Voronoi Diagrams of Polygons: A Framework for Shape Representation.

Niranjan Mayya[1]
V. T. Rajan[2]

UF-CIS-TR-93-29

[1]Department of Computer & Information Sciences
University of Florida, Gainesville, Fl 32611

[2] IBM T. J. Watson Research Center
P.O Box 218, Yorktown Heights, New York, NY 10598.

Conditionally accepted by
the Journal of Mathematical Imaging & Vision

Conditionally accepted for publication in the
Journal of Mathematical Imaging & Vision

Voronoi Diagrams of Polygons: A Framework for Shape Representation *

Niranjan Mayya[1] and V. T. Rajan[2]

[1]Department of Computer & Information Sciences
University of Florida,
Gainesville, FL 32611

[2] Manufacturing Research Department,
IBM Thomas J. Watson Research Center,
P.O. Box 218, Yorktown Heights, New York 10598

Abstract

This paper describes an efficient shape representation framework for planar shapes using Voronoi skeletons.

This paper makes the following significant contributions. First a *new algorithm for the construction of the Voronoi diagram of a polygon with holes* is described. The main features of this algorithm are its *robustness* in handling the standard degenerate cases (colinearity of more than two points; co-circularity of more than three points.), and its *ease of implementation*. It also features a robust numerical scheme to compute non-linear parabolic edges that avoids having to solve equations of degree greater than two. The algorithm has been fully implemented and tested on a variety of test inputs.

Second, the Voronoi diagram of a polygon is used to derive *accurate and robust skeletons for planar shapes*. The shape representation scheme using Voronoi skeletons possesses the important properties of connectivity as well as Euclidean metrics. Redundant skeletal edges are deleted in a *pruning step* which guarantees that connectivity of the skeleton will be preserved. The resultant representation is stable with respect to being invariant to perturbations along the boundary of the shape. A number of examples of shapes with and without holes are presented to demonstrate the features of this approach.

*Also exists as IBM Research Report RC 19282, 11/23/93

Please address all correspondence to:

Niranjan Mayya

Computer and Information Sciences Department

CSE Building, Room 301

University of Florida

Gainesville, FL 32611-2024

E-mail: nrm@cis.ufl.edu

Tel: (904) 392-1227

Fax: (904) 392-1220

KEYWORDS: Voronoi Diagrams of Polygons, Skeletons, Medial Axis Transform, Shape Representation.

1 Introduction

The representation and description of planar shapes or regions that have been segmented out of an image are early steps in the operation of most Computer Vision systems; they serve as a precursor to several possible higher level tasks such as object/character recognition.

The symmetric or medial axis transformation (SAT, MAT), also known as the *skeleton* is a well known tool used in shape modelling. Pavlidis [20] provides a formal definition: "Let R be a plane set, B its boundary, and P a point in R. A nearest neighbor of P on B is a point M in B such that there is no other point in B whose distance from P is less than the distance PM. If P has more than one nearest neighbor, then P is said to be a *skeletal point* of R. The union of all skeletal points is called the skeleton or medial axis of R." An analogous definition is given by Blum using the well-known prairie fire analogy [2]; if one applies fire to all the sides of P, and let the fire propagate at constant speed, then the skeleton is the locus of points where fire wave fronts meet.

The skeleton of a planar object relates the internal structure of the object to significant boundary features. It is a compact descriptor of the "natural" shape of an object, that well describes its global *topological and geometric* properties. Skeletons have been widely used in higher level computer vision tasks such as object/character recognition, as they provide a more explicit, compact and stable representation as compared to the original intensity map of an image. Another important feature exhibited by skeletons is that the compact thin line representation is invariant to minor distortions on the contour, which allows for stable representation schemes.

Skeletons have been widely used to solve problems in object recognition [17], to model and characterize the geometry of nonrigid biological forms [5, 11], in character recognition [21] among others.

1.1 Algorithms for Skeletonization

Most implementations for computing the skeleton to date use discrete space concepts that only approximate Blum's definition. In particular, *preserving important properties such as connectivity and Euclidean metrics have been difficult to achieve in the discrete world.* The various algorithms and implementations that have been published to date can be classified into three different groups [11, 17].

- Topological thinning.
- Medial Axis extraction from a distance map.
- Analytic computation of a skeleton based on an approximation of the object contour.

Thinning Algorithms: A large class of thinning algorithms examine the topological relevance of object pixels rather than the metric properties of the shape. Typically, object pixels are repetitively tested and subsequently deleted, whenever their removal does not alter the topology of the thinned shape. The advantage of this approach to skeletonization is that they ensure connected skeletons using fast algorithms. *However the prime disadvantage is that the discrete domain gives rise to non-Euclidean metrics.* Different thinning algorithms applied on the same image can result in skeletons that vary. These problems are in part due to different pixel "removal conditions" that are defined in terms of local configurations.

Medial Axis extraction from a distance map: The second method of skeleton extraction requires the computation of a distance map - that is to determine, for each point inside the object, the distance of the closest point from its boundary. Depending on the metric used for distance, a wide array of possible distance maps can be obtained. Unfortunately, the easiest and simplest algorithms are those based on non-Euclidean metrics, which lead to skeletons that are not very accurate in terms of the fire front paradigm. However, algorithms to compute correct Euclidean distance maps exist [3]. Skeletonization algorithms using quasi-Euclidean and Euclidean maps follow ridges in the distance map to obtain the skeleton. *The problem with this method is that connectivity is not guaranteed* [11].

Analytic computation of the symmetric axes: The third method involves computation of the symmetric axes by a direct analytical method based on the polygonal approximation of a shape. Early work using this approach was by Montanari [15] who solves systems of linear equations to compute loci of equidistant points.

The Voronoi diagram is a useful geometric structure which contains the entire planar proximity information of a set of points [1]. This allows for easy computation of the distance map from the Voronoi diagram. Further, the Voronoi diagram of the boundary line segments of a polygon is closely associated to its medial axis. *In fact, the medial axis is exactly contained in the set of Voronoi edges of the polygon, and is obtained by deleting the two Voronoi edges incident with each*

concave vertex [10]. Thus construction of the Voronoi diagram is one technique for skeletonization of polygonal shapes.

1.2 Skeletons Derived from Voronoi Diagrams

Using the Voronoi diagram to compute the skeleton of a polygonal shape is attractive because it results in skeletons which are connected while retaining Euclidean metrics. Furthermore, we obtain an exact medial axis, compared to an approximation provided by other methods. Thus we may reconstruct exactly an original polygon from its skeleton, (invertibility, one-to-one mapping). Finally, algorithms to compute the Voronoi diagram (and hence the skeleton) are much faster than approaches that compute a distance map.

However, we observe that there are some disadvantages of using Voronoi diagrams to derive skeletons. We suggest that any method that utilizes Voronoi diagrams of polygons to compute skeletons must overcome the disadvantages listed below, before it can be of practical value.

- Natural shapes are non-polygonal. Thus, accurate polygonal approximations of such shapes are required in order to compute skeletons without loss of accuracy.
- The skeleton of a many sided polygon (of very short sides) will have a large number of redundant edges because of the Voronoi edges at these vertices. This results in an increase in the complexity of the skeleton, without significant addition of any shape information.
- Finally, robust and practical algorithms (those affording ease of implementation) for Voronoi diagram construction of polygons are not very common. Most existing algorithms make assumptions about cocircularity of no more than three points, and colinearity of no more than two. These constraints are difficult to satisfy in most practical applications.

1.3 Voronoi Skeletons for Character Recognition

In this paper we present an efficient shape representation framework based on Voronoi skeletons. The basis of our approach is a new algorithm for computing the Voronoi diagram of a polygon. *Our skeletonization algorithm retains the advantages of Voronoi diagrams described in the previous section, (connectivity, Euclidean metrics and high accuracy). This approach is thus a marked*

improvement over traditional skeletonization methods. Furthermore we avoid the pitfalls of Voronoi skeletons, by overcoming the disadvantages identified in the previous section.

