Efficient Computation of Voronoi Diagrams

Matthew C. Humphrey

TR 88-7

Efficient Computation of Voronoi Diagrams

Matthew C. Humphrey
February 1988

Department of Computer Science
562 McBryde Hall
Virginia Polytechnic Institute
Blacksburg, Virginia 24061

Abstract

Voronoi diagrams have advantages of representing geometric information in a compact form which makes them well suited for robot path planning. However, researchers have often found them too complex and inefficient. In this paper, we review two alternate views of Voronoi diagrams, the Standard [SHAMM75a] and the Generalized [KIRKD79]. We also develop a new method to approximate the Voronoi diagram and show how all three can be effectively used in applications. Finally, the three methods are compared showing their relative strengths and weaknesses for robot path planning.

Approximate length including figures: 3830 words
Topic: Robotics (Static path planning)

I. Introduction

Voronoi diagrams were introduced in [SHAMM75]. Their ability to represent geometric relationships in a compact form makes them suitable for robot path planning and other applications. However, researchers often overlook them because of their complexity and calculation inefficiency. We present here some alternative views of the Voronoi diagram that will allow them to be better utilized in the future.

A Voronoi diagram is a net which partitions a plane containing features into Voronoi regions in which there is one feature per region. The points within a Voronoi region are closer to the feature of that region than they are to any other feature of the plane. Each point comprising the net, therefore, is equidistant from the two closest features.

The Voronoi diagram is useful for finding a path between obstacles. The features of the plane become the obstacles and the Voronoi diagram is a set of paths that lie between the obstacles. The path remains equidistant from the two closest obstacles, be they walls or tables or people. The nature of the obstacles is unimportant; the Voronoi diagram avoids all equally well.

II. What kinds of Voronoi diagrams exist

We denote three kinds of Voronoi diagrams: the "standard" as defined by Shamos [SHAMM75b], the "generalized" as defined by Kirkpatrick [KIRKD79] and refined by Yap [YAPC85] and the "approximate" which is presented here in this paper. Each has different characteristics, is calculated differently and provides different advantages and disadvantages.

II.A. Shamos' standard

Shamos [SHAMM75a] defines the "standard" Voronoi diagram in which the features of the plane are points. The Voronoi diagram is a set of line segments that partition the plane into polygons, each containing one point. Shamos proved that this diagram can be calculated in O (n log n) time by recursive merging of two previously defined Voronoi diagrams.

The Voronoi diagram formed like this is not very useful. It is rare that a robot needs to navigate between parking garage pylons with no intervening cars or trucks. Points are not sufficiently detailed to allow the alogorithm to be used. However, the standard Voronooi diagram does work well for circuit board routers, in which the wires connecting vertical posts must stay maximumly distant from other vertical posts.

II.B. Kirkpatrick's generalized Voronoi diagram

Kirkpatrick [KIRKD79] defines a "generalized" Voronoi diagram in which the features of the plane are line segments as well as points. This allows the algorithm to model real-world objects effectively, for even curves can be expressed as a series of line segments. It has the disadvantage that it generates an exceptionally precise curve that must be followed, and this curve will remain exactly between the two closest features. The algorithm itself has an O(n log n) running time, which is good, but the algorithm itself is very complicated. We know of no implementations.

II.C. Our new method

Our method generates an approximation to the Voronoi diagram for the interior of a polygon, which for example, might represent a floorplan. This "approximate" Voronoi diagram is produced by taking the dual of any triangulation of the polygon. While the diagram does not result in perfectly safe paths, it can be calculated easily and still avoids the walls.

