International Journal of Computational Geometry & Applications

TOWARDS WEB-BASED COMPUTING

KIYOKO F. AOKI*
Department of Electrical and Computer Engineering, Northwestern University
Evanston, Illinois 60208, U.S.A.

and

D.T. Lee
Institute of Information Science, Academia Sinica, Nankang,
Taipei, Taiwan,
dtlee@iis.sinica.edu.tw

Received 12 March 1998
Revised 25 September 1999
Communicated by J-D. Boissonnat

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*, which consists of a GUI and a geometric visualization library that enables the user or algorithm designer to (1) execute and visualize an existing algorithm in the library or (2) develop new code over the Internet. The library consists of geometric code written in C/C++. The GUI is written using the Java programming language. Taking advantage of the socket classes and system-independent application programming interfaces (API's) provided with the Java language, *GeoJAVA* offers a platform independent environment for distributed geometric computing that combines Java and C/C++. Users may remotely join a "channel" or discussion group in a location transparent manner to do collaborative research. 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. A chat box is also provided to enable dialogue among the members. Furthermore, this system not only allows visualization of pre-compiled geometric code, but also serves as a web-based programming environment where the user may submit a geometric code, compile it with the libraries provided by the system, and visualize it directly over the web sharing it with other users immediately.

Keywords: geometric computation, visualization tool, distributed system, Java programming language.

*Current address: Institute of Information Science, Academia Sinica, Nankang, Taipei, Taiwan, kiyoko@ieee.org.

1. Introduction

With the advancement of computer and communication technologies, communication via e-mail and over the World Wide Web has become commonplace in our daily activities. In the computing world, collaboration via the Internet has recently gained popularity. 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 worthwhile, 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 solve 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 straightforward, 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. In addition, major C/C++ libraries have been developed that provide comprehensive object classes and algorithms of which researchers can take advantage. Taking into consideration the usefulness of Java and the prevailing use of C/C++, the combination of both programming languages resulted in 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 for dialogue, (3) a C/C++ library of geometric algorithms called GeoLIB, (4) a compilation tool allowing users to implement user-defined algorithms using the GeoLIB library, and (5) broadcast

visualization of geometric algorithms.

There are many potential applications of this system, among which 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 broadcast 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-mail 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 immediately re-executed for the others to see. Dialogue is enabled by means of a 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, which is provided by the "dynamic re-execution" feature of the system. During dynamic re-execution, 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 a variety of 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 and updating of the changing results of the algorithm on all of the sheets in the group.

As technology advances and becomes more readily available, it would be possible to incorporate audio and video communication to the system for a greater "collaboratory" feel. Although currently not supported, this addition can be incorporated easily due to the modularity of each component of the system. At the time

of this writing, Sun Microsystems is developing the Java Media Framework, which will allow developers to incorporate streaming audio and video into Java applets and applications. This API will provide the necessary functionality to incorporate audio and video communication to the system.

The next section gives a review of other projects related to the *GeoJAVA system*, followed by sections containing user-level and system-level descriptions of the design of the system and a sample session to illustrate how a user may use it. For these sections, the reader is expected to be fairly literate in Java and/or C/C++ programming and to have an understanding of networking terminology such as sockets and TCP/UDP. A brief description of this terminology is provided in the Appendices. The final section discusses our plans for future work.

2. Related Work

As is evident from the "Computational Geometry Interactive Software" page, many geometric algorithm visualization tools[a] have been implemented, and the "Complete Collection of Algorithm Animations" gives a comprehensive list of geometric algorithms written in Java.[b] 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, 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 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.[c] 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.

[a] See http://www.cs.duke.edu/~jeffe/compgeom/demos.html.
[b] See http://www.cs.hope.edu/~alganim/ccaa/geometric.html.
[c] See http://loki.cs.brown.edu:8080/pages/Mocha.html.

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 (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.[d]

Although these applets are successful in demonstrating various computational geometry algorithms, if a researcher, say, wanted to test and develop her own algorithm, she would not be able to make any practical use of these applets, let alone demonstrate the same execution of her 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**[e] is a framework for sharing Java objects with colleagues 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

[d] See http://fiaker.ncsa.uiuc.edu/alpha/reference.html for references regarding Alpha Shapes/Ranks.

[e] http://www.ncsa.uiuc.edu/SDG/Software/Habanero/

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.[f]

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.

Although all three of these applications provide distributed collaboration, none of them can allow users to compile their own code remotely. In order to use the APIs provided, the user must download the libraries and compile their code locally.

The differences between JCAT and the *GeoJAVA system* include location independence and remote compilation. While *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 floor manager is beforehand. That is, only the teacher has control of the algorithm, and the students may not request control of the floor. Since no remote compilation is available, users are forced to use Java and specifically write their code for distributed visualization and compile it locally, whereas users on the *GeoJAVA system* can use any existing C/C++ code and not be concerned with which parts of their code is distributed. In addition, they are provided with remote compilation facilities, so they need not download any libraries whatsoever.

The *GeoJAVA system* is based on GeoMAMOS,[g] part of which are GeoSheet[10] and GeoManager,[1] which provide distributed visualization of geometric algorithms over a UNIX-based network. GeoSheet is the 2D GUI for GeoMAMOS, serving

[f] Refer to http://www6.nttlabs.com/papers/PAPER100/PAPER100-java.html for an online version of their paper.

[g] See http://www.ece,nwu.edu/~theory/geomamos.html

as the interface with which users interact to communicate with their algorithms. GeoManager provides the dynamic re-execution 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*. Note that the *GeoJAVA system* has also been augmented with additional tools that make it more of a web-based programming environment as opposed to a visualization tool.

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 the distributed visualization process can begin immediately upon execution of a program by the floor manager.

3. 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.

3.1. MultiServer

MultiServer is a Java application adapted from the Free Internet Conferencing Tools (FICT) web 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*. Originally the FICT version of MultiServer managed chat boxes and channels. We have adapted it so that it keeps track of the floor queue using the Connection and Vulture classes and provides the dynamic re-execution of geometric algorithms in the library. The Vulture class has been slightly modified, as described later, and the Connection class has been augmented to handle a variety of data messages. The following sub-sections describe the MultiServer, Connection, and Vulture classes in more detail.

MultiServer MultiServer is a 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 listens on this socket for connections from users, and, when a new connection is made, a new Connection object is created and appended to MultiServer's queue of connections. A new Vulture object is also created initially, which ensures that all of the connections are valid. 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 (or exits the system), 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 by dynamically re-executing the algorithm when requested. That is, after the initial execution of an algorithm, MultiServer provides the functionality to allow the user to modify the original data input and immediately see the change in the output. So as the user is moving the location of, say, a point that was used as input to the algorithm, the display will be updated with the output of the algorithm corresponding to the modified input. 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 The Connection object is responsible for acting as the interface between its corresponding GeoJAVASheet, chat box, or user program and MultiServer. It receives messages from their interface objects and processes them appropriately. GeoJAVASheets go through MultiServer to broadcast messages to every GeoJAVASheet in its group or to send messages 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 their corresponding channel, hostname and TCP port number.

Vulture 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. One minor addition made to this class is that if the Vulture notices that a GeoJAVASheet has crashed while holding the floor, then it notifies the next GeoJAVASheet in the floor queue that it has been granted the floor.

3.2. 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.

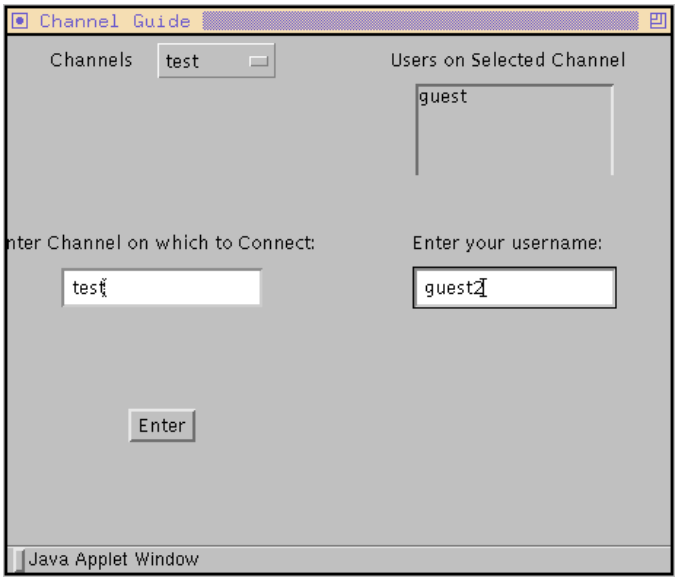

Fig. 1. ChannelGuide Applet.

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.

3.3. 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 (un)weighted and (un)directed 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 running on a Microsoft Windows machine.

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.