The original contributions made by this paper may be enumerated as follows:

1. *A new algorithm for the construction of the Voronoi diagram of a polygon with holes.* This algorithm is *practical* and *robust.* Our Voronoi diagram algorithm is simple to implement. *Assumptions about points being in general position are unnecessary.* Unlike most geometric algorithms, special cases such as cocircularity of greater than three points or colinearity of more than two points are handled elegantly. In addition, our algorithm features a robust numerical scheme to compute non-linear parabolic edges that avoids having to solve equations of degree greater than two.

2. *An accurate scheme for skeletonization that retains the advantages of Voronoi skeletons while overcoming its unattractive features.* In particular, the scheme provides skeletons that maintain Euclidean metrics without sacrificing connectivity. The method is stable with respect to being invariant to perturbations along the boundary. This is achieved by pruning redundant or spurious edges that do not significantly add to shape information. We will see that the pruning step is a simple consequence of the Voronoi diagram algorithm and does not require any postprocessing. *Further, our approach handles shapes with and without holes in a seamless manner, without resorting to special initialization.*

The rest of this paper is organized as follows. In Sections 2 and 3 we present our algorithm for the computation of the Voronoi diagram of a polygon. and discuss its time complexity. In Section 4 we present the details of the skeletonization scheme and present various examples to demonstrate its features. Our conclusions and future research directions are presented in Section 5.

2 The Voronoi Diagram

The Voronoi diagram of a set of sites in two dimensions is the partition of the plane into regions; each region i containing the set of points in the plane closest to the site i.

In the most common case which has been exhaustively researched in the past decade, the sites under consideration are points in the plane. In this case, edges of the diagram are straight line

segments that are perpendicular bisectors of pairs of sites. Optimal algorithms as well as robust implementations have been devised for this version of the problem.

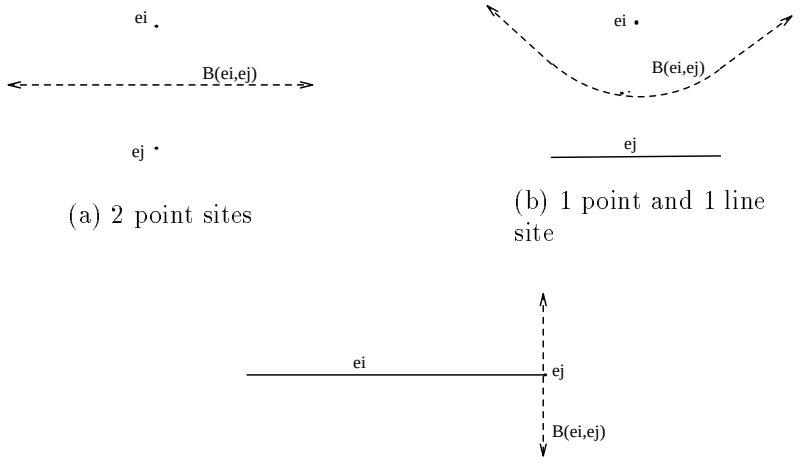
The notion of a site has been generalized to include a collection of two-dimensional objects such as line segments and circular arcs. In the case of line segments, the edges of the ensuing Voronoi diagram are not just straight-line segments, but also arcs of parabola, since they separate the loci of proximity of object pairs of the types $(point, point)$, $(line, line)$, $(point, line)$; the last of these pairs give rise to arcs of parabola. In the case of circular arcs, the Voronoi bisectors can be general conic sections. Much research effort has also been expended in this direction. In particular, Lee and Drysdale [9] have a $O(n \log^2 n)$ time algorithm for the Voronoi diagram of a mixed collection of n objects (including line segments and circles). Kirkpatrick [8] presented the first optimal $\Theta(n \log n)$ solution; Fortune [6] and Yap [22] also have optimal algorithms. Fortune's algorithm computes Voronoi diagrams of line segments; Yap considers simple curve segments. While some of these algorithms are optimal in terms of time complexity, they are not amenable to simple implementation. Some work has been performed in this direction as well; notably the work of Srinivasan and Nackman [18] and Meshkat and Sakkas [14], where the authors present a simple algorithm for the Voronoi diagram of multiply connected polygonal domains (polygons with holes).

In this section we present an algorithm which solves a slightly more general version of the problem than that of Srinivasan and Nackman. While the working examples shown are those of polygons with holes, the algorithm presented here can compute the Voronoi diagram of a Planar Straight Line Graph. The algorithm is easy to implement, and has been fully implemented and tested. Another important feature is its robustness; the algorithm handles the standard degenerate cases inherent in most Computational Geometry algorithms (more than 2 collinear or more than 3 co-circular points). This facilitates its use in most practical applications.

2.1 Preliminaries

In this section, we will introduce some definitions and notation. Definitions 1 through 8 pertain to the definition of polygonal domains and Voronoi diagrams and are taken from [18].

Definition 1 *A closed line segment $[a, b]$ is the union of two endpoints a and b and the open line segment (a, b).*



(a) 2 point sites

(b) 1 point and 1 line site

(c) the point site is an end point of the line site

Figure 1: Bisectors of e_i and e_j

Definition 2 *A multiply-connected polygonal domain P is the closure of a nonempty, bounded, connected, open (in the relative topology) subset of $\mathcal{R}^2$ whose boundary is the union of a finite number of closed line segments.*

The boundary of P, denoted by δP, consists of one or more disjoint subsets. The outer boundary of the polygonal domain is denoted by δP_0 and contains P. The inner boundaries represent the holes of the polygonal region and are denoted by δP_i, $1 \leq i \leq H$, where H denotes the number of holes.

Definition 3 *The vertices of δP are the points of intersection of the closed line segments which constitute δP. The edges of δP are the open line segments obtained by deleting the endpoints of the closed line segments which constitute δP*

Definition 4 *The projection $p(q,[a,b])$ of a point q onto a closed segment $[a,b]$ is the intersection of the line through a and b and the line perpendicular to $[a,b]$ and passing through q.*

Definition 5 *The bisector $B(e_i,e_j)$ of two sites e_i, and e_j, is the locus of points equidistant from e_i and e_j.*

Definition 6 *The half-plane $h(e_i,e_j)$ is the set of points closer to site e_i than to site e_j. Its complement, $\bar{h}(e_i,e_j)$, is the set of points not closer to site e_i than to e_j.*

The nature of the bisector is determined by the nature of the sites (point, line). In particular, when both e_i and e_j are point sites, the bisector is a straight line (the perpendicular bisector of e_i and e_j. When one of the sites is a point, and the other is an open line segment the bisector is in general a parabolic arc. For the special case where the point is one of the endpoints of the open line segment, the bisector is a straight line passing through the point and perpendicular to the open line segment. See Figure 1.

Definition 7 *Given a set of sites S, and a site e_i, $e_i \notin S$, the Voronoi region of e_i with respect to S, denoted by $V(e_i, S)$ is the set of all points closer to e_i than to any site in S.*

Lemma 1 $V(e_i, S) = \cap_{e_j \in S} h(e_i, e_j)$

Proof: Corollary 2, Lemma 1 of [18] ■

Definition 8 *The Voronoi diagram, VOD(S), of a set of elements, $S = e_i$ is given by $\cup_{e_i \in S} V(e_i, S - e_i)$*

2.2 Primitives used in the Algorithm

A site e_i can be either a point or an open line segment. Each such site is associated with the following information.

1. **A contact point** x_{p_i}. For a point site, the contact point is the site itself; for a line site, the contact point is one of the endpoints.

