Visualizing the Connection Among Convex Hull, Voronoi Diagram and Delaunay Triangulation

John Fisher
Department of Computer Science
Michigan Technological University
Houghton, MI 49931–1295, USA
E-mail: jnfisher@mtu.edu

Abstract

The convex hull, Voronoi diagram and Delaunay triangulation are all essential concepts in computational geometry. Algorithms for solving the convex hull problem are commonly taught in an algorithms course, but the important relationship between convex hulls and the Voronoi diagram/Delaunay triangulation is usually not discussed. This paper presents **Hull2VD**, a visualization tool that illustrates the connection among these three important concepts. We provide a short definition and discussion of each of the three problems and some of the algorithms used to solve them. The important relationship between the three problems is also presented. Finally, we discuss the details of **Hull2VD**, which allows these concepts and their interrelationships to be learned visually in an interactive and easy to understand environment, without the need of complex mathematics.

1 Introduction

The convex hull, Voronoi diagram and Delaunay triangulation are essential problems in computational geometry as well as many other scientific fields. In addition to being important individually, these three concepts are closely related. Many visualization tools exist for the individual problems, such as Paul Chew's Voronoi/Delaunay applet (Chew, nd) and Icking, et al.'s *VoroGlide* (Icking et al., nd). These tools solve the three problems but do not attempt to address the important relationship among them. There are also tools available for providing a step-by-step visualization of the individual algorithms associated with each problem, such as *Mocha* (Baker et al., nd). Algorithm visualization tools are excellent aids in teaching specific algorithms, but do not present how the problems and their solutions are interrelated. The connection between convex hull and Voronoi diagram/Delaunay triangulation is usually not presented because it is considered too complex for most undergraduates, and there is no visualization tool available. Furthermore, textbooks that address these problems are written for beginning graduate courses (de Berg et al., 2000; O'Rourke and Goodman, 1997; Preparata and Shamos, 1985). To be able to understand the connection between these concepts as presented in a computational geometry textbook, students must already have a strong background in geometry.

Hull2VD is a visualization tool that illustrates the connection among these three essential concepts in computational geometry: convex hull, Voronoi diagram and Delaunay triangulation. It is written in C using OpenGL and GLUT, and is currently available on Microsoft Windows, Sun Solaris, Mac OS X and Linux operating systems as part of the DesignMentor v2.0 package. The application was written using Ken Clarkson's QuickHull implementation (Clarkson, 1996). **Hull2VD** displays the Voronoi diagram and Delaunay triangulation of planar point sets as well as the convex hull of a projected point set in space. All displays are updated in real time as the user modifies and adds points using GUI controls. **Hull2VD** is a unique tool that enables instructors to present the important relationship between these concepts visually, without cumbersome mathematics.

This paper contains three main sections. Section 2 presents definitions, historical background and a discussion of solutions for the three problems. Section 3 describes how the problems are related and covers **Hull2VD's** method for computing the Voronoi diagram and Delaunay triangulation in a plane from the convex hull in space. Section 4 provides a description of **Hull2VD**, as well as a discussion of how it is used to illustrate the connection among the three problems.

2 Definitions

2.1 Convex Hull

The convex hull of a set of points is the smallest convex set that contains the points (Fig. 1) (Preparata and Shamos, 1985). It is a fundamental concept in computational geometry, and many solutions have been proposed, such as Graham's Scan (Graham, 1972), Jarvis's March (Jarvis, 1973), and QuickHull (Barber et al., 1996).

