'91 KACC 1991. 10. 22~24

Approximate Voronoi Diagrams for Planar Geometric Models

Kwan-Hee Lee and *Myung-Soo Kim*

Department of Computer Science
POSTECH
P.O. Box 125, Pohang 790-600, Korea.
TEL:(0562)-79-2249 FAX:(0562)-79-2299

Abstract

We present an algorithm to approximate the Voronoi diagrams of 2D objects bounded by algebraic curves. Since the bisector curve for two algebraic curves of degree d can have a very high algebraic degree of $2 \cdot d^4$, it is very difficult to compute the exact algebraic curve equation of Voronoi edge. Thus, we suggest a simple polygonal approximation method. We first approximate each object by a simple polygon and compute a simplified polygonal Voronoi diagram for the approximating polygons. Finally, we approximate each monotone polygonal chain of Voronoi edges with Bezier cubic curve segments using least-square curve fitting.

1 Introduction

Voronoi diagram has many important geometric properties which have attractrated much research interests in diverse application areas such as biology, solid-state physics, pattern recognition, geography, stock-cutting, wire layout, geometric optimization, facilities location, computer graphics, and robotics [7]. In collision-avoidance robot motion planning, the problem of finding collision-free paths among obstacles can be reduced to a graph search problem in Voronoi diagram [7]. Voronoi diagram has been an active research area in computational geometry. Much of the earlier works have constructed the Voronoi diagrams for discrete points. In this simple case, the Voronoi edges are line segments. There are also many algorithms designed for a collection of line segments and circular arcs [7], where the Voronoi edges are conic curves. However, for general algebraic curve segments the corresponding Voronoi edges many have very high degree algebraic curves. For example, the bisector curve for two irreducible algebraic curves of degree d can have degree $2 \cdot d^4$ in the worst case. This means the bisector curve for two simple conic (resp. cubic) curves may have degree 32 (resp. 162), which is extremely high for practical applications. For high degree bisector curves, it is also computational quite expensive to recognize the exact portions of bisector curves which belong to the Voronoi diagram. Thus, in this paper we suggest a simple approximation method to compute the Voronoi diagram for 2D curved objects bounded by algebraic curve segments.

Most of the previous algorithms for Voronoi diagrams are mainly concerned about designing asymptotically efficient algorithms with optimal time complexity $O(n \log n)$ [6, 7], where n is the input size. However, it is not straight-forward to implement these algorithms. The data structures and algorithms even for the simple case of planar n discrete points are quite involved for implementation [6]. Further, when the input is given as line segments and circular arcs, the corresponding Voronoi edges may have conic curve segments [7]. This fact can be explained by the following simple observation. When we assume each object is growing with respect to a certain distance criteria, say Euclidean distance. After time t, each linear edge is translated towards its normal direction by a distance t, and each vertex is grown into a circular arc of radius t centered at the vertex. When we visualize this fact in the xyt-space by lifting each object boundary at time t by a vertical distance t, each line segment generates a planar surface patch and a vertex generates a conic surface patch both making slope $\pi/4$ with the xy-plane. Since each Voronoi edge is defined to be the trace of intersection points between two such growing object boundaries, each Voronoi edge is the xy-projection of an intersection curve between two such planar/conic surface patches in the xyt-space. Thus, it is a conic plane curve of degree 2.

To make the Voronoi edges as simple line segments, Canny suggested a simplified Voronoi diagram which grows each polygonal object boundary by a similar polygonal boundary [4]. In this case, there are only planar surface pathes in the xyt-space and the intersection curves and their projections are all line segments. However, polygon vertices with acute inner angles may generate disconnected Voronoi regions and the corresponding Voronoi edges may have large deviations from the exact Voronoi edges under Euclidean metric, see Figure 4. To reduce these undesirable effects, we round each sharp corner vertex by a convex polygonal arc with line segments with no length and use a construction method similar to Canny's simplified Voronoi diagram [4]. The case of Euclidean metric is an extreme case where each vertex is rounded by a convex polygonal arc with infinitely many null line segments with no length. Replacing each vertex by such a convex polygonal arc essentially adds many extra dummy verices and thus increases the vertex inner angles larger.

