GeoLab: An Environment for Development of Algorithms in Computational Geometry

P. J. de Rezende* W. R. Jacometti†

Departamento de Ciência da Computação
IMECC — C.P. 6065
Universidade Estadual de Campinas
13081-970 Campinas SP Brazil

Abstract

We describe a Computational **Geo**metry **Lab**oratory which we developed as a programming environment for implementation, testing and animation of geometric algorithms. **GeoLab** was conceived to be a tool for the working researcher or a group of them, since at its root lie the use of shared libraries of algorithms and an incremental approach to aggregating new types of geometric objects, data structures and extensions accessed through dynamic linking.

1 Introduction

Programming environments directed towards problems of specific fields have been a way of coherently organizing a set of tools needed in those fields. Such is also the case in Computational Geometry.

The need for programming environments for the development of geometric algorithms has been evident for more than a decade. The intricacy of many solutions to problems of geometric nature calls for the compilation of several resources into a single programming environment.

1.1 Related Work

Earlier attempts to achieve this goal include XYZ GeoBench [Sch90] and the Workbench for Computational Geometry [KMM+90]. The former was implemented in Object Pascal

*Supported in part by CNPq Grants 300157/90-8 and 500787/91-3, and a Fapesp Grant.
†Supported in part by CNPq Grant 130767/90-5 and Fapesp Grant 91/4497-6.

and runs on Macintosh computers while the latter was written in SmallTalk also for Macintoshes. Other related works are Ericson and Yap's LineTool [EY88] and Mehlhorn and Näher's LEDA [MN89]. Furthermore, the usefulness of algorithm animation for better understanding of the behavior of algorithms and as a teaching tool has been recognized for a number of years. Brown's Balsa [Bro87], and more recently Zeus [Bro92], Stasko's Xtango [Sta90] and Amorim and de Rezende's AnimA [AdR] are witnesses to this fact.

1.2 The Geometry Laboratory: GeoLab

GeoLab binds together support for software and algorithm development and support for real time interaction. Essentially, these consist of:

Support for development:

- Built-in abstract data types for representation of geometric (and other) entities;
- Data structures and general purpose algorithms;
- Basic algorithms and some complex geometric data structures (as building blocks);
- Mechanisms for creation and incorporation of new components (algorithms, interactive modes, data structures and *even* data types) *without* requiring recompilation of the environment;
- Tools to allow the production of reusable shared libraries;
- Methods to convert between data representations, and one core (yet powerful) data structure (half-edge) to simplify the task of conversions and of external storage.

Support for interaction:

- An interactive graphical environment for the manipulation of geometric models;
- Facilities for the construction of sets of input data to test average performance and degenerate cases;
- Support for debugging and obtaining statistical data on run-time performance;
- Tools for handling secondary storage;
- Customization of algorithm animation (levels, speed, and other attributes).

GeoLab is written in C++ and makes extensive use of object oriented programming for a hierarchical modeling of geometric objects *and* of geometric algorithms. It currently has about 40 algorithms in its dynamically linked libraries which are made up of roughly 45000 lines of C++ code (plus an additional 50000 lines for the environment itself). It runs on SparcStations under Sun/OS using the XView graphics library, following the OpenLook graphical user interface guidelines.

2 Presenting GeoLab

The environment consists of a central kernel which requires no changes in order for one to include new components to the environment, regardless of whether they are new external modes, algorithms or *even* data types. *All* these are aggregated to the system by means of shared libraries.

We should emphasize here that the kernel itself contains *no* geometric algorithms whatsoever; they are *all* externally provided as part of dynamically linked shared libraries.

This keeps the system as small as possible which is a significant plus when one has several dozens of algorithms incorporated. That way, memory consumption by the environment is kept to a bare minimum, leaving as much memory as possible to the algorithms themselves. By virtually eliminating the need to swap portions of the system off to disk (even when only 8Mbytes of RAM is available), measurements of CPU time for the sake of run time comparisons are much more reliable.

2.1 Graphical Interface

GeoLab is based on a graphical representation of the geometric objects on which it operates. This is specially useful since the visual outcome greatly helps the understanding and debugging of algorithms and manipulation of geometric data.

The interface consists of an Editing Area onto which objects can be manually or automatically generated. From the palette of Operation Modes, the user can pick a tool to create, select, move, reshape, zoom or scroll the objects on the Editing Area, while from the various menus, the user can choose a variety of actions.