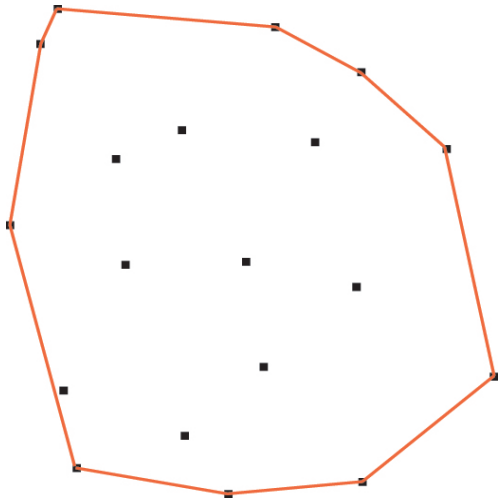

Figure 1: The convex hull of a planar set of points

The convex hull has been used in a multitude of scientific fields (especially computer graphics) for applications such as pattern matching (Soille, 2000), finding bounding volumes (de Berg et al., 2000) and analysis of spectrometry data (Boardman, 1993). Many problems can be reduced to the convex hull problem, such as halfspace intersections, power diagrams, Voronoi diagram and Delaunay triangulation (de Berg et al., 2000).

Students often learn one or more convex hull algorithm in their undergraduate algorithms course because of the many different computational approaches that have been proposed (i.e., gift wrapping, divide and conquer, incremental). **Hull2VD** uses QuickHull, which is an incremental algorithm based on the Beneath-Beyond algorithm (Grünbaum, 1993). The QuickHull algorithm can be generalized and applied to any dimensionality; however in **Hull2VD** we are only concerned with finding the convex hull in three dimensions.

Before any incremental convex hull algorithm can run, a beginning *simplex* of points must be found. In three dimensions, this is a tetrahedron made up of four non-planar points chosen arbitrarily from the input set of points. The convex hull is then constructed as a set of *facets*. In three dimensions a facet is a triangle. A point is *above* a facet if the signed distance from the facet to the point is positive. A facet is considered *visible* to a point p if p is above the facet. The *horizon ridges* of a point are defined as the set of boundary edges of the visible facets (Barber et al., 1996). To compute the convex hull of a set P of n points, $CH(P)$, QuickHull and most other incremental algorithms use the following scheme:

```
1:  initialize sets of facets CH(P) and F to empty
2:  set CH(P) to an arbirtary simplex of points in P
3:  for each p ∈ P
4:      for each facet f ∈ CH(P)
5:          if f is visible to p
6:              add f to F
7:              for each boundary edge e = v1v2 ∈ F
8:                  add new facet (p, v1, v2) to CH(P)
9:                  delete F
10:             end for
11:         end if
12:     end for
13: end for
```

The above incremental approach iterates through all the points p in the input set (line 3). If a point lies outside the current convex hull, it is added to the convex hull $CH(P)$. If no visible facets can be found for a given point, then the point lies within the convex hull, and can be ignored. Each point addition adds a new "cone" of facets from the added point to its horizon ridges. Several algorithms exist that utilize this approach. One such algorithm is the randomized incremental algorithm. Randomized incremental algorithms iterate through the list of points arbitrarily. Another incremental algorithm, QuickHull, selects the furthest point from the current convex hull in each iteration. The run time of QuickHull in three dimensions is $O(n \log r)$, where n is the size of the input point set and r is the number of points that are not ignored by the algorithm. **Hull2VD** computes the convex hull using Ken Clarkson's implementation of QuickHull which is written in C and is freely available for download (Clarkson, 1996).

2.2 Voronoi Diagram

The Voronoi diagram is usually attributed to Dirichlet and is sometimes referred to as the *Dirichlet tessellation* (Dirichlet, 1850). However, Voronoi diagrams can be found in part III of Descarte's *Principia Philosophiae*, published in 1644 (de Berg et al., 2000). Voronoi diagrams have been used in a wide array of scientific fields such as modeling forest dynamics (Mercier and Baujard, 1997), animating lava flows (Stora et al., 1999) and neural network design (Bose and K.Garga, 1993). Because the Voronoi diagram can be applied to so many fields outside of geometric computing and computer graphics, it is advantageous for a student to have an understanding of the basic concepts behind it and its connection with other important problems in geometry. In **Hull2VD** we are concerned only with the Voronoi diagram of a set of points in a plane.

