Towards Web-Based Computing[†]

Kiyoko F. Aoki and D.T. Lee
Department of Electrical and Computer Engineering
Northwestern University
Evanston, Illinois 60208, U.S.A.
Email: {dtlee,kaoki}@ece.nwu.edu

Abstract

In a problem solving environment for geometric computing, a graphical user interface, or GUI, for visualization has become an essential component for geometric software development. In this paper we describe a visualization system, called *GeoJAVA* with a GUI, which enables the user or algorithm designer to execute and visualize an existing code in the library or develop a new code over the internet. The library consists of geometric code written in C/C++. The GUI is written using Java programming language. Taking advantage of the socket classes and system-independent application programming interfaces (API's) that come with the Java language, *GeoJAVA* provides a platform independent environment for distributed geometric computing.

Users are able to remotely join a "channel" or discussion group in a location transparent manner to do collaborative research. Then the visualization of an algorithm, a C/C++ program located locally or remotely and controlled by a "floor manager," can be viewed by all the members in the channel through a visualization sheet called GeoJAVASheet. The execution of the algorithm can then be re-run dynamically to demonstrate the changes in the output upon modification of the input data by the floor manager. A chat box is also provided for verbal communication among all the members.

Furthermore, this system not only allows visualization of pre-compiled geometric codes, but also serves as a web-based programming environment where users may submit their own geometric code, compile it with the visualization libraries provided by the system, and visualize these directly over the web, sharing it with other users immediately.

†Supported in part by the Office of Naval Research under the Grants No. N00014-95-1-1007 and No. N00014-97-1-0514.
A preliminary result was presented at the "Workshop on Geometric Software," held in INRIA, Sophia-Antipolis, France in June 1997.

1 Introduction

As the computer and communication technology advances, communication via e-mails or World Wide Web has become commonplace in our daily activities. In the computing world, collaboration via Internet has gained popularity recently. The notion of a "collaboratory" is introduced in a report on "Distributed, Collaboratory Experiment Environment"[8], which refers to an integrated, tool-oriented computing and communication system that supports scientific collaboration. In other words, it is a computing system that allows remote parties to gain access to scientific resources such as expensive and physically large equipment that would otherwise not be accessible. As in any other scientific computing disciplines, the area of geometric computing would find such a collaborative system beneficial because of the large size of the libraries used to implement geometric algorithms. The effort required to download and install these libraries are oftentimes not worth it, especially when the user only needs them for a single program or algorithm that he/she would like to execute or implement.

The idea of a collaboratory is also to enable remote users with expertise in specific areas of a scientific field to collaborate with one another, viewing the data that is pertinent to each user's specialty in order to come up with a solution to a particular problem. For computational geometers or practitioners dealing with geometric data, most everyone is interested in the execution and analysis of geometric algorithms, so a collaboratory for geometric computing would provide remote users in a group with the facilities to view the execution of an algorithm implemented by any member in the group, and to give feedback to one another regarding the algorithm.

In order to implement such a collaboratory, distributed visualization of algorithms or at a bare minimum remote execution of algorithms, needs to be supported. Any user connected to a network should be able to have access to the collaboratory, and immediately begin collaborating with other users currently connected to the collaboratory. This implies that such a collaboratory must be independent of the users' platform. To implement a collaboratory from scratch that meets the requirement is by no means obvious, especially when visualization or graphics output is involved, for which all sorts of display devices have to be supported. However, since Internet and web browsers on the World Wide Web are readily accessible by many researchers on the network, building a collaboratory on the web seems to be a plausible solution. The Java programing language developed by Sun Microsystems, which is considered platform-independent, is a natural choice of language to use to implement such a collaboratory. Thus came the development of the *GeoJAVA system*, a Web-based interactive visualization system that provides (1) a Java-based GUI (graphical user interface) called GeoJAVASheet, (2) a Java-based "chat" box that allows users in separate groups to communicate verbally, (3) a library of geometric algorithms called GeoLIB, (4) a compilation tool allowing users to implement user-defined algorithms using the GeoLIB library, and (5) distributed visualization of geometric algorithms.

There are many applications of this system. Among them are "distance" learning and collaborative research on geometric computing. For example, a "classroom" can be formed by a group in which the teacher of a geometric code, say "A," initially has control of the "floor." That is, A is the user interacting directly with the code, and the rest of the users in the group become students. Each student in the group can then watch the execution of the same code, say a Delaunay triangulation program, that A has executed. Each student will be able to see the same set of points that A is sending to the program as input and the animated execution of the triangulation program on each of their browsers. If students have questions or comments, they may type them in the chat box, and may also receive control of the "floor" upon release by A to input their own set of input points that is distributed to the rest of the group.

In doing collaborative research, the current problem in the development of a new algorithm is in explaining what the actual execution looks like to remote parties. Up to now, researchers have been using e-mails or transferring files of their algorithms, describing verbally what each step of the execution is on a "frame by frame" basis. The *GeoJAVA system* provides a solution to this problem. For instance, a group may consist of several researchers located at different sites. One of the researchers, say "B", may have developed a new algorithm to solve a specific problem for which she would like advice from the others. So upon receiving control of the floor, B may execute her algorithm to present to the others. Any of the other researchers may then receive the floor to give advice or make improvements on the algorithm. The changes to the code may be made by B, the code recompiled, and then immediately re-executed for the others to see. Verbal communication is all performed through the chat box.

Through these examples, one can see the benefits of visualization; the phrase, "a picture is worth more than a thousand words" indeed rings true. However, not only can the *GeoJAVA system* visualize static data, but it can also serve as an *interactive* visualization system. Users may manipulate the visualized data and simultaneously see the change in the algorithm's output. This feature applies to programs that are in the library or are user-defined, and runs on top of a distributed environment, which makes the *GeoJAVA system* a powerful tool with different utilities.

Returning to the first example, then, after the Delaunay triangulation has been executed, user A may demonstrate how the triangulation changes when a specific point is moved to a different location by simply selecting a point and moving it across the sheet. The *GeoJAVA system* automatically handles the dynamic re-execution of the algorithm and updating of the sheets in the group.

As technology advances and becomes more readily available, audio and video communication can easily be added to the system for a greater "collaboratory" feel. This is due to the modularity of each component of the system, which will be described in detail in the following sections.

In the next section, we briefly introduce features of the Java programming language pertinent to the *GeoJAVA system*. This will be followed by a review of related work, then a description of the design of the *GeoJAVA system*, and finally we discuss our plans for future work.

2 Java Programming Language

The Java programming language by Sun Microsystems provides two major features that make it very applicable to distributed geometric computing. They are *sockets* and *GUI objects*. By simply declaring a new ServerSocket() data object in a server application, client applications can begin communicating to it by using a Socket() class, declared similarly, without worrying about the type of system on which the applet or application may be running. GUI objects such as buttons, canvases and panels can also be created easily with predefined classes provided by the Java library.

Both datagram (User Datagram Protocol or UDP) and stream-based (Transmission Control Protocol, or TCP) sockets are provided in the Java application programming interface (API). However, security issues prevent applets from waywardly creating sockets on users' machines; sockets can only be created on the host that provided the applet. Therefore, if a Java application is running on the server, another applet cannot create a socket on the remote host to even connect back to the server application. Datagram sockets require such a configuration. Although Java applications (as opposed to applets), would work without a problem, that would defeat the purpose of allowing users to easily access the system without having to download the application itself. Stream-based sockets, however, can be used in an applet where the TCP socket is created on the server and the applet communicates directly to that port. Therefore, using TCP sockets, applets can be easily created that provide distributed geometric computing.

In addition to the language limitations, TCP is favorable because of its stability, especially in large networks. UDP packets are not "acknowledged" by the recipient, so the farther the distance between the sender and receiver, the more prone the packet is to get lost. This often results in the user's algorithm "hanging" during execution, without any method of recovering itself. The user is unfortunately forced to kill the execution of the algorithm in this case. Adding error checking packets for acknowledgements would most likely only increase the number of lost packets over the network. We create a GeoLIB library that supports TCP messaging. Since setup and disconnect packets are not used for each message sent (it is only required upon connection/disconnection to/from the system), the packet sizes are smaller, and TCP's reliability prevents the transmission speed from getting degraded as much, compared to the UDP transmission protocol.

3 Related Work

As is evident at the "Computational Geometry Interactive Software" page, (http://www.cs.duke.edu/˜jeffe/compgeom/demos.html), many geometric algorithm visualization tools have been implemented, and the "Complete Collection of Algorithm Animations" at http://www.cs.hope.edu/˜alganim/ccaa/geometric.html gives a comprehensive list of geometric algorithms written in Java. These applets demonstrate Java's "write-once-run-everywhere" concept[7]. Once a user implements her Java applet that demonstrates an algorithm, any user with a Java-enabled browser can execute it. The following is a listing of a few notable java applets from these lists.

GeomNet at the Center for Geometric Computing at Johns Hopkins University (http://www.cgc.cs.jhu.edu/geomNet/).
GeomNet is a system for performing distributed geometric computing over the Internet. It provides a list of GeomNet supported algorithms from which a single user can choose an algorithm they would like to execute. Geometric computing is distributed in that the algorithms are available for anyone on the Internet who would like to see the execution of an algorithm. However, it is not implemented for groups of users to simultaneously see the execution of a single algorithm. One of the components of GeomNet is **Mocha** [2] at the Center for Geometric Computing at Brown University (http://loki.cs.brown.edu:8080/pages/Mocha.html). Mocha is a Java applet that communicates with an "algorithm server" which allows users to select geometric algorithms for which they can provide input.

VoroGlide by Christian Icking, Rolf Klein, Peter Köllner, and Lihong Ma (http://wwwpi6.fernuni-hagen.de/java/anja/index.html.en).
VoroGlide is an applet that smoothly maintains the convex hull, Voronoi diagram and Delaunay triangulation of the user's input while points are added or moved. It illustrates incremental construction of the Delaunay triangulation and includes a recorded demo.

ModeMap by David Watson (http://www.iinet.com.au/~watson/modemap.html).
Modemap is an applet that draws Voronoi diagrams, Delaunay triangulations, natural neighbor circles and radial density contours on a sphere. This is a single 3D applet whose only purpose is to illustrate the relationship between these geometric concepts on a sphere. It also allows for moving of points.

The Geometry Applet by David Joyce (http://aleph0.clarku.edu/~djoyce/java/Geometry/Geometry.html).
The Geometry applet illustrates Euclid's *Elements*. It lets users set up simple geometric

objects in 3D as well as constraints through the use of Java parameters, and then displays the effects as objects are moved.

Alpha-shape demo from NCSA, which requires VRML.
(http://fiaker.ncsa.uiuc.edu/alpha/demo.html).
This alpha-shape demo is an online Alvis demo that serves as a web-based interface to Alvis software. It is used to clarify concepts of Alpha Shapes and Alpha Ranks. Three data sets are available. Please refer to http://fiaker.ncsa.uiuc.edu/alpha/reference.html for references regarding Alpha Shapes/Ranks.

Although these applets are successful in demonstrating various computational geometry algorithms, if a researcher, say, wanted to test and develop their own algorithm, they would not be able to make any practical use of these applets, let alone demonstrate the same execution of their algorithm simultaneously on remote parties' machines. This lack of interactivity and customizability motivates the development of the *GeoJAVA system*.

Other Java-based collaborative systems also worth noting are Tango [3], Promondia [6], and NCSA's Habanero. **Tango** is a Java-based system that allows remote users to collaborate over the Web. Users with applications that they would like to make distributed may incorporate Tango's API into their code, which would allow their application to communicate to a central server that handles the "distribution" of the application. It provides nice multimedia features and is geared towards medical and scientific research. **Promondia** is a system that provides a framework for real-time group communication. Its focus is on group-conferencing using a shared whiteboard, video, and chat system. **Habanero** (http://www.ncsa.uiuc.edu/SDG/Software/Habanero/) is a framework for sharing Java objects with colleagues distributed over the Internet. It is similar to Tango where single-user applications are transformed into multi-user, shared applications using their provided API.

Finally, a Java-based implementation of **Collaborative Active Textbooks (JCAT)** on algorithms was developed by Digital Equipment Corporation [4]. This system, which takes advantage of a new feature in Java version 1.1 called Remote Method Invocation (RMI) technology, allows applets on different machines to communicate with each other, with the views of an algorithm located on different machines. Although JCAT runs on all Java-enabled browsers, at the time of this writing, only HotJava 1.0 can support the collaborative features because it requires JDK 1.1. The algorithms that are visualized are written in Java and are based on BALSA's notion of interesting events to communicate the operations of the algorithm to the views [5], and "group communication" is implemented by having each "student" specify the name of the "teacher's" machine where the algorithm is running.

The focus of Tango is different from that of the *GeoJAVA system* in that it is geared towards medical and scientific researchers. It is very useful in an environment where collaboration is needed from different people with completely different specialties. For example, a consultation for a certain surgical procedure may require the expertise of a neurologist, cardiovascular specialist and a physical therapist, where all three need different views of the same data. Technically speaking, the full-fledged Tango requires the installation of a plug-in for the browser and only works with Netscape 3.0+, whereas the *GeoJAVA system* is "java pure," and so any browser can be used to access it.