In this paper, we assume the given objects are all convex. For non-convex planar curved objects, we can easily compute their convex hulls in linear time [1]. Our goal of this paper is to present a simple algorithm which can be implemented quite easily. Though its worst case time complexity is $O(n^2)$, its performance in practical applications can be significantly improved by adding a reasonable range searching algorithm to detect the edge pairs which can generate the Voronoi edges. As explained above, the main idea is to transform the problem of Voronoi diagram construction to a more general problem of computing the union of certain polyhedra. Thus, in an application programming environment with a solid modeler, one can easily construct Voronoi diagrams using this algorithm without struggling with much hassles of implementing a non-trivial geometric algorithm.

The rest of this paper is as follows. In §2 we briefly review the equations for bisector curves and their degree complexity. In §3 we describe the primitive data structures for implementation. In §4 we present the algorithm to construct an approximate Voronoi Diagram for planar objects bounded by algebraic curves. Finally, in §5 we conclude this paper. This algorithm is implemented on SUN4/330GX Sparc Station using C for the objects bounded by cubic Bezier curve segments.

2 Bisector Curves

In this section, we analyze the degree complexity of the bisector curves by deriving their defining equations. Due to the high degree complexity, it seems reasonable to approximate the bisector curves by polygonal chains or lower degree curves.

2.1 Bisector for an Algebraic Curve

Let C be an algebraic curve defined by an implicit equation $f = 0$, then the bisector of C is the set of points which are equidistant from two different points on C. Suppose p is equidistant from p_1 and p_2 on C, i.e., $d(p,p_1) = d(p,p_2) = r$ for some $r > 0$, then the circle of radius r with center at p is tangent to C at p_1 and p_2. Note that p is a singular point of the constant radius offset curve of C with respect to an offset radius r. The algebraic equation $F(x,y,r) = 0$ for this offset curve can be derived from [2]. Since p is a singular point of the curve, $p = (x,y)$ satisfies $F = F_x = F_y = 0$. By eliminating the variable r from any two of these three equations, we can derive three algebraic equations $S_r(F,F_x), S_r(F,F_y), S_r(F_x,F_y)$, where S_r is the Sylvester resultant eliminating the variable r. The bisector curve of C satisfies these three S_r equations simultaneously and thus the common factor of the three S_r's. Since F has algebraic degree $O(d^2)$ and S_r's have degree $O(d^4)$, the bisector of C may have degree $O(d^4)$ in the worst case.

2.2 Bisector for Two Algebraic Curves

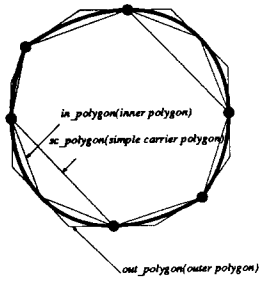
Let C and D be two different irreducible algebraic curve segments defined by implicit equations $f = 0$ and $g = 0$ respectively, then the bisector of C and D is the set of points which are equidistant from two different points p_1 and p_2 on C and D respectively. The bisector equation is given by the following equation. Thus, the bisector can have degree $O(d^4)$ in the worst case.

$$\begin{cases} f(x_1,y_1) = 0 \text{ and } p_1 = (x_1,y_1) \in C & (1) \\ g(x_2,y_2) = 0 \text{ and } p_2 = (x_2,y_2) \in D & (2) \\ f_x \cdot \beta_1 - f_y \cdot \alpha_1 = 0 & (3) \\ g_x \cdot \beta_2 - g_y \cdot \alpha_2 = 0 & (4) \\ \alpha_1^2 + \beta_1^2 = \alpha_2^2 + \beta_2^2 & (5) \end{cases}$$