The Voronoi diagram can be defined using the Euclidian distance in a plane between two points p and q, dist(p, q), where $p = (p_x, p_y)$ and $q = (q_x, q_y)$, such that:

$$\text{dist}(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Let P be a set of n distinct points (known as *sites* for the purpose of this definition) in the plane. The Voronoi diagram of P is the subdivision of the plane in n Voronoi cells, one for each site in P, with the property that a point q lies in the cell of site p_i if and only if dist$(q, p_i) <$ dist(q, p_j) for each $p_j \in P$ with $j \neq i$ (Fig. 2a) (Preparata and Shamos, 1985; O'Rourke and Goodman, 1997; de Berg et al., 2000).

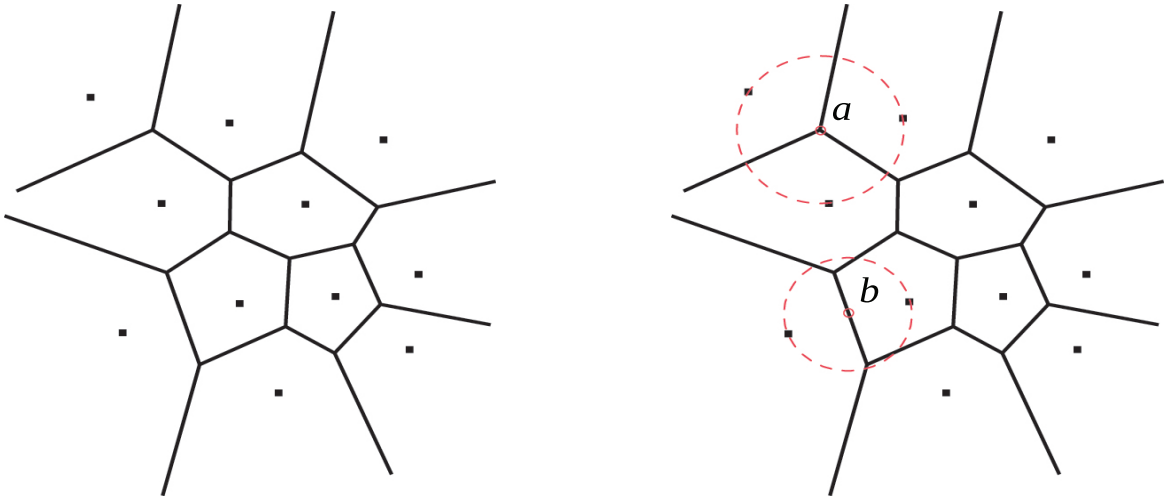

Figure 2: The Voronoi diagram of a planar point set (a) and associated $C_{P(a)}$, $C_{P(b)}$ (b)

We denote $V(P)$ as the Voronoi diagram of P. Each edge in $V(P)$ is a bisector of a pair of two neighboring sites in P, and the vertices in $V(P)$ are the intersection points of these bisectors. The bisectors and intersection points that are included in $V(P)$ can be determined using the following definition. Given a point q its largest empty circle with respect to P, denoted $C_{P(q)}$, is the largest circle with q as its center that does not contain any site in P (de Berg et al., 2000). The following two traits hold:

1. A point q is a vertex in $V(P)$ iff $C_{P(q)}$ contains three or more sites on its boundary.
2. A bisector between sites p_i and p_j is an edge in $V(P)$ iff there is a point q on the bisector such that $C_{P(q)}$ contains both p_i and p_j on its boundary but no other site.

Figure 2b illustrates these two traits. The upper circle, $C_{P(a)}$, is centered on a vertex of $V(P)$, and therefore contains three or more sites in P. $C_{P(b)}$ is centered on a point that lies on an edge in $V(P)$ and correspondingly contains only the two sites that the edge bisects.