Promondia's focus is also different in that it attempts to give users a foundation for real-time communication using Java, as opposed to having any distributed application-based purpose. Their focus is on satisfying the increasing demand for other network services, such as real-time data feeds, group communication and teleconferencing (refer to http://www6.nttlabs.com/papers/PAPER100/PAPER100-java.html for an online version of their paper).

Habanero uses Java applications as opposed to applets, which means it is not necessarily web-based. Its components need to be downloaded, and only Java components can be used for collaboration. Thus, Habanero has a limited scope.

The main difference between JCAT and the *GeoJAVA system* would be location independence. Whereas *GeoJAVA system* users may access the system through a single page and form a group using a single channel name, JCAT requires channels to be formed by forcing "students" to specify the hostname of the "teacher's" machine. This requires knowledge of who the the floor manager is beforehand. That is, only the teacher has control of the algorithm, and the students may not request control of the floor. Also, the algorithm being displayed must be written in Java. Therefore, current algorithms written in C++ must be re-written in Java in order for it to be useful under JCAT.

The *GeoJAVA system* is based on GeoMAMOS (http://www.ece,nwu.edu/~theory/geo-mamos.html), part of which are GeoSheet [10] and GeoManager [1], which provide distributed visualization of geometric algorithms over a UNIX-based network. GeoSheet is the 2-D GUI for GeoMAMOS that is the interface with which users interact to communicate with their algorithms. GeoManager provides the dynamic manipulation of algorithms by allowing users to execute their program, then modify the original input data and simultaneously see the changes in the algorithm's output. A drawback of GeoMAMOS is that it was written in C/C++ for the X-Windows environment running Unix, so users who do not have access to such machines installed with the GeoMAMOS software are not able to make use of the visualization tool. In view of the above, a system-independent version has been implemented in the form of the *GeoJAVA system*.

The following section will describe the design of the *GeoJAVA system*, which allows visualization of users' algorithms written in C/C++ in a distributed fashion. Groups, or channels, are formed by simply specifying a common channel name when entering the system, and distributed visualization process can begin immediately upon execution of a program by the floor manager.

4 Design Description

The *GeoJAVA system* consists of six major components: (1) MultiServer, (2) ChannelGuide, (3) GeoJAVASheet, (4) GeoLIB, (5) Chat box, and (6) a compilation tool. The design of each of these components will be described next.

4.1 MultiServer

MultiServer is a Java application adapted from the Free Internet Conferencing Tools (FICT) home page at http://www.sneaker.org/fict/. Slight modifications were incorporated for it to provide the services for the collaboration management of the *GeoJAVA system*. It keeps track of the groups and the floor queue using the Connection and Vulture classes and also provides the dynamic manipulation of geometric algorithms.

MultiServer(int port, boolean verbose) The MultiServer class is a separate Java program running on the web server which creates the server thread and establishes the socket at the port number specified by the DEFAULT_PORT global constant. It then listens for connections from users. Whenever a new connection is made, a new Connection class is created and appended to the MultiServer's queue of connections. A new Vulture object is also created which ensures that all of the connections are valid. Each of these components will be described later. Note that because all new users connect through MultiServer, groups of users need not be concerned with the actual location of a "server" host. Thus location transparency is supported.

MultiServer handles the floor control for each group by maintaining a FIFO queue. When a new group is created, the floor queue for this group is empty. The first user to press the "Floor Request" button is added to the queue and becomes the "floor manager." Other users in the group who press this button thereafter are appended to the queue. When the floor manager presses the "Floor Release" button, he/she is then removed from the queue, and the successive user in the queue becomes the floor manager for the group.

Finally, MultiServer also functions as the "GeoManager" of the system. After the initial execution of an algorithm, if the user modifies the original data input, MultiServer will send messages to GeoJAVASheet to update the output dynamically. This allows for "true animation" and easier debugging of algorithms for the developer. Furthermore, when users join a channel in the middle of the execution of an algorithm, MultiServer allows these users to "catch up" on the algorithm execution.

Connection(String channel, String username, int port_number, String hostname) class The Connection object is responsible for receiving the messages sent by its corresponding GeoJAVASheet, chat box, or user program and then processing it appropriately. Messages from GeoJAVASheets go through MultiServer to broadcast to the GeoJAVASheets of every member of its group or to send to their user program, messages from chat boxes are broadcast directly to the chat boxes of every member of its group, and the user program's messages are sent to the GeoJAVASheets in its channel. Messages from GeoJAVASheets requesting for or releasing the floor are forwarded to MultiServer with its corresponding channel, hostname and TCP port number.

Vulture class The Vulture class is a simple thread that informs MultiServer when a connection has been closed or lost and cleans up the lists. Whenever possible, the Connection object will notify the Vulture thread when a connection is closed. But even if the Connection objects never notify the Vulture, this method wakes up every five seconds and checks all connections, in case a Connection unexpectedly crashes before it is able to send a "close" message.

The Java source code for these three classes are given in the Appendix.

4.2 ChannelGuide

ChannelGuide is an applet that ensures that multiple users do not enter the system with the same username. This is the applet that the user first sees when entering the *GeoJAVA system*. ChannelGuide takes the user and channel names requested by the user, communicates with MultiServer to check the current user lists for duplicates, and responds with the appropriate information, either allowing the user to start up GeoJAVASheet or prompting for a different user name. The ChannelGuide applet running on an X-Windows system is shown in Figure 1.

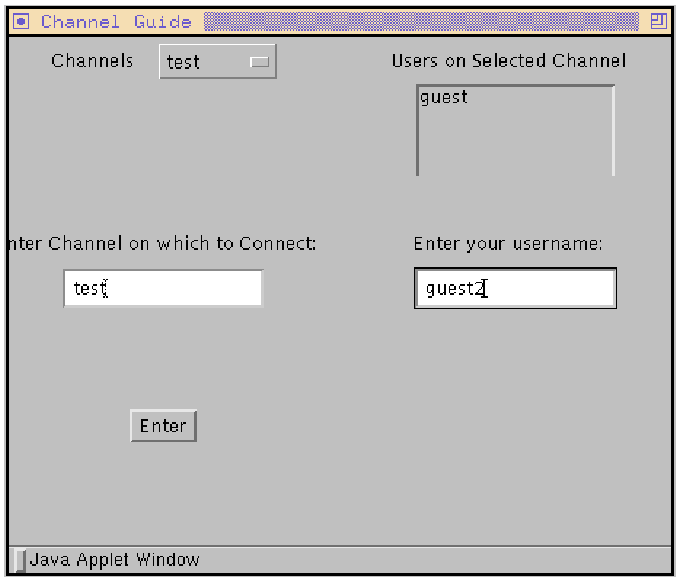
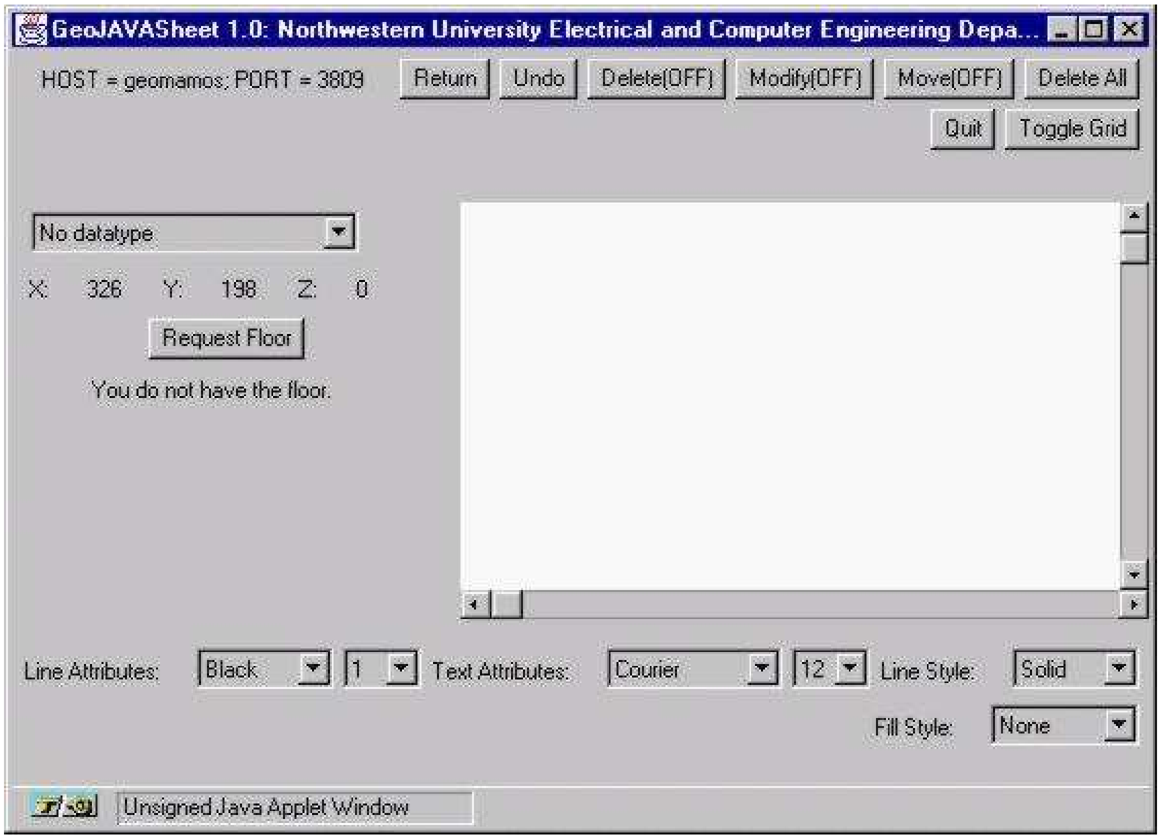


Figure 1: ChannelGuide Applet.

ChannelGuide functions by first communicating with MultiServer, requesting the lists of channels and users currently on the system. Once these lists are received, it processes them to display. If a used username is entered, then a message is displayed indicating that the entered name is invalid. Otherwise, the ChannelGuide window disappears, and a GeoJAVASheet and chat box are initiated with the user's user and channel name.

4.3 GeoJAVASheet

The GeoJAVASheet applet is actually a frame that contains (1) a panel onto which users may input graphical objects such as points, line segments, triangles, rectangles, polygons, polylines, circles, arcs, and various types of graphs, (2) a row of buttons on top: Return (for communication with the user's application program), Undo (undo the previous action), Delete (a specific object on the panel), Modify (an object's component), Move (an entire object), Delete All, Quit, Toggle Grid (reference lines), (3) a choice box on the left to select an object to input onto the panel, (4) a "floor" button under the choice box (this will be explained later), and (5) a row of property selectors on the bottom, such as line widths, line colors, font styles, font sizes, line styles, and fill styles. Figure 2 shows a GeoJAVASheet on a Windows machine.


GeoJAVASheet 1.0: Northwestern University Electrical and Computer Engineering Depa...

HOST = geomamos; PORT = 3809

No datatype

X: 326 Y: 198 Z: 0

You do not have the floor.

Line Attributes: Black 1 Text Attributes: Courier 12 Line Style: Solid

Fill Style: None

Unsigned Java Applet Window

Figure 2: GeoJAVASheet Applet.

GeoJAVASheet is simply a GUI that responds to (1) messages received from MultiServer, and (2) the user's actions such as hitting the Return button or requesting control of the floor. Internally, GeoJAVASheet maintains lists of the various geometric objects. Any time a new object is drawn on the panel, a new instance of that object is appended to the list to which its type corresponds. Users may modify or delete objects on the sheet using one of the buttons on the top row.

Geometric objects can also be displayed (and consequently added to the lists) based on messages received from the user program. These messages are in a specific format to determine the action to take, the data object being referenced, and the object's coordinates and properties. For example, if the user's program wants to display a red point of radius ten pixels at location (25, 30), then the message would look like: (IPC_WRITE, GEOPOINT, 25, 30, 10, RED). Once the message is received, it is parsed, added to MultiServer's appropriate internal list of data structures, and displayed on the panel.

The user program receives data for geometric objects by sending a request message and then waiting for a message containing the data for that object. GeoJAVASheet sends a message to its corresponding user program when the user presses the Return button. When the user program has explictly requested an object for input, and the user hits the Return button, then the last object appended to the panel corresponding to that displayed in the choice box (which has been updated with the object requested from the user program) will be stored in a message to be sent to MultiServer. The user program's ID is stored in GeoJAVASheet upon the program's initialization and thus has been stored in this message as well. Once MultiServer receives this message, it forwards it to the appropriate user program.

There is an additional feature in the application version of GeoJAVASheet where the data on the sheet may be saved to and opened from files. Two additional "Open File" and "Save" buttons provide this option. The data is stored in XFig format[14], just as in GeoMAMOS, but other formats will be supported in future versions. Figure 3 is an instance of the GeoJAVASheet application running under Windows.

