FEATURE

GEOMNET: Geometric Computing over the Internet

GeomNet's layered client-server architecture lets users perform distributed geometric computing over the Internet without having to download, install, and interface with software.

Gill Barequet
The Technion—IIT and Johns Hopkins University
Christian A. Duncan and Michael T. Goodrich
Center for Geometric Computing, Johns Hopkins University
Stina S. Bridgeman and Roberto Tamassia
Center for Geometric Computing, Brown University

Consider two real-life problems. First, suppose you have a collection of nails hammered into a board. If you were to stretch a rubber band around the outside of the nails and let it snap, what shape would the rubber band take? Second, imagine two planes whose trajectories and flight patterns show that they must, at one instance in time, fly dangerously close to one another—will they collide?

In geometric terminology, the shape sought in the first problem is called the convex hull of a set of points in the plane (nails on the board). This is the smallest convex polygon containing all the points. The solution sought in the second problem is an efficient algorithm for detecting potential collisions between objects.

Geometric computing emerged from algorithms developed to solve such problems. It has become a central building block in fields like computer graphics, artificial intelligence, CAD, and GIS databases.

Although many representational formats for geometric data exist, only a few fully implement the combinatorial connectivity information that makes geometric data so interesting. Typically, each geometric software package is designed for a unique, incompatible data format. Designing a standard language for describing geometric data would necessitate the encoding of all possible relationships between the numerical and combinatorial components of geometric data—a Herculean task that doesn't seem likely.

In spite of difficulties, there are a number of robust implementations of geometric algorithms. Many of these are available on the Internet (see the sidebar "Geometric Software on the Web").

Our group at the Center for Geometric Computing developed the GeomNet system to provide easy Internet access to geometric implementations via a plug-and-play environment, forming a link between software evaluation and production modes. By providing a progressive migration of software from the host to the client, GeomNet attempts to simplify interfacing, one of the most significant problems in software engineering and in using software developed elsewhere. Users can invoke online, separately or in a pipeline, a rich collection of geometric computations for performing one-time or repeated tasks. The system is suitable for a wide variety of tasks, such as invoking an algorithm with specific input data, checking geometric structures or data for consistency, experimentally studying and/or comparing algorithms, designing new algorithms through the integration of existing algorithms, or demonstrating the course of an algorithm in an educational setting.

A COOPERATIVE COMPUTING ENVIRONMENT

We envisioned GeomNet as a family of geometric computing servers executing various geometric algorithms on behalf of remote clients. These clients can be users interacting through a Web browser interface, or application programs connecting directly through sockets. We refer to negotiations between client and server as *cooperative computing.*

GeomNet is based on a layered object-oriented architecture (see Figure 1), with the highest level responsible for interacting with client processes and the lowest level responsible for interacting with specific geometric programs. This structure divides a GeomNet server's tasks into separate conceptual units and hides the details of specific implementations from components that don't need that information. This makes it easier both to maintain GeomNet and to incorporate new algorithms into the system.

GeomNet supports file formats commonly used in various communities. Formats for describing three-dimensional geometries include .OFF (used by the data visualization and manipulation Geomview;[1] see http://www.geom.umn.edu/software/download/geomview.html), .WRL (for VRML files), and .STL (a manufacturing file format). The system also supports several graph formats. We are continuously incorporating more formats.

Briefly, users submit data in one of the supported data formats and request a computation from a GeomNet server. GeomNet automatically converts the data to the format assumed by the requested computation, performs the computation, and returns the result in one of many popular formats, including interactive two- and three-

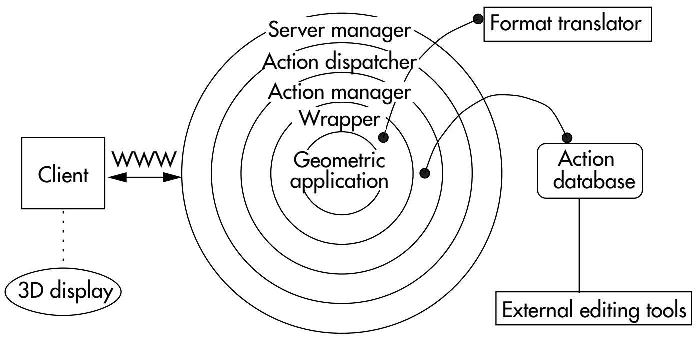


Figure 1. The GeomNet architecture. A client sends a request to one of the GeomNet servers, where the server manager adds it to its queue of incoming messages. The action dispatcher extracts messages from the queue and forwards them to the action manager, which decodes the message and, using the action database, invokes the wrapper of the application that will handle the request. After termination, the output is forwarded back through the layers and returned to the client.