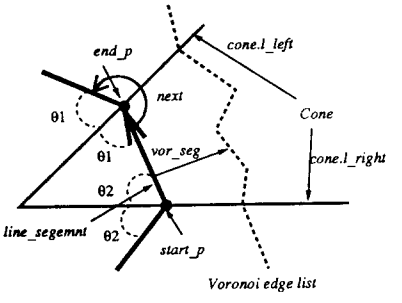
3 Data Structures

Each planar curved object has a boundary representation with edges of algebraic curve segments. Each edge has its begin and edge points, and its defining implicit and/or parametric equation(s). Each point is given as a pair of x and y coordinates. Further, we assume each edge is subdivided into monotone curve segments by adding singular, inflection, and x and y-extreme points as extra vertices if necessary.

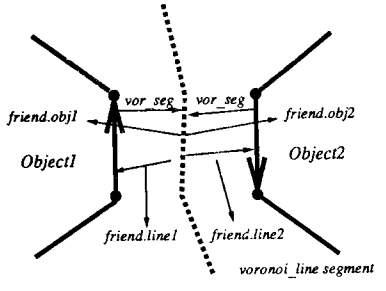
Monotone curve segment is approximated by a polygonal chain of line segments. Each line segment is represented in a data structure with various fields. These include those fields such as the unique *id*, the line equation, the vertex coordinates, a pointer to its corresponding Voronoi edges (*vor_seg*) and a pointer to the conic angular region (*cone*) determined by this line segment and the two neighboring line segments. The *cone* structure is very important because it is the region generated by the line segment growing towards its normal direction by distance t at time t. Vor_seg (Voronoi segment) is a data structure representing a Voronoi edge. It contains the fields for the Voronoi edge *id*, the starting (*start_t*) and ending (*end_t*) times, and the Voronoi edge line equation (*vline*), the starting and ending point coordinates for the edge (*start_p*, *end_p*), and the informations for the objects generating the Voronoi edge (*friend*), etc. The data structure for objects contains additional fields for the inner and outer approximating polygons when each edge of the object is approximated by line segments within an error bound ϵ. Figure 1 shows each field of the data structures. The data structures in C are shown below.



(a) The Field of Object



(b) The Field of Line Segment



(c) The Field of Vor_seg and Friend

Figure 1: Data Structures and its Field

```
struct line_seg
{
    int id;
    LineEq line;
    Cone cone;
    Point *start_p, *end_p;
    Vor_seg *vor_seg;
    Line_seg *next;
};

struct vor_seg
{
    int id;
    float start_t, end_t;
    LineEq vline;
    Point *start_p, *end_p;
    Friend friend;
    Vor_seg *next;
};

struct friend
{
    Object *obj1, *obj2;
    Line_seg *line1, *line2;
};

struct cone
{
    Vector v_right, v_left;
    LineEq l_right, l_left;
    float sign;
};
```

4 Algorithms to Approximate Voronoi Diagrams

In this section, we present algorithms to approximate the exact Voronoi diagrams for planar curved objects by simplified Voronoi diagrams consisting of linear Voronoi edges. In §4.1, we approximate each curved object by certain simple polygons. In §4.2, we compute the simplified Voronoi diagrams for the simple polygons approximating the given objects. To improve the approximation, we subdivide the normal angular region for each vertex by adding redundant coincident vertices into the same vertex. We call this operation as *vertex cracking* and the effect is the same as rounding each vertex by a convex polygonal arc of dummy line segments of length zero. In §4.4, we approximate each monotone polygonal chains of linear Voronoi edges by Bezier cubic curve segments.

4.1 Related Polygons

The *carrier* polygon for a planar curved object is the polygon which is obtained by replacing each curved edge by a line segment connecting the two adjacent vertices [5]. By adding $\Theta(n^2)$ extra vertices if necessary, the carrier polygon can always be made to be simple so that there is no intersection between polygon edges except adjacent edges touching at their common vertex [1, 5]. The *characteristic* polygon is a simple carrier polygon which satisfies an additional property, i.e., the region bounded by each polygon edge and its corresponding curved edge is totally contained either in the object interior or in the exterior. Further, the *inner* (resp. *outer*) polygon is a simple polygon which is totally contained in (resp. containing) the object. Bajaj and Kim [1] presented algorithms to construct the *characteristic*, *inner*, and *outer* polygons for planar curved objects.