Optimal algorithms for constructing the Voronoi diagram of a set of n points in the plane have a runtime bounded by $O(n \log n)$. This can be proven by reducing constructing the Voronoi diagram to sorting a set of n real numbers (Preparata and Shamos, 1985). Many approaches have been proposed to compute the Voronoi diagram in optimal time such as randomized incremental, divide and conquer (Preparata and Shamos, 1985) and sweepline (Fortune, 1987).

2.3 Delaunay Triangulation

Triangulations of a planar point set are extremely important in computer graphics. Triangulation is essential in polygon-based rendering (used by APIs such as OpenGL and Direct3D). Abstract representations of objects, called primitives (such as cones or spheres), must be tessellated into polygons. Usually these polygons are triangles, as triangles have proven to be easy to render using scanline based rendering. Triangulation is used in a multitude of other applications, such as terrain modeling, mesh decimation, and radiosity rendering. The optimal triangulation of a set of points is one that maximizes the minimum angle in each triangle, leading to a set of triangles that are as equilateral as possible. This triangulation will give us the fewest sharp "skinny" triangles, which can cause visual problems in our model.

The Delaunay triangulation was named after the Russian mathematician Boris Nikolaevich Delone, who discovered it in his work on the regular partitioning of space and the theory of Dirichlet partitionings (e.g., Voronoi diagrams) (Delone, 1934). The Delaunay triangulation is optimal; it maximizes the minimum angle over all triangulations of a set of points P. We define the Delaunay triangulation of P, $DT(P)$, as follows:

1. Three points $p_i, p_j, p_k \in P$ are vertices in the same face of the Delaunay triangulation iff the circle through p_i, p_j, p_k contains no other points. This circle is known as the *circumcircle* of the triangle defined by (p_i, p_j, p_k).

2. Two points $p_i, p_j \in P$ form an edge in the Delaunay triangulation iff there is a circle that contains two points p_i, p_j on its boundary and does not contain any other point.

It follows that the circumcircles of all triangles in $DT(P)$ will contain exactly three points in P on their boundaries if and only if no more than three points in P are co-circular. This property is illustrated in **Hull2VD**.

The relation between the Delaunay triangulation $DT(P)$ and Voronoi diagram $V(P)$ can be described using a concept known as the *dual graph*. The dual graph of a planar graph G has a node for each of the face in G and an arc joining two nodes if their corresponding faces share a common edge (O'Rourke and Goodman, 1997). $DT(P)$ is the "straight line" dual graph of $V(P)$. It follows that every edge in $V(P)$ has a corresponding edge in $DT(P)$ and every cell in $V(P)$ has a corresponding point in $DT(P)$ (Fig. 3). It is possible to compute $DT(P)$ from $V(P)$, and vice versa. **Hull2VD** allows the user to see the Delaunay triangulation of P in real time, as well as the circumcircle corresponding to each face in the triangulation.

Many algorithms for computing the Delaunay triangulation exist, such as plane sweep, divide-and-conquer (O'Rourke and Goodman, 1997) and randomized incremental (Seidel, 1991). The randomized incremental approach is of special note because the same method has been applied to calculating the convex hull (Clarkson, 1987). Not surprisingly, the algorithm's runtime is bounded by $O(n \log n)$. This is in agreement with the runtime of the randomized incremental algorithms used in computing the convex hull and Voronoi diagram.