GEOMETRIC SOFTWARE ON THE WEB

Collections of geometric software can be found on the Internet at Amenta's "Directory of Computational Geometry Software" site (http://www.geom.umn.edu/software/cglist/), and a host of geometric computing applications can be found at Eppstein's "Geometry in Action" site (http://www.ics.uci.edu/~eppstein/geom.html). Erickson has an even more extensive collection of geometric software at his "Computational Geometry Software" site (http://www.cs.duke.edu/~jeffe/compgeom/software.html).

In addition, there is a large collaborative effort underway to develop an extensive Computational Geometry Algorithms Library (http://www.cs.ruu.nl/CGAL/),[1,2] which itself is built upon the successful Library of Efficient Data Types and Algorithms (http://www.mpi-sb.mpg.de/LEDA/leda.html).[3,4] This effort is directed at building a large collection of C++ routines for solving computational geometry problems. This collection should grow significantly in the years ahead.

REFERENCES

1. A. Fabri et al., "The CGAL Kernel: A Basis for Geometric Computation," *Applied Computational Geometry (Proc. WACG 96)*, Lecture Notes in Computer Science 1148, Springer-Verlag, 1996, pp. 191-202.
2. M.H. Overmars, "Designing the Computational Geometry Algorithms Library CGAL," *Applied Computational Geometry (Proc. WACG 96)*, Lecture Notes in Computer Science 1148, Springer-Verlag, Berlin, 1996, pp. 53-58..
3. C. Burnikel et al., "Exact Geometric Computation in LEDA," *Proc. 11th Ann. ACM Symp. on Computational Geometry*, ACM Press, New York, 1995, pp. C18-19.
4. K. Mehlhorn and S. Näher, "LEDA: A Platform for Combinatorial and Geometric Computing," *Comm. of the ACM*, Vol. 38, 1995, pp. 96-102.

WHERE TO ACCESS GEOMNET

A first implementation of GeomNet is available on the Internet through the support of the Center for Geometric Computing at Johns Hopkins University (http://www.cs.jhu.edu/labs/cgc/) and at Brown University (http://www.cs.brown.edu/cgc/), with prototype servers available at the GeomNet sites (http://www.cgc.cs.jhu.edu/geomNet/ and http://loki.cs.brown.edu:8081/geomNet/) at the respective universities. New releases are planned throughout 1999 and 2000.

dimensional visualization programs already plugged in to the user's Web browser or, alternatively, the user's application.

Client-Server Dialog Protocol

The dialog protocol between GeomNet's servers and clients was designed and implemented such that a client can call applications on the server's side. Messages exchanged between GeomNet clients and servers have a keyword-value structure. That is, every message is composed of a list of pairs consisting of a keyword, which identifies the name of a field in the message, and a value, which represents the contents of that field. This type of syntax is not only simple to parse and to understand, but also very powerful and expressive.

The client's application (either an interactive applet embedded in a Web page or a stand-alone program) sends a message to a GeomNet server, which expects the user to select a geometric operation (called an action) from the list of available operations (algorithms). Implicit in this selection are any parameters required for the specific action. Some parameters are mandatory, for example, the action name and the input filename. Some parameters are optional. If, say, the input file type is omitted, the system automatically derives it from the filename extension. Still other parameters are specific to the algorithm, for example, a tolerance for an approximation algorithm. Either the user or the client program could select the exact values assigned to these parameters. In either case, the message sent from the client encodes the requested action and the accompanying parameters in a Web-like syntax:

```
GN_action=convex-hull&GN_infile=large-set.off& \
GN_outfile=small-set.wrl&GN_intype=stl&mode= \
2-d&type=upper-hull
```

The message is further encoded by using the MIME type

```
application/x-www-form-urlencoded
```

so that special characters in the keyword and value strings are replaced by their ASCII hexadecimal values.

