Collaborative Scientific Data Visualization

Byeongseob Ki (bki@top.cis.syr.edu)
Scott Klasky (scott@npac.syr.edu)

Northeast Parallel Architectures Center
111 College Place,
Syracuse University
Syracuse, NY 13244-4100

http://kopernik.npac.syr.edu:8888/scivis/

Abstract

We have designed a collaborative scientific visualization package that will aid researchers from distant, diverse locations to work together in developing scientific codes, providing them with a system to analyze their scientific data. We have utilized Java to develop this infrastructure. Two important areas which we have concentrated on developing are 1) a collaborative framework from which the scientific data is interpreted and utilized, and 2) a framework, which is customizable to the suit the needs of a particular task and/or scientific group.

1. Introduction

One of the major benefits of using Java is that it is platform-independent and is the programming language of the World Wide Web. Only a short time after its introduction, Java has become the dominant technology to bring interactive content to the Internet. One of the major advantages of using Java over other programming languages is that it contains a programming environment with the Abstract Windowing Toolkit (AWT) and a built-in networking library to make it easy to develop complex applications. We have utilized these unique features in building our software.

One of the major design criteria's in designing our system (Scivis) was to provide end-users with a client-server data analysis system, by taking full advantage of Java 1.1. A major advantage in using Java to build this application is that it supports peer-to-peer communications, which is vital for collaboration that is spread across a network.

Our experience in designing Scivis comes from initial work by M. Choptuik (University of Texas at Austin), who wrote a 1-dimension client-server visualization program in C using the native SGI language GL[16]. We have extensively built upon this system while using some of its basic functionality's. Our system is currently being used in the NSF Grand Challenge, "The Binary Black Hole Grand Challenge", which is an alliance between 10 or more universities.[17] Since researchers in this community must collaborate over long distances, we identified the need to develop our system for this research community.

One of the key features of our system is that the filters are user definable. A filter is an action, which is performed on data set(s) to create new data based on this action. An example of a filter would be to take the absolute value of a function. When the user wants to perform such an action a new data set is created from the original data set. Since our system is collaborative, users can share their filters with other collaborators on the project. This creates a powerful platform where many different users can exchange their own filters by showing other users how his/her filter performs. Since it would be impossible to design filters for all applications we have created APIs for users to design their own filters. In the following section, we describe how users can create new filters to help analyze their data.

This software is collaborative. Each researcher runs a server on his/her machine. Multiple servers can be run on multiple machines, which can collaborate in a master-master or master-slave configuration. There are several known applications in Java, which have been developed to give applications porting APIs [1][2][3][4]. Since these applications were still in the development stage when we started designing our software, we felt it was necessary to develop our own collaborative system. To support collaboration among users, we use the *java.net* package, which provides two classes -- *Socket and ServerSocket* -- to implement the client and server side of the connection respectively.

Computational steering is another key feature of our software. We can read a list of variables from a Fortran 90 or C program, and then at any time ask to see a variable from the server. This prompts the client to process this information and send back data that the user had requested.

2. The advantages of using Java

Although Java is less than two years old, Java computing and the language on which it is based has attracted more interest than C or C++ has in their lifetime**.** This is the major reason why we choose to implement this software in Java.

Java is a modern, object-oriented language, and comes complete with single-state and multiple interfaces. This is an efficient and powerful combination that works well with the Common Object Request Broker Architecture (CORBA). CORBA is a language-independent industry standard for remote invocation. Using the Object Request Brokers, objects in one location on a network can invoke methods on other objects on the network in a location-independent manner. Since CORBA can hide implementation distinctions between different languages, we can use it to build multi-lingual distributed object systems. Although we are not currently using CORBA in our software, we plan to in the future.

Because Java runs on the Windows, Macintosh and UNIX platforms, it is now possible to create "Write Once, Run Anywhere" applications. This eliminates the need to port applications to separate, incompatible visualization server environments, and overcomes the lack of availability of certain application features and functions on inherently limited platforms. Because the Java platform includes a language as well as a set of standard class libraries, it's easy to build powerful, mission-critical applications that run on any Java virtual machine in any environment.