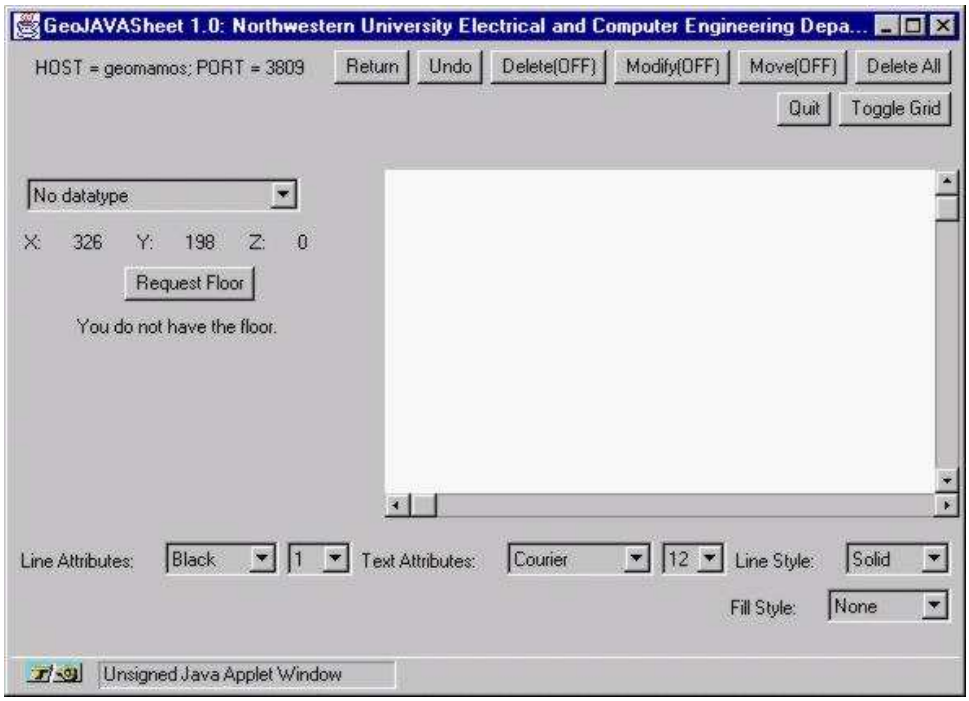

Fig. 2. GeoJAVASheet Applet.

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 GeoJAVASheet's appropriate internal list of data structures, and displayed on the panel. Please refer to Section 4 for more details.

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, indicating that an object has been entered and is ready to be sent to the user program. 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's Connection object, so once MultiServer receives this message, it forwards it to the appropriate user program.

As will be illustrated in the example in Section 5, the user program is able to control the geometric characteristics of the display, such as color, fill style, and line widths. However, this does not prevent the user interacting with the algorithm

from modifying these display properties as well. Anytime before, during or after the execution, the user is free to adjust any of these properties to his/her liking. This is a feature of the modularity of the system, where the user interface is not tied to the user program, or vice versa.

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,[16] just as in GeoMAMOS, so that these same files can be read in by GeoSheet and XFig, but other formats will be supported in future versions. Figure 3 is an instance of the GeoJAVASheet application running under Microsoft Windows.

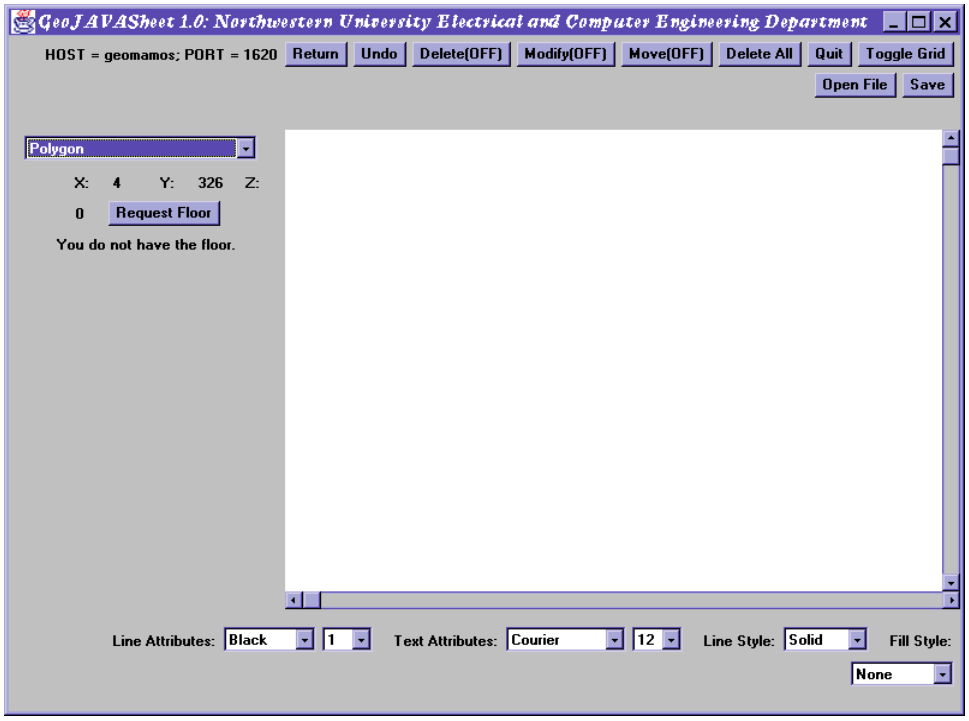

Fig. 3. GeoJAVASheet Application.

3.4. GeoLIB Geometric Library

The GeoLIB library in the current version consists of two parts: the LEDA[11,12] and GeoLEDA libraries.[10] Both libraries are written in C/C++. An advantage of this is that users who have already developed algorithms written in C/C++ may continue to use their algorithms 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)[15] into GeoLIB to be able to provide an even more comprehensive library for users.

LEDA The GeoLIB library is based on the basic geometric classes and member functions of the Library of Efficient Datatypes (LEDA) library (currently version

3.7)[13,14]. It provides the foundation of GeoLIB by supplying the base classes for all of the geometric objects used in the system.

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 was originally developed and used by the GeoMAMOS system. However, since GeoMAMOS uses UDP, all of the IPC functions have been modified for TCP for the *GeoJAVA system*, due to the reasons explained in the Appendices. This is advantageous in that although initializations are slower, communication while connected is faster and more reliable. Next, we briefly describe the main functions from the GeoLIB library that the programs use in order to visualize algorithms.

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 creates a socket connection via which the user program communicates with its Connection object, which acts as an interface to MultiServer. Any messages sent to the Connection object (or MultiServer) can be forwarded to the appropriate GeoJAVASheet since the Connection object stores the GeoJAVASheet ID to and from which it should send and receive input data. 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, or methods, implemented in every geometric object class and are issued from the user program.

Graphic_Read will cause GeoJAVASheet to return to the program the last object inputted onto the sheet. The process is as follows: (1) Graphic_Read is requested from the user program (GeoJAVASheet sets its choice box 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 in turn forwards it to the user program.

Note that the choice box setting on GeoJAVASheet is set automatically when GeoJAVASheet receives the Graphic_Read message. So the user is not expected to modify the choice box setting during a Graphic_Read. If for some reason the user decided to explicitly change the geometric object to input on a Graphic_Read, the GeoJAVASheet would display a message indicating that an invalid operation is being attempted when the user clicks on the sheet and no objects will be added.

If the user wished to implement and incorporate more complex objects, however, checks for the validity of the specific characteristics of the object should be performed at the library level. For example, if a user would like to use a recti-

linear polygon, he/she can create a class that inherits from the polygon class and have the Graphic_Read method ensure that the polygon it receives indeed consists of only horizontal and vertical edges. This can be done by either re-requesting a Graphic_Read or modifying the received data to make it valid. The same idea applies to other complex classes such as planar maps, which can inherit from graphs. Although GeoJAVASheet may not provide a choice box for these complex classes, by inheriting from the classes that are provided and using their Graphic routines, any complex class can be supported by the system.

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 is 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.

Because of the comprehensiveness of GeoLIB, the user is able to implement practically any geometric class and use any of the functions provided by the LEDA library. Therefore, a user may develop geometric algorithms that can perform any geometric computation without concern for implementing the display of intermediate or final results in their code.

3.5. 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. 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 list of the users currently in their channel has been additionally implemented to allow users to send private messages to individual members on the channel. 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.