GeomNet keywords begin with the prefix GN_. Each application may also have its own set of additional keywords. In the example above, the application itself can use the keywords mode and type. Indeed, the same keyword may be used for different actions. Action-specific keywords are not necessarily unique; for example, keywords such as epsilon, mode, and angle are expected to be shared by many algorithms.

Note the similarity between this protocol and Java's remote-method-invocation mechanism: A keyword-value structure is simply a way to invoke a (remote) procedure by setting actual values to formal parameters. In addition, our protocol allows GeomNet to have default values, aggregate arguments (encoded appropriately in the message), and indirect referencing (by specifying a URL as an argument). In particular, arguments are passed by name, and their order is insignificant.

Computation Requests at the Server

GeomNet servers are responsible for decoding and fulfilling client requests for geometric computations. A GeomNet computation begins with the server manager, which tracks client requests and sends responses back to the appropriate clients. The server manager handles a queue of incoming messages to maintain the incoming requests. To process the next awaiting message, the server manager calls its action dispatcher, which extracts the message from the queue and forwards it to the action manager on that machine.

The action manager decodes the message according to the agreed-upon syntax and looks for the mandatory field GN_action. The action is then handled using the action database, where all available actions are fully specified by their names and parameters. This database, saved in an ASCII control file written in Java, identifies the tool that should handle the action. More specifically, the action database maintains details concerning the availability of an action and the name of the Java wrapper class that should be used for performing that action. The action database does not contain information about the interface of the actual tool (usually a stand-alone program) that performs the action. It should know only what input and output formats are needed so that it can, if necessary, also schedule a format conversion routine.

The action manager performs file-type conversions, if necessary, before and after calling the wrapper level. Then the action manager invokes the wrapper of the application that should handle the requested action and passes it the necessary data. When the wrapper terminates, the action manager returns the action's output to the action dispatcher, which then forwards it to the server manager. This is done by passing back a locator (filename) for the result or, if it is small enough, by passing back the actual result encoded in a URL.

Wrapper-Application Interaction

The knowledge of what actual code or environment must be invoked to satisfy an action is at the bottom layer of the architecture, where we store the Java wrapper class that interacts directly with the application. The action manager invokes the serving application's wrapper class by name. It obtains this name at runtime (rather than during compilation) and binds the wrapper by using Java's runtime class loader. This allows for the addition of new algorithms (through new wrappers) without having to recompile, or even restart, the server.

An application's wrapper is the only system component that knows how to invoke the application. Thus, modifications made in the interface to the application, or even in the actual code for implementing an application, require only recompilation of the wrapper. No other GeomNet component is affected by changes made in the actual implementation of its geometric components. The wrapper performs all necessary preparations for properly invoking the associated application, then invokes it, receives the application's output, and returns the result to the action manager. The wrapper terminates an application running too long by sending a time-out interrupt.

Adding a new application requires coding a suitable wrapper (that agrees with the interface conventions), updating and recompiling the control file, and restarting the system. Entire system recompilation isn't needed; thus new applications can be added to every system installation without the system having the application's source code. The end user, however, can use only system-embedded applications.

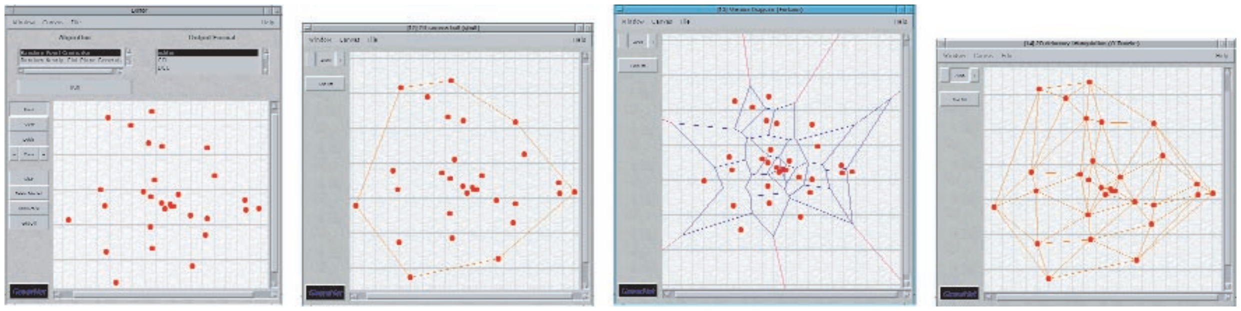
Typically, geometric applications belong to one of three types: programming-language functions, stand-alone computer programs, and embedded applications. We have implemented wrappers for all three types.