2. **A distance function** $d_i(x_0)$; returning the square of the distance of a point x_0 from site e_i. For a line site, the distance of a point to the site is defined as the distance from the point x_0 to its projection on the line site e_i. Given this definition, we can define the distance of a point to any site (either a point or a line) as follows, Given a line site, let $\vec{n_l}$ be its unit vector. If I is the identity matrix, and T denotes the transpose of a matrix, we define a 2×2 matrix M as

$$\begin{aligned} M &= I && \text{for a point site,} \\ &= I - \vec{n_l}\ \vec{n_l}^T && \text{for a line site} \end{aligned}$$

Now we define the distance function as,

$$d_i(x_0) \;=\; (x_0 - x_{p_i})^T \; M \; (x_0 - x_{p_i})$$

3. **A gradient vector** $g(x_0)$; which is defined by.

$$g_i(x_0) = 2(x_0 - x_{p_i})^T \; M$$

4. **the quadratic polynomial** $f_i(x_0 + \vec{v}\tau)$; giving the square of the distance of the point $(x_0 + \vec{v}\tau)$ from the site i. Here $\vec{v}$ is a direction vector, and τ is a scalar multiple. We have

$$\begin{aligned} f_i(x_0 + \vec{v}\tau) &= (x_0 - x_{p_i})^T \; M \; (x_0 - x_{p_i}) + g_i(x_0) \; \vec{v} \; \tau + \vec{v}^T M \; \vec{v} \; \tau^2 \\ &= d_i(x_0) + g_i(x_0) \; \vec{v} \; \tau + \vec{v}^T M \; \vec{v} \; \tau^2 \end{aligned}$$

2.3 Algorithm

2.3.1 Overview

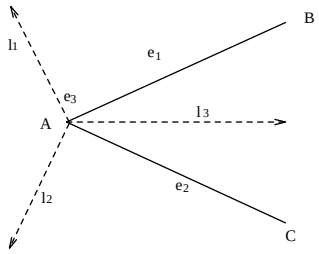
The Voronoi diagram gives us a complete description of the function $t(x)$, that returns the distance of the point x to the closest site in the set S. In particular,

- A Voronoi region (face) is characterized by a single site e_i; the function is given by $t_i(x)$.
- A Voronoi edge is characterized by two sites, e_i and e_j; the edge comprises the set of points where $t_i(x) = t_j(x)$.
- A Voronoi vertex is characterized by 3 or more sites, i, j, k,...m; the vertex satisfies the locus $t_i(x) = t_j(x) = t_k(x) = ...t_m(x)$.

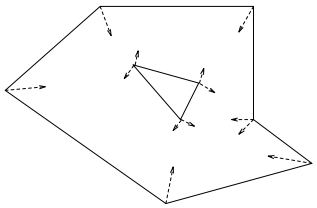
Given an initial Voronoi vertex and the initial direction of the Voronoi edge emanating out of that vertex, we follow the path traced by this edge to determine the vertex at the other end. Every other site is examined to find the closest site that determines the new vertex. The new vertex is equidistant from three or more sites, every pair of which gives rise to a possible new Voronoi edge. The new Voronoi edges are added to an *unexamined Edge List*. The program terminates when all the edges have been traced.

2.3.2 Initialization

Given a pair of successive segments of any polygonal region, this gives rise to the following 3 Voronoi edges. See Figure 2a. BA and AC are 2 successive sides of a polygon. There are 3 sites corresponding to these 2 sides; two line sites e_1 and e_2 corresponding to the open line segments BA and AC, and the point site e_3. The 3 Voronoi edges corresponding to these sites are shown by the dashed lines. l_1 is the bisector of e_2 and e_3, l_2 that of e_1 and e_3, and l_3 the e_1, e_2 bisector. If we are considering the Voronoi diagram of the interior of the polygonal region alone, we are only concerned with those Voronoi edges inside the region. These are easily obtained by considering only the (line,line) bisectors (the l_3 type edges) of every convex pair of successive polygon segments. for the outer boundary of the polygon, and the (point,line) edges (the l_1 and l_2 type edges) for every concave pair of successive polygon segments. For the inner boundaries (the holes), we perform the reverse. Figure 2b shows the complete initialization for a simple case. The vertices of the polygon are also Voronoi vertices. Each of the edges determined in this step are added to the unexamined Edge List.



(a) Voronoi edges of 2 sides of a polygon



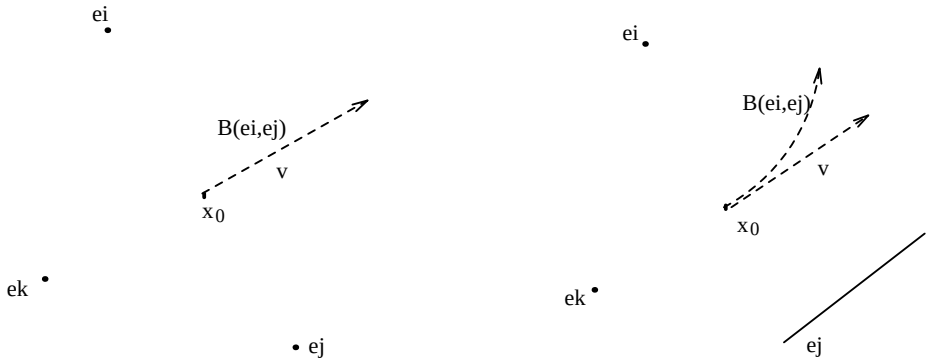
(b) Initialization for a polygon with holes

Figure 2: Initialization

The unexamined Edge List holds edges which have been only partly determined. Specifically, the information known is the starting point, and the initial direction along which we must traverse to determine the other points along the edge. As has been noted earlier, if the Voronoi edge is a $(point, point)$ or $(line, line)$ bisector, it will be a straight line; in which case there is only a single terminating Voronoi vertex to be determined. If we have a $(point, line)$ bisector, the Voronoi edge is a parabolic curve; this curve has to be traced and intermediate points along it computed to fully determine the Voronoi edge. The curve tracing step is the topic of the next section.

2.3.3 Curve Tracing

Consider a Voronoi edge E from the Edge List. The initial starting point is x_0, which is a Voronoi vertex equidistant from 3 or more sites, e_i, e_j, $e_k ... e_m$. Assume, without loss of generality that the edge being traced is a bisector of sites e_i and e_j. The initial direction of the bisector is determined upto the linear order and is given by $\vec{v} = (g_i(x_0) - g_j(x_0))^{\perp}$, where $a^{\perp}$ denotes a unit vector perpendicular to the vector a. For the linear case, the bisector is along the direction of $\vec{v}$. When the bisector is of quadratic order (in the $(point, line)$ case), $\vec{v}$ gives the tangent to the curve at the point x_0. See Figure 3.



(a) $B(e_i, e_j)$ is along the direction of $\vec{v}$

(b) $\vec{v}$ is tangent to $B(e_i, e_j)$ at x_0

Figure 3: Initial Direction for a Voronoi edge

Now consider the function $f_l(x_0 + \vec{v}\tau)$. τ parameterizes points along the vector $\vec{v}$. Recall that f is a quadratic polynomial in τ that gives the square of the distance from a site e_l to the point $(x_0 + \vec{v}\tau)$.

Let $f = \max(f_i, f_j)$. When the bisector is a straight line edge, it is along the direction of $\vec{v}$, and hence in this case $f_i = f_j$. In the non-linear case, f_i and f_j will not be identical beyond the linear term and f represents the greater of the two.

At x_0, $\tau = 0$, and by definition of a Voronoi vertex, we have

$$f_i = f_j = f_k = ...f_m < f_l \qquad \forall\, l \in (S - (e_i, e_j, e_k, ..e_m))$$

We need to trace a path starting at x_0, such that

$$f < f_k, .., f_m, f_l \qquad \forall\, l \in (S - (e_i, e_j, e_k, ..e_m))$$

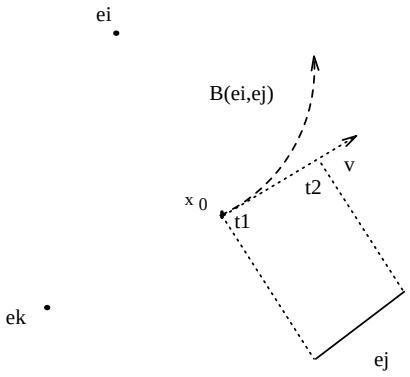


Figure 4: Relevant region for e_j: $\tau \in (t1, t2)$

A new vertex is found when we reach a point where

$$f_i = f_j = f_l \text{ for some site } e_l$$

With respect to every line site e_i is associated a *relevant region* of a given bisector $\vec{v}$ expressed as an interval of τ, determined by the contact points from the bisector to the endpoints of the site. See Figure 4.

