A Java Applet for the Dynamic Visualization of Voronoi diagrams* (Short Version)

Christian Icking[1] Rolf Klein[2] Peter Köllner[3] Lihong Ma[4]
FernUniversität Hagen, Praktische Informatik VI, D–58084 Hagen, Germany

Summary

We present a Java applet named *VoroGlide* that visualizes how the Voronoi diagram (or the Delaunay triangulation or the convex hull) of n points continuously changes as individual points are dragged across the plane. The algorithm owes its speed to a new point location technique which is both simple and efficient in practice, and to a partial redraw strategy, for a screen dump see Figure 1. *VoroGlide* can be found and tried out at `http://wwwpi6.fernuni-hagen.de/java/anja/`.

Introduction

The Voronoi diagram and its dual, the Delaunay triangulation, are among the most important structures in computational geometry; see e. g. Aurenhammer [2] or Aurenhammer and Klein [3] for surveys. The study of their structural properties belongs to the core of this area.

As drawing a complex Voronoi diagram by pencil is an arduous task—particularly so for students who have not yet studied the algorithms—computer based visualization is very desirable.

There exist, by now, a good number of tools that allow to insert point sites by mouse click and then display their Voronoi diagram. Numerous references[5] can be found in the World Wide Web. With some systems, like *XYZ GeoBench* by Nievergelt et al. [7], *Mocha* by Baker et al.[6], or *GeoLab* by de Rezende and Jacometti [4], the user can even pick a point and move it across the scene. The Voronoi diagram of the altered point set is then computed and displayed, after some delay that depends on the size of the point set.

We think that this feature is extremely helpful in teaching and understanding geometric structures and algorithms. The ideal system should respond *instantaneously* to the user's actions: While dragging a selected site we would like to see, at each point in time, the Voronoi diagram of the point sites that remain fixed and of the moving site in its current position.

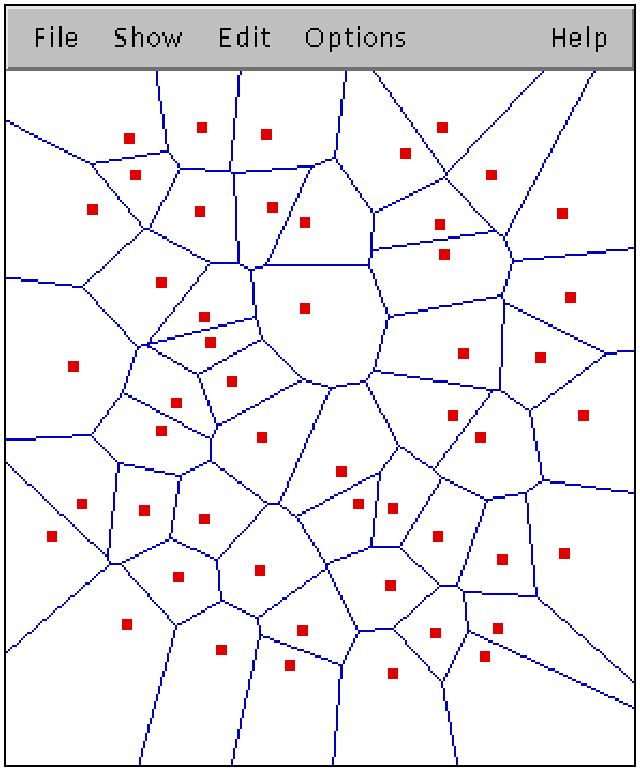

Figure 1: *VoroGlide* showing the Voronoi diagram of 50 points.

Such a perfect dynamic animation, however, is hard to achieve. In practice, all we can hope for is to recompute the Voronoi diagram and update the display frequently enough so that the impression of a continuous motion results. Our experiments indicate that the *frame rate*, i. e. the number of screen updates per second, should not drop below 6 while the mouse is moved at normal speed. Between 10 and 18 frames per second result in a very satisfying animation. At a rate of 25, TV quality is reached. Note that extreme mouse speeds are not likely to create a problem: a slowly mov-

*This work was partially supported by the state NRW.

[1] `christian.icking@fernuni-hagen.de`
[2] `rolf.klein@fernuni-hagen.de`
[3] `peter.koellner@fernuni-hagen.de`
[4] `lihong.ma@fernuni-hagen.de`

[5] For example:
`http://www.geom.umn.edu/software/cglist/`
`http://www.ics.uci.edu/~eppstein/junkyard/software.html`
`http://www.scs.carleton.ca/~csgs/resources/cg.html`
`http://www.cs.duke.edu/~jeffe/compgeom/demos.html`
or the *Software* link in
`http://pine.fernuni-hagen.de/GI/fachgruppe_0.1.2/`

[6] Look at URL:
`http://loki.cs.brown.edu:8080/pages/Mocha.html#Delaunay`

ing mouse can be served all the better, and if the user moves the mouse very fast he cannot expect to follow the structural changes anyway.