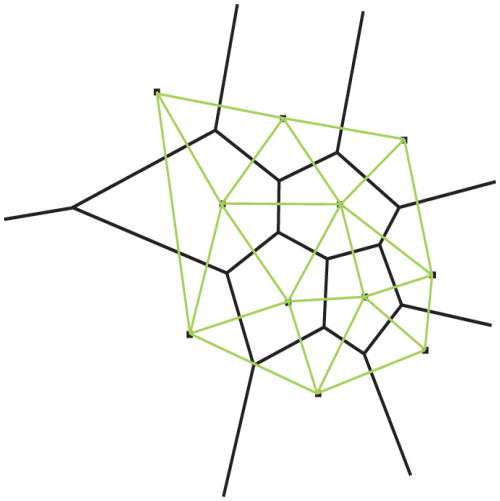

Figure 3: Dual graphs: the Voronoi diagram and Delaunay triangulation

3 Connection

In his 1979 paper K.Q. Brown presented a beautiful connection between the convex hull in space and Voronoi diagram and Delaunay triangulation in a plane (Brown, 1979). This connection shows that it is possible to compute the Delaunay triangulation and Voronoi diagram of a set of points in a plane from the convex hull of a set of points in space. This reduction from the convex hull to the Voronoi diagram and Delaunay triangulation is what **Hull2VD** illustrates. This section contains a short discussion on the connection itself, and a description of how it can be computed.

Given a set P of n points in the plane $z = 0$, we first project them onto the unit elliptic paraboloid $z = x^2 + y^2$ to yield a point set P' such that for each $p = (p_x, p_y) \in P$ a point $p' \in P'$ is computed as follows:

$$p' = (p_x, p_y, p_x^2 + p_y^2)$$

The convex hull $CH(P')$ will contain every $p' \in P'$. The downward-facing facets of $CH(P')$ are those whose normal vectors have a negative z-value. Surprisingly, projecting the edges of the downward-facing facets in $CH(P')$ onto the $z = 0$ plane yields the Delaunay triangulation of P, $DT(P)$! After determining the Delaunay triangulation, it is a simple task to compute the Voronoi diagram. This is done by connecting all circumcircle centers of the triangles in $DT(P)$ that share an edge. Each circumcircle center in $DT(P)$ is a node in $V(P)$. Because $V(P)$ is the dual graph of $DT(P)$ there is a one-to-one correspondence between faces in $DT(P)$ and nodes in $V(P)$ as well as edges in $DT(P)$ and edges in $V(P)$. The procedure **Hull2VD** uses to compute these problems is very similar:

```
1:  initialize P', DT(P) and V(P) to empty
2:  for each p = (p_x, p_y) ∈ P
3:      add p' = (p_x, p_y, p_x^2 + p_y^2) to P'
4:  end for
5:  compute CH(P')
6:  for each facet f ∈ CH(P')
7:      if f's normal faces downwards
8:          for each edge e of f
9:              set z-values of each vertex in e to 0
10:             construct new edge e' = e
11:             add e' to DT(P)
12:         end for
13:     end if
14: end for
15: for each triangle t ∈ DT(P)
16:     for each triangle t' that neighbors t
17:         create edge m by connecting circumcircle centers of t' and t
18:         add m to V(P)
19:     end for
20: end for
```

The above algorithm will compute $DT(P)$ and $V(P)$ from $CH(P')$. One issue that must be addressed is infinite length edges in $V(P)$. **Hull2VD** handles infinite edges by extending those edges outside of the OpenGL view volume, then using clipping planes to clip them at the border of the unit square in the $z = 0$ plane.

4 Hull2VD

Hull2VD is an application written in OpenGL and C used to illustrate the reduction from the convex hull in space to the planar Voronoi diagram and Delaunay triangulation. **Hull2VD** is completely graphical and displays all information on-the-fly. When a user adds or changes the position of a point, **Hull2VD** recalculates the convex hull and resulting Voronoi diagram and Delaunay triangulation and refreshes. This behavior allows the student to explore the concepts presented by the program in real time. **Hull2VD** is intended for use in the classroom as an aid to the instructor, or as a freely downloadable application for interested students and enthusiasts.

Hull2VD consists of two main windows: a 2D Points window and a 3D Projection window. When the user launches **Hull2VD**, both 2D Points and 3D Projection windows will appear blank, with no points on the screen. Once the user adds four points (required to create a simplex) by right-clicking on the 2D Points canvas, **Hull2VD** will begin calculating the convex hull, Voronoi

Diagram and Delaunay triangulations. Each subsequent point addition will cause **Hull2VD** to re-calculate and refresh all windows. By default, the Voronoi diagram and Delaunay triangulation visualization is toggled on. The Voronoi diagram is represented by randomly colored cells on both windows, with black lines for edges. Delaunay triangulation is shown with heavy blue lines. When the user selects the current point by left-clicking, the point is colored magenta on both windows. Similarly, when the interior of a Delaunay triangle is left-clicked it is highlighted in light red.