4.4 GeoLIB Geometric Library

The GeoLIB library in the current version consists of two parts: the LEDA [12] and GeoLEDA libraries. Both libraries are written in C/C++. An advantage of this is that users who have developed algorithms written in C/C++ prior may continue to use their algorithms

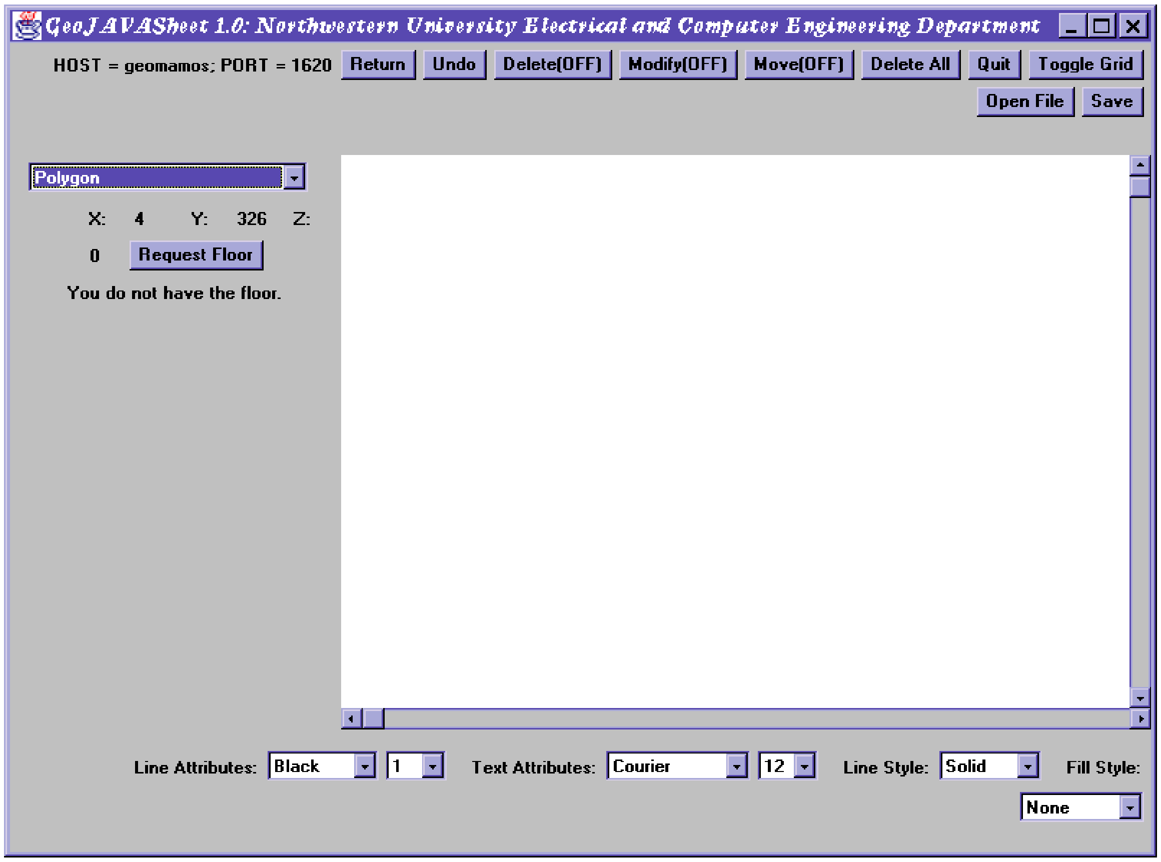


Figure 3: GeoJAVASheet Application.

without re-writing their code, and "new" users need not download a Java compiler if they do not already have one. In the future, we plan on incorporating the Computational Geometry Algorithms Library (CGAL) [13].

LEDA The GeoLIB library is based on the basic geometric classes and member functions of the Library of Efficient Datatypes (LEDA) library (currently version 3.5.2). By inheriting from this comprehensive library of geometric classes, GeoLIB provides both a complete library of geometric objects as well as several basic geometric algorithms.

GeoLEDA The visualization portion of the GeoLIB library is contained in the GeoLEDA library. GeoLEDA consists of geometric objects that (1) inherit from the objects in the LEDA library and (2) contain visualization member functions as well as interprocess communication (IPC) functions that provide the basic socket infrastructure for communication between the components of the *GeoJAVA system*. It also contains functions that implement basic geometric algorithms. This library is developed and used by the GeoMAMOS system. However, since GeoMAMOS uses UDP, all of the IPC functions have been modified to TCP functions, due to the reasons explained in Section 2. This is advantageous in that although initializations are slower, communication while connected is faster and more reliable. We briefly describe the main functions from the GeoLIB library that the programs use in order to visualize algorithms next.

IPCServiceSetup(), IPCServiceSetup(char* host, int portnum) This function sets up the initial TCP connection between the user program and GeoJAVASheet. It can have no arguments, in which case the user will be prompted for the host and port number at the command line, or it can take the host and port number for an input and output GeoJAVASheet that has the floor. It then establishes a socket connection between itself and MultiServer (with a Connection object serving as an interface). Any messages sent to MultiServer contain the GeoJAVASheet ID to which it corresponds so that MultiServer can forward them to the appropriate GeoJAVASheet. The user program must begin with the IPCServiceSetup() function before any visualization functions are called.

Graphic_Read and Graphic_Write (initiated from user program)
The Graphic_Read and Graphic_Write visualization functions are member functions implemented in every geometric object and are issued from the user program.

Graphic_Read will cause GeoJAVASheet to return to the program the last object input onto the sheet. The process is as follows: (1) Graphic_Read requested from user program (set choice box on GeoJAVASheet to the requested object type), (2) user inputs the object onto the sheet, (3) user presses the "Return" button located at the top of GeoJAVASheet, which (4) sends the data for the object to MultiServer, which forwards it to the user program.

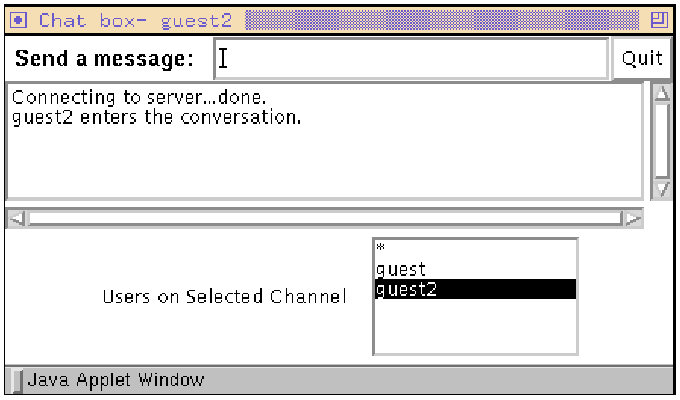
In Graphic_Write, the "opposite" action is performed. The user program sends object data to GeoJAVASheet (through MultiServer), and GeoJAVASheet displays the object. The process is as follows: (1) Graphic_Write command sent to GeoJAVASheet with the object's data, (2) the object is added to the corresponding list of geometric objects, and (3) GeoJAVASheet displays the object.

Using these libraries, a user may develop geometric algorithms that can perform any geometric computing they would like, without concern for implementing the display of intermediate or final results in their code.

4.5 Chat Box

The Chat Box is another applet adopted from the Free Internet Conferencing Tools web page and is a simple GUI consisting of a text field in which to enter text and a textbox in which all messages from users within the same channel are displayed. In addition, it lists the users currently in their channel. It maintains a PrintStream object that handles the displaying of all of the messages, a DataInputStream that receives the messages, and a Socket class with which the connection to MultiServer is made.

A user may select a specific username on the userlist in order to send messages to users privately, or messages may be broadcast to everyone on the channel by selecting the asterisk (*), also on the list. Figure 4 is an instance of the Chat Box running under X-Windows.



```
Connecting to server...done.
guest2 enters the conversation.
```

```
*
guest
guest2
```

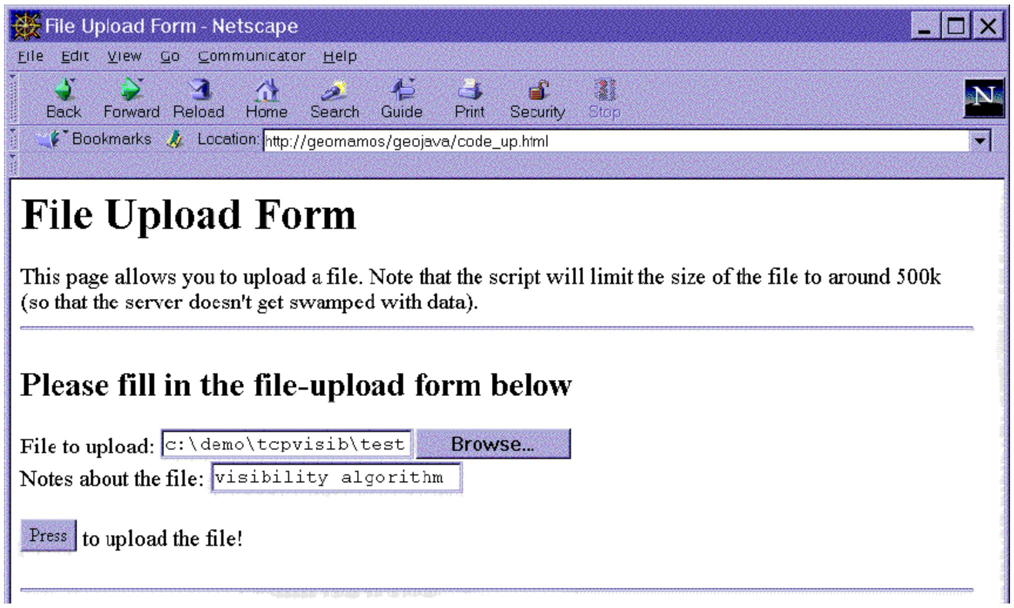
Figure 4: Chat Box Applet.

4.6 Compilation Tool

The compilation tool allows user-defined programs written in C/C++ to be compiled and executed directly on the *GeoJAVA* server. It is a series of common gateway interface (CGI) forms that upload the code, create a corresponding Makefile for it, compile it, and, if the compilation is successful, execute it with the user's corresponding host and port number. Unsuccessful compilations will result in a page listing the compilation errors so that the user may debug their own code and restart the process with the corrections.

With this tool, users no longer need to worry about obtaining the appropriate libraries for their system, installing, and compiling them. Of course, security precautions must be taken in order to ensure that the user's program does not contain any malicious code. It is possible for users to include system code which may corrupt the system. Therefore extra checks during the upload stage of the tool ensure that such malicious code is not used. Records of those who have been using the compilation tool keep track of the users and their actions, while still maintaining a certain level of privacy.

The following series of illustrations demonstrates the steps in the compilation process. Figure 5 is the first page, where the file is uploaded from the user's local disk to the server.


File Upload Form - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://geomamos/geojava/code_up.html
Browse...
Press

File Upload Form

This page allows you to upload a file. Note that the script will limit the size of the file to around 500k (so that the server doesn't get swamped with data).

Please fill in the file-upload form below

File to upload: c:\demo\tcpvisib\test

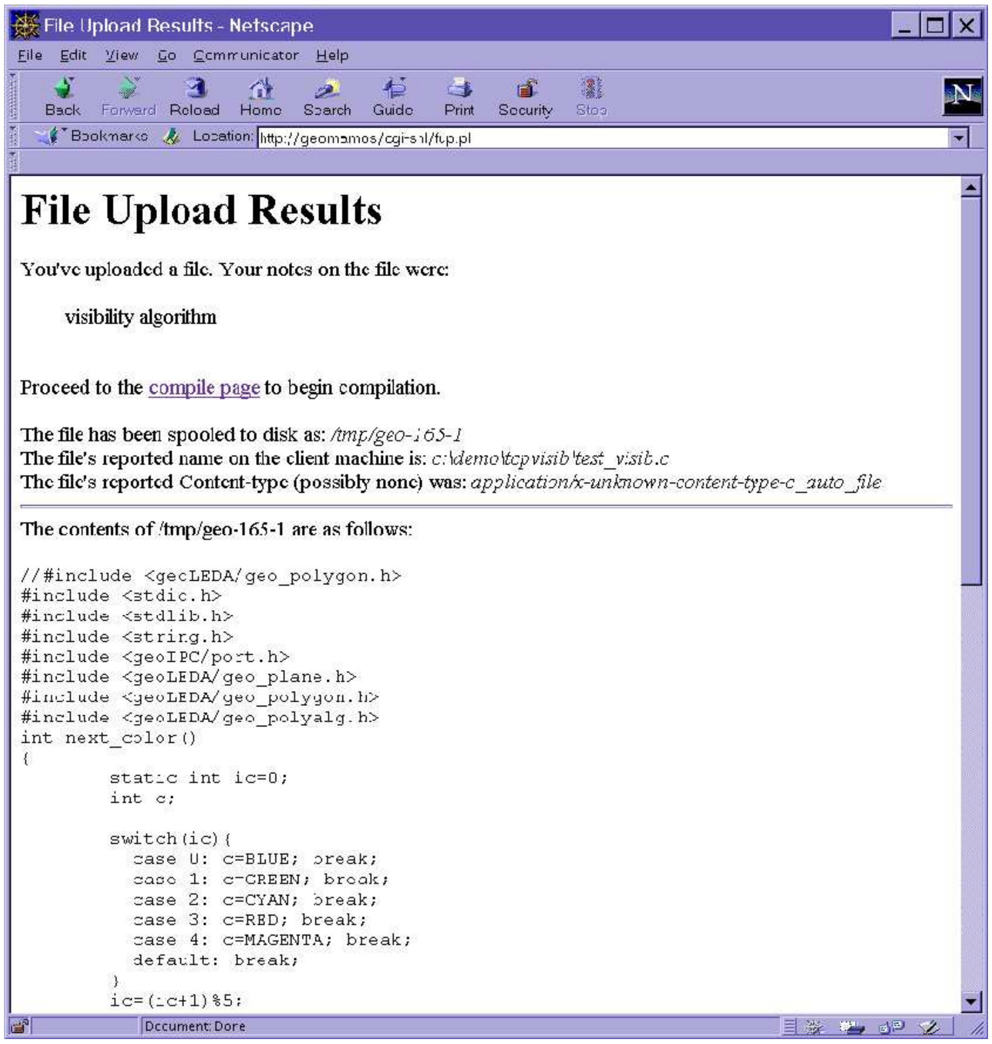
Notes about the file: visibility algorithm

Press to upload the file!