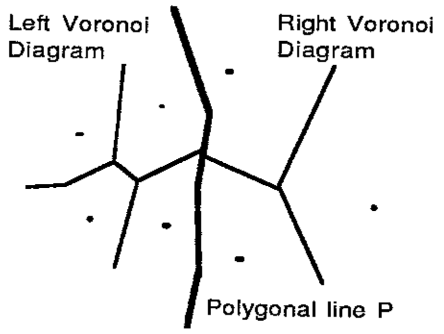
The dual of a triangulation can be computed in O (n) where 'n' is the number of points in the polygon being triangulated. Likewise, the dual can be computed as the triangulation is formed since the dual of one triangle may be found in constant time.

III. How do you calculate them

III.A. Standard

The algorithm originally detailed by Shamos [SHAMM75a] [SHAMM75b] introduces the Voronoi diagram as a useful representation of a set of points for extracting geometric information. Shamos' algorithm operates on a set of points. To begin, assume that the points have been separated into two groups, Left and Right as in figure 1, such that all Left points are to the left of all Right points. Also assume that each group already has its Voronoi diagram calculated.

Shamos proves the existence of a polygonal line (P) that separates Left from Right. All points left of P are closer to Left points than Right points. All points right of P are closer to Right points than Left points. He also shows that this polygonal line is contantly increasing in y value; it does not turn back (Figure 1).



Building Shamos' Voronoi diagram by recursive merging of two previous diagrams.
Figure 1.

He completes his proof by showing that this new polygonal line (P) plus all existing Voronoi edges except those that intersect the new line P form the new Voronoi diagram. This merge process is completed as the polygonal line is traversed. Since (P) is always increasing in value, it can be traversed in linear time, allowing the merging of the Voronoi Diagrams use only O (n log n) time.

Shamos' algorithm treats the objects of interest as points, rather than as independent polygons. There are no walls that must be avoided, all is open space; only the points must be avoided. This greatly limits the use of the algorithm to applications where points define regions.

III.B. Generalized

Kirkpatrick's algorithm [KIRKD79] finds the internal and external skeletons of a collection of polygons. The Voronoi diagram is easily derived from the external skeletons. The polygons represent the obstacles directly. A bounding window surrounds them and the Voronoi diagram is generated within this box.

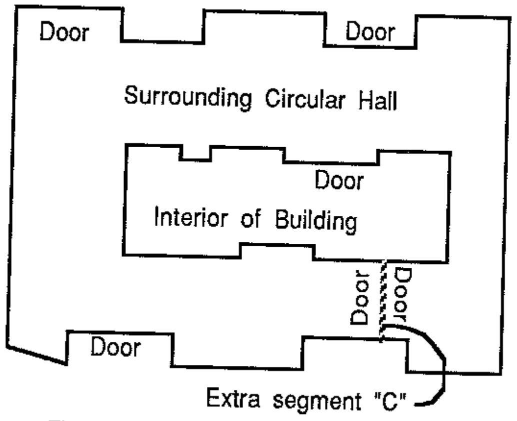
Kirkpatrick defines his polygonal features as points and open line segments. Together, these can be used to describe all manner of planar figures. They provide an adequate representation of static, two dimensional robot obstacles.

Since his obstacles are line segments, the Voronoi diagram that his algorithm produces will contain curved segments. This is because the Voronoi digram produced by various interactions between line segments must be curved to remain equidistant from the nearest features. Likewise, the Voronoi line appearing at the end of one line segment near another line segment must have a complex curve to it. Two parallel lines, though, generate a straight line.

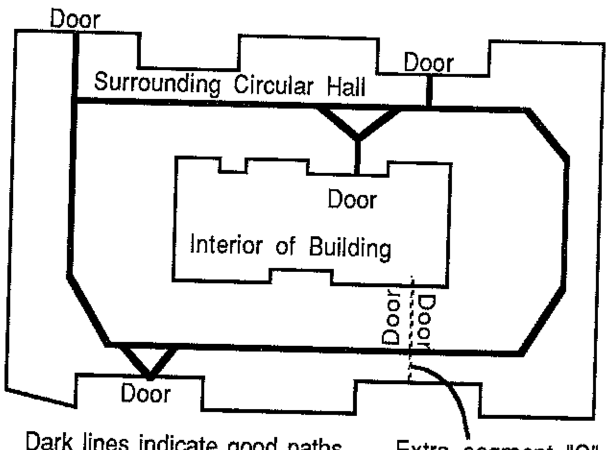
Kirkpatrick proves that there are a fixed number of such interactions that result in curves. His algorithm assembles these basic geometric curves into the Voronoi diagram by analyzing how the line segments interact about their endpoints. He claims that by analyzing the polygonal figures in the plane, it is possible to determine which curve units apply and how they interconnect to form the internal and external skeletons of the scene.

