# Robust Construction of the Voronoi Diagram of a Polyhedron

Victor Milenkovic*
Harvard University
Division of Applied Sciences
Center for Research in Computing Technology
Cambridge, Massachusetts 02138

## Abstract

This paper describes a practical algorithm for the construction of the Voronoi diagram of a three dimensional polyhedron using approximate arithmetic. This algorithm is intended to be implemented in floating point arithmetic. The full two-dimensional version and significant portions of the three-dimensional version have been implemented and tested.

The running time[1] of this algorithm is $O(n_p n_v \log^2 b)$, where $n_p$ is the size of the input polyhedron, $n_v$ is the size of the output Voronoi diagram, and $b$ is the number of desired bits of precision. This algorithm can be made more practical with the use of a binary partition of space. In the worst case, binary partition does not improve the running time, but it should reduce the running time to $O(n_v b)$ on well-behaved inputs. Since $b$ is constant, this eliminates a factor of $n_p$. The algorithm can be generalized to higher dimensions and the order $k$ Voronoi diagram.

## 1 Introduction

This author has been approached by researchers from companies who have a great need for a practical program that can construct the Voronoi diagram of a polyhedron. They require the Voronoi diagram for purposes of computing good decompositions of polyhedra for finite element analysis. The Voronoi diagram is also useful for $\epsilon$-fattening the surface of a polyhedron which in turn might be useful for constructing approximations to polyhedra with the smallest number of triangles (see [20] and [8]). Another application is robotic path planning [13] [14]. The order $k$ Voronoi diagram [11] [21] can be used to rapidly determine the $k$ nearest sites (vertices, faces, or edges) to a point inside a polyhedron. This can be useful to a physical simulation of the properties of a potential function based on the boundary of the polyhedron. At present, there is no efficient[2] exact algorithm for constructing the Voronoi diagram, and no one has a good working implementation. Currently, only the Voronoi construction for point sites in the plane [22] and in space [10] have been treated robustly, and the only other robust algorithms are for planar arrangements of lines [7], line segments [18], algebraic curves [19], for the maintenance of planar triangulations [5], for the construction of convex hulls of points in the plane [5, 15], and for performing set operations on polygonal regions in the plane [9, 16]. There has been no robust treatment of the numerical issues inherent in any geometric construction in more than two dimensions involving quantities of higher than linear algebraic degree–such as the Voronoi diagram of a polyhedron.

The ordinary Voronoi diagram [4] of a set of point sites is a subdivision of space into convex cells, each cell representing the set of points closer to a particular site than any other. The faces, edges, and vertices of the diagram are the faces, edges, and vertices of the convex cells. Each face is shared by two cells; each edge is shared by three cells (in general position); and each vertex is shared by four cells (in general position). The Voronoi diagram of the interior of a polyhedron generalizes the notion of site to mean a vertex, edge, or face of the polyhedron. The cell is the set of points in the interior which are closer to one site (vertex, edge, or face) than any other. The cells of a Voronoi diagram are not necessarily convex; the faces can be quadratic surfaces; the edges can be fourth degree curves; and the vertices can be eighth degree algebraic points. These facts, unfortunate from the point of view of computational complexity, follow from the observation that the locus of points equidistant from a point and a plane is a paraboloid and the locus of points equidistant from two lines is a hyperbolic sheet. Any exact algorithm for constructing the Voronoi diagram of a polyhedron would have to perform computations on quantities of high algebraic degree. It would be very difficult to make such an algorithm practically efficient.

This paper describes a simple, easy to implement, numerically stable algorithm for computing the Voronoi

*Supported by NSF grants NSF-CCR-91-157993 and NSF-CCR-90-09272

[1]This is *expected* time since a hash table is used to store the Voronoi vertices and to enable the algorithm to avoid duplication.