The Java platform's tightly designed set of bytecodes is easy to transport across the Internet or Intranet environments due to its virtual machine orientation. For example, to write an object to a file, Java provides us with the Object Serialization mechanism to simplify the problem. As its name implies, Object Serialization converts Java objects into a byte stream, which can then be easily stored to a file. When the file is later read, the byte stream can be also easily converted back to the original object[9]. Thus, we have adopted the Object Serialization mechanism to read data from a file or socket.

Another reason to choose Java as an implementation language is that Java is a more dynamic language than C or C++. It was designed to adapt to an evolving environment [8]. A major problem with C++ in a production environment is a side effect of the way in which code can be implemented. If we produce a class library (a library of plug-and-play components) and programmer `A` uses it in his/her product, then we then change our library and distribute a new release, programmer `A` will probably have to recompile and redistribute his/her own software. In an environment where an end user `B` gets our software and `A`'s software, problems can result. For example, if we distribute an upgrade to its libraries, then all of the software from the user `B` may break. It is possible to avoid this problem in C++, but it is extraordinarily difficult and it effectively means not using any of the language's Object Oriented features directly.

By making these interconnections between modules later, Java avoids these problems and makes the use of the object-oriented paradigm straightforward. Libraries can freely add new methods and instance variables without any effect on their clients.

3. Software Components

3.1. Client-Server Model

Scivis obtains data from clients connecting to it with socket connections. We use the *BufferredInputStream* class library provided by the Java Development Kit (JDK) to send data to Scivis. This class implements a buffered input stream. Data is read block by block into a buffer, which allows subsequent reads to directly access the data from the buffer, which speeds up sending the data from client to server.

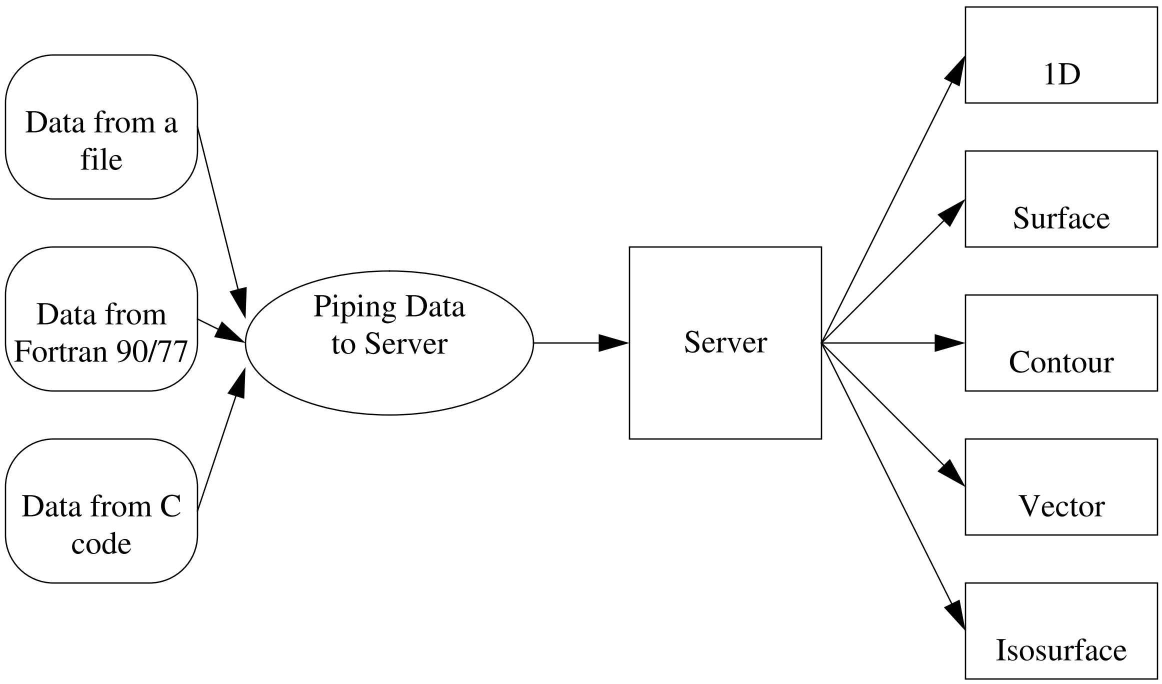
Since the client can send data to Scivis by opening up a socket, the client may be implemented in any language. We have provided support for Java, C, C++, and Fortran programs to connect to the server via function calls.