III.C. Approximate

Our method produces an approximation to the Voronoi diagram by taking the dual of the most easily calculated triangle decomposition (triangulation) of the interior of a simple polygon. The interior of a simple polygon is useful for describing floor plans and other open spaces (figure 3). Even complex floor plans containing "loops" can be described by the interior of one simple polygon. This is accomplished by allowing two non-adjacent edges of the polygon to face each other. In figure 3, line segment "C" of the polygon is really 2 very close edges.



Floor plan and open space as one polygon
Figure 3.



Dark lines indicate good paths
Figure 4.

If the outer polygon is considered a hall, in which doors are walls traversable by the robot and the inner polygon is considered an space of no interest (outside), then we would like to generate a Voronoi diagram like that in figure 4 in which the darker segments are the actual paths between doors. See how the joining segment "C" is treated as two very close doors.

The dual of the polygon triangulation representing this room will yield such an approximation to the Voronoi diagram. The dual of each triangle is calculated separately, in constant time for each triangle. Therefore, the Voronoi approximation is derived from the triangulation in O(n) time.

While any triangulation can be used, we present an easy method for triangulating the interior of a simple polygon and two alternatives for finding a triangulation's dual. The triangulation we present has running time worse than O(n log n), but better than $O(n^2)$. However, it is easily implemented. For true efficiency, one could use Tarjan's triangulation method [TARJR86], which has a running time of O(n loglog n). In either case, finding the dual can be done in linear time.

We will not be using the dual of a triangle in the strict sense. Our dual of a triangle generates the Voronoi segments of the triangle. There can be many variations on these duals, which will generate optimizations in the Voronoi diagram. The "wye" dual (figure 5a) avoids all walls the best. The "delta" dual (figure 5b) has shorter and straighter paths. A weighted dual (figure 5c) adjusts the point where the path crosses the edge by some function on values associated with the endpoints. This gives certain corners safety margins.

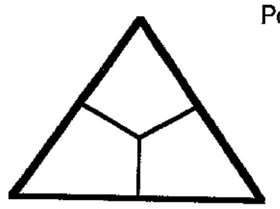


Figure 5a. "Wye" type dual.

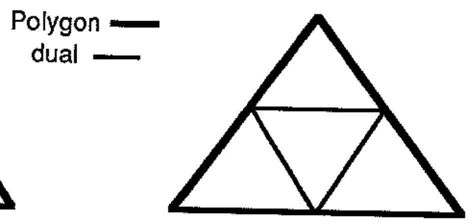


Figure 5b. "Delta" dual.