We need to ensure that the set of constraints admitted by $f < f_l \; \forall \; l$ is maintained at every point on the bisector of i and j. This is achieved as follows. We first determine the *admissible range* of τ, for each site l. The admissible range is computed as the complement of the *inadmissible range* of τ, which is defined as the region where $f > f_l$ in the relevant region of e_l. Thus

$$T^i_{admissible} = [0, \infty] - T^i_{inadmissible}$$

$T_{admissible_i}$ is computed for every site e_i and the intersection of these regions gives us T_{final}.

$$T_{final} = \cap_{e_i \in S} T^i_{admissible}$$

Note that T_{final} can be a set of intervals, one of which will be $[0, \tau_v]$. This is the interval of interest; 0 corresponds to the initial Voronoi vertex, and τ_v the τ value for the new vertex. The new vertex is given by

$$x_t = x_0 + \vec{v}\tau_v$$

Let e_l be the site that determined the value of $\tau = \tau_v$. Then at x_t, $f = f_l$. When the bisector of e_i and e_j is a line, the new vertex determined by this procedure is in fact a Voronoi vertex corresponding to the sites e_i, e_j and e_l, since, for this case $f_i = f_j$ along $\vec{v}$ and at x_t, we have $f_i = f_j = f_l$.

2.3.4 The Non-linear Case: Tracing Parabolic Edges

In the non-linear case, x_t is a point on the linear approximation of the bisector. In this case, we employ an iterative technique to converge onto the Voronoi vertex at the other end of the curve. Having determined x_t, a point on the linear approximation of the bisector, we need to move back to a point on the curve itself. It is easy to see that we will intersect the bisector $B(e_i, e_j)$ if we move along the contact vector $-g(x_t)$ towards the the point site. Thus, it is easy to determine x_{t1}, an intermediate point along $B(e_i, e_j)$. See Figure 5. In Lemma 2, we prove that this procedure is guaranteed not to overshoot the Voronoi vertex; namely, the open interval of the bisector $B(e_i, e_j)$ between the points x_0 and x_{t1} contains no Voronoi vertex. Having determined an intermediate point on the bisector, the procedure is repeated using a new value for $\vec{v}$ that is given by $\vec{v} = (g_i(x_{t1}) - g_j(x_{t1}))^{\perp}$. The procedure terminates when we arrive at a point x_{tn} which corresponds to a site e_n, such that $f_i = f_j = f_n$. x_{tn} is the new Voronoi vertex.

We note that the rate of convergence of the iteration is the same as that of Newton's method, since we are approximating an arc by a straight line.

Lemma 2 *The open interval of the bisector $B(e_i, e_j)$ between the points x_0 and x_{t1} contains no Voronoi vertex. That is, for this set of points $f_i = f_j < f_k; k \neq i, j$.*

Proof: Consider an arbitrary point x in the open line segment (x_0, x_t). (See Figure 6.) If the bisector is non-linear then one of the sites e_i, e_j will be closer and the other one further from x. Let e_j be the closer site, so that $f_j(x) < f_i(x) = f(x) < f_k(x); k \neq i, j$. Draw a circle of radius $\sqrt{f}$ centered at x. This circle will contain portions of site e_j in its interior, and will touch site e_i and all other sites e_k will lie outside this circle. Shrink this circle such that it continues to touch site e_i and it center continues to lie on the line connecting the point x to the contact point of the circle with e_i until the circle just touches e_j. The center of the circle will move in the direction $-g_i(x)$ and the new center will be the point x_1 which lies on the bisector $B(e_i, e_j)$. The shrunk circle lies in the interior of the original circle and hence all sites $e_k, k \neq i, j$ will continue to lie outside the circle. Hence at the point x_1, $f_i(x_1) = f_j(x_1) < f_k(x_1), k \neq i, j$. Hence x_1 cannot be a Voronoi vertex. Since by choosing all the points in the interval (x_0, x_t) in the open line segment we get every point on the bisector $B(e_i, e_j)$ in the interval (x_0, x_t1) this result holds for the entire interval. ■

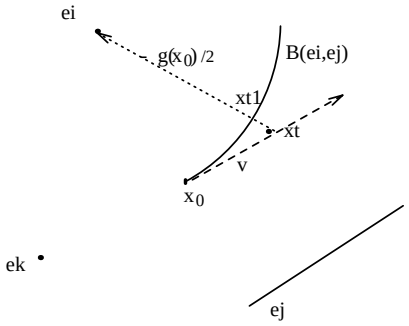


Figure 5: Finding an intermediate point on the curve

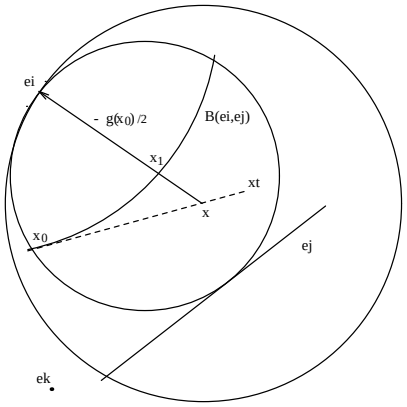


Figure 6: Proof of Lemma 2

2.3.5 Increasing Robustness

While the above curve tracing procedure is guaranteed to converge to a Voronoi vertex. In the non-linear case however, it is possible that as we come closer and closer to the vertex, the very small distances involved in the numerical comparisons give rise to numerical problems that adversely affect robustness. This is particularly true when two Voronoi vertices are extremely close to each other.

The procedure outlined below reduces the number of times we need to employ the curve tracing procedure to obtain parabolic Voronoi edges. The basic idea is to defer the tracing of parabolic Voronoi edges until all (or most) of the Voronoi vertices have been determined. This implies that when we determine the new edges arising out of a newly computed Voronoi Vertex, the non-linear edges are added to the end of the unexamined Edge List and the linear edges are added to the front of the List. Thus most of the linear edges will be examined before the non-linear ones, and consequently most of the Voronoi vertices will be determined before the non-linear edges are traced. The truth of the last statement stems from the fact that *there must be at least one linear*

edge arising out of any Voronoi vertex.

Recall that the unexamined Edge List contains Voronoi edges that have not been completely determined. In particular, only the starting vertex and the direction of the edge is known. Thus, if the two endpoints of a given edge are determined as a result of tracing two different edges that terminate in the respective endpoints, we will have two entries in the unexamined Edge List that represent the same Voronoi edge. The procedure currently followed is to trace a given edge, and if the new vertex determined at the other end of the edge already exists, this implies that there must be another entry in the unexamined Edge List corresponding to the edge being traced.

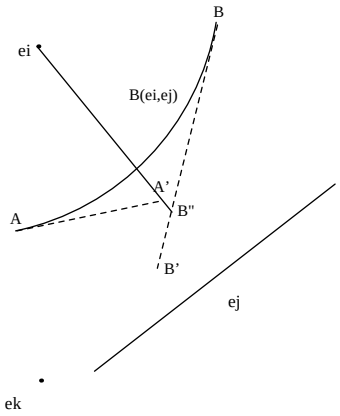
If there are two unexamined parabolic edges that are bisectors of the same pair of sites, each of which start from a different Voronoi vertex, we can avoid the curve tracing procedure if we can confirm that the two vertices are the two endpoints of a single Voronoi edge. Intermediate points on the curve are determined easily by using the same technique used in the curve tracing procedure.

How often will such a case occur - namely, that the two vertices of a parabolic Voronoi edge be determined before the parabolic edge is traced ? Since there must be there must be at least one linear edge arising out of any Voronoi vertex, most of the vertices of the Voronoi diagram can be determined by tracing the straight line edges before the parabolic edges. Tracing a straight line edge is a much simpler (and hence more robust) numerical operation than that for tracing parabolic edges. What remains is to devise a simple procedure to verify that two vertices that are the starting points of two unexamined parabolic edges are the endpoints of a single Voronoi edge. The following Lemma gives us the basis for such a procedure.

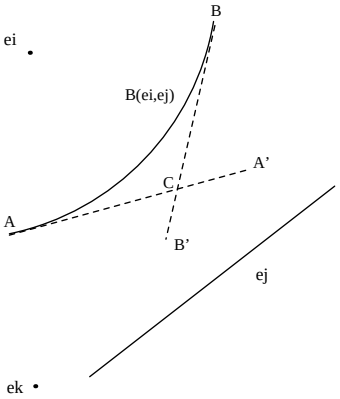
Lemma 3 *Let E_1 and E_2 be two unexamined parabolic edges, corresponding to the same pair of sites e_i and e_j such that E_1 starts from Voronoi vertex A and E_2 from vertex B. Then AB is a single Voronoi (parabolic) edge if the linear approximations from A and B cross each other.*

Proof: Consider Figure 7. Subfigures 7*a*, 7*b* and 7*c* depict the various cases that can arise when iterating from A and B simultaneously.