There are many advantages of using a client-server model for data visualization. One such advantage is that the users can send the data to the server anywhere the socket is supported. Therefore, the user interface of a client/server application can be written on a PC and the server can be written on a mainframe. Clients can be written for DOS- or UNIX-based computers. Since the responsibility of sending data to the server is from the client's side, the server has more computing resources to spend on analyzing queries and visualizing data. This uses the strengths of divergent computing platforms to create more powerful applications.

Another advantage of using a client-server model is that, if the data set is read from a socket, Scivis can be used as a front end to other applications, which produces the data set. For example, if one application, which is written Fortran, produces a data set, it can write the output data set to a socket, which is connected to our server. Our application will then allow the user to analyze and visualize the data.

Figure 1 shows the architecture of Scivis. The client in this system is responsible for sending the data to Scivis, which opens up a window to visualize the data. Scivis can visualize 1-3 dimensional data sets, using (scatter plots, line-plots, vector plots, contour plots, surface plots, and isosurface plots.) When Scivis receives data from a client, it checks the name of the data set. Since Scivis opens up a separate window for each independent data set, if the name of the window is the same as the name of a data set that is already open, the datasets will be merged.

Figure 1: System architecture of this software.



3.2. Data Set

Our software can plot several types of data sets such as 1 or 2 dimensional data, vector data, or isosurface data. Each unique data set will be composed of distinct, sequential timesteps. As the data sets are composed of these time steps, or slices, we need features, which allow animation of this data. Scivis facilitates animations and allows users to choose to loop through the time data, or step through the sequential time slices in either direction. We also allow users to go to any time slice by simply choosing its iteration number.

Another necessary component is the ability to write the data to a file. Since we are working with a client-server model, the data will reside in memory, which may be stored on magnetic media by writing the data out to a file. When a window is closed its contents are cleared from the systems memory. Thus, when a user wants to save a particular data set, he/she will save the results in Scivis, which can be piped into the system at a later date.

There are options to write the data set in ASCII or BINARY format. The first line of each time step data set is a time value. The remaining lines of each time data specify a scalar value for each grid point. We can distinguish each data set by the number of tokens in a line. If there are two number values in a line, it is a 1-dimensional data set, the first value being a *x* value and the next a *f(x)* value. For a 2-dimensional data set, each line will have *x*, *y* and *f(x)* values. For a vector data set each line will have *x*, *y*, *r(x,y)* and *theta(x,y)* values. These data can be also written in BINARY format by simply adding the number of grid points after time value. Each value except the number of grid points is written in an 8-byte double format and the number of grid points is written as a 4-byte integer.

To visualize isosurfaces we provide a C program that calculates the isosurfaces and pipes the calculated triangles into Scivis. When the user wants to change the isosurface threshold value, to calculate a new isosurface, the server sends this parameter to the client. The client recalculates the isosurface from this value, and sends the triangles to the server. This procedure repeats until the server sends a stop or disconnection command to the client.

3.3. Filters

In order to provide a general visualization/data analysis tool we realize that we must provide a system that end users can customize, and provide new filters to analyze their data. We provide APIs so individual users can implement their own custom algorithms.

We have developed our application with the assumption that the "best filter" should be chosen by the end user. Features of Scivis will make it easy for the users to write their own filters to manipulate their data. Our application will then add user-defined filters into the application.

