# Polygonal Approximation of Voronoi Diagrams of a Set of Triangles in Three Dimensions

Marek Teichmann* Seth Teller

MIT Computer Graphics Group

**Abstract**

We describe a robust adaptive marching tetrahedra type algorithm for constructing a polygonal approximation of the Voronoi Diagram of an arbitrary set of triangles in three dimensions.

Space is adaptively subdivided into a set of tetrahedral cells, and the set of Voronoi regions which intersect each cell is determined exactly using a simple primitive we introduce. We obtain a small number of different types of cells in which we then construct the polygonal approximation.

This has applications in the visualization of geometric structures, and in Solid Modeling, for example mesh generation and offset surface computation. Our algorithm can also be used to compute the exact Distance Transform of a three dimensional image, of interest in Visualization, and as a preprocessing step for answering nearest triangle queries.

We also present an exact method for computing the Voronoi Diagram (Medial Axis) of a convex polytope in three dimensions with worst case running time of $O(n^2)$ based on a reduction to convex hull in four dimensions. The practical advantage of this method over previous algorithms is that several robust implementations exist for computing the convex hull.

## 1 Introduction

Computing the Voronoi Diagram of a set of polygons or of a polyhedron, of which the Medial Axis is a subset, is an important problem in Solid Modeling. It can be used for example for hexahedral mesh generation for finite element applications [GP92, STG+97] and offset surface generation [CHL91]. Voronoi diagrams of triangles are also an fundamental structure in Computational Geometry, and it is interesting to be able to visualize such structures.

Our approximation algorithm in effect polygonizes the non manifold surfaces that occur in the Voronoi diagram, inside each cell of a hierarchical adaptive tiling of three dimensional space. In this respect, it generalizes the algorithm of Bloomenthal [BF95] to be adaptive. This adaptive nature simplifies the cases that occur when examining a given tetrahedron, and improves handling of surface intersections, at the cost of increasing the number of triangles produced. In addition, because we are dealing with Voronoi diagrams and not arbitrary non-manifold surfaces, we can deal with certain cases correctly that their algorithm would not. Our algorithm is based on a new primitive for determining exactly which Voronoi regions intersect a given cell.

We present two related algorithms, and a third exact algorithm for convex polytopes:

**Adaptive subdivision and cell labeling algorithm.** The first algorithm constructs a tetrahedral subdivision of a region of interest based on octrees, then performs a wavefront propagation step. The wavefront propagation determines, for each tetrahedral subdivision cell, the exact set of Voronoi regions intersecting that cell, using a new primitive. This generalizes the algorithm of Vleugels and Overmars [VO95], which only the Voronoi regions intersecting cell *corners*.

The first step is analogous to inserting the input triangles into a standard octree and is bounded by $O(nd)$ where $n$ is the number of input triangles, and $d$ is the subdivision depth. The propagation step takes time $O(c \log c)$ where $c$ is the number of cells occupied by input triangles if there are no degeneracies.

**Polygonization of Voronoi surfaces.** In the second algorithm, we start with the cells produced by the algorithm given above, further subdivide these cells in certain complex cases, and create a polygonal approximation of the Voronoi diagram. The error in the approximation is bounded due to the cell size and

*Supported by an NSERC postdoctoral fellowship.

bounds on the curvature of the surfaces involved [VO95]. Barring severe denegeracy, Voronoi vertices are located exactly (up to numerical precision.) The advantage of tetrahedral cells is that the number of different labelings of their vertices is much smaller than for a cube, the same reason for which cubes are divided into tetrahedra in isosurface polygonizing algorithms [Blo94].

**Medial Axis of a polytope.** For the important special case of a convex polytope, we also present a simple, easily implementable, exact algorithm to compute the Voronoi Diagram (Medial Axis) of a convex polytope. It is based on a reduction of this problem to the Convex Hull problem that is different from the classical reduction for the point Voronoi Diagram. The worst-case running time is $O(n^2)$, for $n$ input bounding planes. While slower than the $O(n \log n)$ algorithm of [Hel94], this algorithm relies on the computation of a convex hull in 4 dimensions, for which several implementations, both in exact and floating point arithmetic are available [GO97, chapter 52].