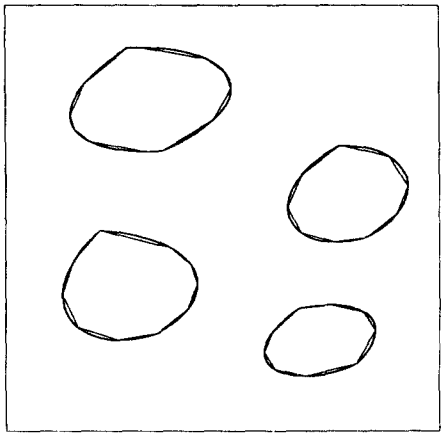


Figure 2: Inner Polygon of the Curved Object.

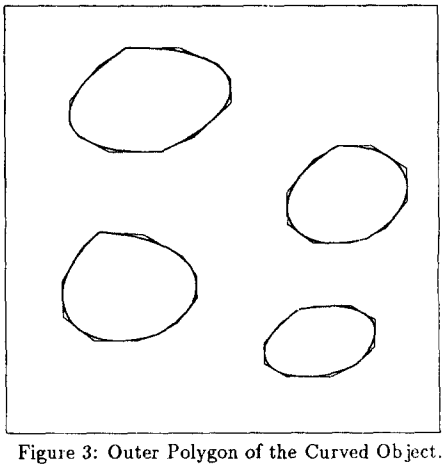


Figure 3: Outer Polygon of the Curved Object.

In this paper, we consider only the planar curved objects which are convex. Thus, all the carrier polygons become simple carrier, characteristic, and inner polygons automatically. Further, they can be constructed quite efficiently in linear time. We can also construct the *outer* polygon using the tangent lines at the vertices. We can refine the *inner* and *outer* polygons even further to approximate the given curved object boundary arbitrarily closely by adding more extra vertices to the curved edges if necessary.

4.2 Simplified Voronoi Diagrams

In this section, we present a simple algorithm to construct the Voronoi diagram for non-overlapping convex polygons. Once we compute the *inner* and *outer* polygons for each convex curved object, we can approximate the Voronoi diagram of these curved objects by computing the Voronoi diagram of the corresponding *inner* or *outer* convex polygons. To make the Voronoi edges constructed to be simple line segments, we can use Canny's simplified Voronoi diagram construction to these convex polygons

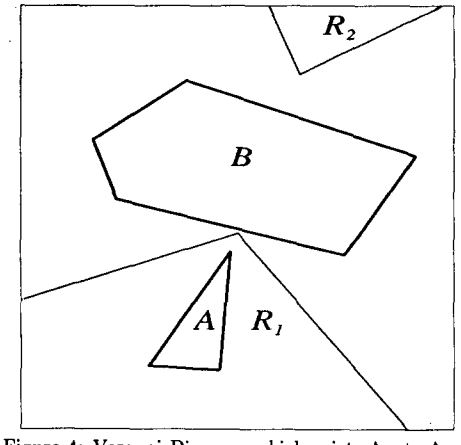


Figure 4: Voronoi Diagram which exists Acute Angle.

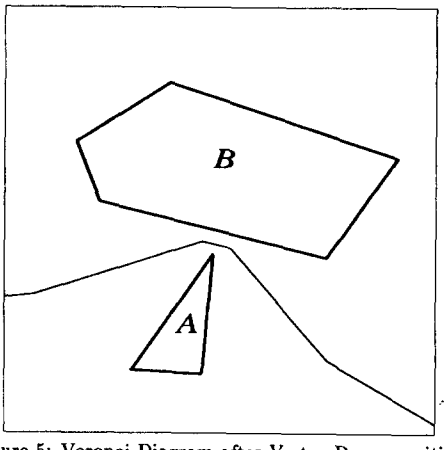


Figure 5: Voronoi Diagram after Vertex Decompositions.