We currently support two types of filters. This is based on the desired input of data from either one data set or multiple data sets. The following code segment shows the class *Filter.*

```
package sv.kernel;
public abstract class Filter {
   public static final int SINGLE_DATASET   = 1;
   public static final int MULTIPLE_DATASET = 2;
  /**  filter type   */
  protected int type;
  /**
   * DataSet. The user should manipulate over this DataSet.
   */
  protected Vector dataSets = null;
  /**
   * argument vector if necessary
   */
  protected Vector args = null;
  /**
   * set filter type
   * (this function will be called when read sv_menu file)
   * @param t filter type
   */
  public void setFilterType(int t) {
    type = t;
  }
  /**
   * assign data set to be manipulated
   * @param d data set vector    */
  public void setDataSet(Vector d) {
     dataSet = d;
  }
  /**
   * set arguments for the filter
```

```
   * @param v argument vector
   */
   public void setArgument(Vector v) {
       args = v;
   }
      ...................
   /**
    * calculation over DataSet
    */
  public abstract DataSet performFilter();
}
```

Using the above code an end-user can implement his/her own filter by just filling in the section for the *performFilter* method over the class *DataSet*. When Scivis reads in the *sv_menu* file, a file which tells Scivis which are the user-definable filters, and how the menus will look in scivis, it also reads in the name of the class that the filter is defined in.

Some filters need data from multiple data sets, which are distributed on multiple windows. We define such filter types as *MULTIPLE_DATASET*. One such example of this filter type is when the user wants to add data from one window to data in another window. We provide support for the user to point and click their mouse to specify which data sets he/she wants to operate on in a class *Filter*.

If a user defines his/her own filters, they can become a part of our application by simply adding a simple interface definition file (*sv_menu*) as follows:

```
<menu> Approximation
  <item>  NAME="Cubic Spline", CLASS="CubicSplineFilter", TYPE=1 </menu>
```

This HTML-like script is a definition of the user interface menu. There are three fields in the script item. The first field is the NAME, which is the name of the filter. The second field is the CLASS field, which is the name of the Java class that was coded. Finally, the TYPE field is used to let this application know whether it need single data set or multiple data set before executing the filter.

4. Software Features

4.1. Collaboration

There are several collaborative applications in Java, which were developed to give applications porting APIs. Since these applications are in the stage to be upgraded from JDK 1.0.2 to JDK 1.1, they can not use all of the features of JDK 1.1, which are vital to our system. Thus, we decided to write our own collaborative system.

Another reason to implement our own collaborative system instead of using existing collaboration tools is that we need to have a system that lets one create user definable filters. *Collaboration Toolsets* developed at the Old Dominion University is based on replacement of the java.awt package. Each java.awt component is replaced with equivalents, but the new component intercepts all the user-events in the application and sends them to the collaboration server. The collaboration server then forwards the events to all other instances of this application, where it is posted to the corresponding component. Therefore, to port an application

Collaboration Toolsets, the application writer has to override method *handleEvent*. Since this collaborative tool needs predefined methods to collaborate, we could not port the filters since they are not predefined. The end users must also have to know how their filters interact with others. Another difficulty is that the filters may be different among different servers. If a server performs a filter, which is not defined on the other server, the application can not be collaborative.

To support collaboration among users, we have made use of the Java Socket and ServerSocket classes. In the beginning stage of our software, we intended to develop collaboration work using RMI (Remote Method Invocation) since Java 1.1 supports RMI. This feature provides simplicity and inherent portability, and, coupled with the Delegation Event Model, it lets us create an extensible collaboration framework that can be used even with proprietary components. RMI also provides a simple solution to distribute the processing of intensive tasks from the client to the server among different servers. But we found that RMI is much slower than using socket as shown in Table 1. The test results of Table 1 are based on Ping performance[14]. Since execution speed is a major factor in designing our system, we choose to develop our own collaborative system by using Socket.

However, we must consider other methods to implement the collaboration features. One of these is a collaboration tool which will be released from Sun Microsystems. When it is available, we can implement the collaboration features much easier and provide the APIs to customize the collaboration. It can also make it possible to avoid expensive maintenance and nasty programming details of socket programming.