# 2 Related Work

There are several ways of approximating Voronoi diagrams. The first was introduced in a paper by Vleugels and Overmars [VO95]. The paper describes the algorithm for any dimension and objects more general than triangles, but we shall specialize to 3 dimensions, and a set of possibly vertex or edge-adjacent, but otherwise disjoint triangles $\mathcal{T}$.

In [VO95], space is divided into axial cells of fixed size, and the Voronoi diagram is approximated by the set of cells intersected by the diagram (with a caveat, see below.) This approximation labels each cube *corner* with the triangle of $\mathcal{T}$ closest to that corner and marks those cells with more than one label as containing part of the diagram. This is an approximation, as a surface may enter a cell undetected, and Voronoi vertices and edges are not identified. This approximation, while convenient for motion planning, is not ideal for visualizing the diagram.

Another method is to approximate Voronoi region boundaries with polygons. For example we could start with the approximation mentioned above, and apply an algorithm for polygonizing non-manifold implicit surfaces, which is a generalization of an isosurface extraction algorithm such as Marching Cubes [LC87] or a tetrahedral version of it such as [Blo94]. The most recent published available algorithm is due to Bloomenthal [BF95], but does not deal with more than one vertex of the surface in a given (tetrahedral) cell, or on a given cell face.

Furthermore, each of the above algorithms require repeated evaluation of the query: given a point $p$, to which Voronoi region does it belong? It seems that one would need a fully constructed Voronoi diagram or some other data structure to answer it. For example, no practical method for doing this is given in [VO95]. In our algorithm, such computations are ordered in such a way as to take advantage of coherence in the Voronoi diagram, that is the queries can be answered definitely "during construction".

There is also an extensive literature in the Solid modeling field on computing or approximating the Medial Axis. A recent survey can be found in the paper by Sherbrooke and Patrikalakis [SP95]. They also give an algorithm for computing the Medial Axis of a Polyhedron exactly (up to numerical precision) by following Voronoi edges and detecting vertices numerically. Finally, a survey of Voronoi diagrams and related algorithms can be found in [Aur91].

# 3 Preliminaries

For two points $p, q \in \mathbb{R}^3$, denote their Euclidean distance by $d(p,q)$, and for two sets $P, Q$, let $d(P,Q) = \inf\{d(p,q) : p \in P, q \in Q\}$.

We are given a set of triangles, possibly sharing edges or vertices. We will define the set of Voronoi sites $\mathcal{T}$ to be the set of relatively *open* triangular faces, along with their open edges, and vertices.

Following [VO95], we define the *bisector* of two Voronoi sites $S, T$ as

$$\mathrm{bis}(S,T) = \{p \in \mathbb{R}^3 : d(p,S) = d(p,T)\}.$$

The bisector is be a portion of either a plane, a paraboloid, a parabolic cylinder or cone or a hyperboloid. We call $S$ and $T$ *generators* of the bisector $\mathrm{bis}(S,T)$. The bisector divides space into two regions: the *dominance* *region* of $S$ over $T$ is

$$\mathrm{dom}(S,T) = \{p \in \mathbb{R}^3 : d(p,S) < d(p,T)\}.$$

Now define the Voronoi region of site $S \in \mathcal{T}$ as

$$\mathcal{VR}(S) = \bigcap_{T \in \mathcal{T}, T \neq S} \mathrm{dom}(S,T).$$

Its boundary is composed of bisectors with other sites, or Voronoi *facets* of dimension 2. Two or more Voronoi faces meet at Voronoi *edges* of dimension 1, and three or more at a Voronoi *vertex*, a point. If more than two faces meet at an edge or more than three at a vertex, we call this situation a *degeneracy*. Finally, let the Voronoi *diagram* of $\mathcal{T}$,$\mathcal{VD}(\mathcal{T})$, be the set of Voronoi facets generated by $\mathcal{T}$.

Note that, by including triangle vertices and edges as sites, we guarantee that for every point $p \in \mathbb{R}^3$ in the interior of a Voronoi region, there is exactly one site closest to $p$. See for example [Hel91] for a similar approach in the plane. Furthermore every bisector has two generator sites associated with it and hence each bisector has one algebraic formula.