3.6. 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 pre-

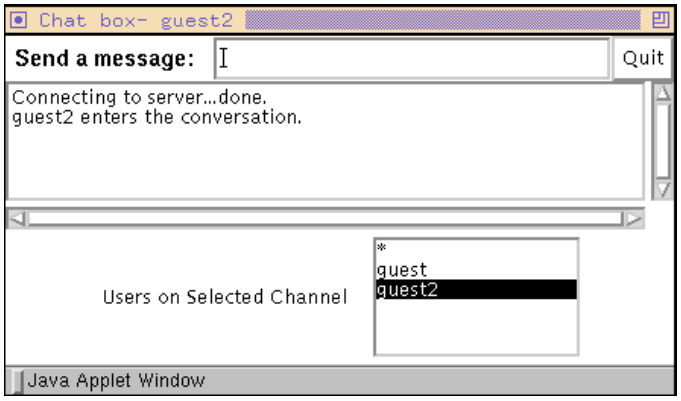

Fig. 4. Chat Box Applet.

cautions 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 compilation stage of the tool ensure that such malicious code is not used. A combination of linking with a reduced C library and relocating the executable to a directory on the server that limits the permissions of the program can provide the necessary security. Also, 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 illustrations demonstrate the steps in the compilation process. Figure 5 is the first page, where the source code is uploaded from the user's local disk to the server. The user may specify a note for their code, which is normally the name of the algorithm. The source code may consist of multiple files, so multiple fields are provided. Currently, there is a limit of five source code files.

Another restriction is that the source code must be written in native C++, as the libraries that they are compiled with are C/C++ libraries. In the future, however, the quickly growing popularity of Java may make the integration of Java libraries useful to those who prefer Java to C++.

Upon completion of the upload, the contents are displayed to ensure that the correct file(s) has(have) 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. 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. The compilation produces an executable located on the web server. Thus, when the user enters the host and port number for their GeoJAVASheet and clicks the Submit button, the CGI script will invoke a

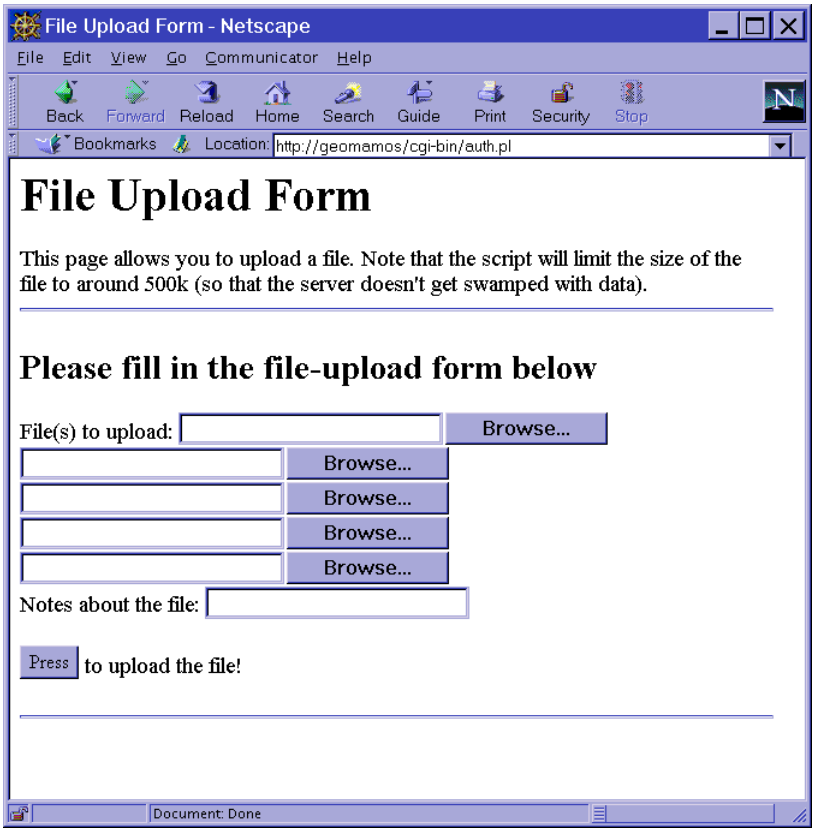

Fig. 5. Code upload page.

process on the algorithm executable and the output will be displayed on the specified GeoJAVASheet (and other GeoJAVASheets on the same channel). In this way, the executable is transferrable between any DOS- or Windows-based machine and is not necessarily "linked" to the GUI. So users may execute their algorithm from a DOS- or Windows-based machine, but collaboration and visualization may take place system-independently. Since the current web server is Windows-based, the executable returned is a Windows executable, but certainly a UNIX-based *GeoJAVA system* can be developed, thus offering UNIX-based executables to be compiled as well.

Once the execution of the program completes, the script will display any messages that were output during execution. So run-time errors can be checked at this point. The program can be run any number of times by simply re-submitting the form. At this point, the user may submit their code to add to the Algorithm Browser, which is shown in Figure 9. The Algorithm Browser lists the available algorithms for execution. Anyone with the floor may bring up this page and sample any of the algorithms provided.

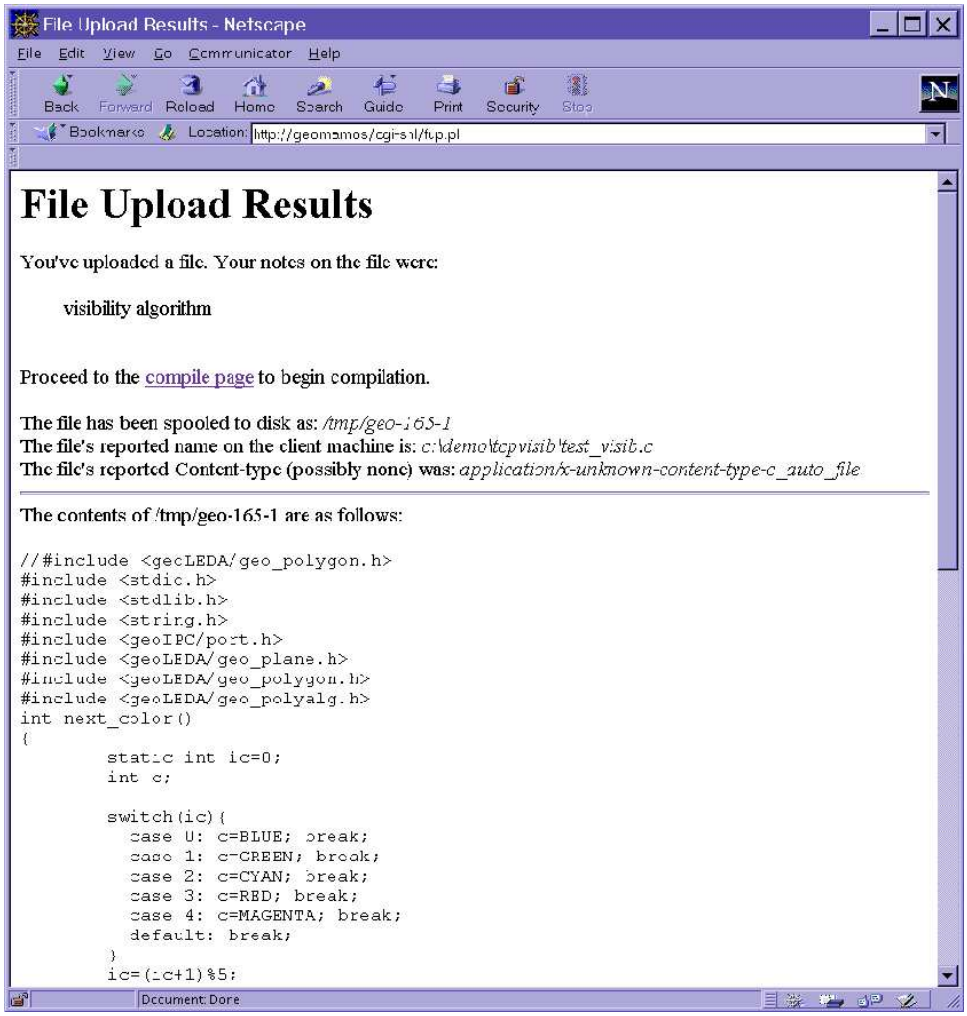

Fig. 6. Intermediate page.

4. System Design

This section will highlight the conceptual aspects of the *GeoJAVA system*.

4.1. Architectural Design

Combining the components described in the previous section, the result is a complete system as illustrated in Figure 10.