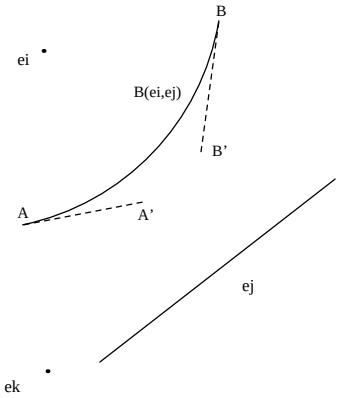
In Case 1 (7*a*), the linear approximations AA' and BB' do not cross, but the line drawn from the point site e_i through A' (respectively B') meets the line BB' (respectively AA') in a point that belongs to the open interval BB' (respectively AA'). Let the line from e_i through A' meet BB' in B''. Then at B'' $f_j < f_i = f < f_k$, for all $k \neq i, j$. From Lemma 2, we know that this relation



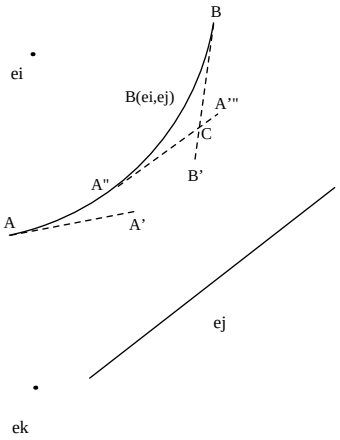
(a) Case 1



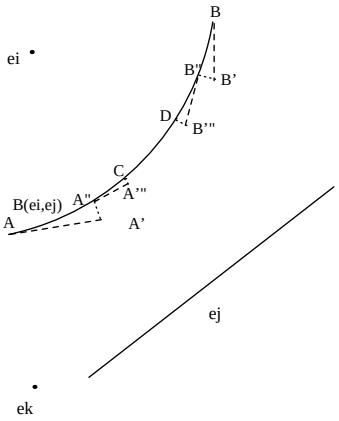
(b) Case 2



(c) Case 3



(d) Case 3a



(e) Case 3b

Figure 7: Figure for Lemma 3

holds as we move towards e_i. Hence at A', we must have $f_j \leq f_i = f < f_k$. But since A' is the linear approximation of the Voronoi vertex starting from A, we must have $f_j = f_k$ at A' for some site k. Thus the linear approximation from A cannot terminate at A'. Hence Case 1 is impossible, and we must have either Case 2 or Case 3.

In Case 2, the two linear approximations cross. Lemma 2 guarantees that we cannot overshoot a Voronoi vertex; hence there can be no Voronoi vertex from BB' or AA' along the curve. Hence AB must be a single parabolic edge. It is possible to determine intermediate points on the curve by projecting back from points on the lines AC and BC to the site e_i.

Finally, it is possible to have a situation as depicted in Case 3. Here the two linear approximations do not intersect. In such a case, we need to repeat the procedure starting at one of the points A' or B'. If the new edge starting from A' crosses the one starting from B, as shown in Case 3a, we have verified that AB is a single Voronoi edge, or repeated iterations may converge to two new Voronoi vertices C and D as in Case 3b, and we have found two Voronoi edges AC and BD. ■

2.3.6 Adding New Edges

Having computed a new Voronoi vertex x_t, we must first perform a check to determine if this vertex has been found before. If this vertex has been computed earlier, it means that there must exist in the Edge List, an unexamined edge with starting point x_t. This edge must be deleted from the Edge List, since it has now been fully determined. If the new vertex has not been computed before, we proceed on determining all the edges coming out of this vertex.

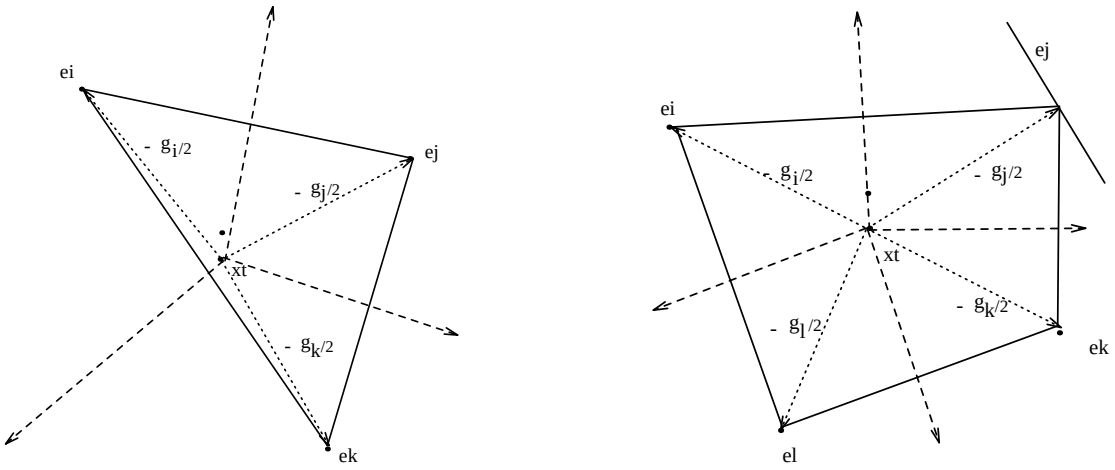
We first need to determine all the other sites equidistant from this vertex, and then determine the new Voronoi edges that emanate out of the new vertex.

The first step is easy to perform; all sites e_l, which satisfy

$$d_l(x_t) = d_i(x_t) = d_j(x_t)$$

lie on the circle of radius $d_l(x_t)$, centered at the new Voronoi vertex x_t.

The second step is to determine all the edges coming out of this vertex. In general, if we have k sites equidistant from a vertex, there can be $\begin{pmatrix} k \\ 2 \end{pmatrix}$ pairs of possible edges, but all of these will not be Voronoi edges. From the vector $g(x_t)$, for each site e_i, we can determine the point of contact



(a) 3 co-circular point sites

(b) 4 co-circular sites with 1 line site

Figure 8: Determining edges from the convex hull of points of contact

of $-g_i/2$ with e_i. If we denote the points of contact as y_i, the convex hull of all the y_i determines which pairs of sites correspond to Voronoi edge. Every convex hull edge, gives a pair of sites that correspond to a Voronoi edge. Thus if we have k equidistant sites from the vertex x_t, there are exactly k Voronoi edges. See Figure 8.

One of these k edges is the edge which was just traced. The other $k-1$ edges are appended to the unexamined Edge List. The head of the Edge List is then examined and the procedure repeats until the List is empty.

Due to the incremental nature of the edge tracing algorithm, we do not need to make any assumptions about points in general position. Hence co-circularity of more than 3 sites can be handled easily.

2.4 Time Complexity

Let the number of sites in the input be n_p, the number of Voronoi edges be n_e and the number of Voronoi vertices be n_v.

The initialization step is of the order of the number of sites in the input, since a single pass over the input suffices to create the initial edges and append them to the unexamined Edge List. This step takes $O(n_p)$ time.

Each edge that is added to the diagram in the edge tracing step requires us to examine each site to determine the closest site. Also, we need to search the list of current Voronoi vertices to check for the existence of a new vertex. Hence, we need $O(n_p + n_v)$ time for each new edge found. The total time required for the algorithm is therefore $O(n_p n_e + n_v n_e)$. By Euler's formula the number of Voronoi edges is $2n_p - 3$ and the number of Voronoi vertices $n_p - 2$. Hence the time complexity is $O(n_p^2)$.

