Seeing Dual

Developing a Tool for Visualizing the Relationships between Convex Hulls, Upper Envelopes, Voronoi Diagrams, and Delaunay Triangulations

John Ingraham
Advised by: Dave Mount

April 25, 2013

1. Introduction

The Delaunay Triangulation, Voronoi Diagram, Convex Hull, and Upper (or Lower) Envelope are widespread geometric structures seen in many fields such as computational geometry and computer graphics. Their ubiquity and usefulness in various scenarios make them important constructs to examine and study if possible, but they can be difficult to visualize or draw without some sort of aid. In addition, there are some important and intuitive relationships between the different structures that are most easily conveyed with a cohesive and complete visualization.

The Dual and Projection Visualizer is a program that provides an interactive visualization of the above structures; more specifically, the program provides the delaunay triangulation and voronoi diagram of a set of points in the plane, along with the three dimensional convex hull and upper envelope of these same points projected onto a surface. Programmed in C++ and utilizing OpenGL (see [5]) and GLUT (see [3]), the essential function is to give the user various modes of interacting with the structures drawn in the interface in order to probe both the relationships between them and the individual structures themselves. The program is available for download here.

The remainder of this article is organized as follows. Section 2 introduces the basic structures in computational geometry explored in this program, as well as how they are related to one another. Section 3 covers the interface and design of this program. The algorithms used to produce these geometric structures in the form of pseudocode are presented in Section 4. Finally, a discussion of the merits of this program along with some potential improvements can be found in Section 5.

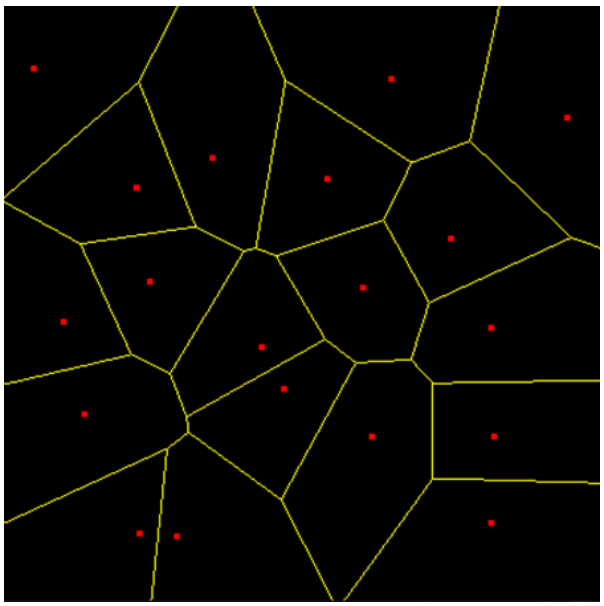

Figure 1: A voronoi diagram (in yellow) generated from an input set of sites

2. Some Geometric Structures

From [4, p. 68], given a set of points in the plane, $P = \{p_1, p_2, p_3, \ldots, p_m\}$ (called sites), the *Voronoi Cell* $V(p_i)$ is the set of points q in the plane where q is closer to p_i than any other site:

$$V(p_i) = \{q \mid \|p_i q\| < \|p_j q\|, \forall j \neq i\}$$

The *Voronoi Diagram* is then the convex subdivision of the plane formed by the union of the voronoi cells. Edges of the diagram are simply the points q equidistant from both p_i and p_j, and the vertices are the single point q equidistant from p_i, p_j, and p_k (assuming no set of four cocircular points in P). See Figure 1 for a visualization.

Moving onward, the *Delaunay Triangulation* $DT(P)$ of a set of points in the plane, $P = \{p_1, p_2, p_3, \ldots, p_m\}$ (called sites) is a planar graph where an edge exists between two sites p_i and p_j if and only if there exists a circle passing through p_i and p_j that contains no other sites (also from [4, p. 75]). See Figure 2 for a visualization (note that the circles in the right image reflect the empty circumcircle property described below). As a corollary to this definition, the triangles in the delaunay triangulation also have a property known as the *empty circumcircle property*. That is, for a triangle to exist in the delaunay triangulation, its circumcircle must contain no other sites in P.

> **Proof of Empty Circumcircle Property:** Given the empty circumcircle of a triangle in the delaunay triangulation, for each edge we can perturb the circumcircle's center infinitesimally away from the third vertex of the triangle to obtain the empty circle required by the original condition (assuming no four sites cocircular).