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

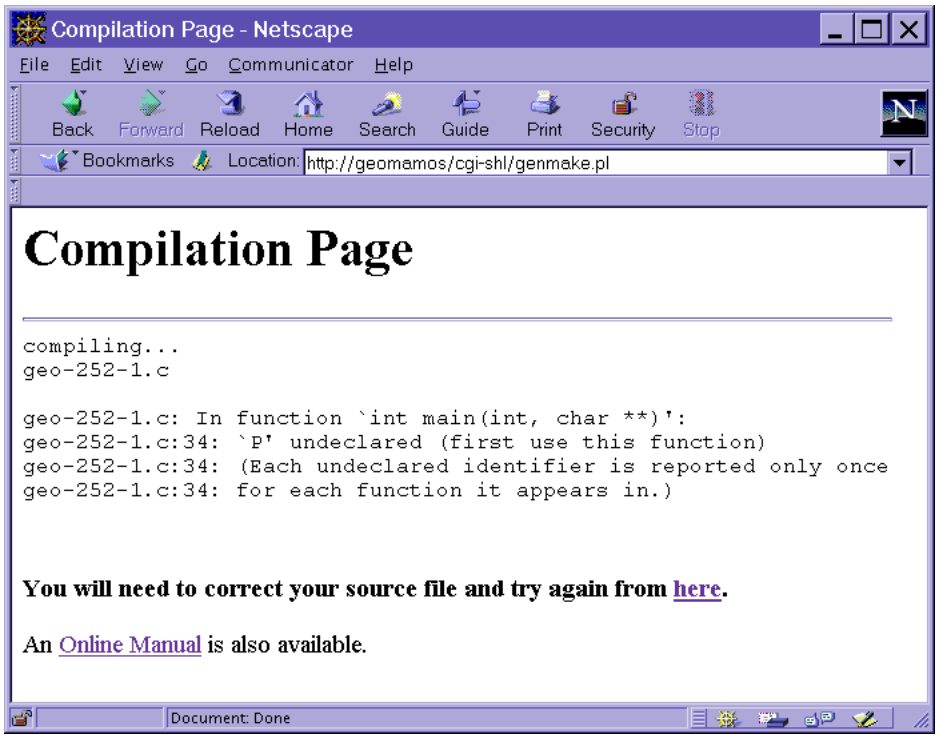

Fig. 7. Compilation output with error messages.

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 web page.

Currently, users on multiple channels all connect to MultiServer in the same way, on one port. However, it is apparent that MultiServer may eventually become a bottleneck. Therefore, as more channels are added to the system, we can make the system more scalable by having MultiServer spawn sub-MultiServers that each manage, say, a subset of channels.

The internal data structures used to efficiently manage the various Connection objects are fairly simple. The original architecture only needed to handle chat boxes, so there is one global vector of Connection objects, to which new Connections are appended whenever a new connection is requested. Additionally, three other data structures have been implemented to manage (1) the floor, (2) GeoJAVASheets, and (3) user programs.

Since each Connection object serves as an interface to MultiServer, it is instantiated as a "generic class" that only receives messages until it knows what type of object it is interfacing. Once it is informed of whether it is interfacing a GeoJAVASheet, chat box, or user program, it stores different information. For example, user programs would need to store the GeoJAVASheet ID to which it communicates, and GeoJAVASheets need to store the user program's ID that it is visualizing. In-

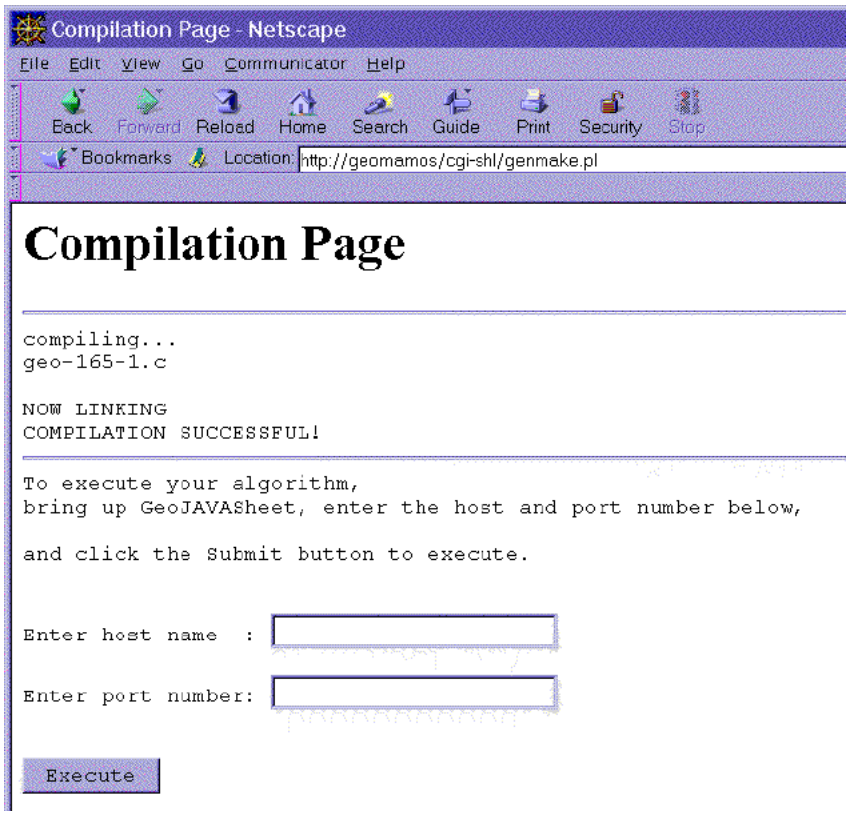

Fig. 8. Compilation output.

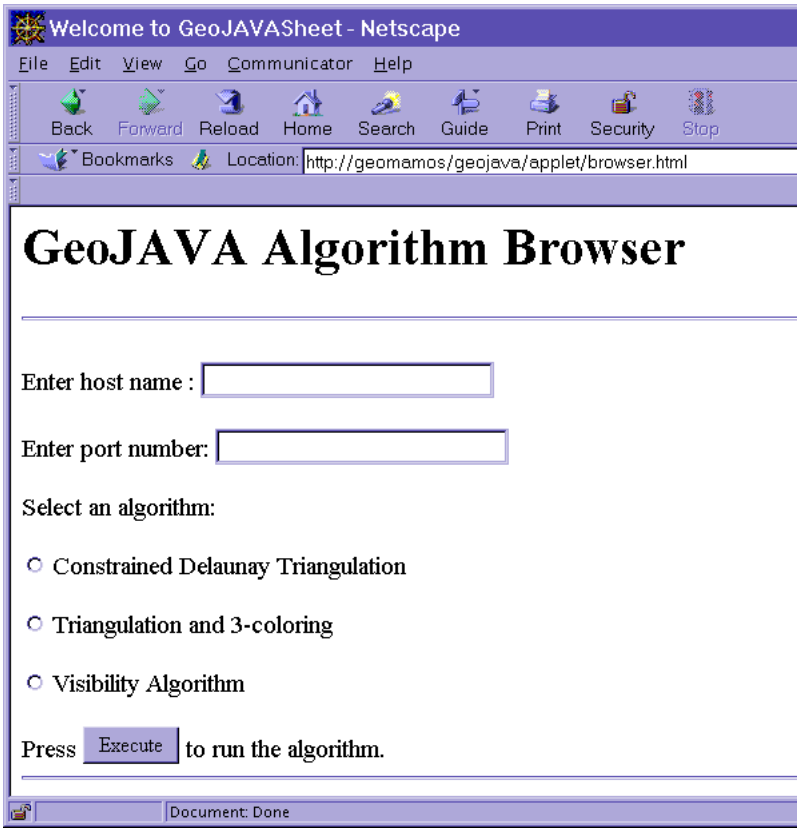

Fig. 9. Algorithm Browser.

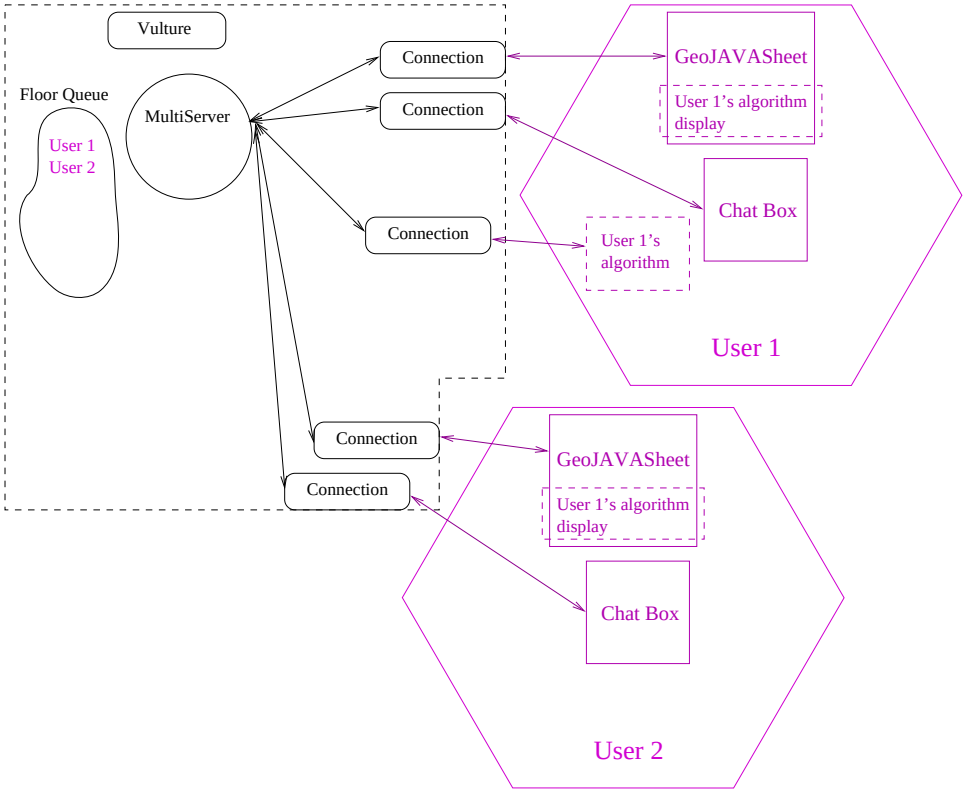

Fig. 10. *GeoJAVA System* Architecture.

