A Geometric Workbench for Degree-Driven Algorithm Design

Jack Snoeyink * Clinton Freeman *

Abstract. We recall the design and implementation of a geometric algorithm workbench for implementing and presenting degree-driven geometric algorithms.

1 Introduction

Two and three dimensional geometric algorithms are often difficult to implement and convey to others. Algorithm *implementers* need to correct programming errors and ensure that degenerate situations are handled correctly. Traditional debuggers provide only textual or numerical representations of geometric data structures, and generating degenerate geometric input is a nontrivial task for which there is often little recourse. Algorithm *presenters* need to convey their ideas to audiences of researchers and students. Many presenters tend to use static depictions with verbal explication of algorithm mechanics. This type of presentation does not fully capture the dynamic nature of algorithms, and can be difficult for the audience to follow.

A *geometric algorithm workbench* aids algorithm implementers and presenters by providing facilities to dynamically visualize geometric algorithms. Implementers can visually inspect geometric relationships and properties of data structures, enabling quick recognition of erroneous computations. Presenters can produce animations of their algorithms, affording a clearer means of conveying essential ideas to their audience. Both types of geometers can interactively control the flow of execution and easily generate or visually specify degenerate input data.

Degree-driven algorithm design encourages robust geometric computing by minimizing an algorithm's arithmetic precision with its running time and space [5]. Millman built a C++ library (`DDAD`) to facilitate the implementation of these algorithms; our aim is to build a workbench to support users of `DDAD`. The creation of this workbench mostly requires the application of techniques developed in previous software visualization research, but the addition of precision as a constraint provides for new avenues of exploration.

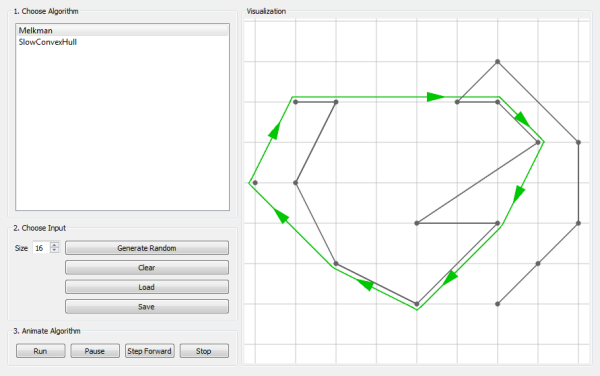

Figure 1: A user watches as Melkman's algorithm handles a degenerate situation.

We begin by briefly reviewing previous workbench systems. Next, we explain how our current system's facilities satisfy each user type and significant decisions we made during their implementation. Finally, we conclude with a discussion of how we can extend our system moving forward.

2 Previous Work

Initially, we spent some time reviewing existing software visualization systems to see if any might help us implement and present our algorithms. In 1997, Dobkin and Hausner [4] reviewed four geometric visualization systems: Workbench, XYZ GeoBench, Geomview, and GASP. Unfortunately, while these systems presented different ways of solving foundational challenges, support has long been discontinued. We also considered the Geometry Center's GeoLab [2] and Stasko's [7] more general algorithm animation software such as POLKA, SAMBA, and XTANGO, and found them similarly unsupported. Lacking a working geometric workbench, we decided to build our own system.

3 User Facilities

Two simple 2D convex hull algorithms, `SlowConvexHull` [1] and Melkman's algorithm [6], provided

*Department of Computer Science, University of North Carolina at Chapel Hill. Email: {snoeyink,freeman}@cs.unc.edu

the initial target inputs for the system. For each algorithm, we first programmed an implementation, then recorded animations of them running on example input data to produce a corresponding Youtube video [1]. This development process placed us in the position of both implementer and presenter, and led us to develop facilities appropriate for both user types.

As implementers, we desired a system with four major capabilities. First, we needed a means of manipulating input data into degenerate situations to test that special cases were handled correctly. Second, we needed to run the algorithm and visually display the results of final and intermediate calculations. Third, upon discovering an incorrect result, we needed to single step the algorithm from the beginning on the same input data in order to see where the algorithm went awry. Finally, we needed visualization code to minimally invade our implementation code.

In response, our system provides four corresponding facilities. First, our system randomly generates either a random point set or simple polyline, and clicking and dragging moves vertices into different configurations. Second, our system maintains display lists which are updated as *interesting events* [3] occur. Third, our system uses threading to control the speed of execution and provides UI controls for starting, pausing, single stepping, and resetting the algorithm. Finally, our system embeds low level visualization functions in higher level geometric types to maintain code readability and ensure visualization consistency.

As presenters, implementer facilities already satisfied many of our needs, producing animations that captured the essential characteristics of each algorithm. However, we desired two additional capabilities: we needed to convey information not necessary for implementation purposes, and view the same algorithm in different ways. In response, our system provides two corresponding facilities: visualizations of arbitrary primitives that aren't directly used by the algorithm, and a passive *model-view-controller* architecture that can be extended with custom views.

4 Engineering Decisions

Two engineering decisions are of particular interest. First, we wanted to use model-view-controller to structure our design, but needed the traditional model concept to encompass a geometric algorithm. Extracting visual representations of operations and data structures without user guidance is a difficult, if not impossible, task. We decided to maintain a separate visual model of display lists, which the algorithm implicitly updates. Second, we needed to track visual semantics on geometric primitives so previous states could be restored (e.g. a hull segment is invalidated). We decided to store a semantic stack for each primitive; low level visualization functions push and pop new states as the algorithm executes.

5 Conclusion

Implementing a geometric algorithm workbench is a challenging task with a rich set of problems encompassing a variety of disciplines. While we continue extending our system by animating more algorithms, two questions provide opportunities for further exploration. First, how can we extend our workbench to highlight precision as a resource? An answer will help thematically differentiate the project from previous work. Second, given that so many past systems fell into disuse, how can we build our workbench to have better longevity? An answer will help solidify a foundation for future work.

Our workbench continues to grow and has not yet reached a state suitable for distribution to the geometry community. As the underlying design and outward interface stabilize, we will release a version for download [1].

[1] See: `http://cs.unc.edu/~freeman/GAV/`

References

[1] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. 3rd edition, 2008.

[2] P. de Rezende and W. Jacometti. Animation of geometric algorithms using GeoLab. In *Proceedings of the ninth annual symposium on Computational geometry*, SCG '93, pages 401–402, 1993.

[3] C. Demetrescu, I. Finocchi, and J. Stasko. Specifying Algorithm Visualizations: Interesting Events or State Mapping? In *Revised Lectures on Software Visualization, International Seminar*, pages 16–30, 2002.

[4] A. Hausner and D. P. Dobkin. Making Geometry Visible: An Introduction to the Animation of Geometric Algorithms, 1997.

[5] G. Liotta, F. P. Preparata, and R. Tamassia. Robust proximity queries: an illustration of degree-driven algorithm design. In *Proceedings of the thirteenth annual symposium on Computational geometry*, SCG '97, pages 156–165, 1997.

[6] A. Melkman. On-line construction of the convex hull of a simple polyline. *Inf. Process. Lett.*, 25(1):11–12, 1987.

[7] J. Stasko. Software Visualization research at GVU, 2011. `http://www.cc.gatech.edu/gvu/ii/softvis/`.