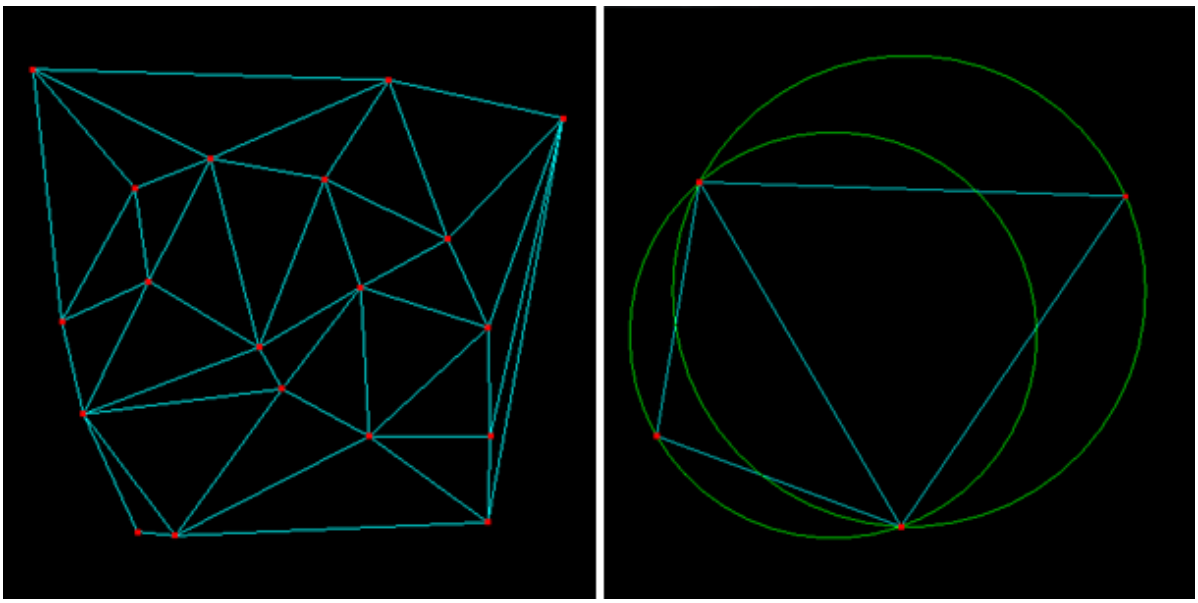

Figure 2: Two delaunay triangulations, one with circumcircles drawn

This proves the forward direction; now we look at the reverse direction. Now, for each edge $\overline{ab}$ in the triangle, we have to have an empty circle passing through the edge's endpoints. Call the other vertex of this triangle c and the other vertex of the other triangle sharing this edge v. If v was positioned on or inside of this triangle's circumcircle, there could not exist an empty circle passing through only a and b because c and v bound the possible circles on either side. Therefore, v must lie outside of the circumcircle of this triangle, and since this is true for all edges, there must be an empty circumcircle around this triangle.

Now that these two structures are fully defined, their relationships can be established and explored. As it happens, the voronoi diagram and delaunay triangulation are what are called *dual structures* of each other. In mathematics, a duality relationship is, loosely, any function that transforms an entity A to the result B, while also transforming B back to A ([7]). In computational geometry, the input sets usually involve points, lines, or planes, so duality in this context attempts to relate these entities to one another (see [2] for more detail). In this case, lines in the voronoi diagram *separating* two sites are related by a dual relationship to lines in the delaunay triangulation *connecting* those two sites. Similarly, every triangle in the delaunay triangulation corresponds to a vertex in the voronoi diagram, which lies at the center of the triangle's circumcircle. With these relationships established, it turns out that it is easy to calculate one of these structures from the other, a property that is very nice to have in practice. Both structures are visualized together in Figure 3.

The *Convex Hull* ($Conv(P)$) of a set of points $P = \{p_1, p_2, p_3, \ldots, p_m\}$ in dimension n is defined as the intersection of all *convex sets* that contain all points in P, where a

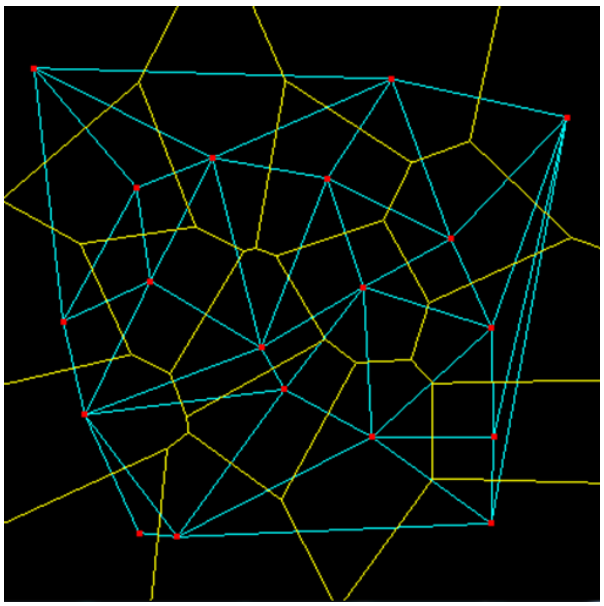

Figure 3: A voronoi diagram and its dual delaunay triangulation, overlaid

convex set is a set of points $S \mid \forall p, q \in S, \overline{pq} \in S$ [1, p. 2]. Alternatively, the convex hull is the intersection of all n-dimensional halfspaces that contain every point in P, where a *halfspace* is merely an n-dimensional plane and all of the space to one side. To illustrate this concept a bit more intuitively, consider a convex hull in two dimensions. The set of points is limited to the plane, and the so-called halfspaces are now just the section of the plane to one side of a given line. Imagine the points are pegs in a pegboard; then the convex hull is the shape a rubber band snapped around all of the pegs would take ([4, p. 12]). This respresentation can actually be seen in Figure 2; the outer boundary of the delaunay triangulation is the convex hull of that set of points. In three dimensions, you can imagine picking the rightmost three points, pressing a piece of paper to their triangle, and then folding it down across each edge until it hits another point, continuing until the paper completely encloses a finite space (in reality, the paper would need to be cut to allow arbitrary folds and then taped back together along an edge). A visualization is presented in Figure 4. Sometimes, it is useful to divide the convex hull into a lower and upper section. For dimension n, the n^{th} component of the normal vector for each face determines which classification it receives; a positive value indicates the upper convex hull and a negative value indicates the lower convex hull. Notice that the upper hull is colored orange and the lower hull is colored yellow in Figure 4.