Figure 5: Code upload page

The user may specify a note for their code, normally the name of the algorithm. During the upload, the CGI script checks the filename for special characters and also parses the code for any "malicious code" such as system calls. Upon completion of the upload, the contents are displayed to ensure that the correct file has been uploaded completely, along with the user's notes, as in Figure 6. A link for a script that performs the actual compilation is also given.

The compilation will then take place, after which the output of the compilation will be displayed. If it is unsuccessful, a page such as in Figure 7 will be displayed, allowing the user to see the compilation errors. In this case, a link back to the first file upload page is given, but the user may also use the Back button on their browser to return to the first page.



File Upload Results - Netscape

File Edit View Go Communicator Help

Back Forward Reload Home Search Guide Print Security Stop

Bookmarks Location: http://geomamos/cgi-bin/fup.pl

File Upload Results

You've uploaded a file. Your notes on the file were:

visibility algorithm

Proceed to the compile page to begin compilation.

The file has been spooled to disk as: */tmp/geo-165-1*
The file's reported name on the client machine is: *c:\demo\tcpvisib\test_visib.c*
The file's reported Content-type (possibly none) was: *application/x-unknown-content-type-c_auto_file*

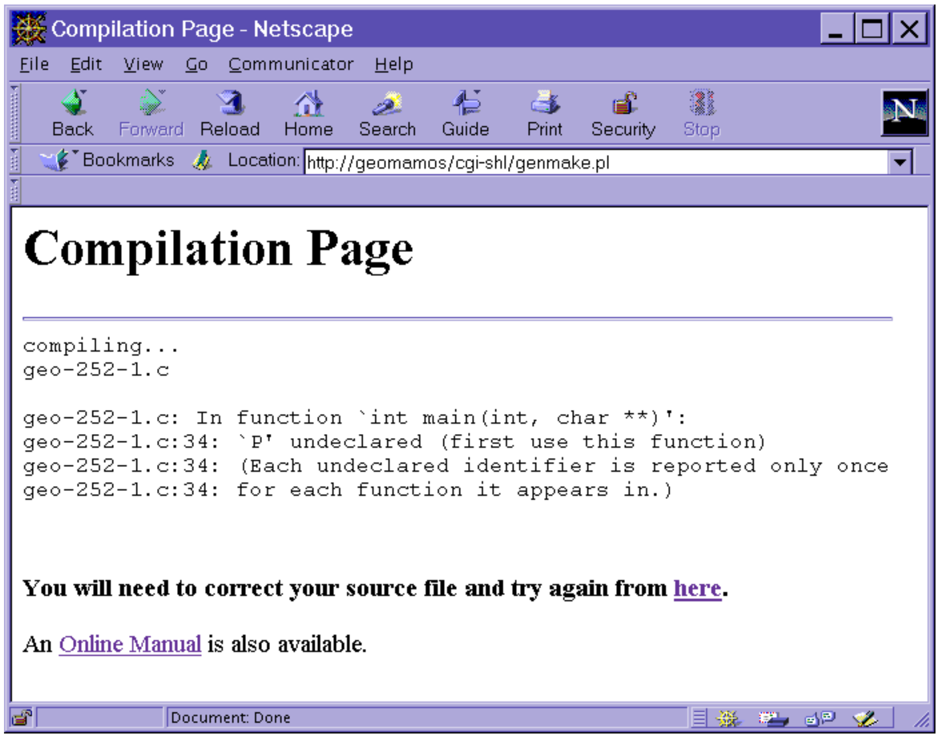
The contents of /tmp/geo-165-1 are as follows:

```
//#include <geoLEDA/geo_polygon.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <geoIPC/port.h>
#include <geoLEDA/geo_plane.h>
#include <geoLEDA/geo_polygon.h>
#include <geoLEDA/geo_polyalg.h>
int next_color()
{
        static int ic=0;
        int c;

        switch(ic){
          case 0: c=BLUE; break;
          case 1: c=GREEN; break;
          case 2: c=CYAN; break;
          case 3: c=RED; break;
          case 4: c=MAGENTA; break;
          default: break;
        }
        ic=(ic+1)%5;
```

Document: Done

Figure 6: Intermediate page


Compilation Page - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://geomamos/cgi-shl/genmake.pl
Document: Done

Compilation Page

```
compiling...
geo-252-1.c

geo-252-1.c: In function `int main(int, char **)':
geo-252-1.c:34: `P' undeclared (first use this function)
geo-252-1.c:34: (Each undeclared identifier is reported only once
geo-252-1.c:34: for each function it appears in.)
```

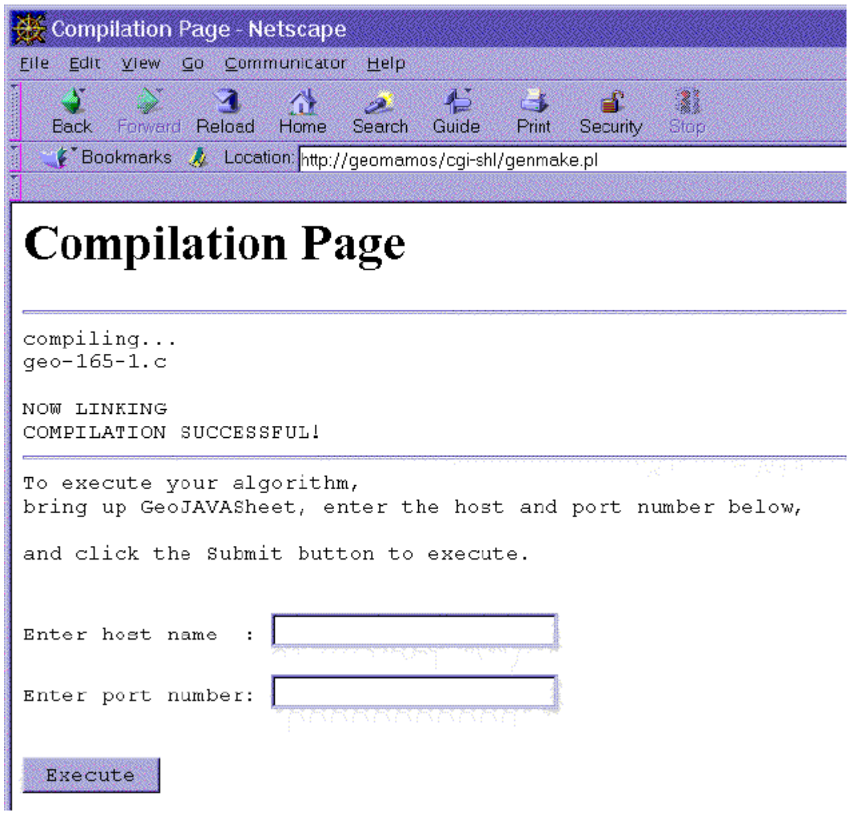
You will need to correct your source file and try again from here.

An Online Manual is also available.

Figure 7: Compilation output with error messages.

In addition, a link for an online manual is available in case the user would like to consult documentation on the usage of the functions in the GeoLIB library. If the compilation is successful, a form for the host and port number will be given as in Figure 8. When the user enters the appropriate data and clicks the Submit button, the algorithm will execute and the output be displayed on the specified GeoJAVASheet (and other GeoJAVASheets in the same channel). The completion of the execution of the algorithm will produce a page like Figure 9. At this point, the user may submit their code to add to the Algorithm Browser, which is shown in Figure 10. The Algorithm Browser lists the available algorithms for execution. Anyone with the floor may bring up this page and sample any of the algorithms given.

The next section gives a general description of the architectural design of the *GeoJAVA system*, followed by an example of a user's code and its execution.


Compilation Page - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://geomamos/cgi-shl/genmake.pl

Compilation Page

```
compiling...
geo-165-1.c

NOW LINKING
COMPILATION SUCCESSFUL!
```

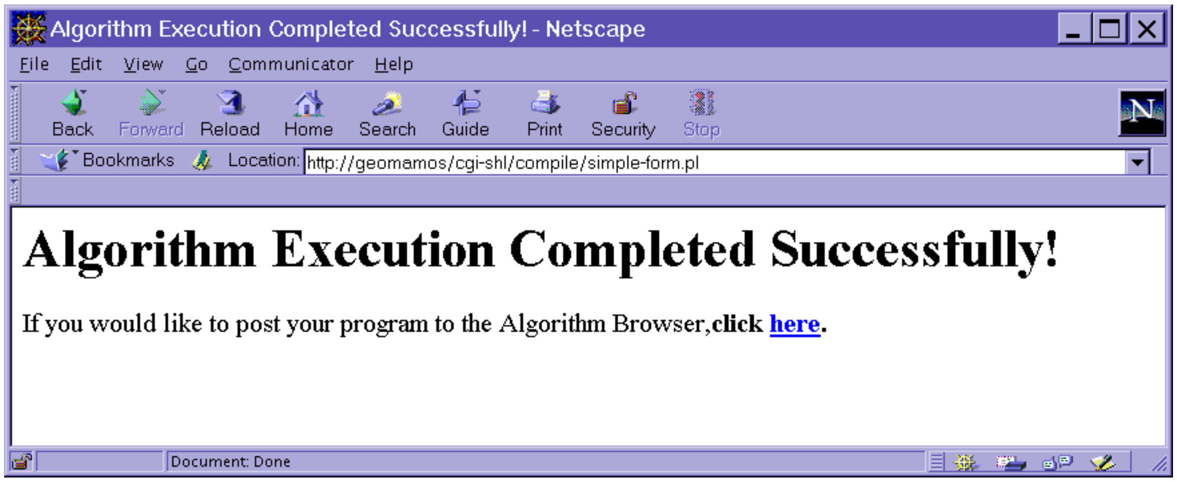
```
To execute your algorithm,
bring up GeoJAVASheet, enter the host and port number below,

and click the Submit button to execute.


Enter host name  :

Enter port number:
```

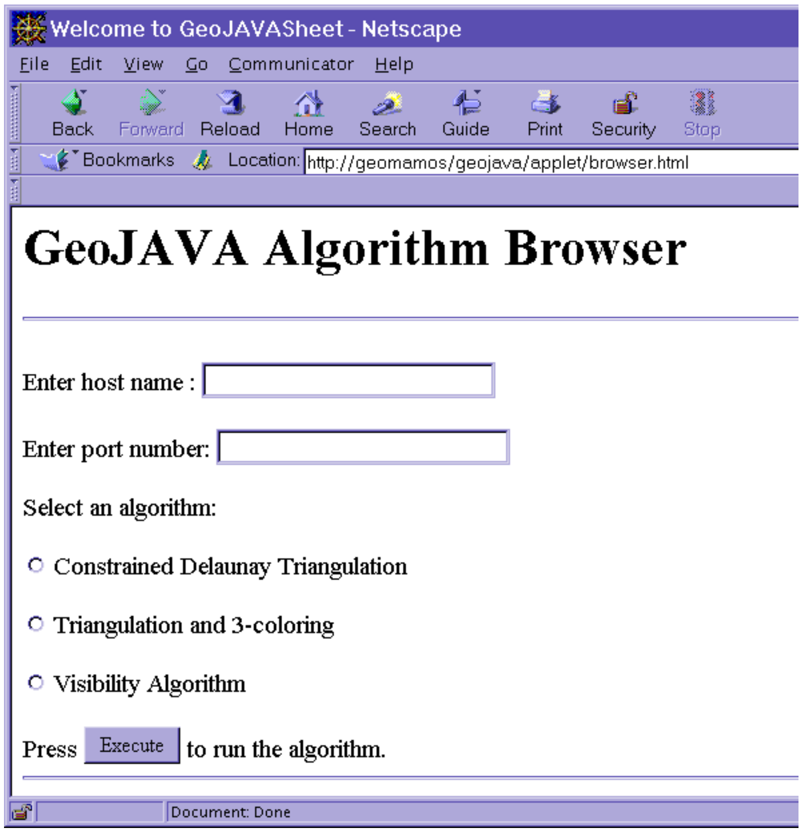
Figure 8: Compilation output


Algorithm Execution Completed Successfully! - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://geomamos/cgi-shl/compile/simple-form.pl
Document: Done

Algorithm Execution Completed Successfully!

If you would like to post your program to the Algorithm Browser,**click** **here**.

Figure 9: Algorithm Execution Completion


Welcome to GeoJAVASheet - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://geomamos/geojava/applet/browser.html

GeoJAVA Algorithm Browser

Enter host name :

Enter port number:

Select an algorithm:

○ Constrained Delaunay Triangulation

○ Triangulation and 3-coloring

○ Visibility Algorithm

Press Execute to run the algorithm.