# 4 The wavefront

We consider a wavefront propagating at constant speed from $\mathcal{T}$. A prairie fire analogy has also been used [Hel91]. The wavefront is the set of points at a given distance to $\mathcal{T}$, also called an *offset* surface [Hel94], (Figure 2). For $t \geq 0$, let the wavefront $F_t$ be the (set of) surface(s) defined by $F_t = \{p \in \mathbb{R}^3 : d(p,\mathcal{T}) = t\}$. We note that $F_t$ is composed of planar, cylindrical and spherical sections. These sections intersect at Voronoi bisectors, and we label each section by the site to which it is closest.

We will use this wavefront as a conceptual aid, without actually maintaining it, or even exactly maintaining the set of cells intersecting it. For clarity, in the next section, we first describe a conceptual algorithm based on maintaining the front. This establishes the structure of a wavefront propagation, and gives the different types of events that can occur.

An implementation of this algorithm would be costly so we then show how to maintain the front only approximately, significantly simplifying the algorithm and its implementation, at a cost of considering an additional constant number of cells at each propagation step.

## 4.1 Conceptual framework for wavefront propagation

We are given the set of sites $\mathcal{T}$, and a subdivision of our region of interest into pairwise disjoint cells. We wish to determine the set of Voronoi regions that intersect each cell.

Let $\delta$ be the diameter of a cell. We start with the wavefront at $t = 0$, and advance the front by increasing $t$, stopping whenever a section of the wavefront enters a cell. We will refer to $t$ as the time.

For each cell, we maintain a *current* set of labels at time $t$, which is the set of Voronoi regions associated with the section of the wavefront that entered the cell at or before time $t$. The label of the section of the wavefront at the point of entry indicates which Voronoi region "enters" the cell at that time; we say that it *propagates* to the cell. (We can also have multiple fronts entering at a given time; we treat them as entering at slightly different times.) Thus for any $t$ there are three types of cells:

- *unvisited cells*: those that are at distance greater than $t$ from $\mathcal{T}$ and have no labels at time $t$,
- *incomplete cells*: those that intersect the wavefront, and have some labels, and
- *complete cells*: those that the wavefront has passed (thus every point in the cell has a distance less than $t$ from $\mathcal{T}$) and will receive no new labels.

An incomplete cell can be updated several times as the front advances and labels are propagated. When it becomes complete, it is labeled with exactly the generators of those Voronoi regions it intersects.

An important observation is that when $t$ goes from $t_0$ to $t_1$ (with $t_1 > t_0$) a given label never propagates further than distance $t_1 - t_0$ (Figure 2). This implies that an incomplete cell never obtains a label from a cell that is further that distance $t_1 - t_0$ from it. This will be used below to show correctness of the propagation

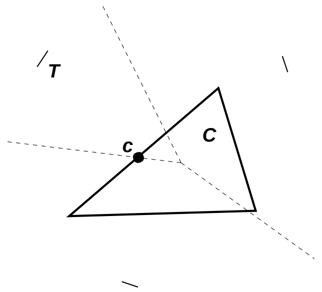


Figure 1: Testing if $VR(T) \cap C$. The Voronoi diagram of the three short open segments is shown as dotted lines.

algorithm. We also note that all cell entries and exits are within a time interval of size $\delta$, after which the cell will not receive any more labels.

We have so far assumed that the cells initially intersecting $\mathcal{T}$ have final labels. This is easy to accomplish by an initial propagation step, in which each cell obtains labels from its neighbors (which are at distance at most $\delta$).

# 5 The Propagation algorithm

Consider a subdivision of our *region of interest* (for example, an enlarged axial bounding box) into axial cubic cells of fixed size. In our implementation each cube is subdivided into 6 tetrahedra (also called Kuhn simplices), which are similar, up to their mirror image, as in [Blo94] and others. This subdivision has the advantage of being compatible with a similar subdivision with half the cell size. We will refer to cell vertices of cells as *corners.*

We are now ready to describe the actual algorithm we use, which computes the set of Voronoi regions intersecting each cell. We begin with a primitive fundamental to our algorithm.

## 5.1 Testing if a Voronoi region intersects a cell

For a cell $C$, let $L(C)$ be the set of all Voronoi regions intersecting $C$, i.e. the set of labels of $C$ when $C$ is complete. We are given a site $T$, and a cell $C$ with labels $L \supseteq L(C)$, and would like to determine if $\mathcal{VR}(T)$ intersects $C$. We describe a new primitive to do this, which we call TestLabel. See Figure 1.