The 2D Points window (Fig. 4) is used for all point manipulation, visualization of the planar Voronoi diagram, Delaunay triangulation and corresponding circumcircles. It also contains other general application controls for tasks such as creating a new set of points and exiting the program. The window consists of a large canvas that represents the $z = 0$ plane, and a bottom bar which includes toggles for graphical options and other GUI controls. A user can add points by simply right-clicking on the canvas. Points are selected by left-click, and modified by clicking and dragging the mouse. Additionally, the user can select individual triangles in the Delaunay triangulation by left-clicking within a triangle's area. The bottom bar of the 2D Points window includes check boxes: Show Paraboloid, Show Delaunay Edges, Show Voronoi Diagram and Show Circumcircles. These check boxes toggle the visibility of their associated visualizations on the screen. Any changes made to the check boxes affect both 2D Points and 3D Projection windows. The bottom bar also contains sliders to translate the projected points along the z-axis, and an opacity slider that controls unit elliptic paraboloid transparency. Users can create a new set of points by clicking New.

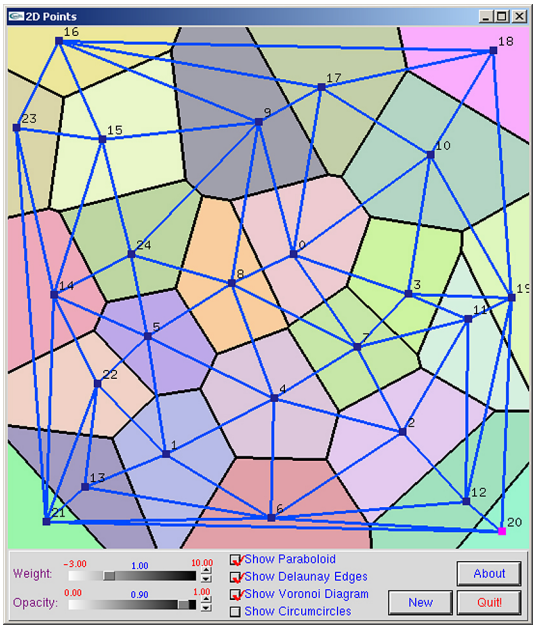

Figure 4: 2D Points window showing Voronoi diagram and Delaunay triangulation

The 3D Projection (Fig. 5) window displays the unit elliptic paraboloid in transparent yellow, as well as the projected points on the paraboloid and the planar Delaunay triangulation and Voronoi

diagram. The convex hull in space is shown as a flat shaded polyhedron. Each downward-facing facet used in the Delaunay triangulation is colored cyan; upward-facing facets are colored green and are discarded in the projection. The 3D Projection window allows the user to toggle dashed vectors from the points in the $z = 0$ plane to their corresponding point on the unit elliptic paraboloid, illustrating their connection. Additionally, the user can rotate, zoom and pan the scene.

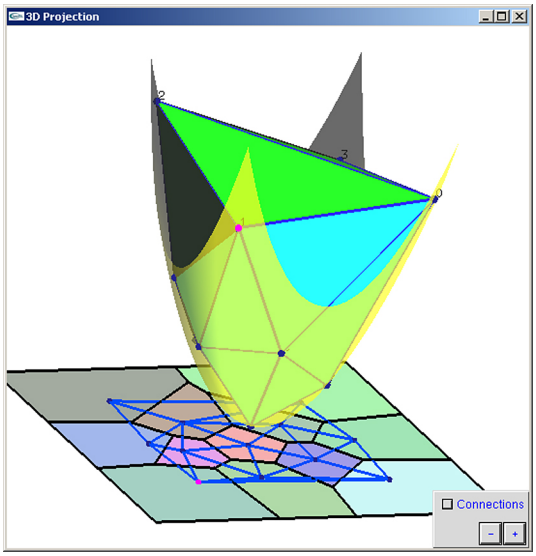

Figure 5: 3D Projection window showing the convex hull and unit elliptic paraboloid