[4]. The simplified Voronoi diagram grows each convex polygon by simply translating each polygon boundary side outwards and constructing a similar larger polygon. However, this method has the following problems. First, a vertex with acute angle may grow too fast compared to the growth of its neighboring edges. Figure 4 shows this problem. In the figure, we can see much difference between the exact Voronoi diagram and the simplified Voronoi diagram. Second, by a similar reason there may appear *islands*. In Figure 4, the convex object A has two disconnected Voronoi regions R_1 and R_2. The disconnedted region R_2 is generated since the sharp corner vertex of A grows too fast. To overcome these problems we use a method called by *vertex-cracking*. In this method, we make the angle of a vertex larger by inserting a convex polygonal arc consisting of extra dummy vertices and null edges at the acute vertex. Then, we can somewhat reduce the undesirable effects of the above two problems. Figure 5 is the same as Figure 4 except that we have constructed the corresponding simplified Voronoi diagram after the *vertex-cracking*. It contains no islands and further it has become very close to the exact Voronoi diagram.

We next consider how to compute the bisector line for two lines defined by the normalized equations $a_1x+b_1y+c_1 = 0$ and $a_2x + b_2y + c_2 = 0$, where $a_1^2 + b_1^2 = a_2^2 + b_2^2 = 1$. After translating these lines by distance t towards their normal directions (a_1, b_1) and (a_2, b_2) respectively, the translated lines satisfy the equations $a_1x + b_1y + c_1 - t = 0$ and $a_2x + b_2y + c_2 - t = 0$ respectively. The points on the bisector line are the set of points which satisfy these two line equations simultaneously at some time t. Thus, the bisector line equation is obtained by eliminating the time papameter variable t from the above two equations, i.e, $(a_1 - a_2)x + (b_1 - b_2)y + (c_1 - c_2) = 0$

For each edge, the *cone* of this edge is defined to be the region bounded by the edge and the two bisector lines for the two outer angles of the vertices. The *cone* corresponds to the sweeping area of the growing edge. A simplified Voronoi edge line segment generated by the two edges from two distinct polygons is the same as an exact Voronoi edge. For this edge, there should be some intersections between the cones of the original edges and the computed bisector line segment. That is, if there is any non-empty intersection, the intersection is an exact Voronoi edge. If not, the two original edges generate no Voronoi edge. A single edge may generate several different Voronoi edges. When we compute all the intersections among them and cut away some redundant parts of them, we can construct a polygonal chain of Voronoi edge line segments. By "classify and select exact Voronoi edges" in the following algorithm, we mean the above process.

```
procedure construct_Voronoi_diagram(object_list, ε)
begin
  { each object in the object_list is a convex polygon. }
  { ε is the maximum permissible deviation of
    the approximated Voronoi diagram from
    the exact Voronoi diagram.}
  Select the vertices whose the inside_angle ≤ π/2 and
  Mark the vertices and all the lines in object_list
  repeat
    { do vertex-cracking on the Marked vertices
      and Mark the corresponding edges. }
    vertex_crack(object_list)
    for each object in object_list
    begin
      for each Marked edge1 in object
      begin
      { when we use an efficient range search
        algorithm to detect only the edges
        which can actually generate Voronoi
        edges, the performance of this algorithm
        can be improved significantly }
        for each edge2 in the rest object in object_list
        begin
          compute the bisector line segment between
            edge1 and edge2
            and push it to Voronoi edge stack
          error = the approximation error
          if error ≥ ε then Mark
            on both vertices of edge1
          else unMark edge1
        end for
        pop all the edges from the Voronoi edge stack,
          classify and select the Voronoi edges
      end for
    end for
  until no more Marked vertex left
end procedure
```

Each Voronoi edge generated by the above procedure has pointers to the object edges which generate it. Figure 6 shows a Voronoi diagram constructed from a set of convex polygons.