**Lemma 5.1** *Let $\mathcal{S}$ be a set of sites, and let $T$ be a site in $\mathcal{S}$. Let $C$ be a cell in $\mathbb{R}^3$ and $c$ be the closest point to $T$ on $C$. Then in the Voronoi diagram of $\mathcal{S}$, $\mathcal{VR}(T) \cap C \neq \emptyset$ iff $d(c,T) \leq d(c,S)$ for all $S \in \mathcal{S} \setminus T$.*

**Proof.** Clearly if $d(c,T) \leq d(c,S)$ for all $S \in \mathcal{S} \setminus T$ then $c \in \mathcal{VR}(T) \cap C$, by definition of Voronoi regions. Conversely, if $x \in \mathcal{VR}(T) \cap C$ for some $x$, $d(T,x) \leq d(S,x)$ for all $S \in \mathcal{S}$ by definition of $\mathcal{VR}(T)$, and since $d(T,c) \leq d(T,x)$, we have $d(T,c) \leq d(S,x)$ and $c \in \mathcal{VR}(T)$. □

Let $c$ be the point closest to $T$ on $C$. Then $\mathcal{VR}(T) \cap C \neq \emptyset$ iff $d(c,T) \leq d(c,S)$ for all $S \in L$. Note that this test also works if $C$ contains a Voronoi region completely. Implementation is done by clipping the query site $S$ against the precomputed Voronoi regions of a generic tetrahedron, and finding distances of the resulting pieces to the appropriate cell vertex, edge or face, which is a simple operation.

If there are $\ell$ labels, this test can be determined in $O(\ell)$ time. Testing all labels against each other takes $O(\ell^2)$ time. In practice this can be somewhat accelerated by ordering the distances and testing them in decreasing order to take advantage of the coherence between points in the cell. Moreover most cells will have only a small number of labels. As a corollary, if $\mathcal{S}$ is a superset of the set of Voronoi regions intersecting $C$, we can remove the labels of the regions that do not intersect $C$ in $O(|\mathcal{S}|^2)$. Finally note that TestLabel can also be used to check if a Voronoi region intersects a triangle or an edge of a cell.

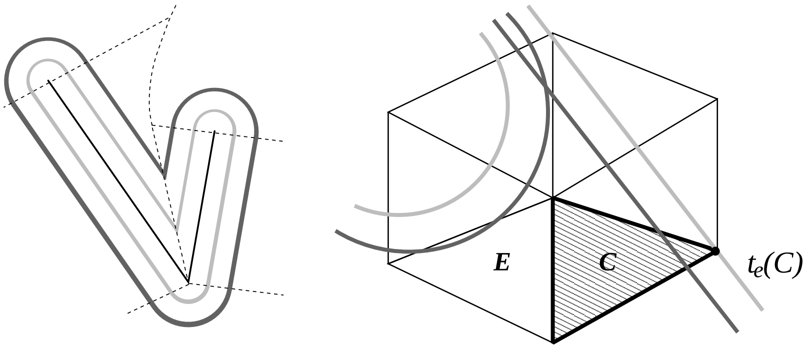


Figure 2: Wavefront during propagation, from light grey to dark grey. Left: the Voronoi diagram of two edges is shown in dotted lines. Right: cell $E$ is incomplete; $C$ is being processed at time $t_e(C)$.

### 5.2 The propagation

In this section we describe the propagation algorithm which simulates the advance of the wavefront, but does so only approximately, i.e. processes events in batches at each step.

There will be two types of cells: *processed* and *unprocessed*. Processed cells correspond to those cells $C$ that the algorithm has seen and labeled, and hence contain the set of labels $L(C)$, and unprocessed ones are the remaining cells that the algorithm hasn't considered yet. We call a label *final* when it labels a complete cell.

During initialization, all input triangles are inserted into the cells they intersect. Then the initializing propagation described above is performed, and from the cell labels, we label each cell corner by the distance from the corner to $\mathcal{T}$. Next all corners are placed into a priority queue $Q$, with the top of the queue at the closest distance to $\mathcal{T}$.