[2]$O(n_V \log^m n_V)$ where $n_V$ is the size of the Voronoi diagram.

diagram of a polyhedron using approximate arithmetic. The robust Voronoi diagram algorithm visits the graph of Voronoi vertices and edges (the *medial axis* [1]) in a breadth-first fashion. Just as gift wrapping requires linear time (in the size of the input) to step from one face of a convex hull to another, the robust Voronoi diagram algorithm requires linear time (in the size of the polyhedron) to follow a Voronoi edge to a Voronoi vertex. Hence the running time is proportional to $n_P n_V$ where $n_P$ is the size of the input and $n_V$ is the size of the output. We give a few details on how to use the output of this algorithm to support numerical primitive for fast point location and other applications (Section 3.3).

Section 2 describes an efficient and numerically stable algorithm for reliably calculating *all* points (the *incenters*) inside a given tetrahedron and equidistant from four sites, each of arbitrary variety: vertex, edge, or face (triangle). This low level algorithm is the engine which drives the construction of the Voronoi diagram, and the total time to find an incenter is proportional to $\log^2 b$, where $b$ is the number of bits of accuracy in the output. Section 3 describes how to follow an edge out of a Voronoi vertex to another Voronoi vertex without accidentally hopping to another branch of the incenter curve. Finding the next vertex requires time linear in the number of sites. Section 4 describes a subdivision technique for speeding up the Voronoi diagram algorithm and the numerical computation. This technique has running time in $O(n_V b)$. In the worst case, it might not speed up the algorithm at all, but in practical cases it reduces the overall running time from $n_P n_V \log^2 b$ to $O(n_V b)$, which is nearly the optimal running time of $n_V \log b$.

For the generalization to $d$ dimensions, the cost of the algorithm grows proportionally to $d^3 n_P n_V \log^2 b$. In the order $k$ Voronoi diagram, each cell is labeled by a set of $k$ sites $\{s_{i_1}, s_{i_2}, s_{i_3}, \ldots, s_{i_k}\}$, which are the $k$ closest sites to each point in the cell. The cost of computing the order $k$ Voronoi diagram is also proportional to $n_P n_V \log^2 b$ where $n_V$ is now the complexity of the order $k$ Voronoi diagram. This abstract discusses only the order 1 version in three dimensions.

The graph-traversal approach to generating the Voronoi diagram necessarily has a running time a factor of $n$ (size of the input) away from optimal in the worst case. Subdivision techniques can reduce this factor to a constant in practice. Fortune [6] gives a sweepline algorithm for the two-dimensional version of the problem, but it would be very difficult to efficiently maintain the planar structure for a three-dimensional sweep. An incremental approach generalizing the Clarkson-Shor algorithm [3] for point-sites is a better candidate for near-optimal worst-case running time. However, there are two obstacles. First, as one randomly inserts triangles (the faces of the polyhedron), parts of the Voronoi vertex-edge graph may be disconnected and therefore hard to locate. It is simple to prove that the Voronoi diagram of the interior of a closed polyhedron is connected, but because of the non-linearity, the Voronoi diagram of an arbitrary set of points, line segments, and triangles might contain "loops," Voronoi edges without any vertices. The second difficulty is that little is known about the complexity of the Voronoi diagram of a polyhedron. It is possible that the complexity of a partially constructed Voronoi diagram could be $n$ or $n^2$ times the complexity of the final result. In any case, the numerical primitives developed here for the graph-traversal algorithm will be necessary for more sophisticated algorithms.

## 2 The Incenter Function

The most essential primitive in the calculation of the Voronoi diagram of a polyhedron is the *incenter function*. An *incenter* of a set of sites $S$ is defined to be a point $p$ equidistant from all sites in $S$. Given a set of sites $S$ and a tetrahedron $T$, Incenter$(S,T)$ returns one of the following:

I. a proof that there are no incenters of $S$ in $T$;

II. all $k$ distinct incenters $\{c_1, c_2 \ldots, c_k\}$ of $S$ in $T$;

III. an incenter $p$ and $k$ vectors $\{v_1, v_2, \ldots, v_k\}$ ($k \leq d$, the number of dimensions) such that the set

$$\{p + \sum_{i=1}^{k} t_i v_i \mid t_1, t_2, \ldots, t_k \in \mathbf{R}\}$$

is the maximum dimension $k$-flat in $T$ tangent to the set of incenters of $S$ in $T$.[3]