The *Upper Envelope* ($UEnv(P)$) of a set of planes in n-dimensional space, $P = \{p_1, p_2, p_3, \ldots, p_m\}$, is just the intersection of the set of halfspaces above each of these planes ([4, p. 117]). The *Lower Envelope* is defined similarly, except that the halfspaces are below each of the planes instead of above. Figure 5 visualizes an upper evelope.

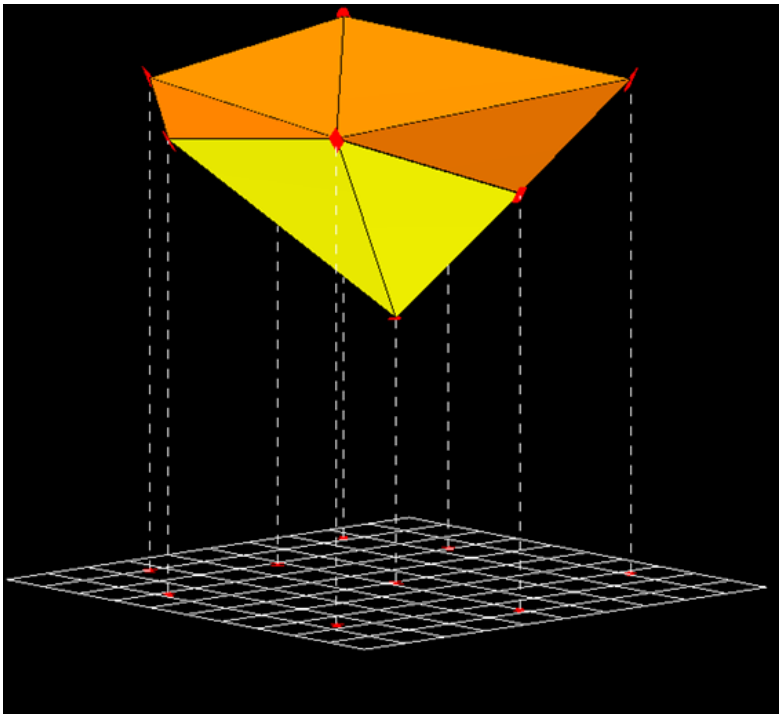

Figure 4: A convex hull of a set of points, which are projected down to the grid to emphasize depth.

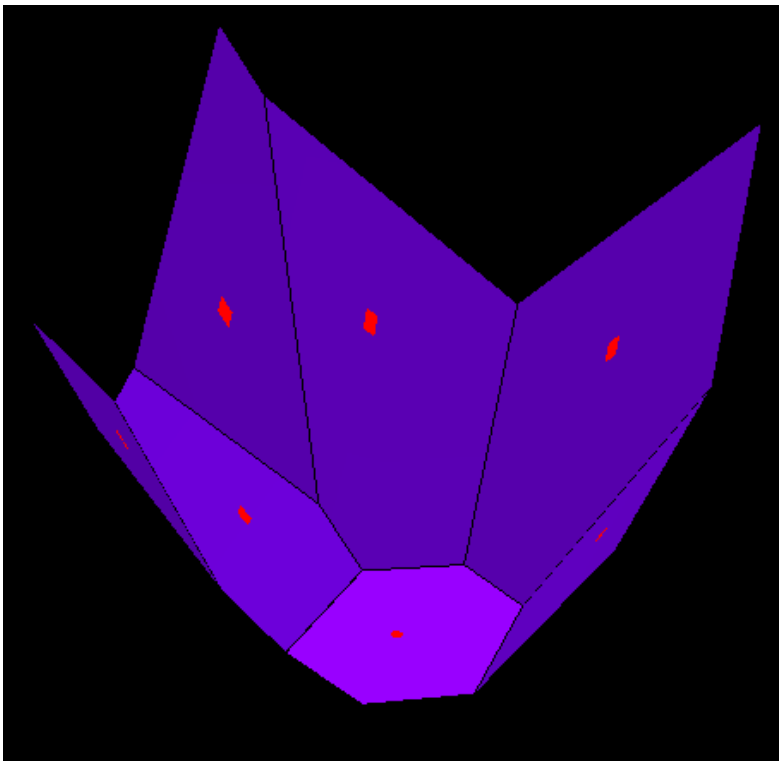

Figure 5: An upper envelope, viewed from below

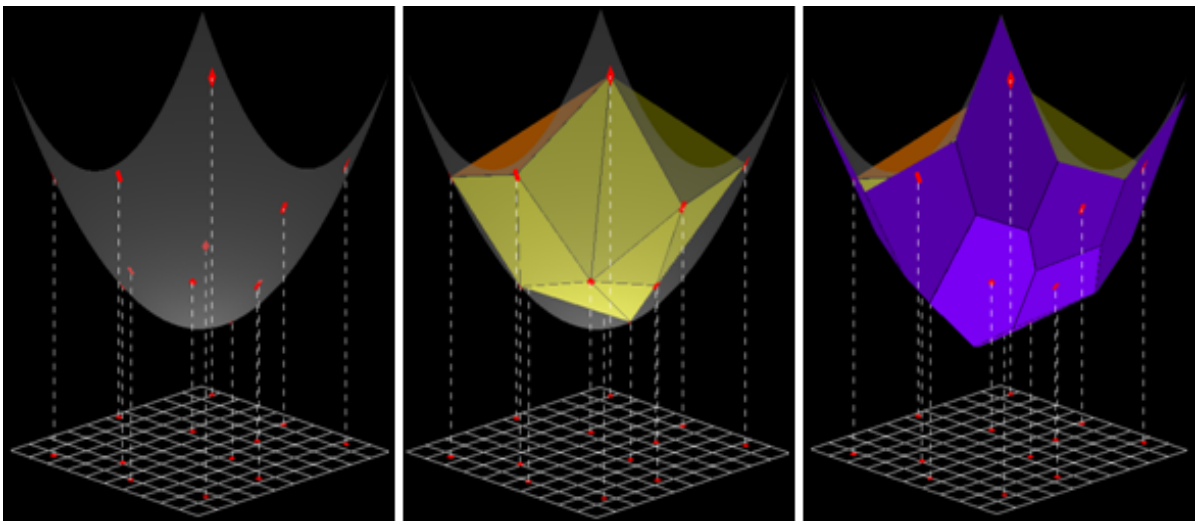

Figure 6: A convex hull, upper envelope, and the paraboloid used in the dual transformation