A *programming-language function* expects the input data in certain data structures to be passed to it as actual arguments. Optionally, an argument can contain the name of a data file. This is usually the case when the function is really a "roof" over a stand-alone program. When calling a function, the wrappers first parse the temporary input file and prepare the data structures required by the function. They then call the function according to its calling sequence. Finally, they scan the data structures returned by the function and write the contents in a temporary output file with the format requested by the server.

Modifications made in the interface to the application require only recompilation of the wrapper.

A *stand-alone computer program* expects all the input in a command line to be extracted by the famous C argc/argv mechanism, or by Java's args[]. These arguments include the names of the input and output files and other arguments to the program. The program expects the input file to be in one of several predefined file formats and usually writes the output in the same file format. If the program does not recognize the syntax of the temporary input file, the wrapper converts the input file into a secondary input file with a syntax the program recognizes. The wrapper runs the program (executable or a shell script) according to its calling sequence. If the syntax of the temporary output file produced by the program is not the one the server requires, the wrapper converts the output file into a secondary output file with the requested format.

An *embedded application* operates only in a specific environment, such as a CAD system or a geometric database. The application expects the data to be preloaded into the environment that the application works in. The environment supplies methods for accessing and modifying the geometric data, so that the application usually needn't be aware of the nature of the actual data structures that store these data. The output is the contents of the environment upon termination of the application. To invoke an embedded application, the wrappers first set the environment. This means they initialize the environment, parse the temporary input file, and load its contents into the environment. The wrappers then invoke the application according to its specifications. To unset the environment, the wrappers scan the environment



(a) (b) (c) (d)

Figure 2. Applet interface for algorithms in two dimensions: (a) point set, (b) convex hull, (c) Voronoi diagram, (d) Delaunay triangulation.

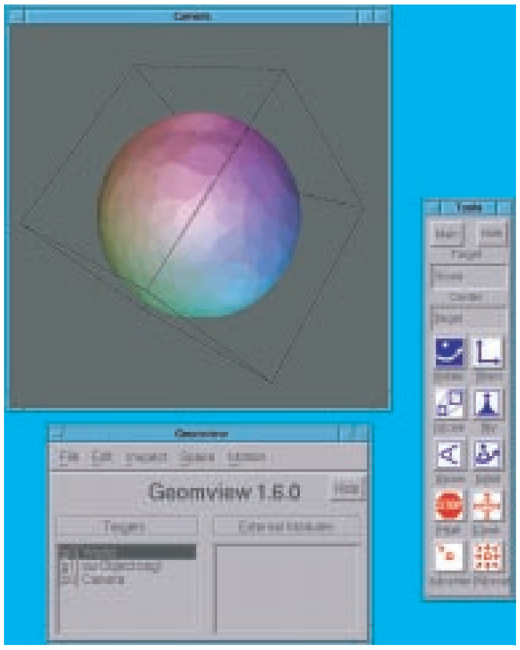


Figure 3. A three-dimensional display plugged into GeomNet.

and save its contents into a temporary output file with the syntax requested by the server. Lastly, they properly close the environment.

The user specifies an output syntax but not an output filename. GeomNet returns the output in a temporary file with the required syntax. The user may then choose a file to save the data into. Finally, GeomNet removes the temporary output file automatically.

INTERFACES AND APPLICATIONS

GeomNet is representative of systems that enable distributed computing over the Internet in the sense that the scheme used in GeomNet can be used to develop similar systems in other disciplines. In this section, we give several examples of how our system can be applied in the field of geometry.