Implemented in rounded arithmetic, Incenter$(S,T)$ returns approximate instead of exact incenters in all three cases. Thus, if $\delta(p,s)$ is the distance from point $p$ to site $s \in S$, define the disparity of $p$ with respect to $S$

$$\mathrm{Disp}(p,S) = \max_{s \in S} \delta(p,s) - \min_{s \in S} \delta(p,s).$$

An approximate incenter $p$ of $S$ satisfies $\mathrm{Disp}(p,S) \leq \tau$ for some tolerance $\tau$. On the other hand, if a tetrahedron $T$ is supposed to have no incenters of $S$ inside it, then $\mathrm{Disp}(u,S) > 0$ for all $u \in T$.

### 2.1 Linear Upper and Lower Bounds

For purposes of calculation, it is easier to work with the square of the distance $\delta^2(p,s)$ from a point $p$ to a site $s$. The calculation of Incenter$(S,T)$ as defined above is based on linear upper and lower bounds on $\delta^2(p,s)$ inside tetrahedron $T$.

Let $T$ have vertices $p_0, p_1, p_2, p_3$. If $u \in T$, then there are unique weights (called *barycentric coordinates*) $w_i(u)$, $i = 0,1,2,3$, such that $0 \leq w_i(u) \leq 1$

$$\sum_{i=0}^{3} w_i(u) = 1 \text{ and } \sum_{i=0}^{3} w_i(u) p_i = u.$$

[3]Actually, there may be more than one connected component of the set of incenters. In this case, Incenter return a point $p$ and a set of vectors $\{v_1, v_2, \ldots\}$ for each component.

Each $w_i(u)$ is a linear function of $u$ (*e.g.* $w_2(u) = \det(p_0, p_1, p_2, p_3)/\det(p_0, p_1, u, p_3)$).

The squared distance $\delta^2(u,s)$ for a site $s$ is a convex quadratic function. Therefore for all $u \in T$,

$$\delta^2(u,s) \leq \sum_{i=0}^{3} w_i(u)\delta^2(p_i, s).$$

This is a linear upper bound $\mathrm{UP}(u,s,T)$ on $\delta^2(u,s)$ in $T$.

To determine a lower bound on $\delta^2(u,s)$ in $T$, we expand the function about some point $p$ in $T$. Let $\pi(u,s)$ and $\pi(p,s)$ be the projections of $u$ and $p$ onto (the flat of) $s$:

$$\begin{aligned}\delta^2(u,s) &= \delta^2(p,s) + 2(u-p)\cdot(p-\pi(p,s)) + \\ &\quad ((u-p)-(\pi(u,s)-\pi(p,s)))^2.\end{aligned}$$

The last term can be thought of as the square of the component of $u-p$ perpendicular to $s$. It follows that,

$$\delta^2(u,s) \geq \delta^2(p,s) + 2(u-p)\cdot(p-\pi(p,s)).$$

This is a linear lower bound $\mathrm{LO}(u,s,p)$ on $\delta^2(u,s)$.

## 2.2 Incenter Location Step

Define the *square disparity* of a point $p$ with respect to a set of sites $S$,

$$\mathrm{Disp}^2(p,S) = \max_{s\in S}\delta^2(p,s) - \min_{s\in S}\delta^2(p,s).$$

The *incenter location step* maps a point $p \in T$ to another point $p' = \mathrm{Step}(p,S,T) \in T$ which has smaller square disparity. In particular, $\mathrm{Step}(p,S,T)$ can have three possible values:

- $p' \in T$ such that $\mathrm{Disp}^2(p',S) < \mathrm{Disp}^2(p,S)$;
- $p$ when location step cannot improve disparity;
- a proof that $S$ has no incenter inside $T$.