Figure 10: Algorithm Browser

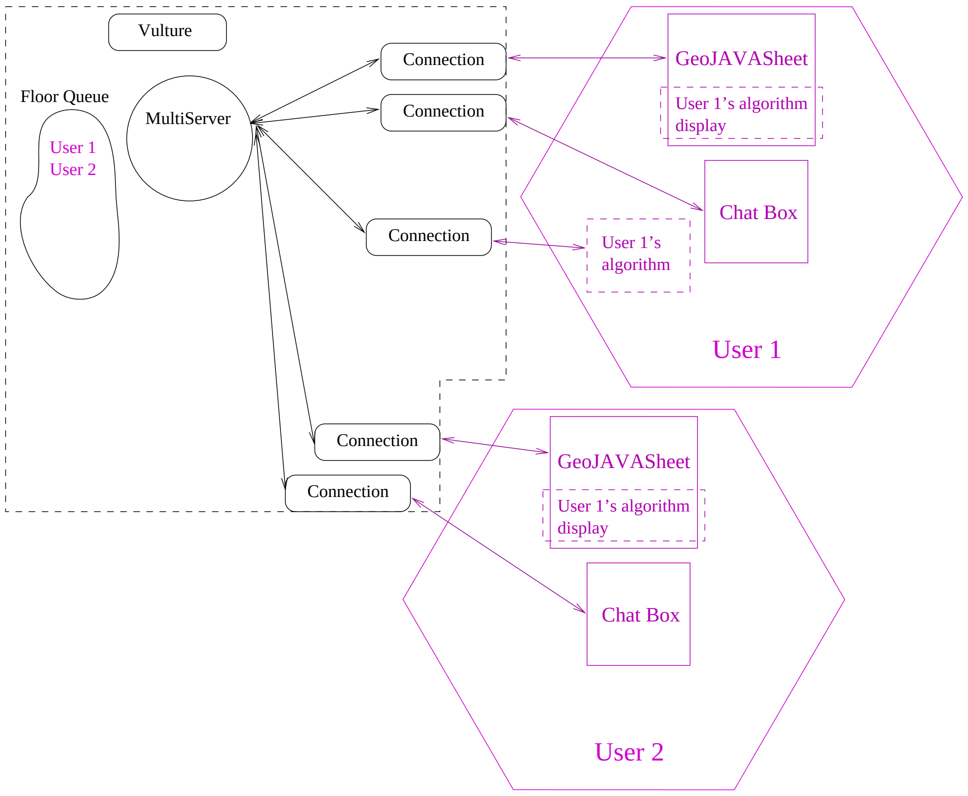


Figure 11: *GeoJAVA System* Architecture

5 Architectural Design

Combining the components described in the previous section, the result is a complete system as illustrated in Figure 11.

The figure is an instance of the system where User 1 and User 2 are on the same channel. User 1 has the floor and is executing an algorithm. The dotted area containing MultiServer is the Java application that runs on the web server at a specified TCP port. When a new user enters, a new Connection instance is created for the user's GeoJAVASheet, chat box, and possibly user program. Note that the Connection objects are clients that MultiServer instantiates when the users enter the system, and these Connection objects communicate to their counterpart applets through a TCP socket. This architecture is an expansion of the AnnoyingChat/MultiServer applets on the FICT home page.

6 Example

The user's program is written in C++ and has only a few simple "rules" for its structure to follow in order to work with the *GeoJAVA system*. These rules consist of including the appropriate include files and calling IPCServiceSetup(...) at the beginning of the program. Then the code may make calls to Graphic_Read() and Graphic_Write() for the geometric objects used. Details of these functions are given in Section 4.4.

The following is an excerpt of a geometric algorithm that computes the visibility polygon from a point interior to a given simple polygon; both the simple polygon and the source point are entered by the user from the GeoJavaSheet. VISIBILITY is a function, based on the algorithm by Lee[9] that takes as input a simple polygon and a point and produces as output the visibility polygon. More details of this implementation can be found in Aoki[1].

```
// Header files
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <geoIPC/port.h>
#include <geoLEDA/geo_plane.h>
#include <geoLEDA/geo_polygon.h>
#include <geoLEDA/geo_polyalg.h>

main(int argc, char* argv[])
{
  geopolygon P, *final;
  GeoPoint pt1;
  node v;

  final = new geopolygon();

  if (argc>1)
     IPCServiceSetup(argv[1], atoi(argv[2]));
  else
     IPCServiceSetup();

  SetOutSheetFillStyle(0);

  GeoPause("Please enter a simple polygon");
```

```
P.Graphic_Read();

GeoPause("Please enter an interior point");
pt1.Graphic_Read();

*final = VISIBILITY(&P, pt1);

SetOutSheetColor(next_color());

GeoPause("Ready to see the visibility polygon?");
final->Graphic_Write();
```

Note that the call to IPCServiceSetup() checks the arguments to the program and assumes that if there are arguments, they are the host and port number for the GeoJAVASheet to which it should communicate. This format should be followed, especially if the user wishes to execute their program off the web server, which will call the program with the host and port number as arguments.

Figure 12 displays the ChannelGuide when user Frank enters the system. User Kiyoko is already on channel "channel1," and Frank is about to join her. At that time, user Kiyoko sees what is in Figure 13. But when she sees Frank join the channel, she can send him messages, as in Figure 14, which is what Frank sees in his chat window.

When Kiyoko runs her algorithm (notice that she does indeed have the floor), both Kiyoko and Frank will see the same display on their sheets. The reader may follow the program along with the following series of figures. First, in Figure 15, Kiyoko and Frank see the request to enter a polygon, brought up because of the call to GeoPause(...) in the program.

Next, after Kiyoko has entered a polygon and hit the Return button, she and Frank both get a view as in Figure 16, requesting a point. Notice the polygon displayed on the sheet. As soon as Kiyoko hits the Return button, the polygon is displayed on all of the other users' GeoJAVASheets on the channel. In this case, only Frank sees the polygon entered.

Figure 17 displays the change in the GeoJAVAPause window to prepare the user for the output, and Figure 18 is the final output to the program.

Figure 19 displays another example of three GeoJAVASheets and three chat boxes connected to the same channel while running on three different hosts.

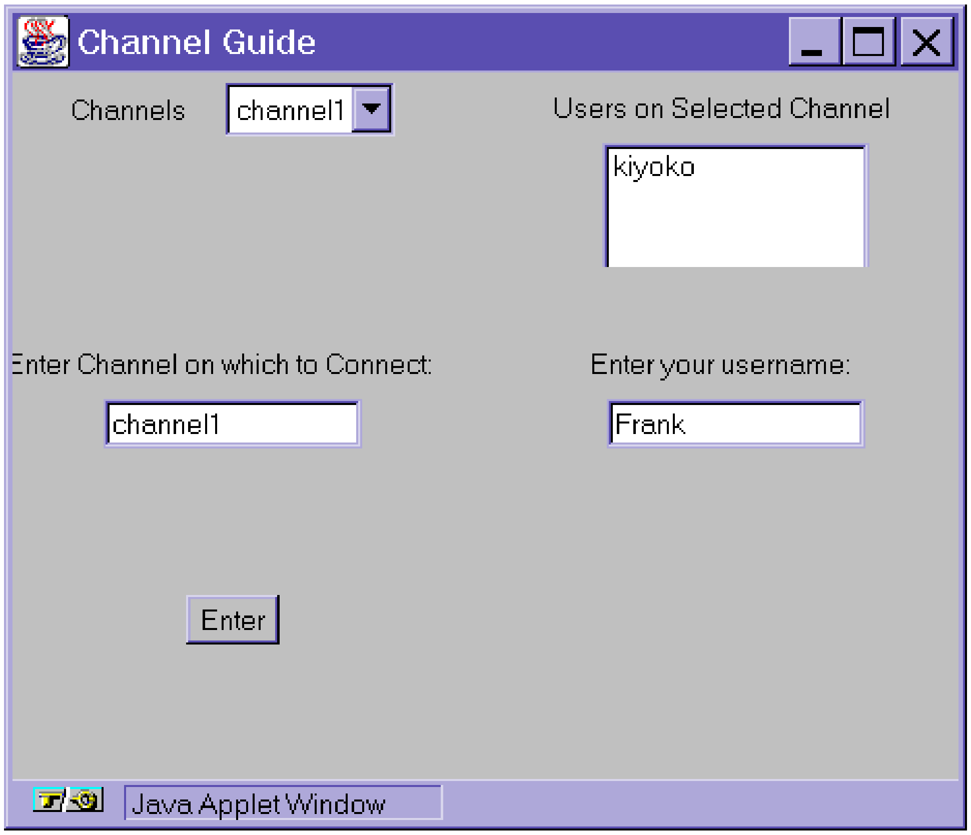
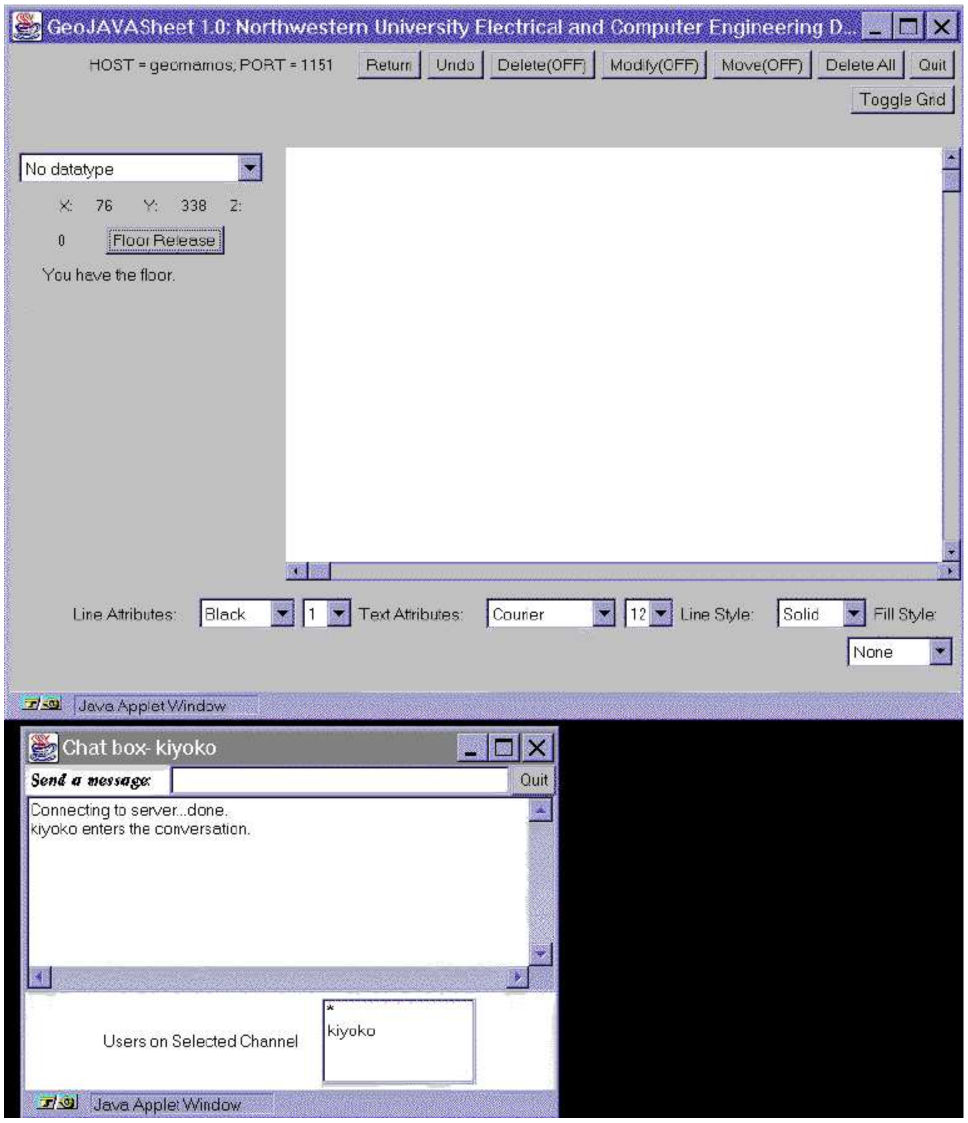
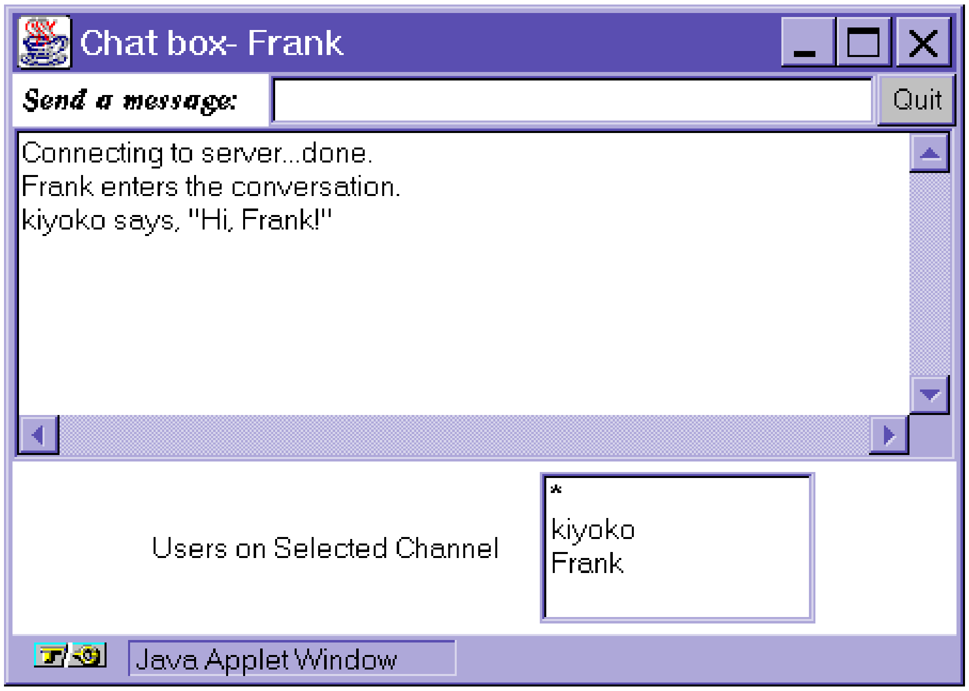


Figure 12: ChannelGuide



Connecting to server...done.
kiyoko enters the conversation.

