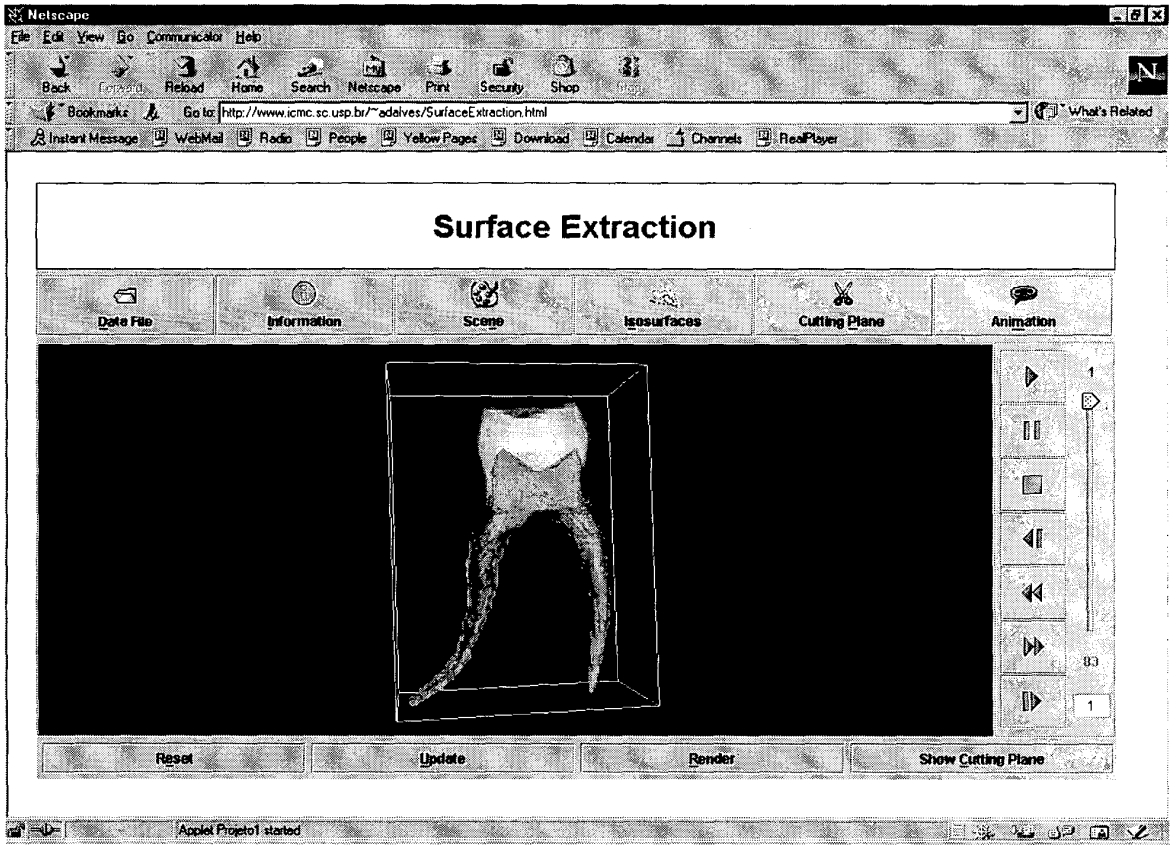


**Figure 4** Interface for the surface extraction module of the visualization applet as seen in the Netscape browser.

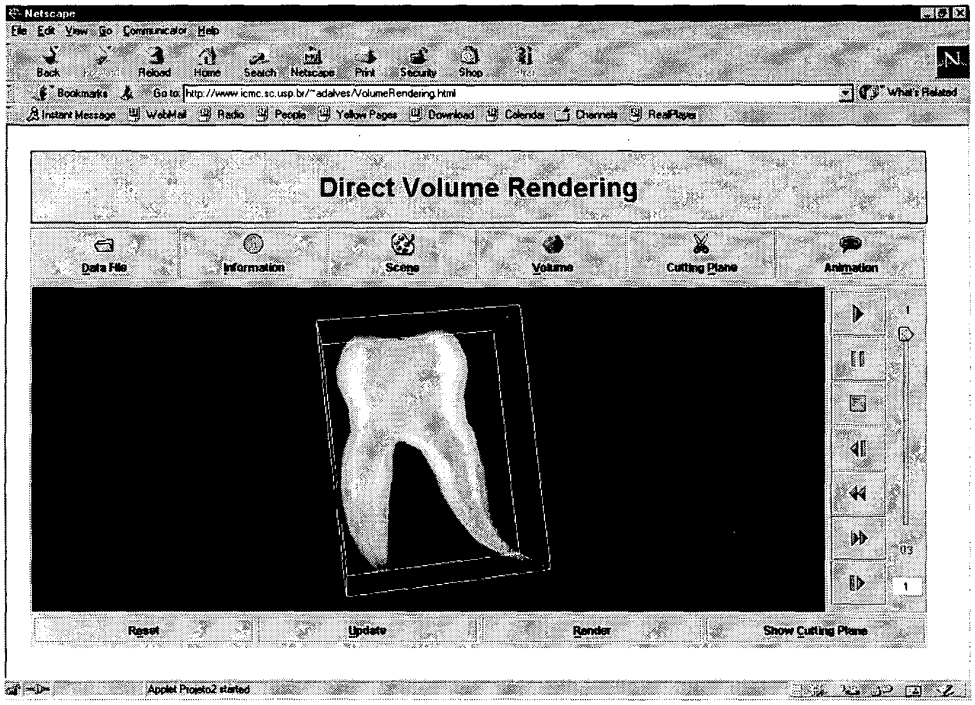