The computer's graphical system keeps track of the moving mouse by generating a sequence of *drag events* that report the mouse's current coordinates. These drag events are generated at a rate that depends both on the system and on its current load. For technical reasons, at most 50 drag events per second are generated with a standard equipment, and a typical rate is less or equal to 33, which is largely sufficient.

As often as a new drag event has been created, we should quickly compute and display the current Voronoi diagram. The problem is that we can typically spend at most 30 milliseconds on this task before the next drag event arrives. If the computation is not finished by then, the next events get delayed or must be skipped. As a consequence, the frame rate drops.

Whereas, under a slight drop of the frame rate, the animation looks jerky, like a video running in slow motion, a severe drop destroys the impression of a continuous motion; only a sequence of seemingly unrelated pictures remains.

In our approach, we wanted to make use of the advantages offered by Java; see Arold and Gosling [1] or Flanagan [5], for example. Java programs run virtually everywhere and can be easily accessed by the World Wide Web, the programmer need not give much thought to different run-time environments, and the code is instantaneously executable. On the other hand, Java programs run considerably slower than compiled code. This calls for a careful selection of algorithms and data structures.

On our system, *VoroGlide* achieves a rate of 15 to 20 frames per second[7] on "average" point sets, nearly independently of the number of points.

Two main features of our algorithm are crucial in achieving this rate: a simple technique for localizing a point in a triangulation, which works very well in practice, and an efficient strategy for maintainig the data structure and the display while a point is being dragged. This strategy allows a drag event to be handled in time proportional to the number of Voronoi neighbors the moving point has at its current position.

The algorithm

When a new point site is inserted, we want to incrementally construct the new structure from the old one. To do this with the Delaunay triangulation is technically a lot easier than with the Voronoi diagram. For this reason, our algorithm computes and maintains the Delaunay triangulation as the primary structure. The Voronoi diagram, or those parts that must be updated, are directly derived from the Delaunay triangulation, as well as the convex hull.

[7] The values stated here refer to a low-cost SGI Indy workstation (R4000 at 100 MHz). A machine with a faster processor will perform better. It is even possible to achieve a frame rate equal to the rate of drag events.

Our algorithm is based on the standard procedure of inserting a new point site, p, into an existing Delaunay triangulation DT; see Figure 2 and e. g. Klein [6] or use *VoroGlide* to provide an animation.

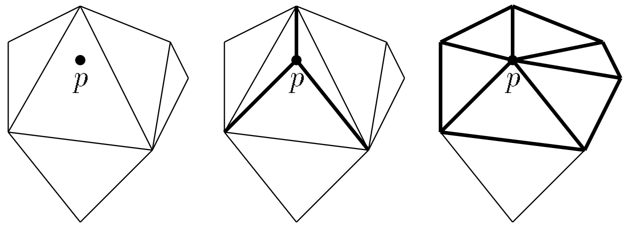

Figure 2: Three steps on inserting a new point into a Delaunay triangulation.

Special attention has been paid to *moving points*. When the user picks a point p in order to move it, we compute from scratch the Delaunay triangulation DT of the remaining $n-1$ points, by using the incremental algorithm. This has to be done only once, at the beginning of each drag operation. Afterwards, only those parts of the Delaunay triangulation are updated and redrawn that have actually changed. The moving point is inserted only temporarily into the structure, such that it can easily be deleted and reinserted at another location.

For locating the point in DT, we are working on a simple but realistic locality assumption: the coordinates of two consecutive drag events are close together, in most cases the moving point is still in the same triangle as before. As long as the current triangle does not contain the point p, we switch to an adjacent triangle which is, in some sense, closer to p; a detailed explanation and a proof can be found in the full paper.

References

[1] K. Arold and J. Gosling. *The Java programming language.* Addison-Wesley, Reading, Mass., 1996.

[2] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23:345–405, 1991.

[3] F. Aurenhammer and R. Klein. Voronoi diagrams. In J. Sack and J. Urrutia, editors, *Handbook on Computational Geometry.* Elsevier, Amsterdam, to appear, 1997.

[4] P. J. de Rezende and W. R. Jacometti. Animation of geometric algorithms using GeoLab. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 401–402, 1993.

[5] D. Flanagan. *Java in a Nutshell.* O'Reilly & Associates, Sebastopol, 1996.

[6] R. Klein. *Algorithmische Geometrie.* Addison-Wesley, Bonn, 1997.

[7] J. Nievergelt, P. Schorn, M. de Lorenzi, C. Ammann, and A. Brüngger. XYZ: A project in experimental geometric computation. In *Computational Geometry — Methods, Algorithms and Applications: Proc. Internat. Workshop Comput. Geom. CG '91*, volume 553 of *Lecture Notes in Computer Science*, pages 171–186. Springer-Verlag, 1991.