3 Skeletonization using Voronoi diagrams

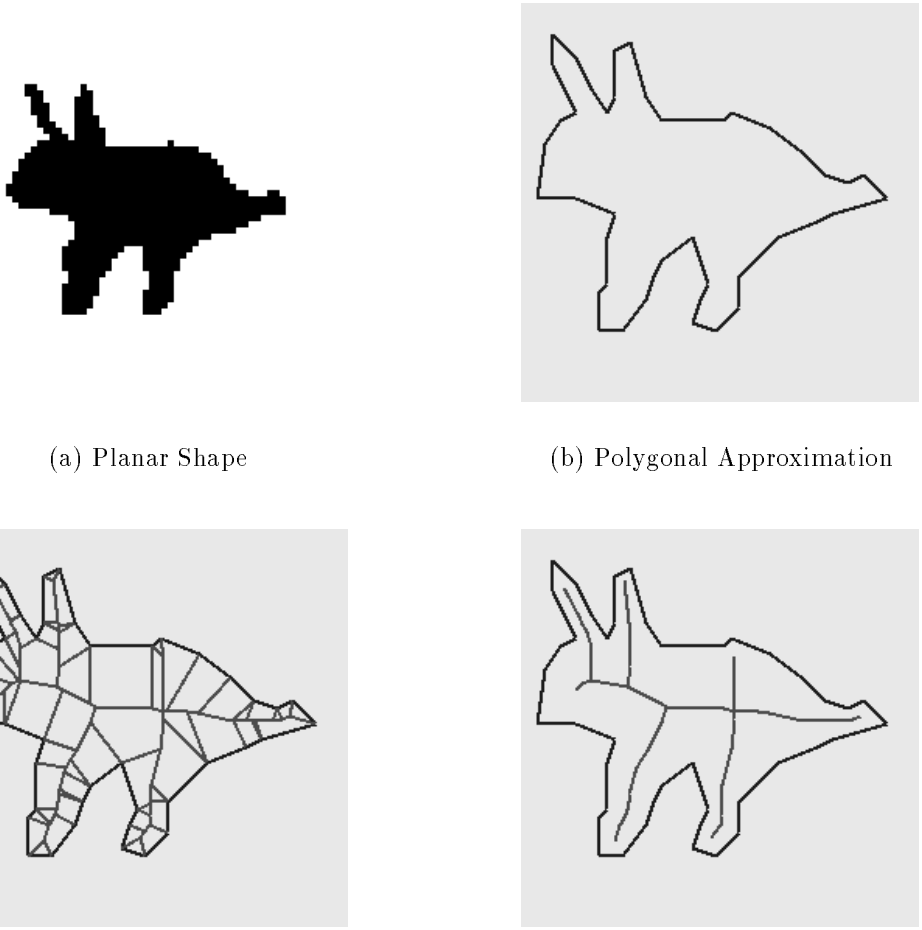
In this section, we describe our technique for obtaining skeletons of planar shapes that are derived from Voronoi diagrams.

3.1 Preprocessing: Computing a polygonal approximation

In order to compute the Voronoi diagram of a shape, we first segment the shape's boundary and derive a polygonal approximation of its bounding contour. In our implementation, we used the zero crossings obtained from an image filtered with a Laplacian of a Gaussian [13], to obtain a bounding contour. The advantage of this technique is that it guarantees closed contours, which ensures that a contour tracing algorithm such as chain coding will converge quickly. From the chain coded bounding contour, critical points (those that reflect significant change in curvature) are retained; these serve as the vertices of the polygonal approximation for each (planar) shape. The input shapes were approximated to ensure that significant changes in curvature would be retained, while minor distortions that could result in noise spurs in the skeleton representation would be smoothed over. Thus, *we limited the occurrence of redundant edges in each representation.*

3.2 Computing the Voronoi Diagram and extracting Skeletons

Given the vertices for a polygonal approximation as determined above, we computed the Voronoi diagram using the algorithm described in Section 2. The medial axis [10] of a polygonal shape can be derived from its Voronoi diagram by deleting those edges that arise from concave vertices of the polygon. (See Section 2.) The data structure we employ for Voronoi edges contained the information required to determine these edges. Hence it is easy to identify (and delete) edges that emerge from concave vertices of a polygon. However, the resultant medial axis is characterized



(a) Planar Shape

(b) Polygonal Approximation

(c) Voronoi Diagram

(d) Pruned Skeleton

Figure 9: Voronoi Skeletonization of Planar Shapes

by a very large number of edges, as every vertex of the polygonal approximation (both concave and convex) gives rise to a Voronoi edge. Many of these Voronoi edges are redundant; i.e. they do not provide any additional structural information about the planar shape. Hence we include a pruning technique to delete these redundant edges.

3.3 The Pruning operator

As shown in Figure 9, the vertices of the initial polygonal approximation of the shape lead to a large number of edges that do not contribute to overall shape information. In a Voronoi diagram every Voronoi edge is a bisector of two sites on the boundary of a polygon; in particular the Voronoi edges at the vertices of a polygon are bisectors of adjacent sites. If the sides of a polygon

are numbered in counterclockwise order along the boundary, we can define the *adjacency* of a Voronoi edge as follows. Let E be a Voronoi edge that is a bisector of 2 sites numbered i and j. Then $Adjacency(E) = |i - j|$. We observe (trivially), that Voronoi edges that lie deep inside an object have higher adjacency than those on the perimeter. Furthermore segments that describe "global" topological (symmetry) relations are bisectors of high adjacency. This fact gives us a means of filtering out unimportant segments. We simply discard edges that are of adjacency lower than some preset threshold. Since the adjacency information is implicitly contained in our Edge data structure (see Section 2), no additional post-processing is required to obtain this reduced set of edges.

We note that the authors of [17] use a similar (but not identical) technique for pruning the edges of Voronoi diagrams. The problem in their case was more critical, since they started out with a set of raster crack end-points along the boundary of an object and hence had an extremely large number of Voronoi edges to process.

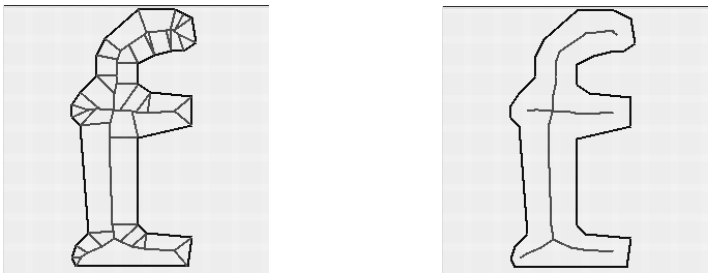
Figure 9 shows the result of the 3 step procedure described above on a planar shape of an animal-like figure. Figure 10 presents more examples of skeletons derived using our technique for skeletonization.

3.4 Preserving Connectivity

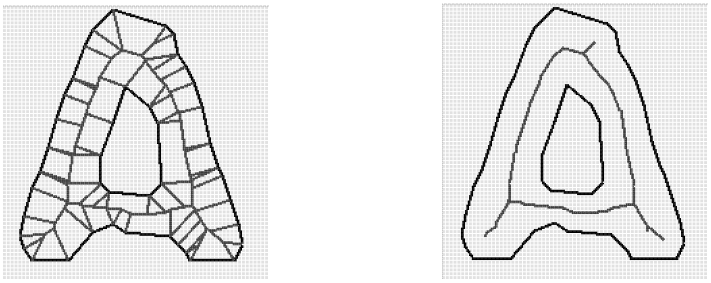
As described in Section 1, one of the advantages of skeletons derived from Voronoi diagrams is that we are guaranteed connected skeletons. However, what is the effect of the pruning step on connectivity ? Furthermore, how does one determine an optimal threshold for deleting the maximum number of redundant skeletal edges without losing connectivity ? Unfortunately, it is not possible to determine a threshold value a priori and apply it to a large class of objects. However, by constraining the threshold value to 1 (retaining only those edges that are of adjacency greater than 1), we can prove that the resultant skeleton remains connected.