stead of creating a different class for each of the GeoJAVASheets, chat boxes and user programs, one Connection class was used because at the point of the initial connection, MultiServer would not be able to determine which object to instantiate. So the Connection object is instantiated, which determines for itself the type of object it is interfacing.

Note the Vulture object is repeatedly checking the global queue of Connections in the case that a Connection "dies" unexpectedly. So it was more convenient to create lists of Connections in the GeoJAVASheet and user program data structures as opposed to creating lists of say, usernames. The Vulture object also updates the floor queue when it finds that a GeoJAVASheet has unexpectedly died and grants the floor to the next GeoJAVASheet in the queue.

As for the dynamic re-execution of algorithms, one requirement is that the algorithm to re-execute must be defined in the library. Obviously, user's programs cannot be dynamically re-executed, as this would cause problems when, for example, intermediate results are being displayed and the program is being called repeatedly. That is, if some program uses many Graphic_Writes to display the intermediate results of its algorithm and takes say ten seconds to complete, and this program is being re-executed every five seconds, the results would be quite unsatisfactory. Users who wish to incorporate their algorithms for dynamic re-execution will have to follow a specific format before being compiled into the library. This format consists of simply creating an overloaded function that takes as input an array of pointers to generic objects for the input (called GeoObjects) and to return an array of pointers to GeoObjects. The user's program to dynamically re-execute would thus call this function and pass in an array of pointers to its input objects after reading them in with Graphic_Read and store the output in a similar data structure. Further details of this format is beyond the scope of this paper.

Let us take as an example an algorithm which we would like to add to the library so that it can be dynamically re-executed. The algorithm is a Delaunay triangulation that takes as input a set of points and outputs a graph, the triangulation. The user would write the algorithm as a function that takes one argument, a pointlist, and returns a graph. Then this function would be overloaded so that another function of the same name is written, but it takes as input a GeoObject list and returns a GeoObject. The overloaded function only needs to store the GeoObjects passed in as a pointlist, call the original function, take the triangulated graph result, cast it as a GeoObject and return it. Once these two functions are incorporated into GeoLIB, any user can call the Delaunay triangulation in their code and dynamically re-execute it. After a user calls the function and presses the Modify button to update the original input points, MultiServer will repeatedly pass the updated pointlist to the overloaded function so that it can re-run the triangulation on the new list and return the updated triangulation in a graph to be re-displayed.

This process of re-execution is made possible by Java's Java Native Interface (JNI). Albeit this technology is provided in Java version 1.1, it is only used by MultiServer, which is running as a Java application on the web server and thus has no effect on clients' web browsers and still upholds GeoJAVASheet's "Java pure"-ness.

"Re-executable algorithms" are provided by a shared library to which MultiServer has access. The algorithms in this library are all compiled in the format mentioned previously. When a user's program that calls a re-executable algorithm runs, MultiServer "registers" the algorithm by accessing the corresponding algorithm name in the library. When the user program completes and a user (with the floor) selects the "Modify" button, GeoJAVASheet will repeatedly send the updated input object data to MultiServer, which calls the algorithm with the updated data and returns the updated output object data to the GeoJAVASheets in the channel. As long as the Modify button is turned on, this process repeats itself continuously. The floor can be transferred to other users who can also modify the data in the same way and watch the algorithm re-execute.

4.2. Conceptual Design

Conceptually, the *GeoJAVA system* can be viewed as consisting of two major parts, the GUI and the underlying "brains" of the system. The GUI consists of GeoJAVASheet, chat box, and the browser, while the "brains" correspond to GeoLIB and the low-level communication implementation such as that provided by MultiServer. However, as will be described later, the user need not use the GUI in order to run their application.

The advantage of this *GeoJAVA system* design is that this modularity allows the GUI to be system independent, and the user never needs to worry about how to implement the display of their algorithm. They only need be concerned with the computations involved. Other systems tend to incorporate these two modules together (i.e. Mocha, written in Java, and LEDA). However, in these cases, users who wish to share their geometric results with others would be required to write code that explicitly broadcasts the visualization of the program to remote hosts.

This system was originally designed as a tool for users to more easily implement geometric algorithms. However, once a user has completed his/her implementation, they may still use their program apart from the tool as a "stand-alone" application, using files or standard I/O to obtain input and interact with the user. This is made possible by a check that is performed in the Graphic_Read and Graphic_Write methods which tests for a valid socket connection before sending or receiving data. If this test fails (i.e., the program is running "stand-alone"), then instead of the normal Graphic_Read or Graphic_Write operation, the method will invoke a read or write from/to file(s) or standard I/O. In the case of using file(s), IPCServiceSetup just needs to be replaced by FileSetup call, which opens two globally defined files for input and output, respectively. (The user is responsible for formatting his/her input files in the format specified for each object as is described in the GeoSheet Manual). Each data type class also contains F_Read(char*) and F_Write(char*) methods which read from and write to the filename supplied to the functions, allowing users to save their objects in distinct files. For standard I/O, no setup procedure needs to be followed; the user will be prompted for input during the Graphic_Reads and output will be displayed on standard out. Thus, these methods give users the option to run their programs separately from the GUI.

5. 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 3.4.

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>
#include <LEDA/graph.h>

int main(int argc, char* argv[])
{
  // defined in geo_polygon.h
  geopolygon P, *final;

  // defined in geo_plane.h
  GeoPoint pt1;

  // defined in graph.h
  node v;

  // intialize the geopolygon
  final = new geopolygon();

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

  // defined in port.h
  // set fill style to 0 = no fill
  SetOutSheetFillStyle(0);

  // also in port.h
  GeoPause("Please enter a simple polygon");

  // Graphic_Read method from geopolygon class
  P.Graphic_Read();
```

```
  GeoPause("Please enter an interior point");

  // Graphic_Read method from GeoPoint class
  pt1.Graphic_Read();

  // VISIBILITY defined in geo_polyalg.h
  // takes as arguments a reference to a geopolygon and
  // a GeoPoint, and returns a geopolygon
  *final = VISIBILITY(&P, pt1);

  // set the output color to blue
  SetOutSheetColor(BLUE);

  GeoPause("Ready to see the visibility polygon?");