Hull2VD allows the user to see the connection between these problems in several significant ways. First, **Hull2VD** uses a learning-by-doing approach; users are given the flexibility to explore the problems by themselves. For example, the trackball feature in the 3D Projection window can be used to align the view with the z-axis, essentially "looking up" at the convex hull in space. It is apparent that the planar Delaunay triangulation lines up exactly with the downward-facing facets of the convex hull in space. Both the Delaunay triangulation and Voronoi diagram are displayed on the $z = 0$ plane in the 3D Projection window, this gives the user a "global" perspective by seeing all problems displayed at once. Vectors connecting the planar points to the points in space illustrate the projection and corresponding 3D convex hull (Fig. 6). Second, all calculations are updated on-the-fly. When the user adds a new point or changes a point location, he/she can see the effect it has on the Voronoi diagram, Delaunay triangulation and convex hull. Third, when a point is selected in the 2D Points window, its color is changed in the plane and in space. This allows the user to see which point is being modified and how those modifications affect the surrounding structure of the Voronoi diagram, Delaunay triangulation and convex hull.

Other interesting properties of the Voronoi diagram and Delaunay triangulation can be seen in the 2D Points window. When Show Circumcircles is toggled on, the circumcircles associated with each triangle in the Delaunay triangulation are displayed in light blue. The user can left-click within a triangle to mark it as the "selected triangle". The selected triangle's circumcircle will be highlighted in red, allowing users to verify that the circle contains exactly three points on its boundary if no more than three points in the input point set are co-circular (Fig. 7). Individual points

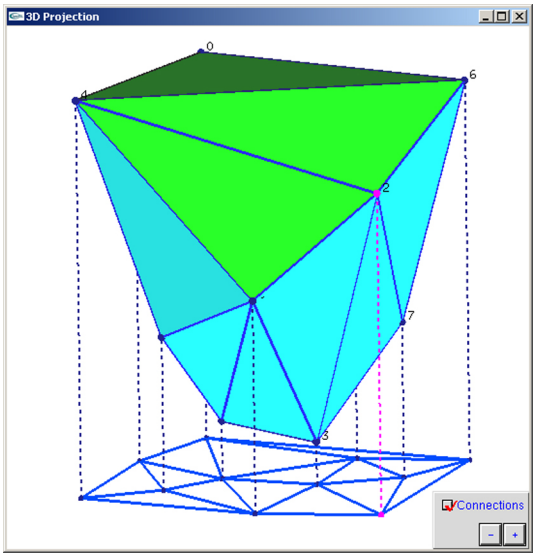

Figure 6: Projections of planar points into space and the resulting convex hull

can be dragged around the canvas, and the user can see the Voronoi cells and Delaunay triangles change in response. Enabling the visualization of both the Delaunay triangulation and Voronoi diagram by selecting Show Delaunay Edges and Show Voronoi Diagram demonstrates the duality between both problems; it is clear that each edge in the Delaunay triangulation corresponds to an edge in the Voronoi diagram, and each cell in the Voronoi diagram corresponds with a point in the Delaunay triangulation.

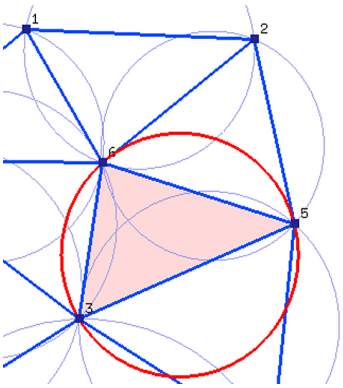

Figure 7: Circumcircles associated with Delaunay triangles

5 Conclusions

We have presented the three problems: convex hull, Voronoi diagram and Delaunay triangulation, as well as their important connection and a visual tool, **Hull2VD**, used to illustrate this connection. **Hull2VD** allows students and other interested parties to explore these problems without any

complex mathematical language or prior background in computational geometry. The underlying mathematics and theory involved in the connection among these problems is left up to the instructor who may use **Hull2VD** as a supplementary tool.