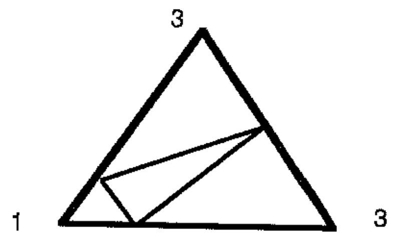
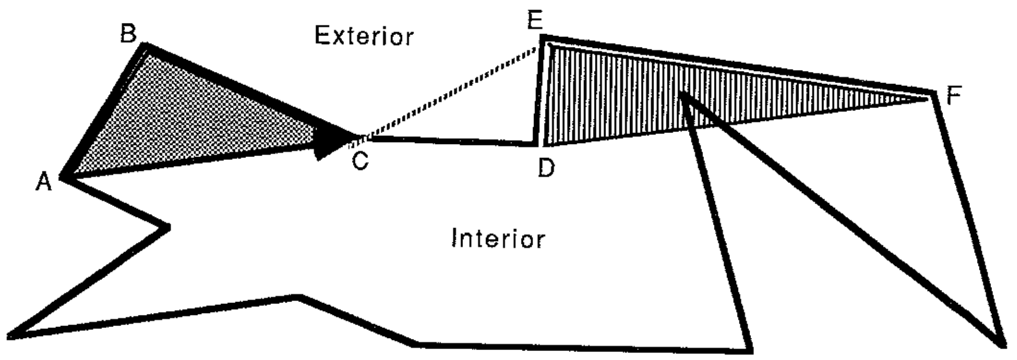


Figure 5c. Weighted dual.

This next section, describing our Voronoi approximation method, is divided into three subparts. The first shows how to calculate a triangulation of the interior of a polygon. The second shows how to convert a triangulation of a polygon into the dual, thereby producing an approximation to the Voronoi diagram. The last describes how to extend the method to three dimensions.

III.C.1 Forming the triangularization

The first step is to generate a triangulation of the interior of the simple polygon. A triangulation partitions the interior of polygon P of N vertices into (N-2) triangles by adding (N-3) diagonals such that no diagonals intersect [TARJ86]. Our method forms this set of triangles iteratively using the set of polygon points. We assume that the points are listed in clockwise order about the polygon.



Triangulating interior of the polygon
Figure 6.

If the polygon has three points, then they form the last triangle and the algorithm terminates. Otherwise of the remaining points, we select three adjacent points. If these points enclose a triangle that lies inside the polygon and that triangle itself encloses none of the polygon's other verticies, then we add the triangle to the triangulation set and eliminate the middle point of the three selected points, which reduces the size of the polygon and we repeat the process. In the figure 6, points A, B and C are adjacent on polygon P. They form a triangle which contains none of the polygon's other vertices. Therefore the triangle may be closed, point B eliminated and the polygon reduced to (N-1) points.

The triangle CDE did not fall within the polygon and triangle DEF contains points of the polygon, so we must select a new set of three points. It is easy to prove that there must always exist at least one triangle to enclose. Assume that it is not possible to close any three adjacent points. This implies that all sets of three adjacent points form concave edges around the polygon, which is not possible. Or it implies that all triangles formed from the boundry edges contain other polygon points. However, those points that cause the triangles to be non-empty are themselves part of the polygon, which means that they are candidates for forming triangles. Points cannot be both contained within a triangle and a vertex of the triangle.

III.C.2. Closing triangles

It is easy to determine if the triangle formed by points A, B and C in figure 6 lies within the polygon, provided that we know on which side of the vector AB the interior of the polygon lies. This information is provided by listing the points in clockwise order. The interior of the polygon is always to the right of vector AB.

Therefore, if point C lies to the right of vector AB then the triangle is in the interior of the polygon. If point C is on the left then the triangle is exterior to the polygon. If the triangle formed by the vectors contains no points, then it may be closed and point B may be eliminated from the set of polygon points. Likewise, edges AB and BC are replaced by a new edge AC.

A problem with forming triangles in this way is that narrow triangles may be generated. Narrow triangles generate less safe paths than triangles that are more equilateral. A safe path stays equidistant from the closest walls. Figure 8a shows a polygon with an arc, which results in many narrow triangles.