**Figure 6** Interface of the Direct Volume Rendering module.

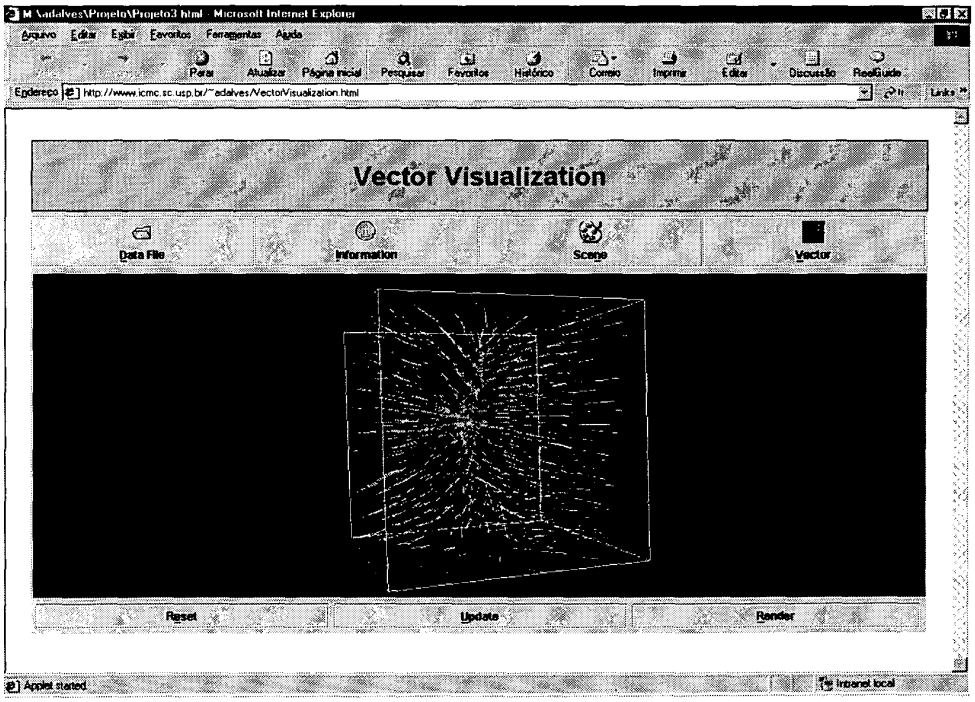


**Figure 9** Interface for the vector visualization module of the visualization applet as seen in the Explorer browser.