$\mathrm{Step}(p,S,T)$ is the solution to a linear program based on the lower and upper bound of the previous section. The five variables are $rr$, $l$ and $u = (u_x, u_y, u_z)$. For each $s \in S$, the linear constraints are as follows:

- for each $s \in S$, $u \in T$ and the projection of $u$ onto the flat of $s$ lies in $s$;
- for each $s \in S$, $l \leq \mathrm{LO}(u,s,p)$ and $\mathrm{LO}(u,s,p) \leq rr \leq \mathrm{UP}(u,s,p)$;
- for every pair $s_1, s_2 \in S$ of vertices or planes, $u$ lies in the bisecting plane of these two sites;
- for each $s_1, s_2 \in S$ such that $s_1$ is incident on $s_2$ (for example, $s_1$ is a vertex of triangle $s_2$), $u$ lies in the line or plane through $s_1$ perpendicular to $s_2$. (This is very important to guarantee quadratic convergence.)

The objective is to minimize $rr - l$. In effect, solving this linear program finds a point $u$ in $T$ at which all the upper bounds are greater than all the lower bounds. It also minimizes the *estimated square disparity,*

$$\mathrm{EstDisp}^2(u,S,p) = \max_{s\in S}\mathrm{LO}(u,s,p) - \min_{s\in S}\mathrm{LO}(u,s,p).$$

If the linear program is infeasible, then there can be no incenter of $S$ in $T$. On the other hand, if $p$ is sufficiently close to an incenter and if $\mathrm{Disp}^2(p,S) = \epsilon$ then $\mathrm{Disp}^2(\mathrm{Step}(p,S,T),S) \in O(\epsilon^2)$. Finally, if $p$ is outside the region of convergence of an incenter, then it is possible that $\mathrm{Disp}^2(u,S) > \mathrm{Disp}^2(p,S)$ for the value of $u$ which minimizes the estimated square disparity. In this case, an addition calculation is necessary to determine a point $p'$ with smaller square disparity. Since $\mathrm{EstDisp}^2$ is a linearization of $\mathrm{Disp}^2$, it follows that $f(t) = \mathrm{Disp}^2(p + t(u-p), S)$ has negative derivative at $t = 0$. The function $f(t)$ is piece-wise quadratic. We can find its minimum on $t \in [0,1]$ by constructing its upper and lower envelope (using time in $|S|\log|S|$). In practice, it is easier to use binary subdivision of the interval $[0,1]$ since we only need an approximation to the minimum.

The location step can return its input $p$ when $p$ is on the boundary of $T$ and the nearest incenter is outside $T$.

## 2.3 Algorithm for Incenter$(S,T)$

This section describes how to use the incenter location step to compute $\mathrm{Incenter}(S,T)$. The basic idea is to set $p$ equal to the centroid of tetrahedron $T$ and then iterate $p \leftarrow \mathrm{Step}(p,S,T)$ until it converges or reports no solution. In practice, we stop the iteration when the square disparity fails to diminish by at least a factor of two.

If the iteration converges to a point $p$ that is not an incenter ($\mathrm{Disp}^2(p,S) \geq \tau$), it is necessary to subdivide $T$. Let $p_ip_j$ be the longest edge of $T$ and let $p_{i'}p_{j'}$ be the edge joining the other two vertices of $T$. The incenter algorithm cuts $T$ by the plane through $p_{i'}$ $p_{j'}$ and $(p_i + p_j)/2$ and recurses on the two halves. This recursion descends until $T$ is small enough and $\delta^2(u,S)$ is "flat" enough inside $T$. In the worst case, a constraint similar to that described in Section 4 stops the recursion. Let $p$ be the centroid of $T$, and let $r$ be the distance from $p$ to the farthest vertex of $T$. There can be no incenter of $S$ in $T$ when $\mathrm{Disp}(p,S) > r$.