Hull2VD was developed under Microsoft Windows XP using Visual Studio .NET and has been ported to Linux, Solaris and Mac OS X operating systems. It has been tested on a wide variety of system configurations and has been found to run smoothly on most modern hardware. **Hull2VD** is available for free download as part of the DesignMentor v2.0 package (Shene, nd).

References

Baker, J. E., Cruz, I. F., Lejter, L. D., Liotta, G., and Tamassia, R. (n.d.). Mocha. Retrieved from `http://loki.cs.brown.edu:8080/pages/`.

Barber, C. B., Dobkin, D. P., and Huhdanpaa, H. (1996). The Quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483.

Boardman, J. (1993). Automating spectral unmixing of AVIRIS data using convex geometry concepts. *4th JPL Airborne Geoscience Workshop*, 1:11–14.

Bose, N. and K.Garga, A. (1993). Neural network design using Voronoi diagrams. *IEEE Transactions on Neural Networks*, 4(5):778–787.

Chew, P. (n.d.). Voronoi/Delaunay applet. Retrieved from `http://www.cs.cornell.edu/Info/People/chew/Delaunay.html`.

Clarkson, K. L. (1987). New applications of random sampling in computational geometry. *Discrete and Computational Geometry*, 2:195–222.

Clarkson, K. L. (1996). A program for convex hulls. Retrieved from `http://cm.bell-labs.com/netlib/voronoi/hull.html`.

de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. (2000). *Computational Geometry: Algorithms and Applications*. Springer, second edition.

Delone, B. N. (1934). Bull. acad. sci. *USSR: Classe Sci. Mat*, 7:793.

Fortune, S. (1987). A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–147.

Graham, R. L. (1972). An efficient algorithm for detemining the convex hull of a finite planar set. *Information Processing Letters*, 1:132–133.

Grünbaum, B. (1993). Measure of symmetry for convex sets. *7th Symposium in Pure Mathematics of the American Mathematical Society, Symposium on Convexity*, pages 233–270.

Icking, C., Klein, R., Kllner, P., and Ma, L. (n.d.). VoroGlide. Retrieved from `http://wwwpi6.fernuni-hagen.de/GeomLab/VoroGlide/`.

Jarvis, R. A. (1973). On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2:18–21.

Mercier, F. and Baujard, O. (1997). Voronoi diagrams to model forest dynamics in French Guiana. *Proceedings of the 2rd International Conference on GeoComputation*, pages 161–171.

O'Rourke, J. and Goodman, J. E. (1997). *Handbook of Discrete and Computational Geometry*. CRC Press.

Preparata, F. P. and Shamos, M. I. (1985). *Computational Geometry: An Introduction*. Springer-Verlag.

Seidel, R. (1991). A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry: Theory and Applications*, 1:51–64.

Shene, C.-K. (n.d.). DesignMentor 2.0. Retrieved from `http://www.cs.mtu.edu/~shene/NSF-2/`.

Soille, P. (2000). From binary to grey scale convex hulls. *Fundamenta Informaticae*, 41(1-2):131–146.

Stora, D., Agliati, P.-O., Cani, M.-P., Neyret, F., and Gascuel, J.-D. (1999). Animating lava flows. *Graphics Interface*, pages 203–210.

Acknowledgements

I would like to acknowledge the help of Dr. C-K Shene, whose conversations and guidance throughout this project has been most helpful. Also, Dr. John Lowther for helpful insights on this paper. Ken Clarkson's convex hull implementation, QuickHull, is available for free download (Clarkson, 1996). The development of **Hull2VD** and DesignMentor 2.0 is supported by the National Science Foundation under grant number DUE-0127401.