  // Graphic_Write method from geopolygon class
  // since final was defined as a pointer to a geopolygon,
  // the -> must be used instead of . to call its
  // Graphic_Write method
  final->Graphic_Write();
}
```

When a C/C++ program is executed with arguments, these arguments are stored in the argc and argv variables, where argc is an integer indicating the number of arguments (including the program name) and argv is an array of strings, each string storing the argument(s) passed to the program. For example, say the program name is "visib" and its arguments are the host and port number, "geo" and "1234." Then argc would equal 3 and the array argv would contain { "visib", "geo", "1234"}. Since `IPCServiceSetup()` is overloaded as `IPCServiceSetup(char* host, int port)`, the second argument to `IPCServiceSetup()` must be converted to an integer, which is done using `atoi()`.

In this example, before the call to `IPCServiceSetup()`, the program checks its arguments and assumes that if arguments exist, 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 execute the program with the host and port number as arguments.

Figure 11 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, Figure 12 is what Kiyoko sees on her machine. But when she sees Frank join the channel, she can send him messages, as in Figure 13, 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 14, 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 15, 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,

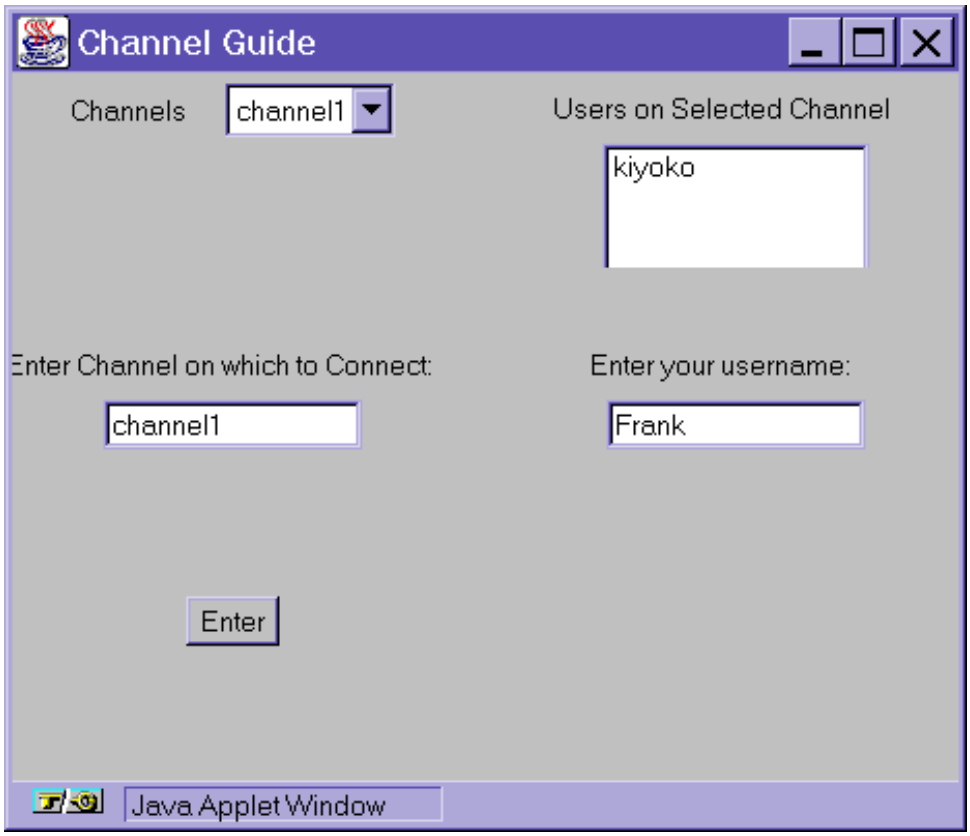

Fig. 11. ChannelGuide

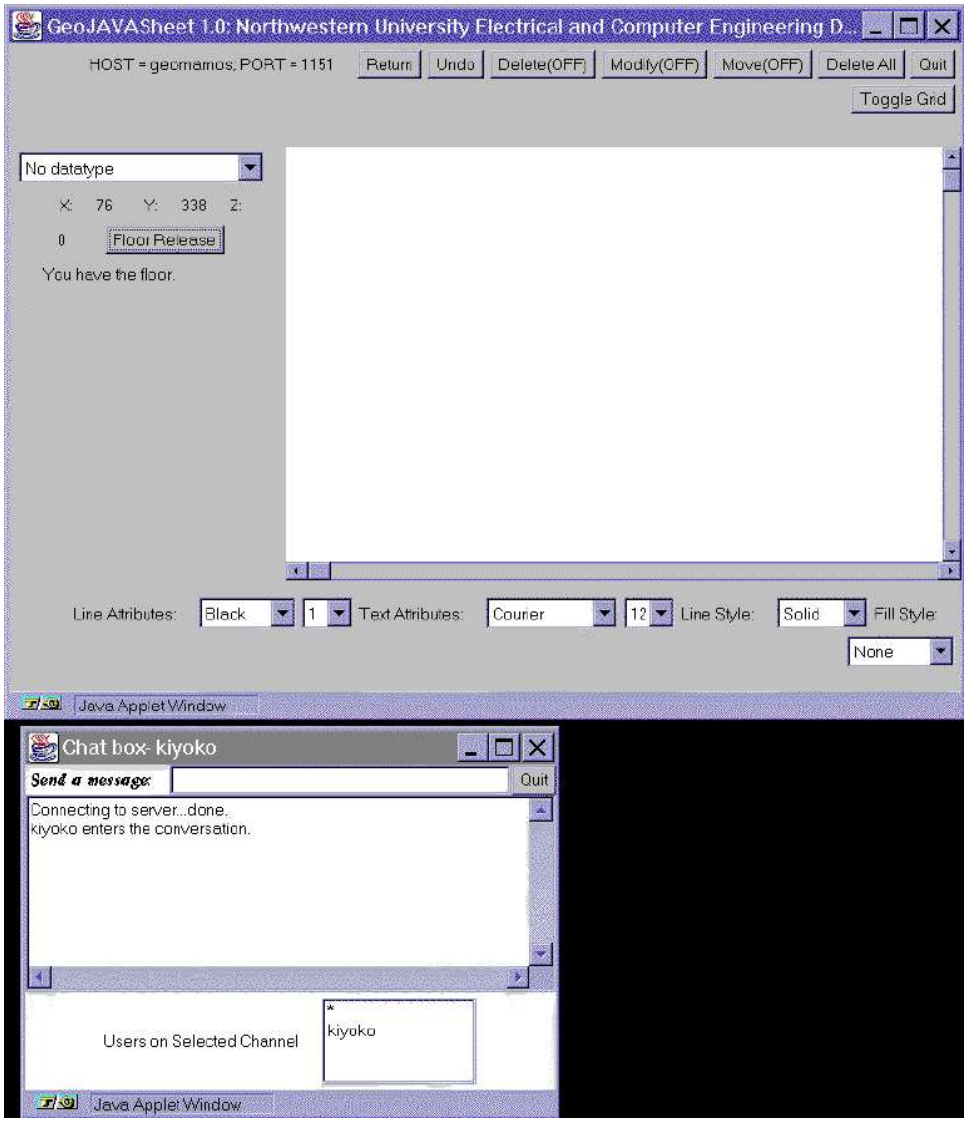

Fig. 12. Kiyoko's View

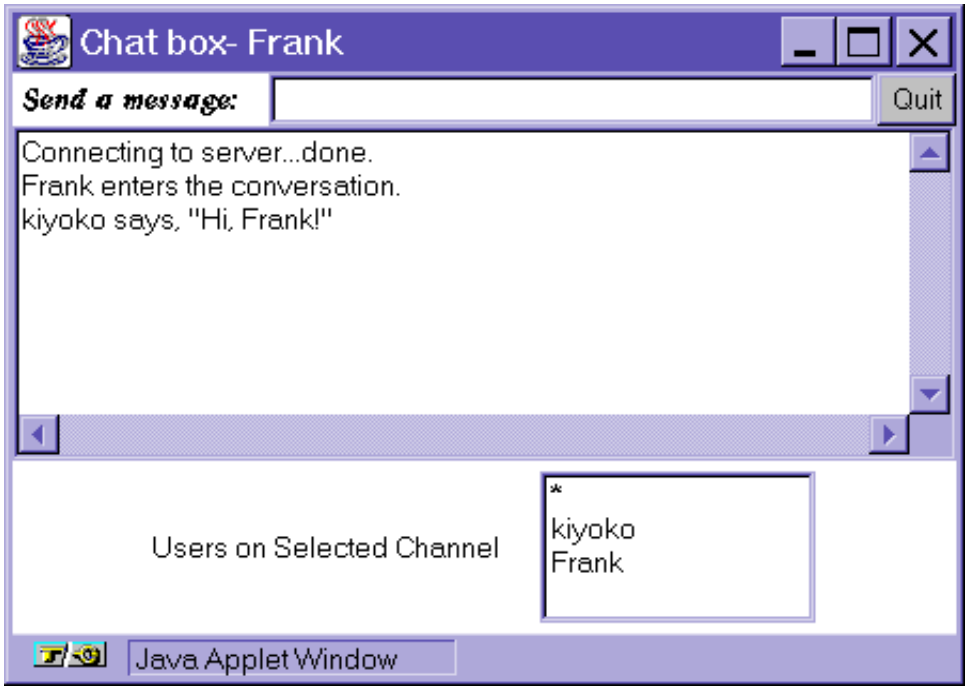

Fig. 13. Frank's Chat Box View

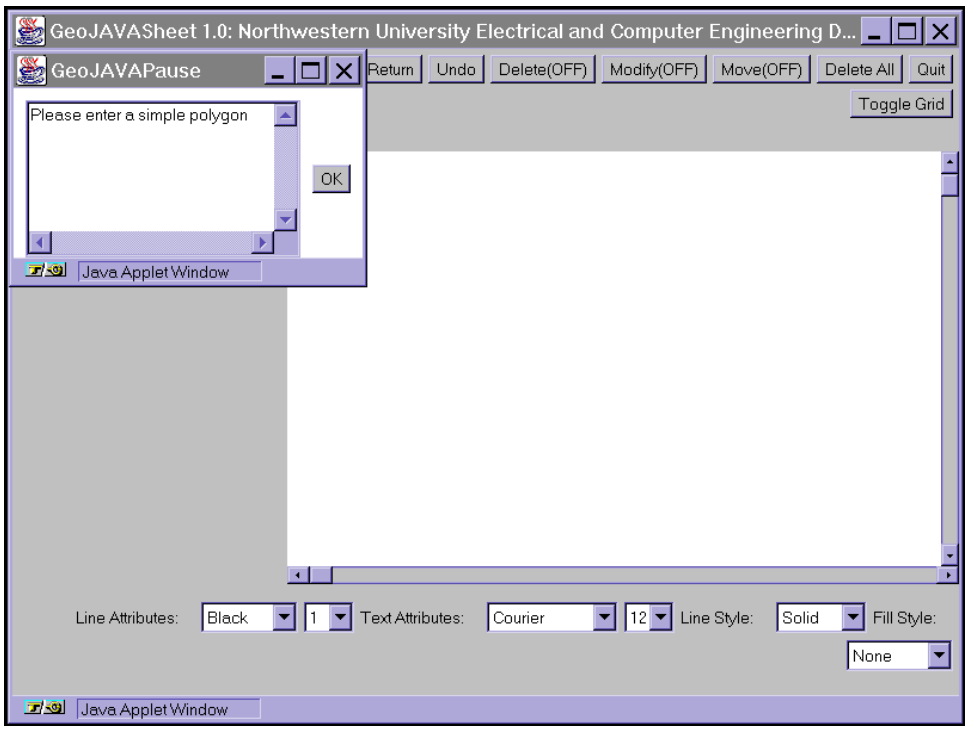

Fig. 14. Enter a polygon request.

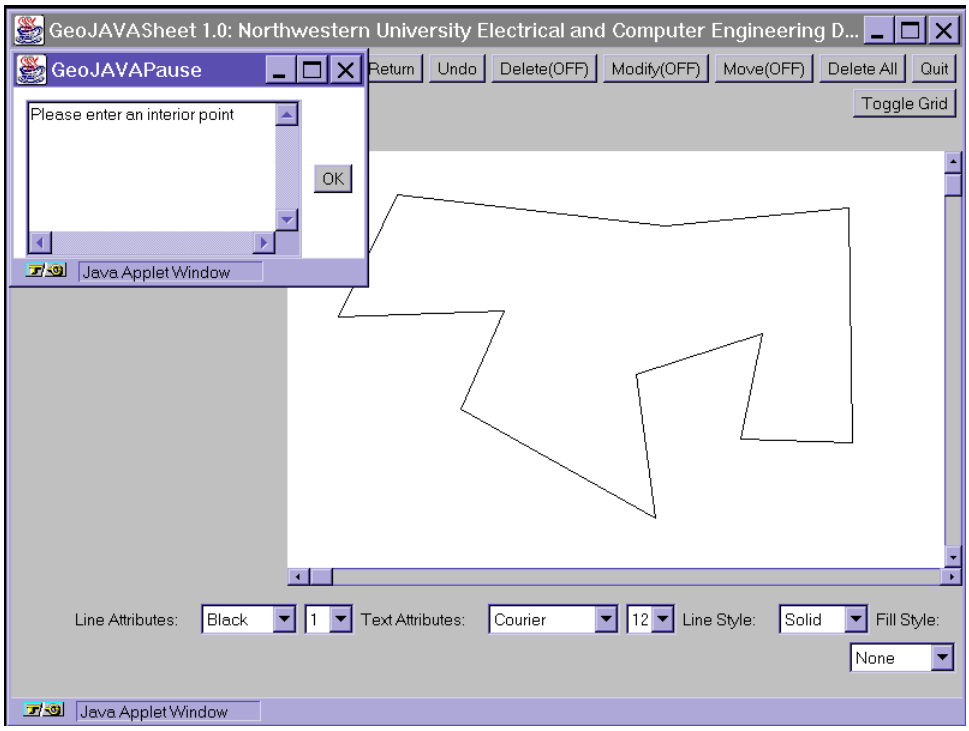

Fig. 15. Enter a point request.

only Frank sees the polygon entered.

Figure 16 displays the change in the GeoJAVAPause window to prepare the user for the output, and Figure 17 is the final output to the program.

Figure 18 displays another example of three GeoJAVASheets and three chat boxes connected to the same channel while running on three different hosts.

6. 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 make the system scalable, and the dynamic re-execution 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.

Furthermore, a Java version may be developed of the GeoLIB library, which may eliminate the need for a large C/C++ library (since Java already provides classes

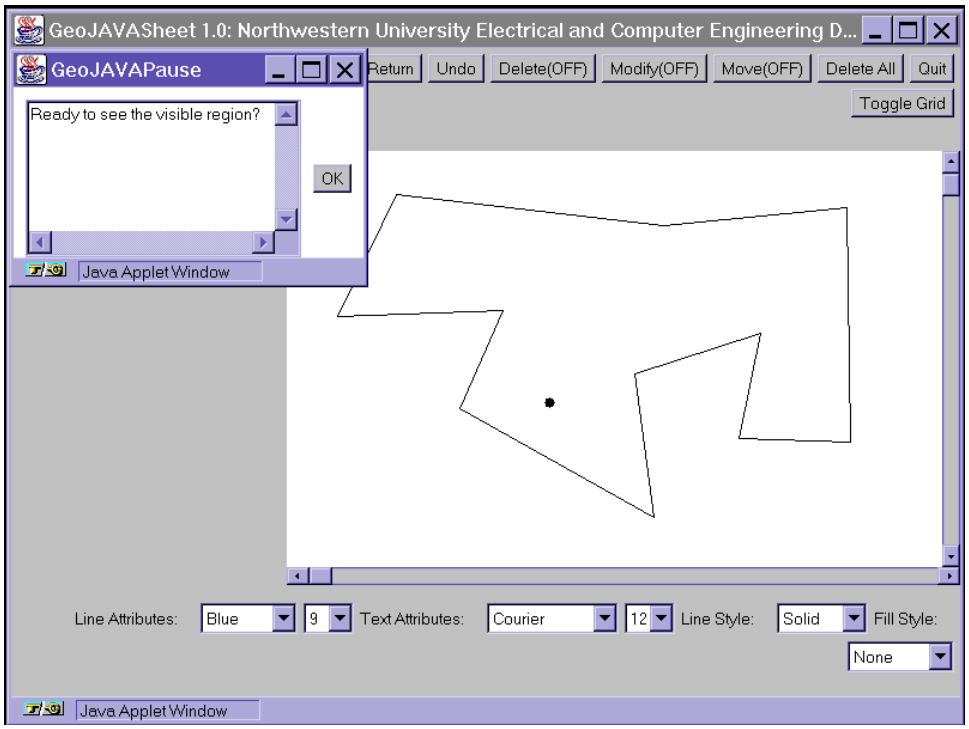

Fig. 16. Ready request.

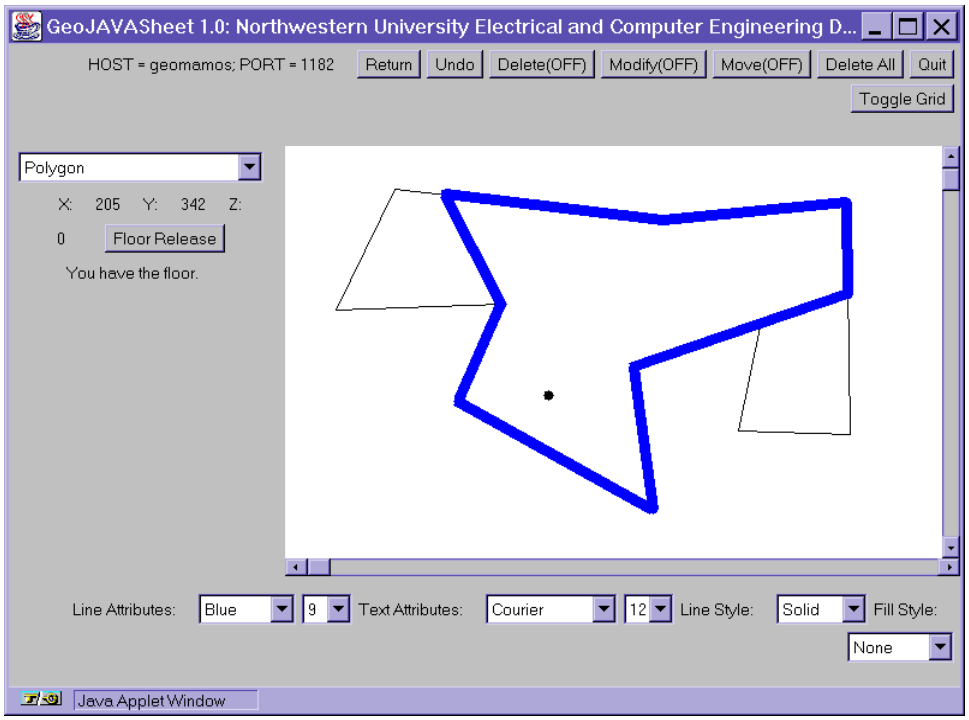

Fig. 17. Final program output.

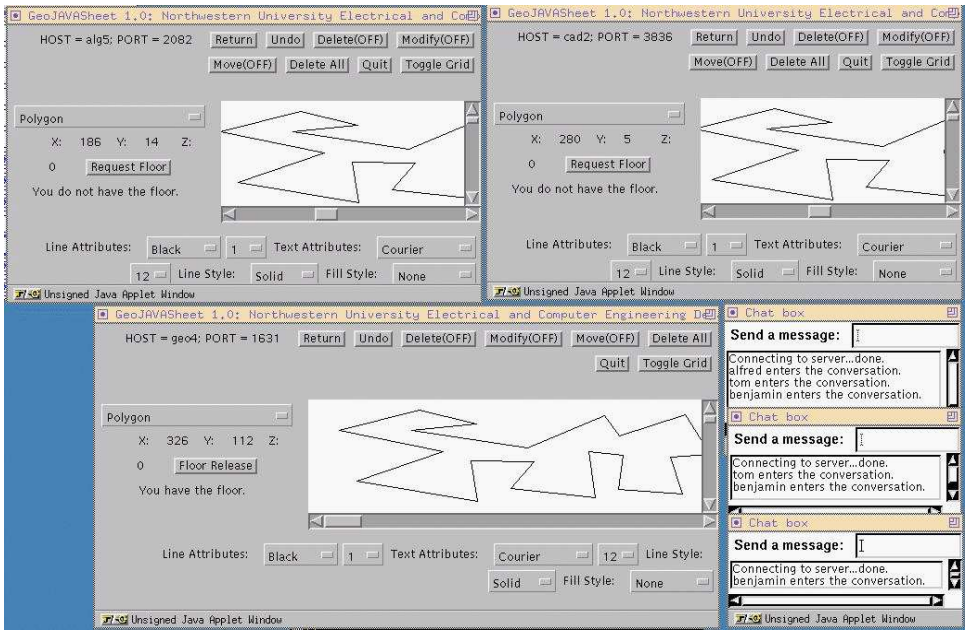

Fig. 18. *GeoJAVA System* Demo.

for basic geometric objects) and provide another option for those who do not know or do not wish to implement their algorithms in C/C++. This would also offer users a wider range of programming languages, especially since the trend seems to indicate that Java will continue to gain popularity.

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 compilation system can also be expanded to networks of computers that are accessible locally to the user in addition to over the Internet. Users will be able to compile their C/C++ and/or Java programs from their local computers, using libraries located remotely. Consequently, computers that are not necessarily connected directly to the Internet, but to a local area network (LAN) from which an network connection is available through another computer, will still be able to visualize their programs and collaborate with other users. In this way, the user's working space can be augmented even more.