Figure 13: Kiyoko's View

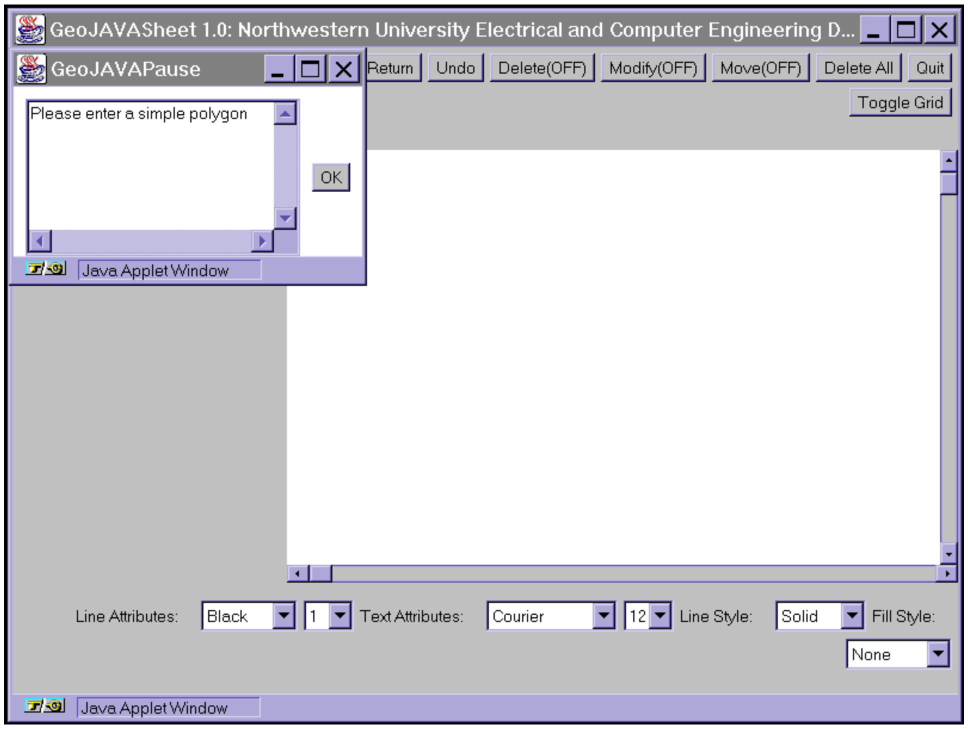


Connecting to server...done.
Frank enters the conversation.
kiyoko says, "Hi, Frank!"

Users on Selected Channel

*
kiyoko
Frank

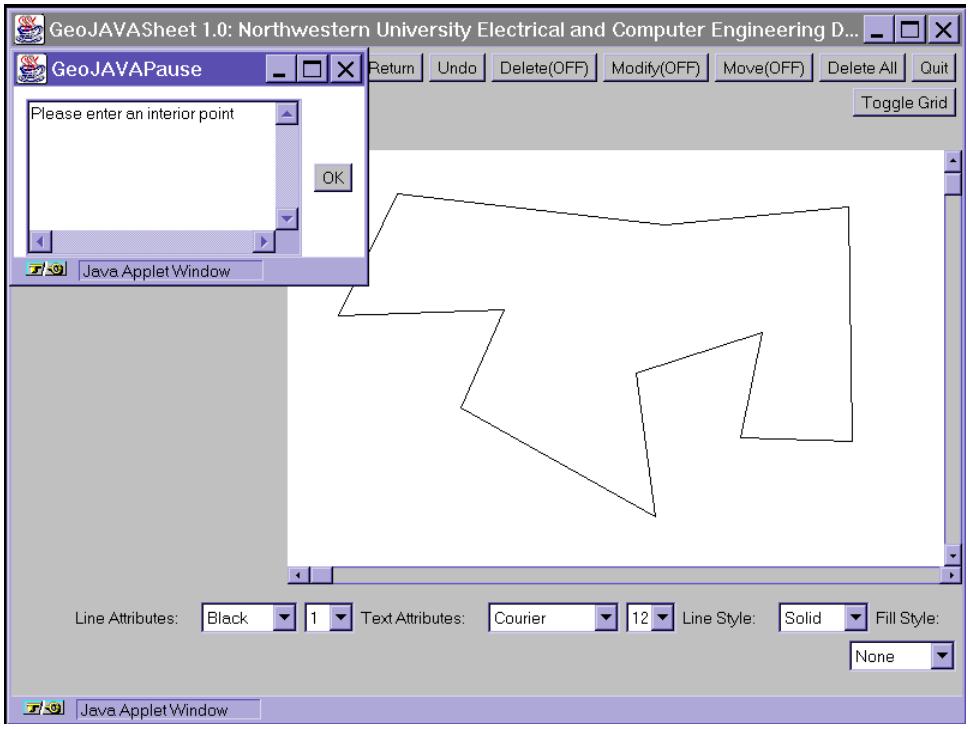
Figure 14: Frank's Chat Box View



Please enter a simple polygon

Line Attributes: Black 1 Text Attributes: Courier 12 Line Style: Solid Fill Style: None

Figure 15: Enter a polygon request

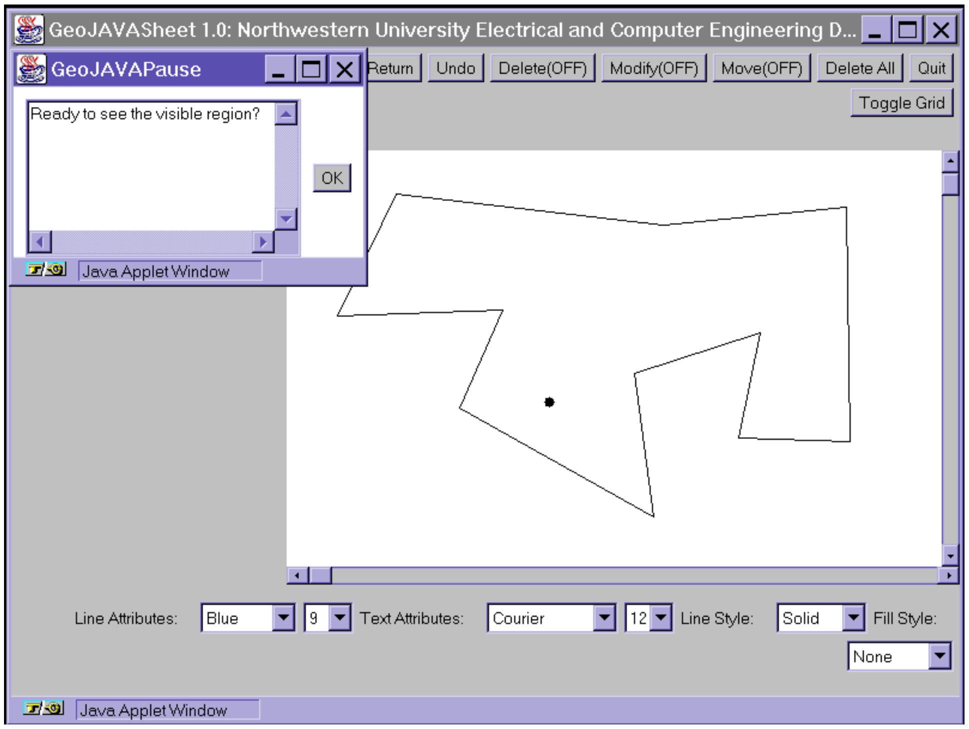

GeoJAVASheet 1.0: Northwestern University Electrical and Computer Engineering D...

GeoJAVAPause

Please enter an interior point

Line Attributes: Black 1 Text Attributes: Courier 12 Line Style: Solid Fill Style: None

Figure 16: Enter a point request

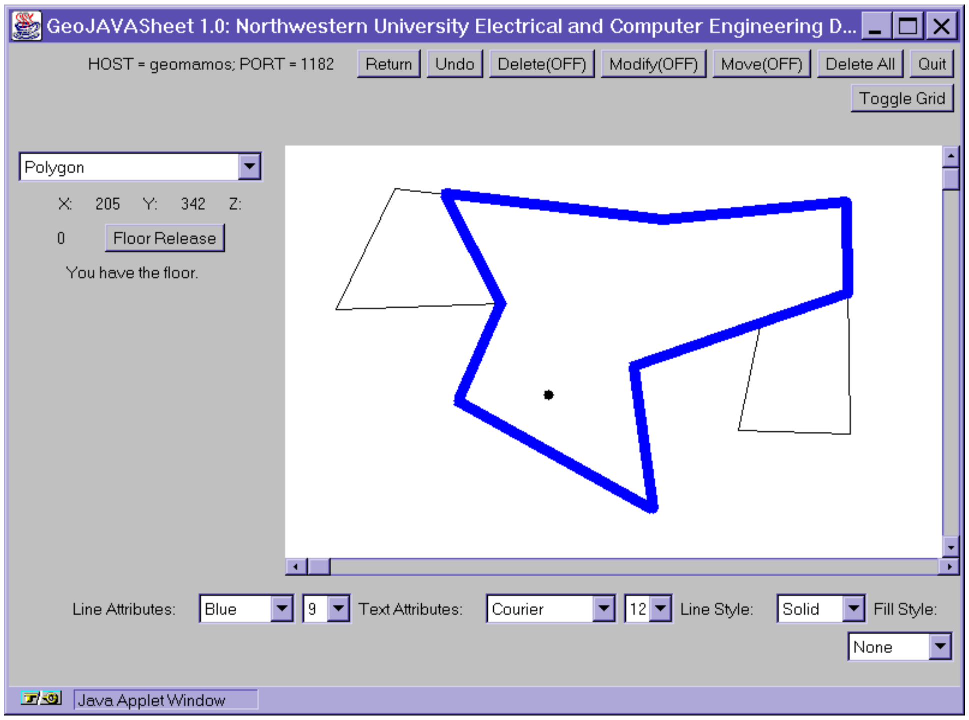

GeoJAVASheet 1.0: Northwestern University Electrical and Computer Engineering D...

GeoJAVAPause

Ready to see the visible region?

Line Attributes: Blue 9 Text Attributes: Courier 12 Line Style: Solid Fill Style: None

Figure 17: Ready request



GeoJAVASheet 1.0: Northwestern University Electrical and Computer Engineering D...

HOST = geomamos; PORT = 1182

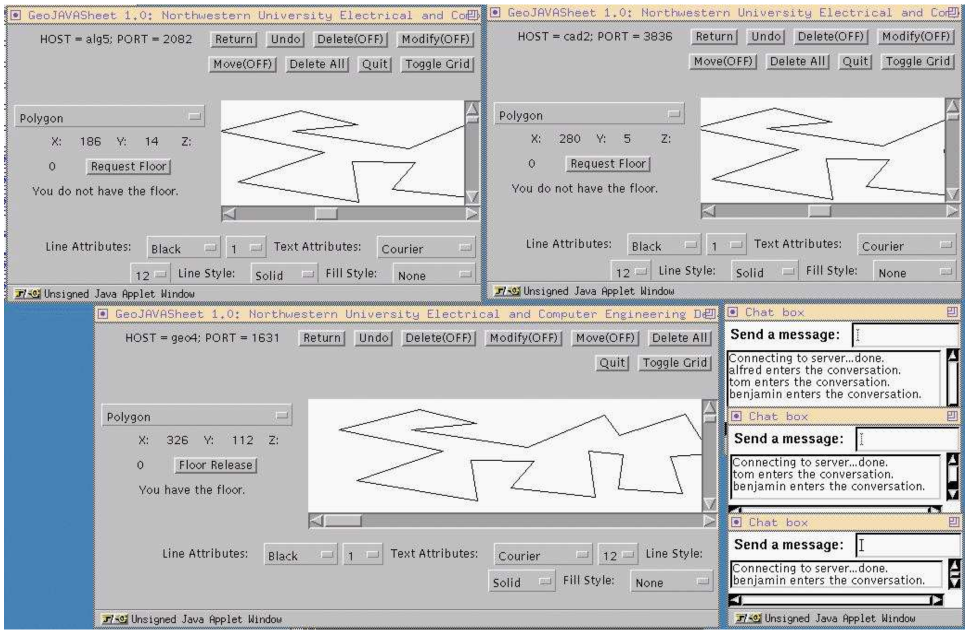
Polygon

X: 205 Y: 342 Z: 0

You have the floor.