Classic Geometric Algorithms

A prime application for GeomNet is invoking implementations of fundamental geometric algorithms. Therefore the current GeomNet release includes implementations of several algorithms, such as *d*-dimensional convex hulls (including a random-box generator for code testing purposes), Voronoi diagrams, and Delaunay triangulations.

We developed several interface mechanisms for invoking these routines. The simplest is the interactive applet interface. Figure 2a shows a Java point-set editor embedded in GeomNet.

The purpose of the interactive applet interface is clearly not to perform intricate large-scale computations but rather to interact with the system so as to gain intuition about geometric algorithms and to easily test GeomNet servers. The user can then apply any algorithm for which an appropriate application is bound to the system. Figures 2b, 2c, and 2d show the convex hull, the Voronoi diagram, and the Delaunay triangulation, respectively, of the point set shown in Figure 2a, as computed by the appropriate applications and displayed by GeomNet's applet. Some operations don't require graphical input, only parameter settings. For these operations GeomNet provides a general form for obtaining user input.

For production-level usage we provide a socket interface that can be invoked directly from inside an application program running on the client's site. This interface lets any program, not just Java applications, communicate with the server via the

GeomNet protocol. We also provide a forms interface for downloading geometric data files. In both cases the server output can be sent immediately to an interactive display system running on the client, or it can be piped into a client application (including the calling application). Figure 3 shows such a response, the result of a client application that first requested GeomNet to generate a random sample of points in R^3 and then returned that set of points for GeomNet to compute its convex hull.

GeomNet's response was then sent immediately to a client invocation of the Geomview system. Here Geomview was used as a plug-in for the client's application. However, it could also have been used as a Netscape plug-in for someone using the forms interface or the applet editor.

Users may choose three-dimensional browsers to serve as plug-ins to the Web browser (when opening the GeomNet page) or as plug-ins to the user's client (if the socket interface is used). Often ,users can request that a GeomNet application's output be specified in a format suitable to the user's plug-ins or helper applications.

Drawing Abstract Graphs

Graph drawing is an evolving research area. The objective is to produce a graphical representation from a set of objects (nodes) and connections between them (edges). A graph is an abstract entity that has many possible drawings—deciding which drawing is of the highest "quality" is rather subjective and depends on the type of graph and the application. Common factors for determining quality include minimizing the number of edge intersections or the total area of the picture.

The Graph Drawing Server, a GeomNet component, can be used for drawing graphs from user applications, studying and comparing graph-drawing algorithms, converting between different formats for describing graphs and their drawings, creating a database of graphs occurring in user applications, and providing educational demonstrations.

The user must specify the graph style, the input format, the type of service desired (drawing or format conversion), the output format, and the drawing algorithm (if the graph is to be drawn). To provide the input graph in HTML form, the user can