Now it's time to present another relationship involving duality. The convex hull and upper envelope are also *dual structures*. More specifically, the *lower* convex hull is the dual of the *upper* envelope, and the *upper* convex hull is the dual of the *lower* envelope. The duality this time is between points and planes in three dimensions. Once a duality relation transforming points into planes and planes into points is established, the points that form the upper or lower convex hull are transformed into the planes that compose the corresponding envelope. An easy way to establish a dual relationship between points and planes is to take a surface defined in space and take the tangent plane at the given point. For example, the image of a given point could be the tangent plane at that point to a sphere (of the proper radius) centered at some arbitrary point (this is known as *polar reciprocation* [6]). In the visualizations provided thus far, the duality relation is actually the tangent plane to the paraboloid $z = x^2 + y^2 + c$, where c is a free variable set so that the point $p_i = (x_i, y_i, z_i)$ lies on the surface. So we have the relation:

$$(x_i, y_i, z_i) => z = 2(x_i)x + 2(y_i)y + (z_i - (x_i^2 + y_i^2))$$

Lines can also be related to lines by this dual relationship by considering that the line between two points in the primal should correspond to the line where the two planes in the dual intersect. Other interesting relations develop once these connections are established, such as a plane rotating about a line being dual to a point moving along that line's dual. A side-by-side illustration of the convex hull, upper envelope, and paraboloid is shown in Figure 6.

So why is a paraboloid used here as the basis of the dual transformation between the convex hull and upper envelope? There is another relationship to be explored here, which we will call *projective equivalence*. A structure in n dimensions is *projectively equivalent* to another structure in $n - 1$ dimensions if the projection of the first structure down

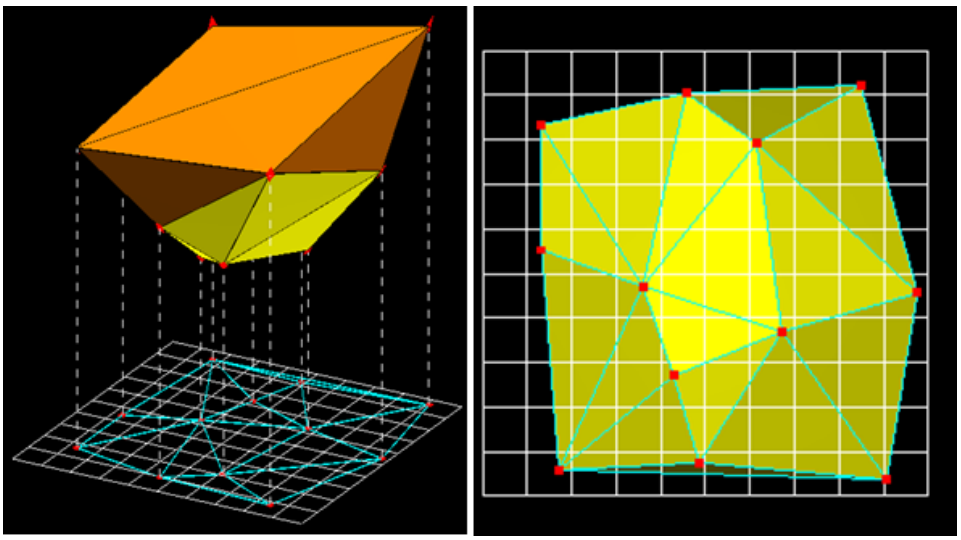

Figure 7: A convex hull and delaunay triangulation, viewed from the side and from below to display projective equivalence

to $n-1$ dimensions is equivalent to the second structure. This is easily checked in the case of $n = 3$ (and an orthogonal basis established by cartesian coordinates), where simply looking at both structures vertically from below will effectively perform this projection. When the set of input points used in the plane for the delaunay triangulation is projected up on to a paraboloid and then used as the input for the convex hull, the (three dimensional) convex hull will be projectively equivalent to the (two dimensional) delaunay triangulation. The same applies to the upper envelope and voronoi diagram.