Lemma 4 *Pruning Voronoi edges of adjacency 1, is guaranteed not to destroy connectivity.*

Proof: The Voronoi diagram of a polygon is a Planar Straight Line Graph [18], which by definition, is connected. For the removal of an edge to result in a disconnected graph, the edge must be a "bridge", namely the edge must be part of every path between any two vertices of



(a) (b)



(c) (d)

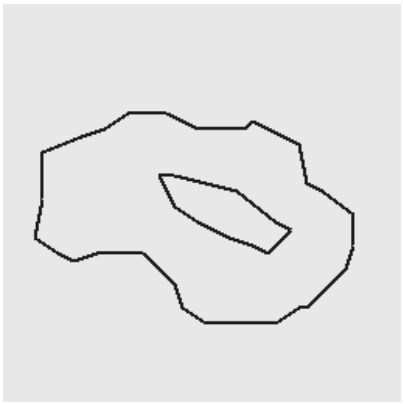
Figure 10: (a),(c) Voronoi Diagrams, (b), (d) pruned skeletons

the graph. It is easy to see that no edge of adjacency 1 can be a bridge, since (by definition of adjacency) every edge of adjacency 1 terminates at a vertex of a polygon. Hence removal of all edges of adjacency 1 is guaranteed to preserve connectivity of the resulting skeleton. ■

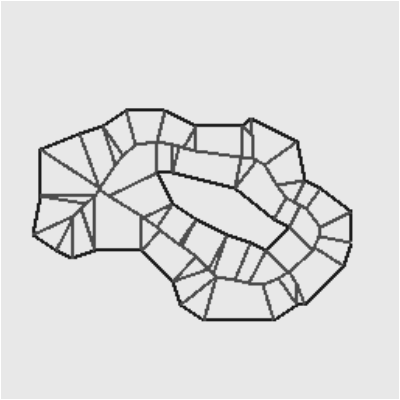
3.5 Skeletonizing Shapes with Holes

One of the attractive features of our method is that it handles shapes with and without holes in a uniform manner. Figure 11 shows an example of our approach applied on a holed object. No additional initialization is required. Note that when carrying out the pruning step, care must be taken to define the adjacency information correctly, in order that the last numbered site on the outer boundary and the first numbered site on the next hole are not considered adjacent. Shapes with multiple holes are handled similarly.

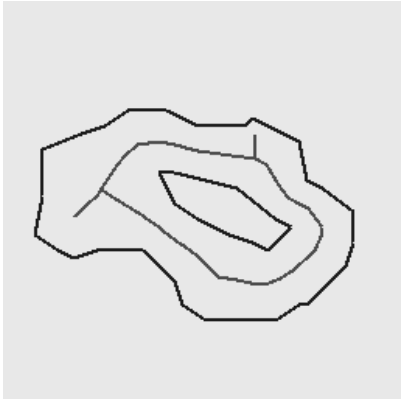
(a) Planar Shape



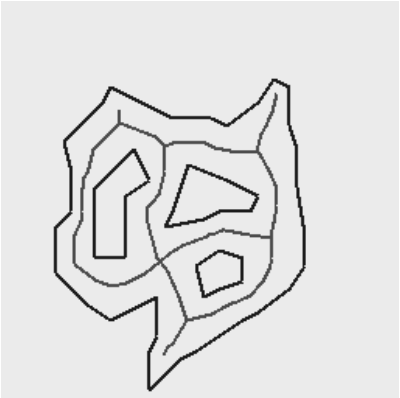
(b) Polygonal Approximation



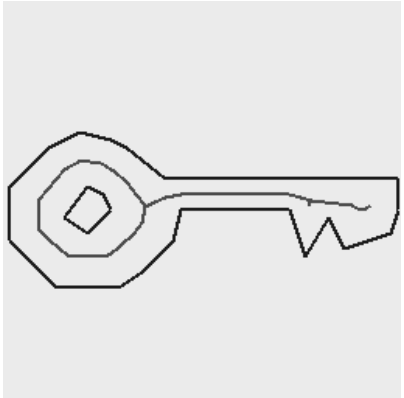
(c) Voronoi Diagram



(d) Pruned Skeleton



(e) Shapes with multiple holes



(f) Skeleton of a Key

Figure 11: Skeletons of Shapes with holes

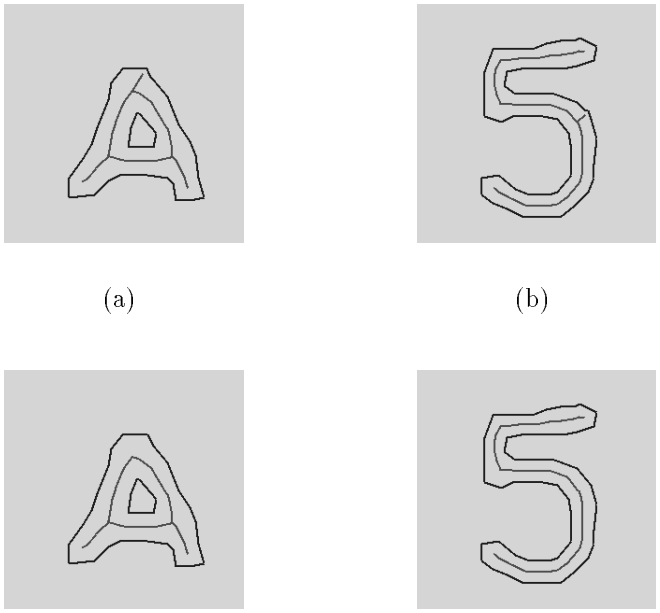


Figure 12: (a,b): Very small edges cause noise spurs. (c,d): Smoothing contour eliminates noise spurs.

3.6 Performance Evaluation

In this section, we briefly look at the time complexity of the entire skeletonization process. In the discussion that follows, let N denote the size of a square image, ($N = M * M$), where M is the length of each row and column. Let B represent the average size of the bounding contour of each planar shape. In Section 2, we saw that the time complexity of the Voronoi diagram algorithm is $O(B^2)$, where B is the number of sites on the bounding contour. Other skeletonization algorithms have time complexities that are a function of N, and hence valid comparisons are difficult to make. In practice, we have observed that in spite of the quadratic complexity, the algorithm is efficient in practice, since the number of boundary points B, is generally an order of magnitude less than N. However, in computing the time for the skeletonization process, the preprocessing steps must also be taken into account. The table below summarizes both the theoretical time complexity of each stage, as well as the actual time taken on a Sparc 10 computer to compute the skeleton of character shapes. The timing measurements are reported for each character of size $64x64$, averaged over a total of over 1350 character images.

	Time Complexity	Actual Running Time
Segmentation	$O(N \log N)$	2.26 sec
Polygonization	$O(B)$	0.10 sec
Voronoi Skeleton	$O(B^2)$	0.81 sec

Table 1: Performance Evaluation: Running times reported per 64x64 character image.

3.7 Discussion

In this section we show that Voronoi skeletons computed by the procedure described above, provide a representation superior to existing skeletonization algorithms.

Our approach exploits the fact that the medial-axis of a polygonal shape is implicitly contained in its Voronoi diagram. This fact immediately ensures (a) *that the computed skeleton lies in* $\mathcal{R}^2$*, (Euclidean metrics) and* (b) *connectivity is guaranteed. We compute the exact medial axis as opposed to a discrete approximation.* This is a major advantage over other skeletonization techniques.

Skeletons computed by thinning algorithms are constrained by the 4 or 8 connectivity of the discrete grid in which the object shapes are embedded [4]. The typical drawback of such approaches is therefore the loss of Euclidean metrics. Furthermore thinning algorithms also have to deal with redundant edges as a result of noise artefacts on the boundary of object shapes.

An alternative approach to skeletonization involves *ridge following* techniques based on distance maps computed from object shapes. The quality of the resulting skeletons is critically dependent on the metric used to derive the distance map. Non-Euclidean metrics (city-block distance, chessboard distance among others), lead to simple skeletonization algorithms, but lead to an inaccuracy of upto 40% with respect to Euclidean distances [11]. While methods to compute Euclidean [3] or quasi-Euclidean [16] distance maps exist, these methods do not guarantee connectivity. Gaps occur due to the discrete domain in which objects are embedded, and need to be filled in postprocessing steps.

More recently, new skeletonization algorithms [17] and [11] that are improvements over the traditional techniques described above have appeared in the literature. In the sequel, we briefly describe their methods, and compare them with our approach.

In [17], the authors compute the Voronoi diagram of *the set of points along the boundary of an object shape.* Two points arise with respect to the general scope of this technique. (a) A very large

number of points are required to correctly approximate object shapes; this is not very efficient. Further the large number of points lead to an extremely large number of redundant Voronoi edges, necessitating the use of complex pruning techniques. (b) A more critical drawback of this approach is that it is not easily applicable to objects with holes. In particular, it is not possible to decide a priori, the number of sampled points on an object, necessary to ensure that the Voronoi edges between the inner and outer boundaries will be computed.