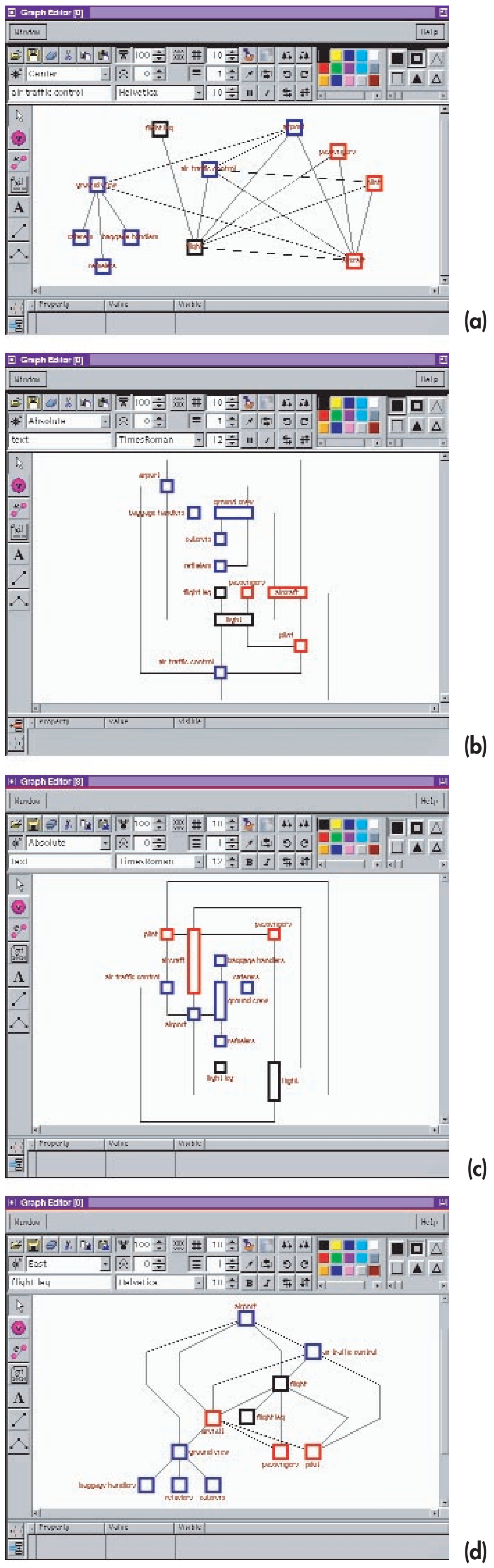
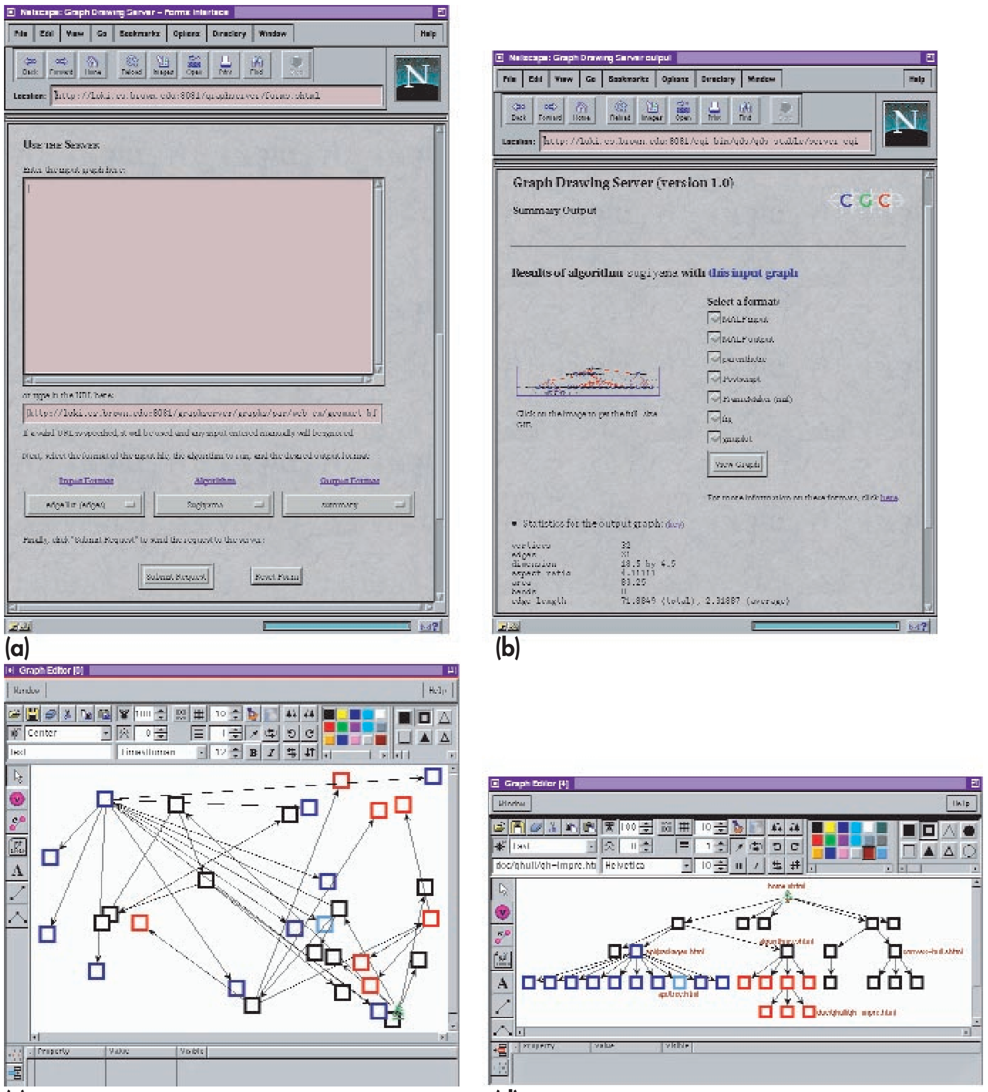