In order to demonstrate why these structures should be projectively equivalent to their respective pair under this construction, a parallel must be drawn between the unique conditions that define each structure. Now, the unique condition for the existence of a face on the three dimensional convex hull is that all of the points in the input set must lie on or to one side of the plane defined by the face's vertices. Equivalently, there *must not* be any points on the other side of the plane. This is similar to the empty circumcircle property that can be used to define the delaunay triangulation. A proof of the projective equivalence of the convex hull and delaunay triangulation under this construction follows (paraphrased from [4, p. 115]):

> **Proof:** Given the paraboloid $z = x^2 + y^2$, a point $p = (a, b, a^2 + b^2)$ on the paraboloid will produce the tangent plane $z = 2ax + 2by - (a^2 + b^2)$. Then add a constant r^2 to perturb this plane upward so that it intersects the paraboloid at more than just one point. The plane's intersection with the paraboloid would be
>
> $$\begin{aligned} x^2 + y^2 &= 2ax + 2by - (a^2 + b^2) + r^2 \\ (x^2 - 2ax + a^2) + (y^2 - 2by + b^2) &= r^2 \\ (x - a)^2 + (y - b)^2 &= r^2 \end{aligned}$$

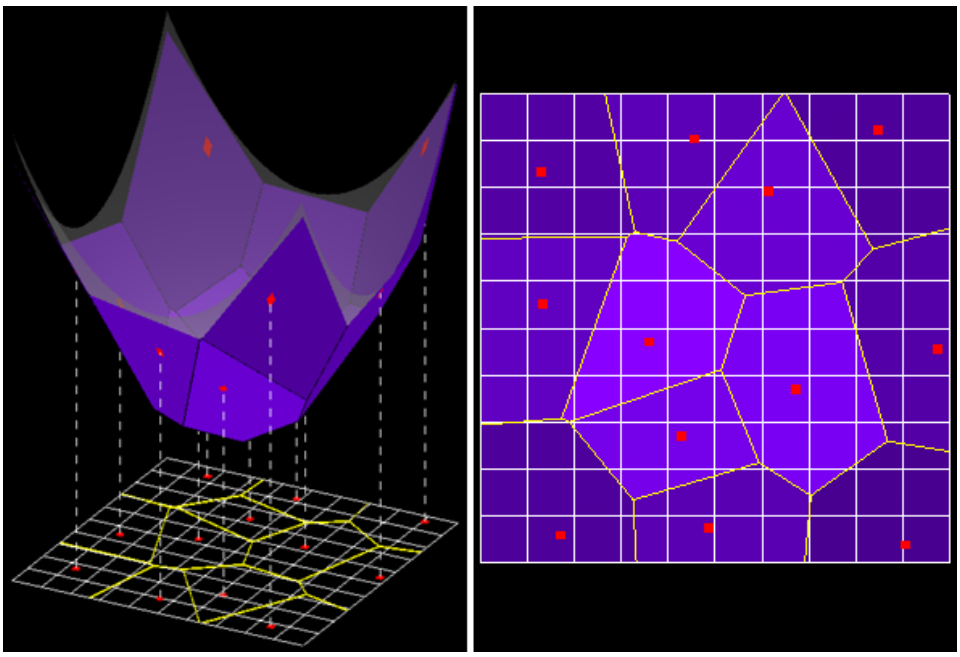

Figure 8: An upper envelope and voronoi diagram, viewed from the side and from below to display projective equivalence

So with z eliminated, this is the projection down onto the xy-plane, demonstrating that any plane's intersection with the paraboloid projects down onto the xy-plane as a circle! Now look at any face of the convex hull. It is composed of three vertices p, q, and r, which uniquely define a plane. The intersection of this plane and the paraboloid projects down to the xy-plane as a circle, with the projections of p, q, and r all lying on the circle. If a fourth point s on the paraboloid lies below this plane, its projection must lie within the projected circle. Thus the conditions required for the delaunay triangulation and convex hull are the same under this construction.

The voronoi diagram and the upper envelope are also projectively equivalent. Given that the voronoi diagram is the dual of the delaunay triangulation, and the upper envelope is similarly related to the lower convex hull, it is elegant that these two dual transformations preserve the projective equivalence of these structures. Here, it is sufficient to show that the intersection of two planes in the upper envelope projects down onto the perpendicular bisector between the two points used to generate the planes. If this is true for all pairs of planes in the upper envelope, then its projection will be the voronoi diagram. The proof follows (also from [4, p. 117]):

Proof: Take any two points $p = (a, b)$, $q = (c, d)$ in the plane. If they are projected up onto a paraboloid, they generate the tangent planes $z = 2ax + 2by - (a^2 + b^2)$

and $z = 2cx + 2dy - (c^2 + d^2)$. Now find the intersection:

$$2ax + 2by - (a^2 + b^2) = 2cx + 2dy - (c^2 + d^2)$$
$$x(2a - 2c) + y(2b - 2d) = (a^2 + b^2) - (c^2 + d^2)$$

Now since z was eliminated, this is the projection of the line at the intersection of the two planes down to the xy-plane. Note that this is also the perpendicular bisector between p and q. First, the midpoint of a and b lies on this line:

$$\frac{a+c}{2}(2a - 2c) + \frac{b+d}{2}(2b - 2d) = (a^2 + b^2) - (c^2 + d^2)$$
$$(a^2 - c^2) + (b^2 - d^2) = (a^2 + b^2) - (c^2 + d^2)$$

Second, its slope is the negative reciprocal of $(\frac{b-d}{a-c})$, i.e. it is perpendicular to the line connecting p and q:

$$x(2a - 2c) + y(2b - 2d) = (a^2 + b^2) - (c^2 + d^2)$$
$$y = -x\frac{2a - 2c}{2b - 2d} + \frac{(a^2 + b^2) - (c^2 + d^2)}{2b - 2d}$$
$$=> slope : -\frac{a-c}{b-d} = -\frac{1}{\frac{b-d}{a-c}}$$

Note that the paraboloid $z = x^2 + y^2$ was selected for simplicity. If desired, the proofs could have been completed with a more general formula for paraboloids, $z = m((x-x_0)^2+(y-y_0)^2)+z_0$. In other words, the steepness and location of the paraboloid are irrelevant to the needed properties in these proofs, although the steepness in the x and y directions must be equivalent.