Once objects have been created (in the editing area), the user may select some or all of them. At this point, the system consults each algorithm available on whether it can handle the current selection. Only those able to handle the selected objects will the enabled on the Algorithms Menu. By selecting an algorithm from this menu, the user requests the system to pass the selection as argument to the algorithm. The result, whenever it is a set of geometric objects with graphical representation (see section), is then placed on the editing area with the current default graphical attributes.

Additionally, two user-chosen externally implemented modes can be installed in the palette of operation modes at any given time. One of these is an Interactive Creation mode and the other is a Functional Mode. These can be just about anything, such as an interactive simple polygon creation tool, or a query type shortest path (among obstacles) mode.

2.1.1 Input Data Generation

One of the difficulties that appears in testing geometric algorithms is the generation of *interesting* test data. While the editor contains tools for that purpose, some automatic means to build either large or very particular sets of data are often desirable.

A few input generation functions are provided within the environment to ease this task. There are generators of pseudo-random convex, star shaped, simple and general polygons as well as generators of random point sets satisfying specified restrictions (bounded areas or contours).

2.1.2 Animation Modes

Two animation modes have been incorporated into **GeoLab** (see [dRJ93a]). The first one is called *dynamic move*. It basically animates the geometric objects produced as output by any given algorithm as the input undergoes changes in real time conducted interactively by the user manipulating the mouse (watch [dRJ93b]). No changes in the code of the algorithms are required for this mode to operate. It serves well the purpose of illustrating the relationships between input data and output geometric constructs, which, for teaching computational geometry is of great value.

The second mode (see figure 1), which truly characterizes as algorithm animation, requires the inclusion of code for graphical display of geometric actions taken by the algorithms. In order to facilitate this task, a library of graphical routines, called **GeoLab** Animation Toolkit, is provided so that programmers need not recourse to the lower level Xlib functions. However, any such code, being conditionally compiled, does not compromise portability, so that even animated algorithms can be ported unchanged for use in contexts other than within **GeoLab**.

2.2 Programming Support

Together with the graphical interface, the environment contains a large set of tools intended to help in the process of construction of geometric algorithms and abstract data types (ADTs) — for representation of geometric and non-geometric objects. Our approach consists of providing an ample set of extensible C++ classes, hierarchically organized.

2.2.1 Object Oriented Programming

GeoLab provides in a simple and efficient way a number of geometric ADTs. Several of these are logically derived from others which makes the inheritance mechanisms very desirable since it eases code maintenance.

Additionally, we decided to approach the construction of geometric algorithms also hierarchically since algorithms in Computational Geometry can be organized according to classes which share the same characteristics (e.g. regarding the paradigms which they employ: plane-sweep, divide-and-conquer, etc.).

2.2.2 Visualization System

In order to keep **GeoLab** as independent as possible of the specific resources of the host, we created an effective abstraction of many such resources within a visualization system called World which defines all the tools for graphical representation of geometric objects. This system offers:

- the use of real (homogeneous) coordinates and orientation of the coordinate plane;
- the manipulation of graphical elements not supported by traditional toolkits (e.g. rays, points at infinity, etc.);
- the control of dimensions and position of the virtual plane of visualization, allowing for the operations of Zoom and Scroll to be easily done;
- the treatment of external events through filters (see [Jac92]);
- the isolation of the machine dependent code with respect to the graphical input and output, facilitating portability;
- integration of the windows management package XView with C++.

2.2.3 Geometric Objects

The approach used for the implementation of the many geometric objects handled by **GeoLab** consists of the creation of a double hierarchy of classes – pure objects and graphics objects. This is of fundamental importance in questions such as *generic libraries*, uniformity of the treatment of objects by the editor, and savings of space in derived and/or composed representations.

The purpose of a double hierarchy of geometric objects is to separate data and essential methods of data (e.g. the coordinates of vertices of an object) from the methods used by **GeoLab** itself (e.g. color, label, thickness of lines, etc.).

For each pure geometric object (e.g. Point2D), there is a graphical geometric counterpart. While the pure objects implement data and methods specific to their geometry, graphical objects implement the set of data and methods which define the protocol of interaction used by the environment to manipulate them. This dual functionality allows the creation of composed objects without the undesirable duplication of information. E.g., a line segment is comprised of a *pair of* pure points and *only one* graphical object (of type Segment). The savings on memory and code to manipulate such objects is an immediate consequence of this approach.