## 2.4 Locating Multiple Incenters

Suppose that a tetrahedron contains a point $p$ which is equidistant from a set $S$ of sites. We want to locate other incenters or prove there are none. First we subdivide the tetrahedron into four tetrahedra with $p$ as a vertex. Assume we have a tetrahedron $T = p_0p_1p_2p_3$ with $p = p_0$. To show that $T$ does not contain *another* incenter, we employ a *volume elimination step.* The function VES$(T, s_i, s_j)$ returns a tetrahedron $p_0q_1q_2q_3$ such that $q_1q_2q_3 \subseteq p_1p_2p_3$ and such that no incenter of sites $s_i$ and $s_j$ lies inside $p_0q_1q_2q_3$. We will describe VES shortly, but first we summarize how to use it to locate all incenters of $S$ inside $p_0p_1p_2p_3$.

Applying VES eliminates a portion of $T$. We can compute VES$(T, s_i, s_j)$ for every pair of different sites $s_i, s_j \in S$. If any portion of $T$ remains, we subdivide it by a plane parallel to $p_1p_2p_3$ and halfway between that face and vertex $p_0$. We then subdivide the results into tetrahedra and apply the Incenter function recursively to each. A few subdivision should suffice because the bisectors of pairs of sites will appear flatter as the tetrahedra become smaller.

The region VES$(T, s_i, s_j)$ for $s_i, s_j \in S$ is computed as follows. We have already defined $\pi(p, s)$ to be the point of projection of $p$ onto (the flat of) site $s$. Define $\Pi(p, s)$ to be the plane through $\pi(p, s)$ perpendicular to segment $p\pi(p, s)$. It is easy to see that for any point $u$

$$\delta(u, \Pi(p, s)) \leq \delta(u, s) \leq \delta(u, \pi(p, s)).$$

Consider the paraboloid

$$P(p, s_i, s_j) = \{u \mid \delta(u, \pi(p, s_i)) = \delta(u, \Pi(p, s_j))$$

and its convex interior

$$P^{<}(p, s_i, s_j) = \{u \mid \delta(u, \pi(p, s_i)) < \delta(u, \Pi(p, s_j)).$$

For any point $u \in P^{<}(p, s_i, s_j)$,

$$\delta(u, s_i) \leq \delta(u, \pi(s_i)) < \delta(u, \Pi(s_j)) \leq \delta(u, s_j),$$

and therefore $u$ is not an incenter of $S$. Since $p = p_0$, $p_0 \in P(p, s_i, s_j)$. We can compute intersections of the edges of $p_1p_2p_3$ with $P(p, s_i, s_j)$ by parameterizing each edge and solving a quadratic equation. Suppose $p_1p_2$ and $p_1p_3$ intersect $P(p, s_i, s_j)$ at $q_2$ and $q_3$ and $p_1 \in P^{<}(p, s_i, s_j)$. In this case, it follows from convexity that the interior of $p_0p_1q_2q_3$ lies inside $P^{<}(p, s_i, s_j)$ and therefore does not contain an incenter. The tetrahedron $p_0p_1q_2q_3$ is returned from VES$(T, s_i, s_j)$. There are other cases depending on the number of times $P(p, s_i, s_j)$ intersects each edge of $p_1p_2p_3$.

## 3 Following a Voronoi Edge

A Voronoi vertex is an incenter of a set of sites, four sites except in degenerate cases. Let $\mathbf{V}_{\{A,B,C,D\}}$ be a vertex for sites $A, B, C, D$. There will be a Voronoi edge $\mathbf{E}_{\{A,B,D\}}$ out of $\mathbf{V}_{\{A,B,C,D\}}$ which will follow a curve of incenters of sites $A, B, D$. This section describes how to use the incenter function of the previous section to find the site $E$ such that $\mathbf{V}_{\{A,B,D,E\}}$ is the Voronoi vertex at the other end of $\mathbf{E}_{\{A,B,D\}}$ from $\mathbf{V}_{\{A,B,C,D\}}$.