So now the various relationships between these structures have been demystified. The delaunay triangulation is dual to the voronoi diagram in the plane and the convex hull is dual to the upper envelope in space. Then, when the set of points used as input for the two dimensional structures is projected up onto a paraboloid for the three dimensional structures, and the dual relationship between the convex hull and upper envelope is the tangent plane of that paraboloid, the convex hull and upper envelope are projectively equivalent to the delaunay triangulation and voronoi diagram, respectively. When this somewhat complicated construction is put in place, all four of these structures become closely interrelated.

3. Program Design and Interface

This program provides various functions aimed at the general goal of visualizing these four structures and their relationships. There are two windows in the interface: an input window on the left (see Figure 9), and an output window on the right (see Figure 10). The input window visualizes the xy-plane, drawing the delaunay triangulation and voronoi

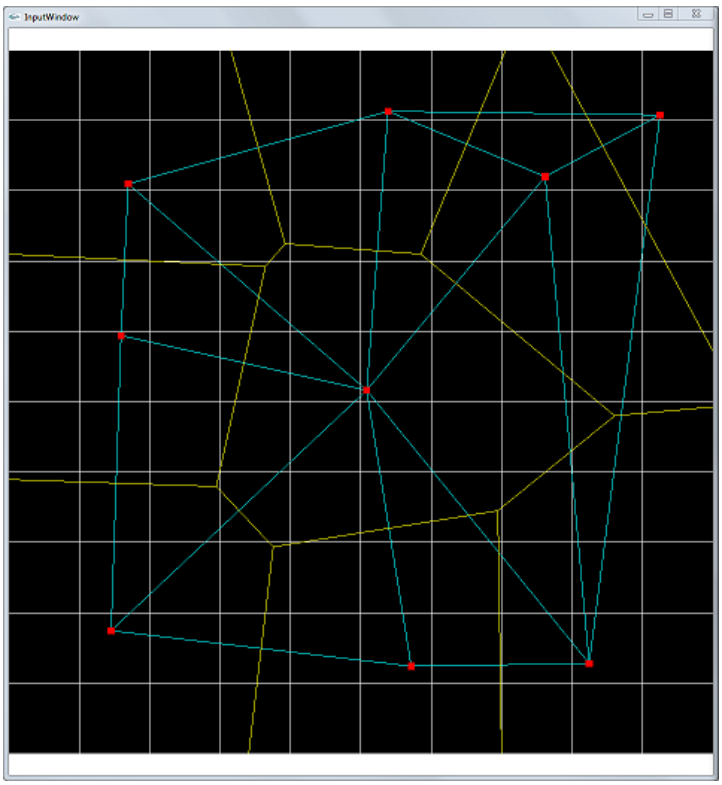

Figure 9: The input window of the program

diagram along with a grid and the input points themselves. Interaction is mostly limited to the ability to add a new input point at the location of a click. The output window shows all four structures drawn in a three dimensional scene. The convex hull and upper envelope surround the parabola used for projection, the voronoi diagram and delaunay triangulation are drawn clipped to the grid in the xy-plane, and the input points are drawn on both the xy-plane and the parabola with dotted lines indicating the projection transformation. Various controls change the viewpoint from which the scene is viewed or mutate the paraboloid being used for projection.

The basic workflow of the program is as follows. When the user adds a point $p = (a, b, 0)$ by clicking on the input window, a point q is also defined by projecting p up onto the paraboloid. At any given point, the paraboloid is defined as $P(x, y) = m((x - x_0)^2 + (y - y_0)^2) + z_0$, so q would be $(a, b, m((a - x_0)^2 + (b - y_0)^2) + z_0)$. Once p and q are determined, p is input into the delaunay triangulation incremental algorithm, and q is input into the convex hull incremental algorithm. These actions trigger the recalculation of the voronoi diagram based on the new delaunay triangulation and the upper envelope based on the new convex hull. The other change that triggers a recalculation of these various structures is a change in any of the paraboloid's parameters. The user may change m, x_0, and y_0 in order to demonstrate that these parameters do not affect the relationships described above. Note that z_0 is dynamically updated so that the

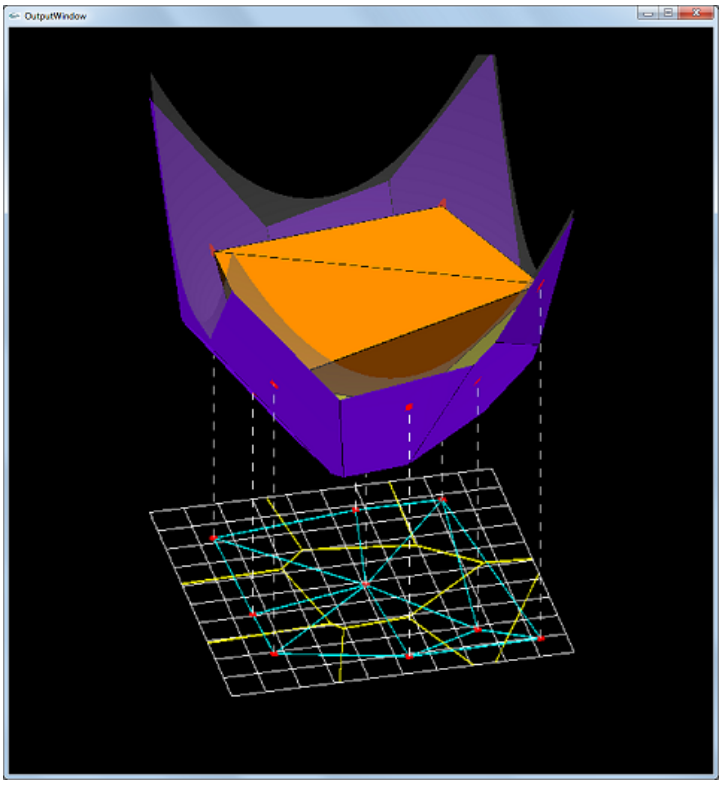

Figure 10: The output window of the program.

paraboloid is always drawn slightly above the xy-plane. If any of these inputs change, the entire set of projected input points is recalculated based upon the new paraboloid equation and run through the convex hull incremental algorithm one by one. Once this is completed, the upper envelope is recalculated as usual. Note that the voronoi diagram and delaunay triangulation are unaffected by this type of change.