The *GeoJAVA system* exemplifies a fresh approach to web-based programming. Incorporating native C/C++ programs to a system-independent interface using Java allows existing programs to be used in combination with the latest technology. In addition to the transparent support for distributed visualization, the on-line compilation facilities provided are features unique only to the *GeoJAVA system*. Compared to Java's RMI technology in version 1.1, where remote objects are directly accessed within the confines of the Java environment, the *GeoJAVA system*

is refreshing in that it expands the tools available to the user by incorporating an already popular programming language. Our approach can potentially allow any program written in any language to communicate with our system because of the use of sockets. The simple communication protocol is all that needs to be observed. The GeoLIB library can certainly be compiled on other platforms to develop programs in other environments, but still communicate with the *GeoJAVA system*. No other system can provide such functionality.

Another goal is that this system can serve as a foundation on top of which a problem solving environment for other fields of scientific research and development can be built. For example, chemical or mechanical engineers could possibly use this tool to dialogue and collaborate on ideas for which geometric objects may be useful.

Ultimately, our desire is to enable users to take full advantage of the newest technology while not disallowing the use of existing tools that are adequate and in use today. With the fast-paced growth of technology, we hope to assist users by providing the most useful and worthwhile of technological developments and empowering people to advance in their respective fields through the use of our system.

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. Furthermore, the anonymous referee's comments on an earlier version of this paper were very helpful in improving its presentation.

Funding was provided in part by the Office of Naval Research under the Grants No. N00014-95-1-1007 and No. N00014-97-1-0514, the National Science Foundation under the Grant No. CCR-9731638, and the National Science Council under the Grant No. NSC 88-2219-E-001. A preliminary result was presented at the "Workshop on Geometric Software," held in INRIA, Sophia-Antipolis, France in June 1997.

References

1. K. Aoki, "The Prototyping of GeoManager: A Geometric Algorithm Manipulation System," Master's Thesis, Dept. EE/CS, Northwestern University, 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 *Proc. 1997 IEEE Symposium on Visual Languages*,

(IEEE Computer Society, September 1997) pp. 372–379.

5. M. H. Brown, R. Sedgewick, "A System for Algorithm Animation," *Computer Graphics* **18**(3) (July 1984) 177-186.
6. U. Gall, F. J. Hauck, "Promondia: A Java-Based Framework for Real-Time Group Communication in the Web," *Proc. 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) pp. 119-155.
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. http://www.mpi-sb.mpg.de/LEDA/leda.html.
12. K. Mehlhorn, S. Näher, "LEDA: a platform for combinatorial and geometric computing," *Communications of the ACM* **38,1** (Jan. 1995) 96–102.
13. S. Näher, "The LEDA3.0 User Manual," technischer Bericht A, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, 1992.
14. K. Mehlhorn, S. Näher, M. Seel, C. Uhrig, "The LEDA User Manual - Version 3.7," Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany. http://www.mpi-sb.mpg.de/LEDA/MANUAL/MANUAL.html.