### 3.1 Constructing a $\theta$-Tetrahedron about $\mathbf{E}_{\{A,B,D\}}$

Let $p$ be an incenter of $\{A, B, D\}$ on edge $\mathbf{E}_{\{A,B,D\}}$. We can construct a vector $v$ tangent to $\mathbf{E}_{\{A,B,D\}}$ at $p$ by intersecting the planes LO$(u, p, s) = 0$ (Section 2.1) for $s \in \{A, B, D\}$. A *$\theta$-tetrahedron* $T(p, v, \theta, \{A, B, D\})$ for $p$ and $v$ has vertices $p, p_1, p_2, p_3$ with the following properties:

- for $i = 1, 2, 3$ the angle between $v$ and $pp_i$ is $\theta$;
- for $1 \leq i < j \leq 3$ the angle between $v \times p_i$ and $v \times p_j$ is 120 degrees;
- Only one other incenter of $A, B, D$ (other than $p$) appears on the boundary of $T(p, v, \theta, \{A, B, C\})$.

The algorithm for constructing $T(p, v, \theta, \{A, B, D\})$ first constructs a large tetrahedron satisfying the first and second properties. If this tetrahedron has incenters of $A, B, D$ in one or more of the sides incident on $p$ ($pp_1p_2$, $pp_2p_3$, $pp_3p_1$), then the tetrahedron is truncated by a plane perpendicular to $v$ and passing through the incenter nearest to $p$ in the direction of $v$. The resulting tetrahedron will have incenters of $A, B, D$ only at $p$ and in the face opposite $p$. We rename $p_1, p_2, p_3$ to be the vertices of this face.

At this point, triangle $p_1p_2p_3$ will contain an odd number of incenters of $A, B, D$: one for the curve $\mathbf{E}_{\{A,B,D\}}$ emanating from $p$ and a pair for every other branch of the incenter curve that enters and leaves the tetrahedron through $p_1p_2p_3$. There is a technique for determining the points on these other branches that are closest to $p$ along the direction of $v$. Practically speaking, it is simplest to perform a binary search on the set of planes perpendicular to $v$ to determine one which has only one incenter of $A, B, D$ in its intersection with the interior of the tetrahedron. Truncating the tetrahedron by this plane yields the desired $T(p, v, \theta, \{A, B, D\})$.

### 3.2 Locating the Next Voronoi Vertex

We construct a sequence of $\theta$-tetrahedron that follow the curve of $\mathbf{E}_{\{A,B,D\}}$. First, set $p$ equal to $\mathbf{V}_{\{A,B,C,D\}}$ and set $v$ equal to the tangent vector pointing away from site $C$. Construct $T(p, v, \theta, \{A, B, D\})$. Set $p$ equal to the "exit point" of $\mathbf{E}_{\{A,B,D\}}$: the incenter of $A, B, D$ in the face $p_1p_2p_3$ of the tetrahedron, and repeat. The value of $\theta$ is arbitrary to start with (say 30 degrees). If the exit point is on an edge of $p_1p_2p_3$, this means that the tetrahedron was truncated because $\mathbf{E}_{\{A,B,D\}}$ initially exited "out the side." In this case, $\theta$ should be made larger. If the exit point is interior to $p_1p_2p_3$, then the tetrahedron was truncated because it hit another branch of the incenter curve of $A, B, D$. In this case $\theta$ should be made smaller. A good value of $\theta$ can be selected by binary search as the $\theta$-tetrahedra are constructed.

The sequence of $\theta$-tetrahedra is stopped when one is discovered that contains an incenter for $A, B, D, s$, where $s$ is some other site. This requires linear time per tetrahedron because each other site must be considered, although in practice the subdivision technique of Section 4 greatly reduces the number of sites $s$ to be considered.

The current $\theta$-tetrahedron is truncated by a plane through the incenter of $A, B, C, s$ nearest to $p$ along $v$. Each remaining site $s'$ is examined to determine if an incenter of $A, B, D, s'$ lies inside the current tetrahedron. If so, it is truncated. The result is a $\theta$-tetrahedron $pp_1p_2p_3$ and a site $E$ such that $A, B, D, E$ has an incenter on $p_1p_2p_3$ and no other site $s$ has an incenter with $A, B, D$ inside the tetrahedron. The incenter for $A, B, D, E$ is the Voronoi vertex $\mathbf{V}_{\{A,B,D,E\}}$ connected to $\mathbf{V}_{\{A,B,C,D\}}$ by edge $\mathbf{E}_{\{A,B,D\}}$.