Table 1 Performance Comparison on Ping performance(msecs)

	CORBA/IIOP	RMI	HTTP/CGI	Socket
Performance	3.3	5.5	603.8	2.0

Figure 2 Collaboration Architecture

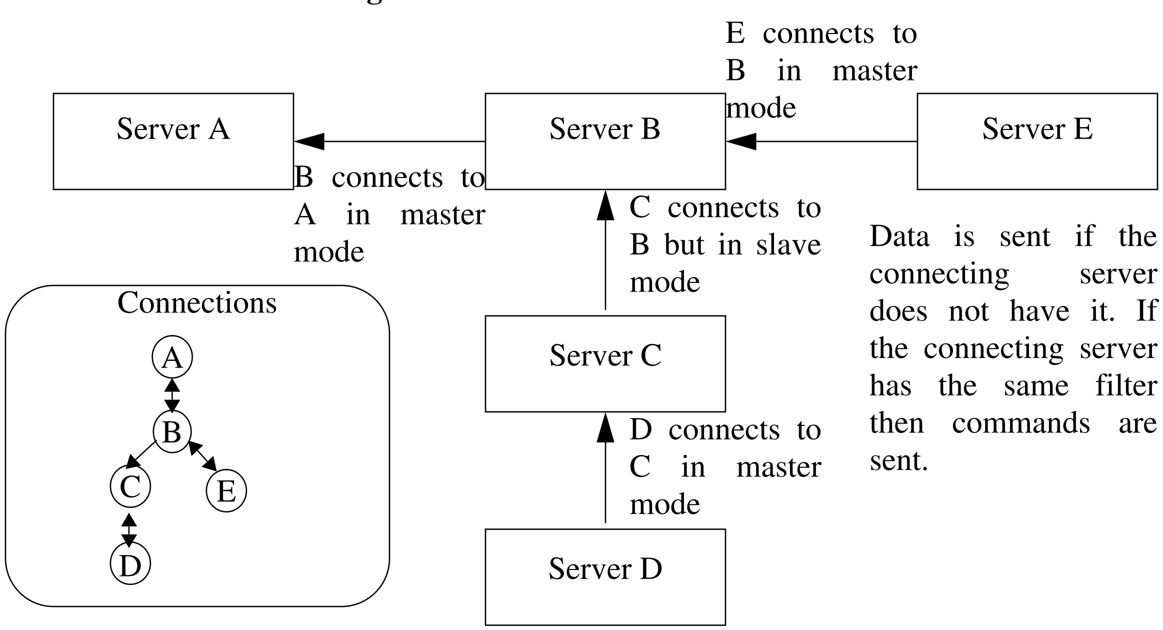


Figure 2 shows our collaborative architecture. To establish a collaboration connection between two servers, A and B, Server B sends the request for a collaboration to Server A, Server A replies to either accept or deny this. If Server A sends an exception message to Server B, the two servers exchange the names of their filters, their data sets, and a connection lists which contains the other collaborator's addresses.

The connection is run in master/slave mode, by selecting the mode when the connection request is received from the other server. The actions of the "master" are reflected on all other collaborator's windows. "Slave" users are limited to simply observing the actions of the "master". Any actions that a "slave" user attempts to initiate on the running program will be ignored. Verbal contact should be maintained between the "master" user (who should be giving a running explanation of his actions as he uses the desired filter) and the "slave" users (who should be allowed to interrupt to ask questions at any time). In the current release, we are using a simple chat program to communicate among the collaborator. We are planning to add a whiteboard for more flexible communications in the next release.

In Figure 2, Server A is started, and Server B request for a connection from Server A. Server A lets Server B be a Master, so now A and B are Masters. Server C connects to Server B, but Server C is only allowed to be a slave of B. Server D connects to C and C allows D to be a master. Server E connects to Server B, and B allows E to be a Master. Now Server A gets some data and all of the other servers obtain this data. When server D filters some data, Server C sees this, but the rest do not. When server B filters some data, all of the servers see the command to filter the data. When Server D executes a filter, we can consider two cases, Server C has the same filter or not. If Server C has the same filter, then Server C executes that filter. Otherwise Server D sends the result data set to Server C.

4.2. Summary of Software Features

Scivis allows users to visualize their data, which is piped in via sockets. The following summarize the basic features and user interfaces of this software:

1. Visualize 1-dimensional *(x, f(x))*, 2-dimensional *(x, y, f(x,y))*, vector, isosurface data and contour plots.

2. This software can show next and previous time data, and animate over time. Users can choose the animation speed by a simple slider interface.