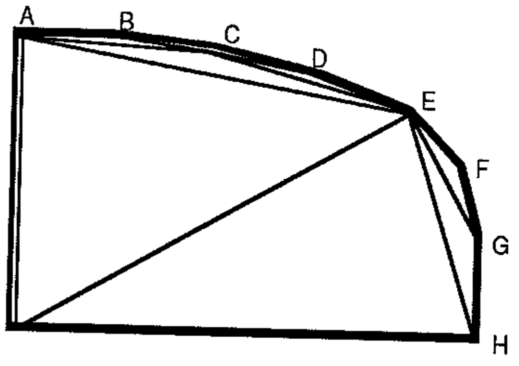


Figure 8a. Arc has poor triangularization

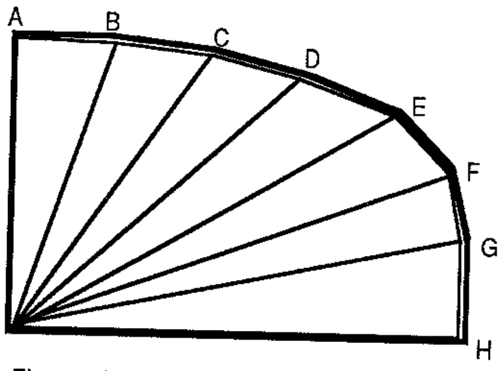


Figure 8b. Better triangularization induced.

Some of these triangles can be avoided by forcing a closed triangle to have certain characteristics. This is most easily done by requiring the angle between the vectors to have a minimal value. Values below the limit result in no triangle. For cases like the arc of figure 8a, limiting the minimum size of triangles will favor the triangulation in figure 8b.

Unfortunately, this may cause some polygons to form no triangles and therefore not have a good triangulation. An approximation of a circle by a polygon causes all adjecent edges to have angles near 180^o. Therefore, any pair of adjacent edges forms a narrow triangle. The narrow triangles cannot be avoided. Imposing a limit may cause the algorithm to never terminate.

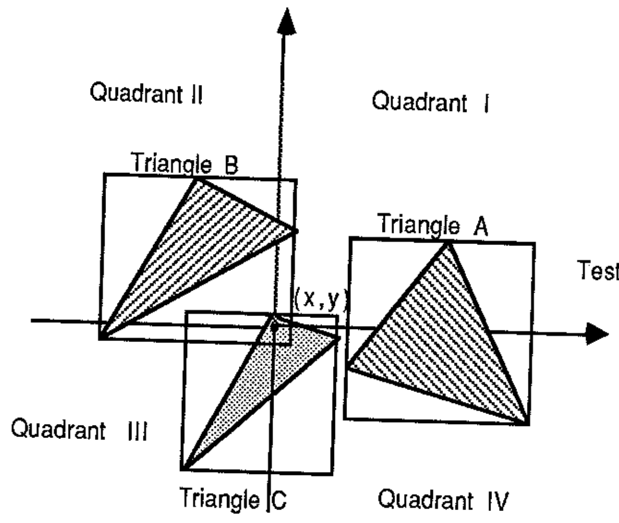
As a final note, in a contrived worst case, there must be $O(n^2)$ comparisons of points. But this worst case relies on a very particular formation of edges, which also must have a very particular starting point. By carefully selecting the starting point or by recognizing the worst case it can be reduced or eliminated. For most applications, this is not a problem. For those which need a faster running time, there are other more complicated algorithms for calculating the triangulation.

III.C.3. Checking emptiness

Once an interior triangle has been found, we must show that it is free of all other points. Figure 6 shows a triangle (DEF) which is not empty. Using a O(n^2) method, it is very easy to check that all remaining points do not lie within the triangle ABC. However, we would prefer better performance than that. Our method uses a quadtree to quickly determine if any points intersect the triangle.

In the beginning we store all points of the polygon in a quadtree. This operation requires O(n log n) time. As each triangle is formed, we intersect the triangle with the quadtree to determine if there are any hits. Any hit at all indicates failure. A complete miss indicates that the triangle is empty of other points.

The flaw with this scheme is that a quadtree cannot be used directly to determine intersections with a triangle. The tree only reports which quadrant, relative to some point, an arbitrary point lies. To overcome this, let's generalize (for the moment) and treat the triangle as a rectangle as in figure 11.