Figure 4. Some graph-drawing examples: (a) the flights database, (b) the Bend-Stretch algorithm, (c) the Giotto algorithm, (d) the Sugiyama algorithm.



Use the Server

Enter the input graph here:

or type in the URL here:

http://loki.cs.brown.edu:8081/graphserver/graphs/par/web_cs/geomnet.hf

If a valid URL is specified, it will be used and any input entered manually will be ignored.

Next, select the format of the input file, the algorithm to run, and the desired output format:

Input Formats	Algorithms	Output Formats
edgelist (edges)	Sugiyama	summary

Finally, click "Submit Request" to send the request to the server:

(a)

Graph Drawing Server (version 1.0)

Summary Output

Results of algorithm sugiyama with this input graph

Click on the image to get the full size GIF.

Select a format:
- GML input
- GML output
- postscript
- Postscript
- FrameMaker (mif)
- fig
- xgraph

For more information on these formats, click here.

- Statistics for the output graph: (key)

```
vertices          32
edges             31
dimension         18.5 by 4.5
aspect ratio      4.11111
area              83.25
bends             0
edge length       71.8849 (total), 2.31887 (average)
```

(b)

(c)

(d)

Figure 5. GeomNet's Web page hierarchy: (a) input form, (b) textual output, (c) graphical input, and (d) graphical output.

either give the URL of the file containing the graph or simply type the graph description in the form itself. The graph editor applet lets the user interactively draw the input graph on the screen. GeomNet currently supports four orthogonal-drawing algorithms for general graphs, as well as several algorithms for drawing specialized graphs. A prototype version of the Graph Drawing Server can forward requests as needed to other servers.

Figure 4 compares the performance of several graph-drawing algorithms on a diagram taken from a flights database.

Figure 5 shows the application of a graph-drawing algorithm on the hierarchy of GeomNet's Web pages. The user specifies the URL of the hierarchy description in the input form (Figure 5a). Figure 5b shows the textual output of invoking the Sugiyama hierarchical graph-drawing algorithm. The output page provides a thumbnail image of the drawing (linked to the full-size image) and some statistics on the output, as well as convenient links for obtaining the output in a different format. Figure 5c shows the same input entered in the graph-editor applet, and Figure 5d shows the graphical output. The graphical output is used, for example, to demonstrate how many links the GeomNet user must follow to find a particular page.

Geometric Algorithm Animation

Animation can be used as a tool to understand an algorithm's operation. We are incorporating geometric algorithm animation capabilities into GeomNet by extending previous work on the Mocha model for Web-based algorithm animation.[2] In the Mocha model, algorithm animation consists of two components: the algorithm server, which executes the algorithm and produces a sequence of interesting events, called algorithm operations, and the animation component, which provides the multimedia visualization of the algorithm operations. The animation component can be further divided into the GUI, which handles the user's interaction with the interface and sends changes in the input to the algorithm server, and the animator, which receives algorithm operations from the algorithm server and updates the display accordingly.

Currently supported are animations of the convex hull, Voronoi diagram, and Delaunay triangulation algorithms provided by the Library of Efficient Data Types and Algorithms (LEDA), and the computation of a proximity graph of a set of points.

Experimental Results

We ran our geometric algorithms server on a Pentium Pro 200-MHz machine using Linux, and measured the running time of the algorithms for two- and three-dimensional Delauney triangulations and three-dimensional convex hulls. For a reasonably sized input (tens of thousands of points) the system provided the output in less than 30 seconds (plus transportation time, which varied).