In addition, it allows for the implementation of geometric algorithms which are essentially independent of the environment since they are restricted to manipulating pure objects.

When the user activates an algorithm to act upon the *selected objects*, a Dispatcher sorts out the pure (geometric) objects' informations from the graphic objects' informations and passes just the geometric data to the algorithm. The Dispatcher is also responsible for adding graphical information to the (pure) objects returned by the algorithm. This mediator is essentially a virtual constructor which hides the environment from the external algorithms.

In order to avoid an explosion of the number of different classes, we recourse to a mechanism which we call dynamic characterization of objects. In a language which permits type migration, this mechanism allows for many class derivations to be avoided. We take advantage of this by making each object able to inform its geometric type and subtype (e.g. type polygon; subtype convex).

Further details about the programming support, built-in types, classes, methods, messages, protocols and geometric primitives can be found in [Jac92].

3 Programming in the **GeoLab** Environment

While the previous section concentrated on a description of the kernel of **GeoLab**, here we confine ourselves to describing the techniques and concepts which were incorporated in order to allow the incremental growth of the environment. There are four fronts where this growth can take place: algorithms, geometric objects, interactive creation tools and functional modes, none of which require any changes to the kernel itself. In this way, we are able to keep separate the stable kernel from the components under development.

The most immediate consequence is the fact that several users can work concurrently and independently in the development of external modules, shaping the set of tools they use in accordance with their particular needs.

This is possible due to the mechanism called dynamic linking available on the host system which enables linking of shared libraries at run-time. These libraries constitute the exportable components of **GeoLab**.

In this context, a geometric algorithm is an entity which perform processing on geometric objects and produce as results geometric objects. In principle, **GeoLab** is unaware of any geometric algorithm since they are *all* external to the kernel. In fact, the kernel only knows about a protocol through which it communicates with the algorithms indicated by the user.

Rather than implementing *all* geometric algorithms as methods of classes, as is done in other geometric workbenches, we draw a distinction between those which are well fit to be methods of classes and those which are best regarded as external applications. Essentially, the separation is based on whether the algorithm depends strongly on the internal representation of the geometric objects it acts on or not. A few of the positive characteristics of this approach are:

- Semantic Organization

  Some algorithms are able to handle objects of several unrelated classes;

- Independence of Representations

  When an algorithm is independent of a representation, to implement it as a method of particular class restricts its use with different representations of the same object which share the same repertoire of methods;

- Restrictions on the Dynamic Characterization

  As said in section , it is not always desirable to introduce new geometric objects through class derivation. When it is more convenient to use dynamic characterization, geometric algorithms implemented as methods of those classes require extra code to recognize this form of characterization.

3.1 Dynamic Linking Specification

Given that at one point there may be hundreds of algorithms in the shared libraries of external components a (very simple) mechanism was developed for the user to specify which ones are to be dynamically linked. This specification is done in textual form in a file (called .geolab-menu) which essentially describes three items per line: a string to appear on the menu, the name of the algorithm (function) to be called and the corresponding module where it lies. This file is written according to a (small) control language and it is read by a parser built into the kernel.

Once linked, these algorithms are manipulated by means of the protocol referred to above, which require the algorithms to be able to:

1. inform the kernel whether they contain animation code;

2. inform the kernel whether they are able to handle the current selection;

3. create (empty) instances of the *external objects* that they build so that the kernel can load externally stored data files of unknown object types;

4. organize input data from the list of selected pure objects passed as arguments;

5. organize output data produced by the algorithms so they can be passed to the kernel to be incorporated into the list of current objects and be displayed in the editing area;

6. cleanup after themselves when large intermediate structures are created (specially by complex external functional modes).

3.2 Conversions between Representations

One of the difficulties in programming items 4. and 5. above is the need for conversion between representations of geometric objects. Most of the algorithms rely on the use of very specific data structures in order to assure efficiency. Conversions between representations become inevitable once we realize that the output of an algorithm can become the input for a number of other algorithms. In order to avoid having a number of converters quadratic on the number of representations, we established a core structure — the **half-edge**, to and from which virtually all representations dealt with to this date can be converted. In this way, the number of converters needed is linear on the number of different representations.

Since a half-edge can easily be saved textually by means of the inverse of the (concise set of) Euler operators required to destroy it, and once a converter of a new structure to and from a half-edge is written, we have, as a very useful side effect of choosing the half-edge to be the core structure, that saving data represented in the form of this new structure to external storage becomes trivial.