Line Attributes: Blue 9 Text Attributes: Courier 12 Line Style: Solid Fill Style: None

Figure 18: Final program output



GeoJAVASheet 1.0: Northwestern University Electrical and Co

HOST = alg5; PORT = 2082

Polygon

X: 186 Y: 14 Z: 0

You do not have the floor.

Line Attributes: Black 1 Text Attributes: Courier 12 Line Style: Solid Fill Style: None

GeoJAVASheet 1.0: Northwestern University Electrical and Co

HOST = cad2; PORT = 3836

Polygon

X: 280 Y: 5 Z: 0

You do not have the floor.

Line Attributes: Black 1 Text Attributes: Courier 12 Line Style: Solid Fill Style: None

GeoJAVASheet 1.0: Northwestern University Electrical and Computer Engineering D

HOST = geo4; PORT = 1631

Polygon

X: 326 Y: 112 Z: 0

You have the floor.

Line Attributes: Black 1 Text Attributes: Courier 12 Line Style: Solid Fill Style: None

Send a message:

```
Connecting to server...done.
alfred enters the conversation.
tom enters the conversation.
benjamin enters the conversation.
```

Send a message:

```
Connecting to server...done.
tom enters the conversation.
benjamin enters the conversation.
```

Send a message:

```
Connecting to server...done.
benjamin enters the conversation.
```

Figure 19: *GeoJAVA System* Demo

7 Conclusion and Future Work

The *GeoJAVA system* provides a comprehensive set of tools with which any user on the World Wide Web can learn and implement concepts of computational geometry as well as collaborate with remote users on algorithm design. This is provided in a distributed and location transparent manner. System independence and modular implementation also makes the system scalable, and the dynamic manipulation of geometric algorithms and the online compilation tool are unique features that make the *GeoJAVA system* useful for computational geometers and students.

Using this *GeoJAVA system* as a stepping stone, many other valuable systems may be developed. For example, Sun Microsystems has recently announced its 3D API. A great contribution to computational geometry can be made with the implementation of a *GeoJAVA system* that visualizes 3D geometric algorithms.

Also MultiServer may become a bottleneck as more and more users and groups participate in collaborations. Therefore, a proposed workaround is to create a new MultiServer instance every time a new group is formed. The check for duplicate group names may be performed by communication among the existing MultiServer instances. This would make the *GeoJAVA system* more scalable.

Furthermore, a Java version may be developed of the GeoLIB library, which may eliminate the need for a large C/C++ library for those who do not know or do not wish to implement their algorithms in C/C++. In the current version of GeoJAVASheet, the JDK version 1.0.2 has been used since JDK 1.1 has yet to be implemented in most of the browsers in use today. However, once its stability and popularity have become the norm, we plan on upgrading GeoJAVASheet to at least JDK 1.1.

The need for compilation is expected to remain, however, so the compilation page may be improved or expanded. In order for users to retrieve their compiled code, an additional tool may be developed which can be downloaded to the user's machine. This tool would connect over the Internet to the appropriate server containing the libraries for the user's system and then compile the code on the local machine itself. This would eliminate the need to use a browser for compilation, and may complement the GeoMAMOS system.

It is our hope that with this visualization prototype, a foundation can be established on top of which a problem solving environment for geometric computing, and other fields of scientific research and development can be built.

8 Acknowledgements

We would like to thank the following people for their help in the initial implementation of GeoJAVASheet: Mehmet Sayal, Lisa Singh, Takashi Yoshikawa. We would also like to acknowledge Benjamin McLean for his implementation of ChannelGuide and file I/O on the application version of GeoJAVASheet and Steve Loranz and Matt Firlik for their help with the compilation tool.

References

[1] K. Aoki, "The Prototyping of GeoManager: A Geometric Algorithm Manipulation System", Master's Thesis, December, 1995.

[2] J. E. Baker, I. F. Cruz, L. D. Lejter, G. Liotta, and R. Tamassia, "Mocha", http://loki.cs.brown.edu:8080/papers/MochaFS.html.

[3] L. Beca, G. Cheng, G. C. Fox, T. Jurga, K. Olszewski, M. Podgorny, P. Sokolowski, K. Walczak, "Web Technologies for Collaborative Visualization and Simulation", NPAC Technical Report SCCS-786, Syracuse University, NPAC, Syracuse, NY, submitted January 6, 1997.

[4] M. H. Brown, M. A. Najork, R. Raisamo, "A Java-Based Implementation of Collaborative Active Textbooks", in 1997 IEEE Symposium on Visual Languages, pages 372-379. IEEE Computer Society, September 1997.

[5] M. H. Brown, R. Sedgewick, "A System for Algorithm Animation", *Computer Graphics*, **18**(3), 177-186, July 1984.

[6] U. Gall, F. J. Hauck, "Promondia: A Java-Based Framework for Real-Time Group Communication in the Web", *Sixth International World Wide Web Conference*, 1996.

[7] J. Gosling, B. Joy and G. Steele, The Java Language Specification, Addison-Wesley Developers Press, Sunsoft Java Series (1996).

[8] W. E. Johnston and S. Sachs, "Distributed, Collaboratory Experiment Environments (DCEE) Program: Overview and Final Report", Lawrence Berkeley National Laboratory, February, 1997.

[9] D. T. Lee, "Visibility of a Simple Polygon," *Computer Vision Graphics and Image Processing*, **22** (1983) 207-221.

[10] D. T. Lee, C. F. Shen and S. M. Sheu, "GeoSheet: A Distributed Visualization Tool for Geometric Algorithms", *Int'l J. Computational Geometry & Applications*, **8,2**, April 1998, to appear.

[11] S. Näher, "LEDA – A Library of Efficient Data Types and Algorithms", Max-Planck-institut für informatik. Technical Report A 04/89, Universität des Saarlandes, Saarbrücken, 1989.

[12] S. Näher, "LEDA3.0 User Manual", technischer Bericht A, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, 1992.

[13] M. H. Overmars, "Designing the Computational Geometry Algorithms Library CGAL." In Proceedings Workshop on Applied Computational Geometry, May 27-28, 1996, Philadelphia, Pennsylvania, pp. 113-119.

[14] B. V. Smith, *The Xfig User Manual*, 1993.

Appendix:

The following is the java source code for MultiServer. Note that it works closely with Classes Connection and Vulture.

```
public class MultiServerTCP extends Thread {

    /** Default port to listen on */
    protected final static int DEFAULT_PORT = 8411;
    protected final static int DEFAULT_PORT2 = 8511;
    private int port;
    private ServerSocket listen_socket;
    protected ThreadGroup threadgroup;
    protected Vector connections;
    protected Vulture vulture;
    protected Vector ports = new Vector();
    protected Vector floorQ = new Vector();
    protected Vector channelQ = new Vector();
    private Vector channelGuideQ = new Vector();
    private Vector names = new Vector();
    private Vector channels = new Vector();

    public boolean locked = false;
    public static final int BUFSIZE = 4096;
    public static final int MAXCHANNELS = 10; // also set in channelGuide.
    protected String last_ss = new String("");

    /** Exit with an error message, when an exception occurs. */
    public static void fail(Exception e, String msg) {
        System.err.println(msg + ": " +  e);
        System.exit(1);
    }

    // Create a ServerSocket to listen for connections on; start the thread.
    public MultiServerTCP(int port, boolean verbose) {
        // Create our server thread with a name.
        super("Server");
        if (port == 0) port = DEFAULT_PORT;
        this.port = port;
        try { listen_socket = new ServerSocket(port); }
```

```
      catch (IOException e) {fail(e, "Exception creating server socket");}
      // Create a threadgroup for our connections
      threadgroup = new ThreadGroup("Server Connections");

      // Initialize a vector to store our connections in
      connections = new Vector();

      // Create a Vulture thread to wait for other threads to die.
      // It starts itself automatically.
      vulture = new Vulture(this);

      // Tell the world we are running
      if (verbose) System.out.println("Multiserver is running...");

      // Start the server listening for connections
      this.start();
  }

/**
  * The body of the server thread.  Loop forever, listening for and
  * accepting connections from clients.  For each connection,
  * create a Connection object to handle communication through the
  * new Socket.  When we create a new connection, add it to the
  * Vector of connections.  Note that we are running asynchronously.
  * We used to use synchronized to lock the Vector of connections.
  * The Vulture class does the same, so the vulture won't be removing
  * dead * connections while we're adding fresh ones. This version seems
  * more resistant to deadlock.
  */
  public void run() {
    try {
        while(true) {
          Socket client_socket = listen_socket.accept();
          System.out.println("socket accepted!!! "+client_socket.getPort());
          Connection c = new Connection(this, client_socket, threadgroup, 3, vulture);
          connections.addElement(c);
        }
    }
    catch (IOException e) {fail(e, "Exception while listening for connections");}
  }
```

```
/** Start the server up, listening on an optionally specified port */
  public static void main(String[] args) {
      int port = 0;
      boolean verbose = false;

      if (args.length == 0)  new MultiServerTCP(port, verbose);
      else if (args.length == 1) {
         if (args[0].equals("-v"))  {
            verbose = true;
            new MultiServerTCP(port,verbose);
         } else {
            try {port = Integer.parseInt(args[0]);}
            catch (NumberFormatException e) {port = 0;}
            new MultiServerTCP(port, verbose);
         }
      }
      else if ((args.length == 2) && args[0].equals("-v")) {
          try {port = Integer.parseInt(args[1]);}
          catch (NumberFormatException e) {port = 0;}
          verbose = true;
          new MultiServerTCP(port, verbose);
      }
      else System.out.println("Usage: java MultiServerTCP [-v] [<port>]");
  }

  public void removeChannel(String hoststr, int portnum, String chanName) {
      floorQ.removeElement(hoststr+"|"+portnum+"|"+chanName);
      channelQ.removeElement(chanName+"|"+hoststr+"|"+portnum);

      Enumeration e = channelQ.elements();
      while (e.hasMoreElements()) {
         Object o = e.nextElement();
         System.out.println(o);
      }
      e = floorQ.elements();
      while (e.hasMoreElements()) {
         Object o = e.nextElement();
         System.out.println(o);
      }
  }
```

```
public boolean add2Q(String hoststr, String portstr, String chanName) {
    String request;
    Integer portint;
    int portnumber, bytecount=0;
    byte[] barray = new byte[BUFSIZE];
    InetAddress address;
    boolean success;

    success = true;

    if (floorQ.isEmpty()) {
       floorQ.addElement(hoststr+"|"+portstr+"|"+chanName);
       success = true;
    } else { // Modified KFA 1/16/98 for multiple channels.
       Enumeration f = floorQ.elements();
       while (f.hasMoreElements()) {
           String s = (String)f.nextElement();
           if (s.endsWith(chanName))
              success = false;
       }
       if (success == true)
    }
    return success;
}

public boolean hasMore(String chanName) {
   Object o;

   Enumeration e = floorQ.elements();
   o = e.nextElement();

   while (e.hasMoreElements()) {
      if (((String) o).endsWith(chanName))
         return true;
   }
   return false;
}

public boolean remfromQ(String hoststr, String portstr, String chanName) {
    String request;
    byte[] barray = new byte[BUFSIZE];
```

```
InetAddress address;
boolean success;
Enumeration e;
String newHost = new String();
String newPort = new String();
String newSS = new String();
int portno, newport = -1;

if (floorQ.contains(hoststr+"|"+portstr+"|"+chanName)) {

   if (hasMore(chanName))
      floorQ.removeElement(hoststr+"|"+portstr+"|"+chanName);

   e = floorQ.elements();
   Object o = e.nextElement();

   while (!((String)o).endsWith(chanName)) {
      o = e.nextElement();
   }

   StringTokenizer st = new StringTokenizer((String)o, "|");

   newHost = st.nextToken();
   newPort = st.nextToken();

   request = new String("floor granted ");

   Connection you;

   e = connections.elements();

   portno = (new Integer(portstr)).intValue();

   while (e.hasMoreElements()) {
      you = (Connection)e.nextElement();
      if (you.myType.equals("GJS") && you.channel.equals(chanName) &&
          you.hoststr.equals(newHost) && you.portstr.equals(newPort)) {
         you.send(request);
         newport = you.portnum;
         if (!you.serv_str.equals(""))
            newSS = you.serv_str;
```

```
               }
            }

            // adjust UP which originally connected with
            // a different GeoSheet that had the floor...

            e = connections.elements();
            while (e.hasMoreElements()) {
               you = (Connection)e.nextElement();
               if (you.myType.equals("UP") && you.UPport == portno) {
                  you.UPport = newport;
                  you.hoststr = newHost;
                  System.out.println("Adjusted UP with "+you.serv_str);
               }
            }

            success = true;
         } else {
            success = false;
         }

         System.out.println("Floor Q now has ");
         e = floorQ.elements();
         while (e.hasMoreElements()) {
            Object o = e.nextElement();
            System.out.println(o);
         }

         return success;
      }

      public void broadcastDraw(String msg, int UPport, String last_serv_str, int sendUP) {
         StringTokenizer stringT, request;
         String mychann, mess;
         String hoststr;
         Integer portInt;
         int portnum;
         int bytecount=0;
         byte[] barray = new byte[BUFSIZE];
         InetAddress address;
         Connection you;
      last_ss = new String(last_serv_str);

      Enumeration e = channelQ.elements();
      while (e.hasMoreElements()) {
         String channstr = (String)e.nextElement();
         stringT = new StringTokenizer(channstr, "|");
         mychann = stringT.nextToken();
         hoststr = stringT.nextToken();
         portnum = (new Integer(stringT.nextToken())).intValue();
         if (portnum == UPport) {
            System.out.println("UP: found "+portnum);
            // find connections with same channel
            Enumeration c = connections.elements();
            while (c.hasMoreElements()) {
               you = (Connection) c.nextElement();
               System.out.println("checking "+you.channel+"="+mychann);
               if (you.channel.equals(mychann) && you.isGJS) {
System.out.println("serv_str is "+you.serv_str);
                  if (sendUP == 0 && you.portnum == UPport)
                     System.out.println("not sending");
                  else if (you.serv_str.equals("GeoSheetStdOut")) {
                     System.out.println("sending");
                     you.send(msg);
                  }
               }
            }
            break;
         }
      } //while
   }

   public void send2UP(String msg, int portnum, int serv_num) {
      Connection you;
      String serv_str;

      Enumeration c = connections.elements();
      while (c.hasMoreElements()) {
         you = (Connection) c.nextElement();
         if (you.myType.equals("UP"))
            if (you.UPport == portnum && you.serv_str.equals(last_ss)) {
```

```
            you.send(msg);
         }
   }
}

public void addChannel(String hoststr, int portnum, String newchannel) {
   channelQ.addElement(new String(newchannel+"|"+hoststr+"|"+portnum));
   System.out.println("adding to channelQ: "+newchannel+"|"+hoststr+"|"+portnum);
}

public boolean GJSHasFloor(String host, int num, String serv_str) {
   String you;
   Connection connect;

   Enumeration c = connections.elements();
   while (c.hasMoreElements()) {
      connect  = (Connection) c.nextElement();
      if (connect.myType.equals("GJS") &&
          connect.portnum == num &&
          connect.hoststr.equals(host)) {
              connect.serv_str = new String(serv_str);

              break;
      }
   }

   c = floorQ.elements();
   while (c.hasMoreElements()) {
      you = (String) c.nextElement();
      if (you.startsWith(host+"|"+num))
         return true;
   }
   return false;
}

public String findFloorGJS(String hoststr, int UPport) {
   String chanName = new String();
   Connection you;
   StringTokenizer st;
   String newHost;
   String newUP;
   Enumeration c = connections.elements();
   while (c.hasMoreElements()) {
      you  = (Connection) c.nextElement();
      if (you.myType.equals("GJS") &&
          you.portnum == UPport &&
          you.hoststr.equals(hoststr)) {
              chanName = you.channel;
              break;
      }
   }

   c = floorQ.elements();
   while (c.hasMoreElements()) {
      Object o = c.nextElement();
      if (((String)o).endsWith(chanName)) {
         return (String)o;
      }
   }

   chanName = new String(hoststr +"|"+ UPport);
   return chanName;
}

//changed by BJM
public void add2CGQ(String newchannel, String newuser)
{
  Enumeration e = connections.elements();
  Connection you;

    channelGuideQ.addElement(new String(newchannel+"|"+newuser));
    channels.addElement(newchannel.toLowerCase());
    names.addElement(newuser.toLowerCase());

    while (e.hasMoreElements()) {
       you = (Connection) e.nextElement();
       you.sendUpdate();
   }
}

public void someoneDied(String chan, String nam)
```

```
{
 channelGuideQ.removeElement(new String(chan+"|"+nam));
    names.removeElement(nam.toLowerCase());
    channels.removeElement(chan.toLowerCase());
}

public Vector getCGQ()
{
 return channelGuideQ;
}

//this method checks the names and channels vectors to see if a user can add
//with the given name and channel. Returns 0 if the user can connect on that
//channel, returns 1 if there are already MAXCHANNELS channels, returns 2 if
//it is an invalid username
public int canIConnect(String req)
{
    locked = true;
    int i = req.indexOf("|");
    String C = req.substring(0,i);
    String N = req.substring(i+1);
    System.out.println("C= "+C+" N= "+N);
    String c = C.toLowerCase();
    String n = N.toLowerCase();

    if (names.contains(n) || n.equals("*"))
    {
       locked = false;
      return 2;
    }

    if (channels.size() >= MAXCHANNELS)
    {
      locked = false;
      return 1;
    } else {
       add2CGQ(C,N);
      locked = false;
       return 0;
    }
    }

}

// This class is the thread that handles all communication with a client
// It also notifies the Vulture when the connection is dropped.
class Connection extends Thread {

    static int connection_number = 0;
    protected Socket client;
    protected Vulture vulture;
    protected DataInputStream in;
    public PrintStream out;
    protected MultiServerTCP serv;
    public String channel = new String();
    public String oldChannel = new String();
    public String username;
    public String myType = new String();
    public String portstr = new String();
    public int portnum;
    public String hoststr = new String();
    public int UPport = -1;
    public boolean isGJS = false;
    public String serv_str = new String();
    public int serv_num;

    // Initialize the streams and start the thread
    public Connection(MultiServerTCP theserver, Socket client_socket,
              ThreadGroup threadgroup, int priority, Vulture vulture)
    {
        // Give the thread a group, a name, and a priority.
        super(threadgroup, "" + ++connection_number);
        this.setPriority(priority);
        // Save our other arguments away
        client = client_socket;
        this.vulture = vulture;
        serv = theserver;
        channel = "The Default Channel";
        oldChannel = "The Default Channel";
        username = "UnKnown";
        // Create the streams
    try {
        in = new DataInputStream(client.getInputStream());
        out = new PrintStream(client.getOutputStream(), true);
    }
    catch (IOException e) {
        try { out.close(); in.close(); client.close();}
        catch (IOException e2) {} ;
        System.err.println("Exception while getting socket streams: " + e);
        return;
    }
    // And start the thread up
    this.start();
}

public void sendUpdate() {
    Vector CHG = serv.getCGQ();
String outline = new String("CHG");
for (Enumeration enum = CHG.elements(); enum.hasMoreElements();)
{
  outline = outline.concat(new String(";"+enum.nextElement()));
}
    out.println(outline);
System.out.println("here's the data" + outline);
}

public void run() {
    String line;
    Integer portint;
    StringTokenizer st;
    String msg = "";

    // Send a welcome message to the client
    send("0 WELCOME " + super.getName());

    try {
        for(;;) {
          yield();
          try {sleep(500);} catch (InterruptedException e){ }
          line = in.readLine();
          if (line == null) break;
          line = line.trim();
```

```
     if (line.startsWith("CHANNEL")) {

if (!isGJS) {
   int i = line.indexOf("|");          //added by BJM
          channel = line.substring(8, i-1);   //added by BJM
           oldChannel = line.substring(8, i-1); //added by BJM
           username = line.substring(i+11);    //added by BJM
           myType = new String("CHAT");
        } else {
           channel = line.substring(8);
        }
        if (hoststr != null) {
           serv.addChannel(hoststr, portnum, channel);
        }
     } else if (line.startsWith("channelGuide request")) { // added by BJM
        Vector CHG = serv.getCGQ();
        String outline = new String("CHG");
        for (Enumeration enum = CHG.elements(); enum.hasMoreElements();) {
        outline = outline.concat(new String(";"+enum.nextElement()));
    }
          out.println(outline);
     } else if (line.startsWith("Can I Connect: ")) {
        line= line.substring(15);
        while (serv.locked) {}

        int response = serv.canIConnect(line);
        if (response==1)  {
           int i = line.indexOf("|");          //added by BJM
           channel = line.substring(0, i);   //added by BJM
           oldChannel = channel; //added by BJM
           username = line.substring(i+1);    //added by BJM
}

        out.println("Connection Request: "+response);

      } else if (line.startsWith("HOST")) {
        st = new StringTokenizer(line.substring(4),"|");
        hoststr = st.nextToken();
        portstr = st.nextToken();
        portint = new Integer(portstr);
        portnum = portint.intValue();
```

```
  myType = new String("GJS");
  isGJS = true;
} else if (line.startsWith("CLOSE ")) {
  portstr = line.substring(6);
  serv.removeChannel(hoststr, portnum, channel);
  serv.remfromQ(hoststr, portstr, channel);
} else if (line.startsWith("GJS")) {
  st = new StringTokenizer(line.substring(4),"|");
  serv_num = (new Integer(st.nextToken())).intValue();
  portstr = st.nextToken();
  msg = new String(st.nextToken());
  if (msg.equals("Xfloor request") || msg.equals("floor request")) {
    if (serv.add2Q(hoststr, portstr, channel))
       out.println("floor granted");
  } else if (msg.equals("Xfloor release")||msg.equals("floor release")) {
    if (serv.remfromQ(hoststr, portstr, channel))
        out.println("floor released ");
    else
        out.println("MSPause only one ");
  } else if (msg.startsWith("Draw")) {
    serv.broadcastDraw(msg.substring(5), portnum, "GeoSheetStdOut", 0);
  } else if (msg.startsWith("XDraw")) {
    serv.broadcastDraw(msg.substring(6), portnum, "GeoSheetStdOut", 0);
  } else { /* reply to UP */
    serv.send2UP(msg, portnum, serv_num);
  }
} else if (line.startsWith("A|connect")) {
  st = new StringTokenizer(line.substring(10), "|");
  serv_str = new String(st.nextToken());
  hoststr = new String(st.nextToken());
  UPport = (new Integer(st.nextToken())).intValue();
  myType = new String("UP");
  if (serv.GJSHasFloor(hoststr, UPport, serv_str)) {
     System.out.println(serv_str+" UPport "+UPport);
  } else {
    hoststr = serv.findFloorGJS(hoststr, UPport);
    st = new StringTokenizer(hoststr, "|");
    hoststr = st.nextToken();
    UPport = (new Integer(st.nextToken())).intValue();
  }
} else if (line.startsWith("A|disconn")) {
```

```
              try {out.close(); in.close(); client.close();}
              catch (IOException e2) { System.out.println("Yow!");};
            } else if (line.startsWith("B|")) {
              processUP(line.substring(2));
            } else {
              if (line.length() > 0)
                 broadcast(line);
            }
    }
}
catch (IOException e) { }
    // When we're done, for whatever reason, be sure to close the socket,
    // and to notify the Vulture object.  Note that we have to use synchronized
    // first to lock the vulture object before we can call notify() for it.
    // Note: running asynchronously now.
    finally {
      try {out.close(); in.close(); client.close();}
      catch (IOException e2) { System.out.println("Yow!");};
      channel = "Kill me please!";
    }
}

// handle user program message
public void processUP(String line) {
    serv.broadcastDraw(line, UPport, serv_str, 1);
}

public  void send(String msg) {
   if (out.checkError()) {
      this.channel = "Kill me please!";
   } else {
      out.println(msg);
   }
}

// Here we send a message to everyone who is on the channel.

public void broadcast(String msg) {
  Connection you;
  String toName;
  StringTokenizer st;
```

```
      st = new StringTokenizer(msg, "|");
      st.nextToken();
      st.nextToken();
      toName=st.nextToken();
      if (toName.equals("*")) {
       Enumeration e = serv.connections.elements();
            while (e.hasMoreElements()) {
                you = (Connection) e.nextElement();
                if (you.channel.equals(channel))
                    you.send(msg);
            }
        } else {
           Enumeration e = serv.connections.elements();
           while (e.hasMoreElements()) {
               you = (Connection) e.nextElement();
               if (you.channel.equals(channel) && you.username.equals(toName))
                  you.send(msg);
           }
        }
      }
   }


// This class waits to be notified that a thread is dying (exiting)
// and then cleans up the list of threads and the graphical list.
class Vulture extends Thread {
    protected MultiServerTCP server;

    protected Vulture(MultiServerTCP s) {
        super(s.threadgroup, "Connection Vulture");
        server = s;
        setPriority(5);
        this.start();
    }

    // This is the method that waits for notification of exiting threads
    // and cleans up the lists.  It is a synchronized method, so it
    // acquires a lock on the 'this' object before running.  This is
    // necessary so that it can call wait() on this.  Even if the
    // the Connection objects never call notify(), this method wakes up
    // every five seconds and checks all the connections, just in case.
    // Note also that all access to the Vector of connections and to
    // the GUI List component are within a synchronized block as well.
    // This prevents the Server class from adding a new conection while
    // we're removing an old one.
    public /* synchronized */ void run() {
      Connection c;
      for(;;) {
         try {sleep(10000);} catch (InterruptedException e) {};
         System.gc();
         // Do we run into trouble here if not synchronized?
         // (Assuming that something changes during the enumeration)
         Enumeration e = server.connections.elements();
         while (e.hasMoreElements()) {
            c = (Connection) e.nextElement();
            // if the connection thread isn't alive anymore,
            // remove it from the Vector and List. And inform
            // the deceased's friends.
            if (c.channel.equals("Kill me please!"))
            {
               server.remfromQ(c.hoststr, c.portstr, c.oldChannel); // added by KFA
               c.stop();
               server.connections.removeElement(c);
               server.someoneDied(c.oldChannel, c.username); //added by BJM
               c = null;
            } else if (!c.isAlive()) {
               server.remfromQ(c.hoststr, c.portstr, c.oldChannel); // added by KFA
               server.connections.removeElement(c);
               server.someoneDied(c.oldChannel, c.username); //added by BJM
               c.broadcast("0 OBITUARY " + c.getName());
               c = null;
            }

         }
      }
    }
}
```