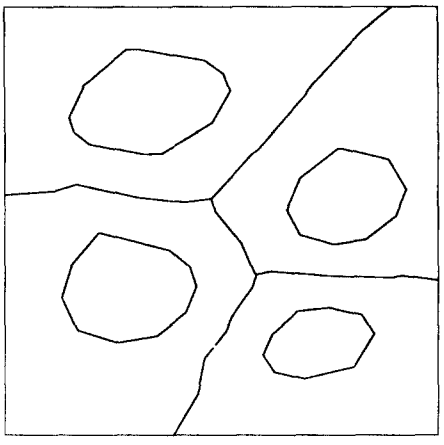


Figure 6: Voronoi Diagram Constructed from Convex Polygons.

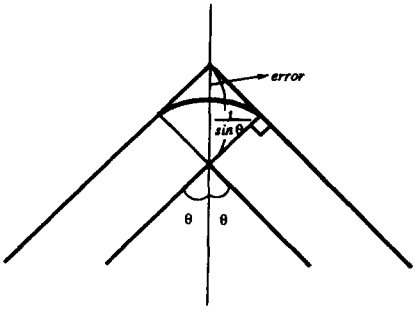


Figure 7: Error in Object Extension.

Each Voronoi edge generated by the above procedure has pointers to the object edges which generate it. Figure 6 shows a Voronoi diagram constructed from a set of convex polygons.

4.3 Further Refinements

Up to now, we have computed the simplified Voronoi diagram which approximates the exact Voronoi diagram under Euclidean metric. The formula for simplified Voronoi edges are explained in §4.2. In the exact Voronoi diagram, each vertex grows into a circular arc, but in the simplified Voronoi diagram it grows into a convex polygonal arc of line segments. Figure 7 shows the error at a vertex. When an object is grown upto time t, the maximum distance error must occur at a vertex. The distance between the edge and the extended edge is always t. However, the distance between the vertex and the extended vertex is $\frac{t}{\sin\theta}$, where 2θ is the inner angle of two neighboring edges. Thus, the maximum extension error between the two vertex growings, i.e, the circular arcs and the convex polygonal arcs, is:

$$Error = \frac{t}{\sin\theta} - t = t \cdot (\frac{1}{\sin\theta} - 1)$$

This equation means that the error term for a vertex growing depends on the angle θ and the time t. To get a Voronoi diagram which has its permissible maximum error within ε compared to the exact Voronoi diagram, we must replace each vertex which has sharp inner angle into a convex polygonal arc of many null edges.

4.4 Curve Fitting

The simplified Voronoi diagram algorithm for the inner or outer polygon yields a list of Voronoi edge line segments. The chain of Voronoi edges is a polygonal chain of line segments which has much difference from the exact Voronoi diagram, especially when the object boundary is curved. The exact Voronoi edges for curved objects are very high degree algebraic curves. Thus, we do curve fittings to the polygonal chains of Voronoi edge line segments to obtain a Voronoi diagram with a smaller number of Voronoi edge segments. We do curve fitting, by using the least squares method. The following is a description of the algorithm. Figure 9 and Figure 10 show the Voronoi diagram for *inner* polygons, and *outer* polygons respectively.

```
procedure curve_fit(voronoi_edge_list)
begin
      divide voronoi_edge_list at each
        intersection point of three Voronoi edges
        {⟹ see Figure 8}
      for each divided voronoi edge list
      begin
          do curve fit using least squares method

      end
end procedure
```

5 Conclusion

We presented a simple algorithm to compute an approximate Voronoi diagram for planer curved objects. Each curved object is first approximated by *inner* and *outer* polygons, and the simplified Voronoi diagrams are constructed for these convex polygons. Though the simplified Voronoi diagram has an important advantage of having only simple line segments, there are some undesirable effects caused by sharp corner with acute inner angles. To eliminate these undesirable effects, we use a method called *vertex-cracking* which essentially rounds each acute corner vertex with a convex polygonal arc of dummy null edges. The simplified Voronoi diagram constructed for these modified simple polygons approximate the exact Voronoi diagram more closely.