4 Other Features

The ability to include new algorithms and geometric objects together with the abstraction of a visualization system make possible the definition of two special classes of geometric algorithms whose purpose is to make the interaction modes of the editor capable of expansion by external tools.

New interactive modes are installed dynamically onto two multi-purpose buttons on the interface. These give the user not just the ability of creation and manipulation of geometric objects on the editing area in new ways but also the design and construction of geometric algorithms that require interaction (e.g. query mode range search, point location, etc.).

5 Algorithms that have been implemented

Some 40 algorithms have been implemented so far in shared libraries that can be accessed by **GeoLab**. Rather than list all of them here, we refer the reader to the more structured display shown in figure 2.

Most of these algorithms are animated and a sample of those animations can be watched in [dRJ93b].

6 Conclusion

We have presented a concise description of some of the main features of **GeoLab** which we view as a **Geo**metric **Lab**oratory in Software where one (or many) can experiment with geometric algorithms, data structures and animations.

A continuously growing library of (dozens of) such algorithms has been used as research and teaching tools at the Universidade Estadual de Campinas. The environment will be made available for `ftp` for educational and research purposes.

References

[AdR] R. V. Amorim and P. J. de Rezende. AnimA – An Environment for Algorithm Animation. To appear.

[Bro87] M. H. Brown. *Algorithm Animation.* ACM Distinguished Dissertations Series. MIT Press, Cambridge, MA, 1987.

[Bro92] M. H. Brown. Zeus: A system for algorithm animation and multi-view editing. Technical Report 75, Digital Systems Research Center, Palo Alto, CA, February 1992.

[dRJ93a] P. J. de Rezende and W. R. Jacometti. Animation of geometric algorithms using GeoLab. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, 1993.

[dRJ93b] P. J. de Rezende and W. R. Jacometti. Video: Animation of geometric algorithms using GeoLab. In *Video Review of the 9th Annu. ACM Sympos. Comput. Geom.*, 1993.

[EY88] L. W. Ericson and C. K. Yap. The design of LINETOOL, a geometric editor. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 83–92, 1988.

[Jac92] W. R. Jacometti. GeoLab – um ambiente para desenvolvimento de algoritmos em geometria computacional. Master's thesis, DCC - IMECC - UNICAMP, 1992.

[KMM+90] A. Knight, J. May, M. McAffer, T. Nguyen, and J.-R. Sack. A computational geometry workbench. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, page 370, 1990.

[MN89] K. Mehlhorn and S. Näher. LEDA - A Library of Efficient Data Types and Algorithms. *Lecture Notes in Computer Science*, 379:88–106, 1989.

[Sch90] P. Schorn. An object-oriented workbench for experimental geometric computation. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 172–175, 1990.

[Sta90] John T. Stasko. TANGO: A framework and system for algorithm animation. *Computer*, 23(9):27–39, September 1990.