15. M. H. Overmars, "Designing the Computational Geometry Algorithms Library CGAL," in *Proc. Workshop on Applied Computational Geometry*, (Philadelphia, Pennsylvania, May 27-28, 1996) pp. 113-119.
16. B. V. Smith, *The Xfig User Manual*, 1993.

Appendix A: Terminology

The following lists definitions of terminology used that should be familiar to the reader, taken from the Free On-Line Dictionary of Computing (http://wombat.doc.ic.ac.uk/foldoc/index.html).

class The prototype for an object in an object-oriented language; analogous to a derived type in a procedural language. The structure of a class is determined by the class variables which represent the state of an object of that class and the behaviour is given by a set of methods associated with the class.

datagram A self-contained, independent entity of data carrying sufficient information to be routed from the source to the destination computer without reliance on earlier exchanges between this source and destination computer and the transporting network.

LAN (or local area network) A data communications network which is geo-

graphically limited (typically to a 1 km radius) allowing easy interconnection of terminals, microprocessors and computers within adjacent buildings. Ethernet and FDDI are examplse of standard LANs.

object In object-oriented programming, a unique instance of a data structure defined according to the template provided by its class. Each object has its own values for the variables belonging to its class and can respond to the messages (methods) defined by its class.

packet The unit of data sent across a network.

socket The Berkeley Unix mechanism for creating a virtual connection between processes. Sockets can be of two types, stream (bi-directional) or datagram (fixed length destination-addressed messages). The socket library function socket() creates a communication endpoint or socket and returns a file descriptor with which to access that socket. The socket has associated with it a socket address, consisting of a port number and the local host's network address.

stream An abstraction referring to any flow of data from a source (or sender, producer) to a single sink (or receiver, consumer). A stream usually flows through a channel of some kind, as opposed to packets which may be addressed and routed independently, possibly to multiple recipients. Streams usually require some mechanism for establishing a channel or a "connection" between the sender and receiver.

TCP (or Transmission Control Protocol) The most common transport layer protocol used on Ethernet and the Internet. TCP is nearly always seen in the combination TCP/IP (TCP over IP). It adds reliable communication, flow-control, multiplexing and connection-oriented communication. It provides full-duplex, process-to-process connections. It is connection-oriented and stream-oriented, as opposed to User Datagram Protocol.

UDP (or User Datagram Protocol) Internet standard network layer, transport layer, and session layer protocols which provide simple but unreliable datagram services. UDP adds a checksum and additional process-to-process addressing information. UDP is a connectionless protocol which, like TCP, is layered on top of IP. UDP neither guarantees delivery nor does it require a connection. As a result it is lightweight and efficient, but all error processing and retransmission must be taken care of by the application program.

Appendix B: 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 means 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.