3. Postscript Output.

4. Collaborative.

5. User Definable Filters.

We provide some basic filters which includes merging of data sets by a simple point-and-click interface, extraction of a 1-dimensional data from a 2-dimensional dataset over the x and y values, extraction some portion of the data, and sample numerical analysis filters such as Cubic Spline, Least Square Methods. As the name implies, the user can add their own filters in the system by referring sample filters provided by us.

6. Customizable color maps.

The user can customize the color by dragging the color graphs. We support RGB and HSB color models. This software can also store the customized color map to a file and can refer stored color map at any time.

7. Different zoom modes for all types of plots.

4.3. Future Work

We have implemented our software in Java 1.1 and have encountered a few problems. The problems stem from the fact that Java 1.1 is not 100% cross-platform compatible. For example, the pop-up menu, 1-3 button mouse, and keyboard mapping are different on different machines. As a result, we are trying not to use these features in order to enhance cross-platform compatibility.

We realize that Scivis has a sluggish performance when the software draws 3D colormapped surface plots. When we try to animate these data sets over time, we need at least 300msec for the screen to redraw. Scivis can be drastically improved if we had hardware acceleration for the graphics. We are going to use the Java 3D API when it is available in the future releases. As a part of Java Media, the Java 3D API is an application programming interface used for writing stand-alone 3D graphics applications or Web-based applets. It gives the developer high level constructs for creating and manipulating geometry and for constructing the structures used in rendering that geometry. Application developers can describe very large virtual worlds using these constructs, which provides Java 3D with enough information to render these worlds efficiently [7].

In the next release, we will provide a much simpler installation and usage environment by using the Java Runtime Environment (JRE) and Java ARchive (JAR) file format. JRE is the minimum standard Java Platform for running Java programs. It contains the Java virtual machine, Java core classes, and supporting files. It makes it possible for the users to install the latest JDK version to run this software.

JAR is a platform-independent file format that allows the user to bundle a Java applet or application and its requisite components. It will compress and bundle the files, minimizing the distribution size. This will assist in reducing the application start up time.

5. C and FORTRAN Interfaces

5.1. Function Calls

We provide users with C and Fortran 90 function calls to send data to Scivis, which are shown below:

1. For 1-dimensional data:

- `java_fser(name, time, x[], fx[], nx)`
  - Writes the x and f(x) array data to a binary file. Time is a double precision scalar, and f and f(x) are vectors of length `nx`.
- `java_ser(name, time, x[], fx[], nx)`
  - Same as above but instead of writing the data to a file, it is sent directly to Scivis.
- `java_mser(name, nt, time[], x[], fx[], nx[])`
  - Sends multiple x and f(x) array values to a server. Time is now an array of size `nt`, and x

and `fx` are size `nx*nt`. The arrays are formed by concatenation of individual time data into one array.

- `java_bbser(name, time, bb, nx, fx[])`

  - Same as `java_ser` but instead of sending the x array it just sends the minimum and maximum values of the x array in a vector of size 2. This assumes that the data has uniform mesh spacing in the x direction.

- `java_bbmser(name, nt, time[], bb[], nx[], fx[])`

  - Sends several bounding boxes and time data to a server.

- `java_pser(name, time, x[], fx[], nx, np)`

  - The difference between this call and `java_ser` is that np = the total number of pieces that the data will be sent in. It is meant for parallel computing runs where there are ghost regions from different processors. Here multiple data will be defined on different pieces and will not be shown on Scivis.

- `java_pmser(name, nt, time[], x[], fx[], nx[], np)`

  - Sends part of several data sets which are produced from the parallel computer to a server.

2. 2-dimensional function calls are similar to 1-dimensional function calls:

- `java_fser2d(name, time, x[], y[], fxy[], nx, ny)`
- `java_ser2d(name, time, x[], y[], fx[], nx, ny)`
- `java_mser2d(name, nt, time[], x[], y[], fx[], nx[], ny[])`
- `java_bbser2d(name, time, bb, nx, ny, fx[])`
- `java_bbmser2d(name, nt, time[], bb, nx[], ny[], fx[])`
- `java_pser2d(name, time, x[], y[], fx[], nx, ny, np)`
- `java_pmser2d(name, time[], x[], y[], fx[], nx[], ny[], np)`
-

3. The following functions are used to send vector data to Scivis:

- `java_vser(name, time, x[], y[], r[], theta[], nx)`

  - Sends *x, y, r(x, y), and theta(x, y)* value arrays to a server. These arrays consist of the class `VectorData`. Class `VectorData` also includes, for example, name and time information. `VectorData` consists of a class `VectorSet`. In this case, class `VectorSet` includes only one `VectorData` as a form of the class `Vector`.