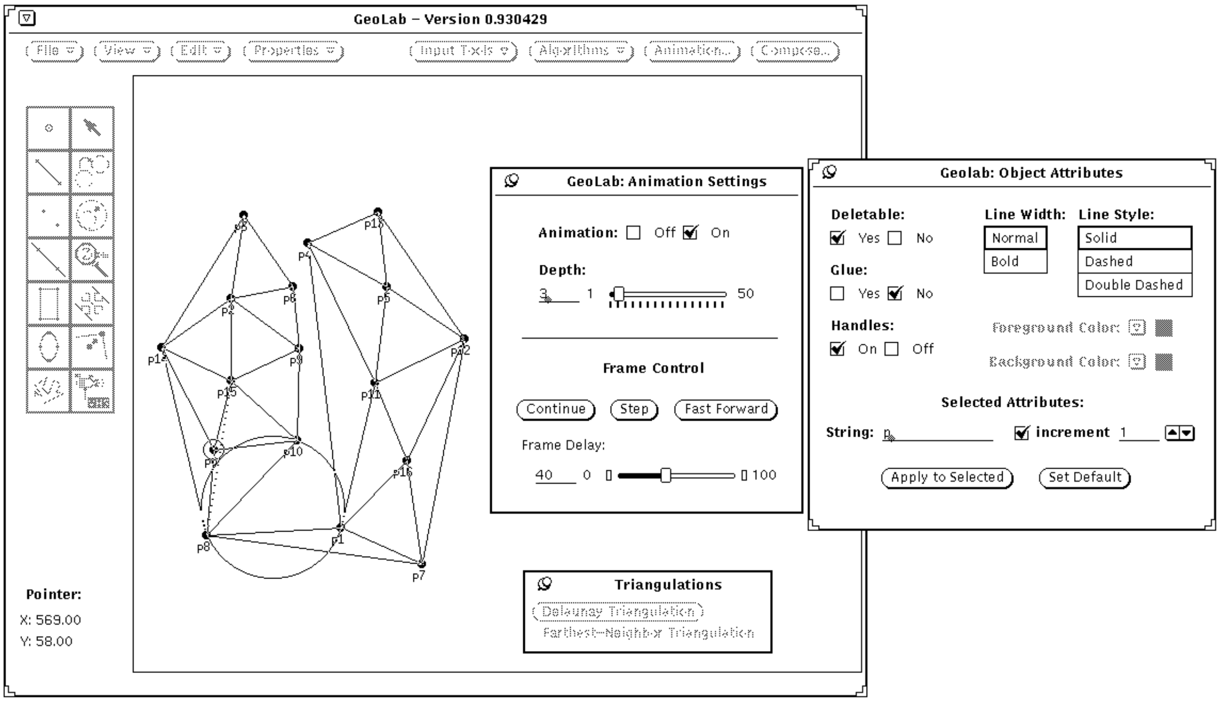

Figure 1: Animation of a Divide-and-Conquer Algorithm for Constructing a Delaunay Triangulation

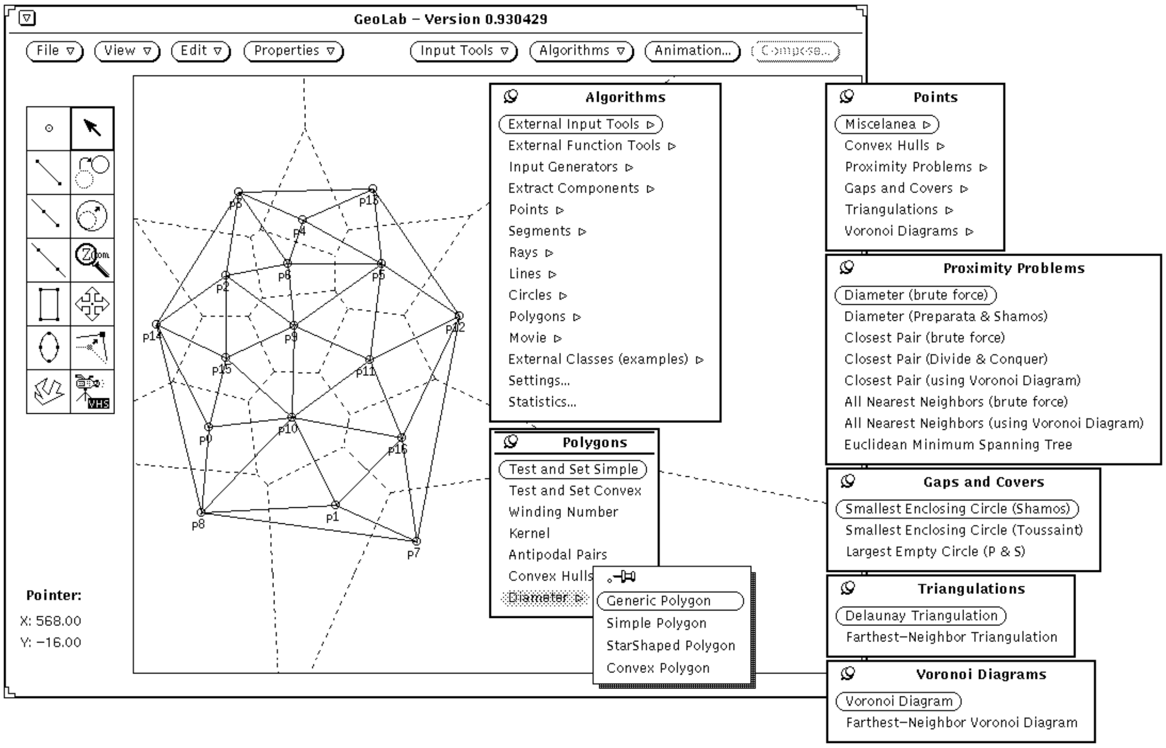

Figure 2: A view of **GeoLab**'s interface and of many of the choices from the Algorithms Menu