As far as the actual visualization of the structures, the code mostly consists of standard openGL polygon calls with a static color applied. Lighting was added to make the angles of the faces of the various polytopes more distinguishable, and the paraboloid is blended so that the convex hull can still be seen with the paraboloid drawn. The lines indicating the projection between points in the plane and on the paraboloid were drawn using a line stipple. Also notable was the use of clipping planes to restrict the voronoi diagram and upper envelope to the grid, as the true structures would have lines or planes continuing to infinity in various directions. Any other occlusions that could get in the way of examining each structure were dealt with by adding the ability to toggle almost every drawn component of the program.

4. Algorithms and Pseudocode

Since the set of input points is specified one at a time by the user, an incremental algorithm for calculating these structures would be beneficial for visualizing these structures as points are added. Also, the size of the input set is restricted, so the simpler and less efficient algorithms may be used without too many consequences. In particular, the simple algorithms for convex hull and delaunay triangulation generation are incremental. Then, the upper envelope and voronoi diagram are calculated using their respective dual structures. Thus, the dual relationships described above are taken advantage of by these algorithms, but the projective equivalence relations are purely coincidental as far as the code used to visualize these structures is concerned.

Four algorithms in pseudocode follow this paragraph, each describing how one of the structures is computed. Assume a standard DCEL representation of these structures, although in actuality the code uses custom data structures with less functionality and specification than a full DCEL.

Most of these algorithms are straightforward, but there is a somewhat lengthy caveat in the upper envelope algorithm. Note that the upper envelope is the dual of the *lower* convex hull. Since all points in the convex hull lie on a paraboloid, they all lie on the lower convex hull. However, some of the faces incident on some of those points will be part of the upper convex hull. Intersecting the tangent planes of each of these three vertices actually gives a point on the lower envelope, which isn't quite the point at infinity needed to accurately represent the upper envelope. Taking advantage of some of the properties of dual transformations, however, an appropriate point can be found:

Algorithm 1 Convex Hull Incremental Algorithm (from [1, p. 246])

if exactly 4 points have been specified so far **then**
    Form a tetrahedron from these four points
else if more than 4 points have been specified **then**
    Find all faces visible to this point (p)
    Create a polytope P out of these faces
    for each edge e in the ring of edges around P **do**
        Make a new face by connecting e and p and add to the convex hull
    end for
    for each face f in P **do**
        Remove all references to this f
    end for
end if

Algorithm 2 Upper Envelope Algorithm

for each vertex $v = (a, b, m((a - x_0)^2 + (b - y_0)^2) + z_0)$ on the convex hull **do**
    Find the tangent plane:
        $T(v) \equiv (z = 2m(a - x_0)x + 2m(b - y_0)y + m((x_0)^2 + (y_0)^2 - a^2 - b^2) + z_0)$
end for
for each face f on the convex hull **do**
    Find the corresponding upper or lower envelope vertex
    (i.e. find the intersection of the planes corresponding to f's vertices)
end for
for each vertex v on the convex hull **do**
    Find all faces on the convex hull that share v
    Sort these faces in CCW order relative to $T(v)$'s normal vector
    Convert each of the faces in this list to its corresponding envelope vertex
    Create a face of the upper envelope by connecting each of these vertices in order
end for

Algorithm 3 Delaunay Triangulation Incremental Algorithm (from [1, p. 199])

Def: Given a vertex v in a triangle T_1, there is an edge e in T_1 that does not contain v and another triangle T_2 that shares e with T_1. Then the *opposite vertex*, as used below, is the vertex in T_2 not shared with T_1.

Require: Start with an arbitrarily large triangle

for each point p added thereafter **do**
  Find the triangle in the delaunay triangulation that contains the new point
  Connect each vertex of the container triangle to p
  Add these three new triangles and remove the old container triangle
  for each new triangle T created **do**
    Find the opposite vertex v and its containing triangle T_v given p and T
    if v is in T's circumcircle **then**
      Perform an edge flip:
        Remove the edge shared by T and T_v
        Add an edge between p and v
        Remove the two old triangles and add the two new triangles
    end if
  end for
end for

Algorithm 4 Voronoi Diagram Algorithm

for each triangle T in the delaunay triangulation **do**
  Calculate the center of T's circumcircle
  Add a vertex in the voronoi diagram at this location
end for
for each triangle T in the delaunay triangulation **do**
  Get the associated voronoi vertex
  Connect this vertex to all neighboring triangles' voronoi vertices
end for

On line 13 of the upper envelope algorithm, a vertex v on the convex hull has been selected, and a sorted list of its incident faces is being traversed to create the dual list of vertices forming a face of the upper envelope. An edge on this face (e) is actually the dual of the intersection (i) between the two faces in question (f_1 and f_2) on the convex hull. Translating a point along e is dual to rotating a plane from f_1 to f_2 about i. So for the sake of visualization, given adjacent faces f_1 on the lower convex hull and f_2 on the upper convex hull, it is sufficient to find the line connecting the dual points of f_1 and f_2, select a point on the line with an arbitrarily large z-component, and use that as the effective point at infinity. Therefore, when a face on the upper convex hull is reached in the traversal, this method is applied.