Testing quadtree for triangles. (x,y) is coordinates of quadtree node.

Figure 11.

If the point in the quadtree being tested lies outside of the rectangle, then the rectangle is empty. If the rectangle is empty, then the triangle must also be empty. The rectangle, however, may lie within one or two quadrants surrounding the tree point (x,y) as does triangle A in the figure. These regions must also be searched, but many vertices are eliminated from the search.

If the point lies within the rectangle, then the triangle may also contain the point as in triangle C. A quick vector check determines if the point lies within the triangle itself. If the point is within the triangle, then the search stops and the triangle must be rejected. Otherwise the search continues with the regions between the triangle and the rectangle.

If the point does not lie within the triangle and yet does lie within the rectangle as in triangle B, then there are at most three quadrants that must be searched for a hit. A rectangle enclosing a triangle always has at most three remainder areas. The first one that responds terminates the search. The goal of the quadtree is to reduce the number of points being checked for intersection with the triangle.

III.C.4. Generating the dual

Once a triangle has been generated, there are several ways of approximating its Voronoi diagram. The method that most resembles the Voronoi diagram operates by drawing three line segments, each from the center of the triangle to the midpoints of the faces (figure 5a) called the "wye" dual. The Voronoi lines from two adjacent triangles will meet at the midpoint of the triangle face because the midpoint is the same from both sides.

An alternate method provides shorter paths, but runs closer to the walls. This method draws three line segments between the midpoints of the faces, forming a delta, rather than a wye path. This is presented in figure 5b. The approxmiate Voronoi diagrams calculated in this paper use the "delta" dual.

It is even possible to draw paths of a particular width between the faces, to mimic an actual roadway of minimum width. Triangles whose faces are too small to allows the path to cross them may be marked as "unpassable". Also, corners (vertices) may be weighted to force the path to be closer or further from one side or the other (figure 5c).

IV. What are the application advantages and disadvantages

The most apparent application of Voronoi diagrams is planning robot paths through static environments. We'll examine each method to see how they are used, what their advantages are and what their disadvantages are.

IV.A. Standard

Shamos' Voronoi diagram has several advantages when applied to robotics. It can be calculated in O (n log n) time and guarantees that the path remains exactly equidistant from obstacles. Due to the recursive merging of the algorithm, it seems well suited for parallel machines.

However, it has one significant disadvantage. The obstacles of the standard Voronoi diagram are points. For most real world applications, this is not acceptable. For those where the obstacles are points in a plane, this works well.

IV.B. Generalized

Kirkpatrick's generalized Voronoi diagram uses line segments and points as obstacles.

Since any 2-D object can be described using line segments, they are ideal for real world applications. The polygon skeleton can be calculated in O (n log n) time, which is efficient for most applications. The path generated is usually curved, and does stay equidistant between the closest obstacles.

However, we know of no implementation of Kirkpatrick's algorithm. Its mathematical complexity makes it unsuitable for realistic application [YAPC85]. Yap has described an algorithm to find the generalized Voronoi diagram of points and line segments in O (n log n) time. It is more robust than Kirkpatrick's, but has the same advantages and disadvantages.

IV.C. Approximate

Our approximation algorithm applies to the interior of a polygon, so it is ideal for floor plans and factory layouts. A floor space that has any number of connected paths can be described by one simple polygon. For example, an ordinary room with two doors can be represented by figure 12a. Doors are marked as "open" polygon edges. A table in the room as an obstacle will yield a polygon as shown in figure 12b. The line segment that is drawn from the table to the wall enables the scene to be interpreted as one polygon. However, the line segment, which is really two edges of the polygon, is labelled as a "door", which allows the path to traverse it.