### 3.3 Degenerate Cases and Applications

There are extra techniques required to handle singular cases and detect Voronoi faces. Basically, these involve constructing a tetrahedron about a Voronoi vertex and computing the intersection of the Voronoi diagram with the surface of the tetrahedron. The same techniques can be used to construct the intersection of the Voronoi diagram with a plane: one simply adds the plane restriction to the linear program described in Section 2.2. In general these techniques are sufficient to implement data structures for fast point location and other applications.

Here is a rough sketch of a very simple point location strategy. Once we have computed the Voronoi diagram, we select a plane that roughly bisects the set of Voronoi regions. We then compute the set of regions to the left or intersecting the plane and the set of regions to the right or intersecting the plane. We apply this bisection process recursively to each set. Repeated plane queries rapidly locates the Voronoi cell containing a query point. Determining the Voronoi cells intersecting a plane is equivalent to constructing the intersection of the Voronoi diagram with the plane, which we know how to do.

## 4 Subdivision Techniques

This section shows how to use a binary subdivision of space to speed up the computation of the Voronoi diagram.

### 4.1 Some Theory

Given a point $p$, define $s(p)$ to be the site of the polyhedron closest to $p$ which $p$ lies over (the projection of $p$ onto the flat of $s(p)$ lies in $s(p)$). Given a rectangular parallelopiped "box" $R$, define

$$\text{nearest-sites}(R) = \{s(p) \mid p \in R\}.$$

The set nearest-sites($R$) is difficult to calculate, but it is easy to calculate a superset near-sites($R$) in the following manner.

Define $|R|$ to be the length of the diameter of $R$ (the major diagonal). Let $c(R)$ be the center of $R$, and let $s(R)$ is the nearest site to $c(R)$ ($s(R) = s(c(R))$); and let $d(R)$ be the distance from $c(R)$ to $s(R)$. Define near-sites($R$) to be the set of sites $s$ with the following properties:

- the (point-set) distance from $c(R)$ to $s$ is less than or equal to $d(R) + |R|$;
- the projection of $c(R)$ onto the flat of $s$ is no farther than $|R|/2$ away from $s$.

A simple proof shows that near-sites($R$) is a superset of nearest-sites($R$).

A standard binary subdivision, starting with the axis-parallel bounding box of the polygon can be used to speed up the detection of the vertex at the end of a Voronoi edge. We create a binary tree of boxes rooted at the bounding box. Each node box can be split along its longest dimension to yield the boxes corresponding to its children in the tree. The set near-sites($R$) can be computed recursively for each box in the tree. As a $\theta$-tetrahedron is checked for new incenters (Section 3.2) all boxes $R$ in the subdivision tree which intersect it are determined by a recursive search, and the set of sites considered for incenters are limited to those in near-sites($R$).

### 4.2 Cost

What is the cost of the subdivision procedure? Assume the subdivision tree is expanded in breadth-first fashion. In the worst case, no branches can be pruned, and the expansion must stop before the total cost grows greater than $n_P n_V$.

Practically speaking, we expect the subdivision to prune off branches not leading to a Voronoi vertex. In this case the tree can be computed in time proportional to $n_V b$.

## 5 Conclusion

This Voronoi diagram algorithm is very likely to be of great practical importance. A two-dimensional version has been implemented and tested by Rajan and Mayya at IBM Watson Research Center. It will become part of automatic meshing software. The incenter algorithm has been implemented (by the author) in three dimensions, and it converges in four steps to an incenter. It also reliably determines multiple incenters. We are currently implementing the reliable edge following of Section 3.

Our next goal will be to determine if the set of reliable incenter and curve primitives can be used to create an incremental Voronoi algorithm.

## References

[1] H. Blum. A transformation for extracting new descriptors of shape. In *Proceedings of the Symposium*

*on Models for the Perception of Speech and Visual Form.* Weiant Whaten-Dunn, Ed. MIT Press, Cambridge, MA, pp. 362–380, 1967.