In addition, these algorithms make some basic assumptions about their input. In particular, all of these algorithms make the simplifying assumption of general position. *General position* simply states that no three points in the input set are collinear and no four points are cocircular. Therefore, in order to account for this assumption in practice, either the algorithms have to be made more robust in their handling of edge cases, or some method of perturbation must be applied to the input set of points in order to (at least probabilistically) ensure that the requirements of the assumption are met. Since the granularity of the input set in this program is limited to the finite number of pixels on the input screen, it is possible to perturb each point in the input set some amount smaller than the size of a single pixel. This adds some randomness to the input set, reducing the chance that three collinear points or four cocircular points will be entered. In addition, this program prevents points that are too close (within a distance threshold) to another point from being added to keep the screen from becoming too cluttered.

Another issue is worrying about the limited accuracy of floating point computations on the computer. Some amount of correction needs to be made to computed values such as determinants of matrices when they stray slightly from their expected values. A simple example is checking that the absolute value of a determinant is below some epsilon instead of checking for equality with zero. These corrections can be found throughout the code in an attempt to keep all inputs and intermediate values for the algorithms in valid ranges.

Efficiency is another obvious concern. The convex hull and delaunay triangulation algorithms perform well, easily finishing before the user has time to input another point. The upper envelope and voronoi diagram calculations are a bit slower since they are calculated from scratch every time their dual structure changes. Stress testing the application led to some slowdown for very dense input sets, especially when changing the paraboloid and rebuilding the convex hull from scratch, but for normal use performance was smooth.

5. Conclusions

This program aims to facilitate the user's exploration of these structures and their relationships. The drawing is modular so that the user may choose which components of the scene he or she wants drawn at any given time. Typical camera manipulation is also provided so that the three-dimensional structures can be viewed from any angle. All in all, this program provides high quality rendered images of voronoi diagrams, delaunay triangulations, convex hulls, and upper envelopes for a dynamic input set of points, as well as illustrating how the structures are related to each other.

Beyond this base functionality, there is a laundry list of potential improvements that could make this program more interactive and perhaps more informative. The input set is static except for adding more points and clearing the entire list. Adding the ability to select given points and either delete them or move them around could add to the intuition gained by potential users. Better algorithms for calculating and keeping track of the various geometric structures could also be implemented for a potential performance boost. Other more aesthetic options that could be implemented include the ability to modify the base colors used for each structure or customization and better use of blending techniques. However, this program in its current state provides a solid implementation for the desired functionality.

References

[1] Mark de Berg et al. *Computational Geometry: Algorithms and Applications*. 3rd ed. Santa Clara, CA, USA: Springer-Verlag TELOS, 2008.

[2] Partha P Goswami. *Duality Transformation and its Applications to Computational Geometry*. 2008. URL: http://www.tcs.tifr.res.in/~ghosh/partha1-lecture.pdf.

[3] Mark J Kilgard. *The OpenGL Utility Toolkit (GLUT) Programming Interface API Version 3*. Silicon Graphics Inc. 1995. URL: http://www.opengl.org/resources/libraries/glut/spec3/spec3.html.

[4] David Mount. *CMSC754 Lecture Notes*. Mar. 2013. URL: http://www.cs.umd.edu/~mount/754/Lects/754lects.pdf.

[5] Mark Segal and Kurt Akeley. *The OpenGL Graphics System: A Specification (Version 1.1)*. Ed. by Chris Frazier. Silicon Graphics Inc. 1995. URL: http://www.opengl.org/documentation/specs/version1.1/glspec1.1/.

[6] Wikipedia. *Dual polyhedron — Wikipedia, The Free Encyclopedia*. [Online; accessed 28-March-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Dual_polyhedron&oldid=539351748.

[7] Wikipedia. *Duality (mathematics) — Wikipedia, The Free Encyclopedia*. [Online; accessed 28-March-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Duality_(mathematics)&oldid=544248126.

Appendix A. Controls

All input is handled through key/mouse interaction: To provide input to a specific window, make sure it is selected before providing any input!

Input Window (left)

Add a new input point	left click
Quit the program	q
Refresh the window	r
Clear out all input points	R
Toggle drawing the delaunay triangulation	d
Toggle drawing circumcircles	c
Toggle drawing the voronoi diagram	v
Toggle drawing the grid	g

Output Window (right)

Change camera viewpoint	left click + drag
Kick off a rotation of the scene	left click + flick
Zoom in or out	right click + drag
Quit this program	q
Reset the paraboloid's parameters	r
Clear out all input points	R
Toggle drawing the delaunay triangulation	d
Toggle drawing circumcircles	c
Toggle drawing the voronoi diagram	v
Toggle drawing the grid	g
Toggle drawing the convex hull	h
Toggle drawing the upper envelope	e
Toggle drawing the paraboloid	p
Toggle drawing the dashed projection lines	l
Increase the slope of the paraboloid	+
Decrease the slope of the paraboloid	-
Move the paraboloid along the +y-axis	up arrow
Move the paraboloid along the -y-axis	down arrow
Move the paraboloid along the +x-axis	right arrow
Move the paraboloid along the -x-axis	left arrow