We have collected statistics on the performance of the graph-drawing server, which was run on a Sun UltraSparc workstation using Solaris 2.5/2.6. For graphs with up to 100 nodes and 140 edges, the typical running time of a drawing algorithm was less than 10 seconds, with less time spent on input and output format conversions and on other system overhead. A little more time was needed for conversion if the output format was graphical (for example, a GIF or PostScript file).

FUTURE WORK

Future GeomNet development will incorporate new types of servers, upgrade existing servers to control more algorithms, and expand the online help, which is now only partially available. These enhancements will make GeomNet functionality richer and appropriate to more research domains. Its modular structure makes these tasks simple.

A super-action mechanism, now under development, will give the user a scripting language for conditioning, pipelining, and iterating regular actions. This feature will enable further automation of complex sequences of operations performed on geometric objects.

We also plan to enhance the system to utilize distributed computing where appropriate. This means implementing a cluster of servers that can assign tasks or subtasks to one another. In a homogeneous environment, where most machines running the servers have the same resources, a task should be solved in parallel, to the extent that the underlying problem is parallelizable. Conversely, if the machines' computing capabilities differ, then each server should handle the problems it is best suited for.

We also intend to explore the possibility of downloading and running an application on the user's site, thus reducing the load on the server. This requires the application author's consent to Internet distribution of the application. Also, the user's machine would need the code type and machine resources to run the application.

Once the system supports a cluster of servers, users can add their own applications by starting a server locally at their site. They can also submit applications to a server's owner for inclusion, although the current server assumes applications are trusted and protects only against crashes and infinite loops. Additional security measures are needed to protect the server from applications that may cause accidental or malicious damage.

Finally, we also plan to implement a mechanism (a metalanguage) that will let the user define the input and output format for the transferred data. This would remove the limitation that the user submit data in one of the formats the system recognizes. ■

ACKNOWLEDGMENTS

We thank Yair Amir, Jim Baker, Ashim Garg, and Subodh Kumar for several useful discussions regarding geometric software and distributed computing.

REFERENCES

1. T. Munzner, S. Levy, and M. Philips, "Geomview: A System for Geometric Visualization," *Proc. 11th Ann. ACM Symp. Computational Geometry*, ACM Press, New York, 1995, pp. C12-13.
2. J.E. Baker et al., "The Mocha Algorithm Animation System," *Proc. Int'l Workshop Advanced Visual Interfaces*, 1996, pp. 248-250.

Gill Barequet is a senior lecturer at the Faculty of Computer Science of the Technion, the Israel Institute of Technology. His research interests include discrete and computational geometry; interpolation and reconstruction algorithms; geometric software and computing over the Internet; and geometric applications in CAD, medical imaging, and molecular biology. Barequet received a BSc in mathematics and computer science and an MSc and a PhD in computer science from Tel Aviv University in 1985, 1987, and 1994, respectively.

Christian A. Duncan is pursuing a PhD in the Department of Computer Science at Johns Hopkins University. His research interests include computational geometry, computational metrology, approximation algorithms, and computer graphics. He received a BS in computer science and mathematical sciences and an MSE in computer science from Johns Hopkins University in 1994.

Michael T. Goodrich is a professor in the Department of Computer Science and codirector of the Center for Geometric Computing at Johns Hopkins University. His research focuses on the design of high-performance methods for solving geometric and combinatoric problems in computer vision, computer graphics, astronomy, and scientific data analysis. He received a PhD in computer science from Purdue University in 1987. He is a senior member of the IEEE and a member of the IEEE Computer Society, ACM, and SIAM.

Stina S. Bridgeman is a PhD candidate in computer science at Brown University. Her research interests include graph drawing and computational geometry. She received a BA in computer science from Williams College in 1995.

Roberto Tamassia is a professor in the Department of Computer Science at Brown University. His research interests include analysis, design, and implementation of algorithms; computational geometry; graph algorithms; data structures; graph drawing; and information visualization. He received a PhD in electrical and computer engineering from the University of Illinois at Urbana-Champaign in 1988.

Readers can contact Barequet at the Israel Institute of Technology; barequet@cs.Technion.AC.IL.