[2] D.R. Chand and S.S. Kapur An Algorithm for Convex Polytopes. *JACM* 17(1): 78-86, January, 1970.

[3] K. Clarkson and P. Shor. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry*, Vol. 4, pp. 387–421, 1989.

[4] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, Springer-Verlag, 1987.

[5] Steven Fortune. Stable Maintenance of Point-Set Triangulation in Two Dimensions, manuscript, 1990, ATT Bell Laboratories. An abbreviated version appeared in *30th Annual Symposium on the Foundations of Computer Science*, IEEE, October 1989.

[6] S.J. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153-174, 1987.

[7] S.J. Fortune and V. Milenkovic. Numerical Stability of Algorithms for Line Arrangements. *Proceedings of the Seventh Symposium on Computational Geometry*, ACM, pages 334–341, June 1991

[8] L. J. Guibas, J. E. Hershberger, J. S. B. Mitchell, and J. S. Snoeyink. Approximating polygons and subdivisions with minimum link paths. *Proc. 2nd Annu. SIGAL Internat. Sympos. Algorithms*, Lecture Notes in Computer Science, Vol. 557, Springer-Verlag, 1991, pp. 151–162.

[9] C. Hoffmann, J. Hopcroft, and M. Karasick. Towards Implementing Robust Geometric Computations, *Proceedings of the Fourth Annual Symposium on Computational Geometry*, 1988, ACM, pages 106-117.

[10] H. Inagaki, K. Sugihara, and N. Sugie. Numerically Robust Incremental Algorithm for Constructing Three-Dimensional Voronoi Diagrams. *Proceedings of the Fourth Canadian Conference on Computational Geometry*, St. John's, Newfoundland, 1992.

[11] D.T. Lee. On $k$-nearest neighbor Voronoi diagrams in the plane. *IEEE Transactions on Computing* C-31:478-487, 1982.

[12] D.T. Lee and R.L. Drysdale III. Generalization of Voronoi diagrams in the plane. *SIAM Journal of Computing* 10:73-87, 1981.

[13] D. Leven and M. Sharir. Intersection and Proximity Problems and Voronoi Diagrams. *Adv. Robotics* Vol. 1, pp. 187-228, 1986.

[14] D. Leven and M. Sharir. Planning a purely translational motion of a convex robot in two-dimensional space using Voronoi diagrams. *Discrete and Computational Geometry*, Vol. 2, pp. 9–31, 1987.

[15] Z. Li and V. Milenkovic. Constructing Strongly Convex Hulls using Exact or Rounded Arithmetic, *Proceedings of the Sixth Symposium on Computational Geometry*, pages 235–243, ACM, June 1990.

[16] Victor Milenkovic. Verifiable implementations of geometric algorithms using finite precision arithmetic, *Artificial Intelligence*, 37:377–401, 1988.

[17] Victor J. Milenkovic. *Verifiable Implementations of Geometric Algorithms using Finite Precision Arithmetic*, Ph.D. Thesis, Carnegie-Mellon, 1988. Technical Report CMU-CS-88-168, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, July 1988.

[18] Victor Milenkovic. Double Precision Geometry: A General Technique for Calculating Line and Segment Intersections Using Rounded Arithmetic, *Proceedings of the 30th Annual Symposium on the Foundations of Computer Science*, IEEE, pages 500-506, October 1989.

[19] V. Milenkovic. Calculating approximate curve arrangements using rounded arithmetic. *Proceedings of the Fifth Annual Symposium on Computational Geometry*, pages 197–207, 1989.

[20] J. S. B. Mitchell and S. Suri. Separation and approximation of polyhedral surfaces. *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, 1992, pp. 296–306.

[21] M.I. Shamos and D. Hoey. Closest-point problems. *Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science*, pp. 151–162, 1975.

[22] K. Sugihara and M. Iri, Construction of the Voronoi Diagram for One Million Generators in Single Precision Arithmetic. *Proceedings of the First Canadian Conference on Computational Geometry*, Montreal, Canada, 1989.