Polygon walls ——— Triangulation .... Voronoi path ———

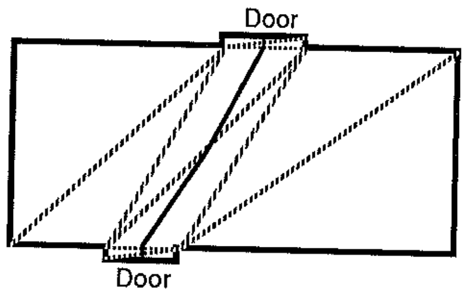


Figure 12a. Triangulating room with a door.

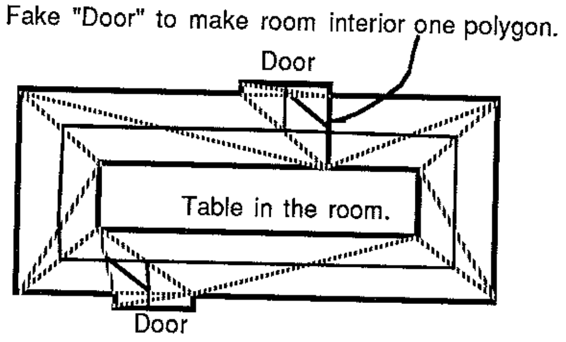


Figure 12b Triangulating a room with a table

The algorithm's triangularization yields additional useful information about the space being traversed. The widths of the passages between obstacles is available to determine if the robot will fit through the door or around the corner. By simply altering the exact calculation of the dual, it is possible to generate more direct paths. The algorithm is implemented and running on an IRIS 2400 graphics work station, where it can convert a floor plan with doors into a set of paths.

There are some drawbacks to the approximation method. It has a longer running time than the other methods, especially for very unusual cases, such as curves that are approximated by short segments. It runs in O(n log n) time, with contrived worst cases at $O(n^2)$. Additionally, the path does not stay exactly equidistant between obstacles, since it is only an approximation. And finally, even for ordinary cases, it can generate poor paths, in which the robot may travel past a target and then come back to it by another route. However, the running time problem can be optimized by using a more efficient triangulation method. Using Tarjan's triangulation yeilds the Voronoi approximation in O (n loglog n) time.

V. Summary of main features

Voronoi diagrams can be used in realistic applications. The different types of Voronoi diagrams "standard", "generalized" and "approximate" are each more suitable for certain kinds of problems. The "standard" works best where the features are isolated points. The "generalized" allows real world objects and provides "safe" paths, but it is difficult to compute. Our approximation works well with fewer separate objects, but has good running time and is easily imlemented. Application to objects found in the real world make the "generalized" and "approximate" Voronoi Diagrams most useful.

The Voronoi diagrams are efficient to calculate. Their runing times of O(n log n) or better are quite good. Additionally, the recurse merging nature of Shamos' algorithm suits it to optimization on a parallel machine.

Finally, these algorithms are not just mathematical curiosities; implementations exist. Shamos' algorithm is straightforward and directly implementable. We have implemented our algorithm on a graphics workstation where it converts floor plan specifications with doors into paths suitable for a robot. We know of no implementation of Kirkpatrick's algorithm.

Bibliography

[KIRKD79]	Kirkpatrick, D. "Efficient Computation of Continuous Skeletons" IEEE publications, 1979.
[SHAMM75a]	Shamos, M.I., Hoey, D. "Closest-point Problems" 16th Annual Symposium on Foundations of CS, October 1975.
[SHAMM75b]	Shamos, M.I. "Computational Geometry" Ph.D. Thesis, Yale University, 1975.
[TARJR86]	Tarjan, R.E., Van Wyk, C.J. "An O (n log log n)-Time Algorithm for Triangulating Simple Polygons" AT&T Bell Laboratories, 1986.
[YAPC85]	Yap, Chee K., "An O(n log n) algorithm for the Voronoi Diagram of a set of simple curve segments." Courant Institute for Mathematical Sciences, Technical Report #161, 1985.