- `java_vmser(name, nt, time[], x[], y[], r[], t[], nx[])`

  - Sends several vector data sets to a server. The arrays *x* and *y* are beginning point arrays and the array *r* represents the size of the vector. The array *t* represents the rotation angle by radian.

4. For volumetric data, we provide the following function:

- `java_iso(name, time, val[][][], shape(nx, ny, nz))`

  - Isosurfaces are calculated on a remote client, which yields triangle information and is then sent to Scivis to visualize in real-time. Triangulation is utilized because it is expensive to send 3D data sets to Scivis.

6. Conclusion

We have implemented a collaborative data analysis and visualization application, named Scivis.

Scivis is flexible and very usable and thus we feel that researchers can benefit from using our software. One such group that is already taking advantage of Scivis is the “Binary Black Hole Grand Challenge” group. We have provided APIs to make it easy for a user to incorporate Scivis into their research.

7. References

[1] K. Mani Chandy, *Caltech Infospheres Project Overview: Information Infrastructures for Task Forces,* via http://www.infoshperes.caltech.edu/.
[2] Lukasz Beca, et. al. *TANGO - a Collaborative Environment for the World-Wide Web*, via http://www.npac.syr.edu/tango/papers.html.
[3] NCSA, *Habanero Developer Guide*, via http://ncsa.uiuc.edu/.
[4] Bjorn J.Kvande, *The Java Collaborator Toolset*, Master's Project Report, Department of Computer Science, Old Dominion University, Aug. 1996. or via http://www.cs.odu.edu/~kvande/Projects/Collaborator.
[5] K, Arnold, J. Gosling, *The Java Programming Language,* Addison-Wesley, 1st edition, 1996.
[6] J. Gosling, B. Joy, and G. Steele, *The Java Language Specification*, Addison-Wesley Developers Press, Sunsoft Java Series, 1996.
[7] Sun Microsystems, *The Java 3D API White Paper*, via http://www.javasoft.com/marketing/collateral/3d_api.html.
[8] Sun Microsystems, *The Java Language White Paper*, via http://java.sun.com/docs/overviews/java/java-overview-1.html.
[9] Sun Microsystems, *Object Serialization Specification,* via http://www.javasoft.com:80/products/jdk/1.1/docs/guide/serialization/index.html.
[10] Sun Microsystems, *RMI Specification,* via http://www.javasoft.com:80/products/jdk/1.1/docs/guide/rmi/index.html.
[11] K. Mani Chandy, E. Schooler, *Designing directories in distributed systems: A systematic framework,* IEEE International Symposium on High Performance Distributed Computing (HPDC-5), Syracuse, NY, Aug 1996.
[12] H. Abdel-Wahab, et. al. *Using Java for Multimedia Collaborative Applications*, Proceedings of the 3rd International Workshop on Protocols for Multimedia Systems (PROMS'96), October 1996, Madrid. or via http://www.cs.odu.edu/~kvande/Projects/Collaborator/.
[13] K. Mani Chandy, Adam Rifkin, Paolo A.G. Sivlotti, *A World-Wide Distributed System Using Java and the Internet,* IEEE International Symposium on High Performance Distributed Computing (HPDC-5), Syracuse, NY, Aug 1996.
[14] Orfali, Robert, C*lient/Server programming with Java and CORBA,* 1997 John Wiley & Sons Inc.
[15] Tom Haupt, *Adaptive Mesh Refinement,* via http://www.npac.syr.edu/projects/bbh/AMR/amr.html.
[16] Matthew Choptuik, *Ser,* via ftp://helmholtz.ph.utexas.edu/pub/rvs/rvs/src/.
[17] The Binary Black Hole Grand Challenge, via http://www.npac.syr.edu/projects/bh/.