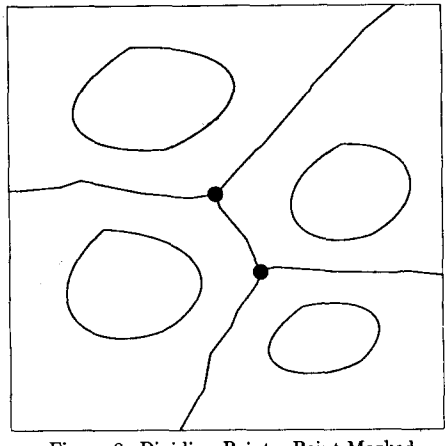


Figure 8: Dividing Point : Point Marked.

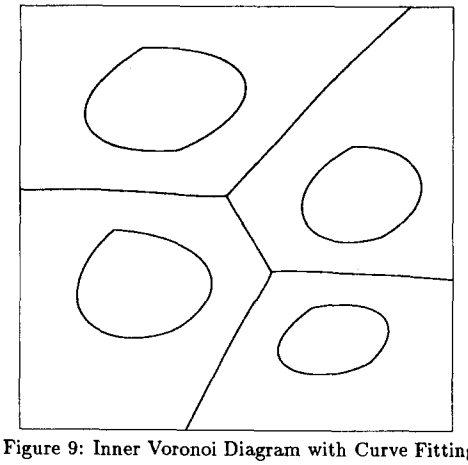


Figure 9: Inner Voronoi Diagram with Curve Fitting.

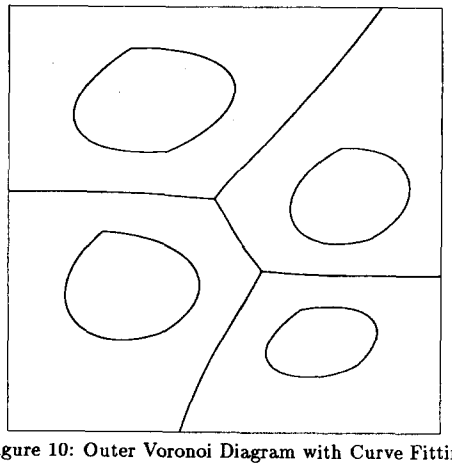


Figure 10: Outer Voronoi Diagram with Curve Fitting.

References

[1] Bajaj, C., and Kim, M.-S., (1988), "Algorithms for Planar Geometric Models," *Proceedings of the 15th International Colloquium on Automata, Languages and Programming (ICALP 88), Tampere, Finland, Lecture Notes in Computer Science*, Springer-Verlag, pp. 67–81.

[2] Bajaj, C., and Kim, M.-S., (1989), "Generation of Configuration Space Obstacles: The Case of Moving Algebraic Curves," *Algorithmica*, Vol. 4, No. 2, pp. 157–172.

[3] Bajaj, C., and Kim, M.-S., (1990), "Convex Hulls of Objects Bounded by Algebraic Curves," to appear in *Algorithmica*.

[4] Canny, J., (1987), *The Complexity of Robot Motion Planning*, The MIT press, pp. 128–147.

[5] Dobkin, D., Souvaine, D., and Van Wyk, C., (1988), "Decomposition and Intersection of Simple Splinegons," *Algorithmica*, Vol. 3, pp. 473–486.

[6] Preparata, F., and Shamos, M., (1985), *Computational Geometry: An Introduction*, Springer-Verlag, New York.

[7] Yap, C., (1987), "An $O(n \log n)$ Algorithm for the Voronoi Diagram of a Set of Simple Curve Segments," *Discrete and Computational Geometry*, Vol. 2, No. 4, 1987, pp. 365–393.