Our approach on the other hand, overcomes both these deficiencies. The number of redundant edges are small as compared to [17]. Thus a simple pruning step, that involves no postprocessing suffices to rid us of almost all spurious edges. In addition, by controlling the polygonal approximation process to eliminate edges of very small length on the contour, we can eliminate the occurrence of those redundant edges that might not be deleted by the pruning step. Figures 8a,b, show examples where the polygonal approximations of the character shapes contain very small length edges that result in redundant edges that are not deleted by the pruning step. By ensuring that the polygonal approximation will not contain very small length edges these spurs can be eliminated. (Figures 12c and d).

Furthermore, polygons with and without holes are handled in a uniform manner. Polygons are defined in terms of line segments, and are therefore well defined at every point $p(x, y) \in \mathcal{R}^2$ along the boundary. This overcomes the drawbacks of discretizing the bounding contour as in [17].

The authors of [11] employ the *snake model* (an active contour model), to compute skeletons. In their technique, initial control points are defined for the snake at curvature extrema of a shape's bounding contour. The *dynamic grassfire* is then simulated by the snake propogation. This technique maintains correct Euclidean techniques by computing the Euclidean distance map, while the nature of the contour model ensures that connectivity is maintained. One limitation of this approach is the problem of computing the correct curvature in the discrete domain. Furthermore, a very ragged boundary implies a large number of curvature extrema leading to redundant skeleton edges; in this case other criterion need to be applied to reduce the number of initial control points. In addition, a special case arises when the bounding contour includes a circular arc whose center (a) lies within the object and (b) is an end-point of a skeleton branch. In such a case, additional control points need to be defined to ensure accurate skeleton computation. *In contrast, our technique offers a elegant and uniform approach to skeletonization that is independent of the shapes topology.*

Another advantage offered by our method, is that *a graph representation of the skeleton is immediately available.* Recall that the Voronoi diagram of a polygon is a planar straight line graph, and that by Lemma 4, the pruning step maintains connectivity. Skeletonization methods that compute the skeleton as a pixel map (as against an edge map that our method provides), require an intermediate vectorization step before a graph representaion may be obtained. Vectorization of the skeleton is an essential step before the skeleton can be used as a higher level of representation. In the context of recognition systems, our technique thus is amenable to the simple extraction of structural features such as end-points and junctions by a traversal of the graph comprising the skeleton.

In summary, Voronoi skeletons are powerful shape descriptors which overcome several disadvantages of existing techniques.

4 Conclusions

In this paper we have presented a new algorithm for Voronoi diagram construction of a polygon with holes. The algorithm employs an elegant formulation which handles point and line sites uniformly. The time complexity of the algorithm is $O(n_p^2)$ where n_p is the number of sites in the input. The incremental nature of the algorithm allows us to handle standard degenerate cases; no assumption need be made about points in general position. *We believe that ours is the first robust and easy to implement algorithm for this problem.* This allows for its wide applicability across various domains. We demonstrate its use in deriving efficient shape representation schemes. Other applications include mesh generation [19].

Voronoi skeletons derived from Voronoi Diagrams of polygons gives us an efficient and compact shape representation scheme. The skeletons are accurate, and are characterized by Euclidean metrics and the preservation of connectivity. We overcome the disadvantages of Voronoi skeletons in particular and the skeletonization approach in general, with the help of pruning methods. The scheme allows us to represent a wide array of shapes including objects with holes. The resulting representations are largely invariant to noisy detail on the object boundary and hence are amenable to higher order tasks such as recognition.

Our future research goals are twofold. The current Voronoi diagram algorithm uses a brute force method to determine new Voronoi edges (by searching every possible site in the polygon).

This allows for easy implementation and robustness, and enables us to handle degeneracies in a straightforward manner. However this is at the expense of time complexity. It is possible to define heuristics that allows us to prune the search space, and hence allow for faster convergence; this is one current area of focus.

We have presented several examples in this paper to demonstrate the usefulness of our shape representation framework. In particular, we have found it to be a stable representation scheme for handprinted characters. A future goal is to employ this representation in recognition applications.

References

[1] F. Aurenhammer [1991], "Voronoi diagrams - a survey of a fundumental geometric data structure," *ACM Computing Surveys*, Vol. 23, 345–405.

[2] H. Blum, [1967] "A transformation for extracting new descriptors of shape," *Models for the Perception of Speech and Visual Form* (W. Wathen-Dunn, ed.), Cambridge MA: MIT Press.

[3] [1980] P. E. Danielsson, "Euclidean distance mapping," *Comput. Graphics Image Processing*, Vol 14, 227-248.

[4] [1981] E. R. Davies and A. P. N. Plummer, "Thinning algorithms: a critique and a new methodology," *Pattern Recognition,* Vol. 14, 53–63.

[5] A. R. Dill, M. D. Levine, and P. B. Noble [1987], "Multiple Resolution Skeletons," *IEEE Trans. Patt. Anal. Machine Intell.* PAMI-9 No. 4, 495–504.

[6] S. Fortune [1987], "A Sweepline Algorithm for Voronoi Diagrams," *Algorithmica*, 2, 153–174.

[7] R. C. Gonzalez and R. E. Woods[1992], "Digital Image Processing", Addison-Wesley.

[8] D.G. Kirkpatrick [1979], "Efficient Computations of Continuous Skeletons," *IEEE 20th Annual Symposium on Foundations of Computer Science*, 18–27.

[9] D. T. Lee & R. L. Drysdale [1981], "Generalization of Voronoi Diagrams in the plane," *SIAM J. Computing*, Vol 10, No. 1, 73–87.

[10] D. T. Lee [1982], "Medial Axis Transformation of a Planar Shape," *IEEE Trans. Patt. Anal. Machine Intell.* PAMI-4, No. 4, 363–369.

[11] F. Leymarie and M. D. Levine [1992], "Simulating the grass-fire transform using an active contour model," *IEEE Trans. Patt. Anal. Machine Intell.* PAMI-14 No 1, 56–75.

[12] N. Mayya and V. T. Rajan [1994] "Voronoi Diagrams of Polygons: A Framework for Shape Representation," *Proceedings of IEEE CVPR 1994*, Seattle, Washington (to appear).

[13] D. C. Marr and E. Hildreth,[1980] "Theory of edge detection," *Proc. Royal Soc. London.,* B 207, 187–217.

[14] Siavash N. Meshkat & Constantine M. Sakkas [1987], "Voronoi Diagram for Multiply Connected Polygonal Domains II: Implementation and Application," *IBM J. Res. Develop.* 31, No. 3, 373–381.

[15] U. Montanari [1969], "Continuous Skeletons from Digitized Images," *JACM*, 16, No. 4, 534-549.

[16] U. Montanari [1968], "A method for obtaining skeletons using a quasi-Euclidean distance," *J. Assoc. Comput. Machinery*, Vol. 15, 60–624.

[17] R. Ogniewicz and M. Ilg [1992], "Voronoi Skeletons: Theory and Applications", *Proc. ICPR*, pp 63–69.

[18] Vijay Srinivasan & Lee R. Nackman [1987], "Voronoi diagram for multiply connected polygonal domains I: Algorithm," *IBM J. Res. Develop.*, 31, No 3, 361–372.

[19] V. Srinivasan, L.R. Nackman, J. Tang and S.N. Meshkat [1992], "Automatic Mesh Generation using the Symmetric Axis Transformation of Polygonal Domains," *Proceedings of the IEEE*, Vol. 80, No. 9, 1485–1501.

[20] Theo Pavlidis [1982], "Algorithms for Graphics & Image Processing", Computer Science Press.

[21] C. Y. Suen, R. Legault, C. Nadal, M. Cheriet and L. Lam [1992], "Computer Recognition of Unconstrained Handwritten Numerals," *Proceedings of the IEEE* VOl 80, No 7, 1162–1180.

[22] C. K. Yap [1984], "An $O(n \log n)$ Algorithm for the Voronoi Diagram of a Set of Simple Curve Segments," *NYU-Courant Robotics Report No 43.*