After initialization, we iterate the following steps until $Q$ is empty. Let $c$ be a corner on the top of $Q$, and $C$ an unprocessed cell with corner $c$. If $C$ was the only such cell, $c$ is removed from $Q$. If $C$ lies in the region of interest, $C$ is processed as follows. All labels of cells at a distance of at most $2\delta$ are collected. Then TESTLABEL is used repeatedly to eliminate labels that are not final. The cell $C$ is labeled by the set $L(C)$ of final labels, and additionally, the corners of $C$ are labeled by the site closest to them. If the number of labels of $C$ is greater than one, each corner is inserted into the queue, if not already present (this ensures that we do not propagate any further from cells with only one label).

**Correctness.** Consider a cell $C$ and let $[t_a(C), t_b(C)]$ be the time interval for which $C$ is incomplete, (hence $t_a(C) = d(C, \mathcal{T})$), and $t_e(C)$ be the time $C$ is actually encountered by the propagation algorithm. First note that the algorithm guarantees that $t_a(C) \leq t_e(C) \leq t_b(C) \leq t_a(C) + \delta$. This follows from the triangle inequality. Hence the time interval between steps is at most $\delta$.

**Lemma 5.2** *At time $t_e(C)$, the set of labels collected from cells at a distance of at most $2\delta$ from $C$ is a superset of $L(C)$.*

**Proof.** Say that we are about to process an unprocessed cell $C$, and assume for all processed cells $C_i$, the set of labels is final when they are encountered by the propagation algorithm (at time $t_e(C_i) < t_e(C)$), but before $t_e(C)$, they do not necessarily contain any of the labels that would arrive earlier for $t$ between $t_a(C_i)$ and $t_e(C_i)$.

Collecting cell labels from cells that are at a distance of at most $\delta$ is sufficient by the observations in Section 4.1. Hence, collecting labels from the processed cells $C_i$ at a distance of at most $\delta$ is sufficient to obtain the final labels from the processed cells. There can however be unprocessed cells $E_i$ (with $t_e(E_i) > t_e(C)$) within distance $\delta$ from $C$ that have $t_a(E_i) < t_e(C)$ i.e. they are incomplete at time $t_e(C)$, and may contain some labels according to the framework. An example can be seen on the right in Figure 2, where cell $E$ is already entered by the left wavefront, but has not been processed hence contains no label according to the algorithm above. Unprocessed, cell $E$ has no labels at time $t_e(C)$ according to our algorithm, but those labels are late by at most $\delta$. Thus it is sufficient to collect labels from cells that are at a distance of at most $2\delta$ from $C$. In effect, we do a propagation for some unprocessed cells. □

We conclude that if we process cells in order of their closest vertex to $\mathcal{T}$ by collecting labels as above, which is from a constant number of cells, we obtain a superset of the final cell labels. Note that it is not necessary to consider cells with only one label since all the information is contained in cells with multiple labels.

**Complexity.** If $\ell$ is a bound on the number of labels per cell, the total time spent in a propagation step is $O(\ell^2)$. The number of propagation steps is the same as $c$, the number of cells processed. Hence the total running time is $O(n + \ell^2 c \log c)$, where $n$ is the number of input triangles. Assuming a constant number of labels per cell, which can be ensured as described later unless degeneracies occur or a cell contains a large number of input sites, the propagation step takes constant time, and the total time is $O(n + c \log c)$.

### 5.3 Adaptive algorithm with propagation

Next, we present an algorithm which will be used as a preprocessing step in the polygonizing algorithm of Section 6. It adaptively subdivides the region of interest into tetrahedral cells, with vertices on a hierarchical grid, such that each cell has a number of labels sufficiently small to allow for efficient propagation. This number must be determined experimentally.

We insert input triangles into a standard octree [Sam90]. Octree nodes are subdivided until we achieve the same size for nodes that contain input triangles (to simplify the propagation), and no node has more than the required number of triangles. In addition, we require that each cell contains at most one input vertex.

The resulting non-empty nodes are tetrahedralized and used as a basis for the propagation algorithm. At this point propagation is efficient due to the small number of labels in each cell, with the tradeoff that more cells must be processed. A conservative worst case bound on the running time is $O(nd + \ell^2 c \log c)$, where $c$ is the number of cells containing input triangles, each with at most $\ell$ labels, and $d$ is the octree subdivision depth.

### 5.4 Purely adaptive labeling

Once the above cells have been computed, the polygonizing algorithm below requires that certain cells with large numbers of labels be further subdivided. In this section, we first describe an adaptive algorithm for doing this which will be used as a subroutine in the polygonizing algorithm, after the initial subdivision is done as in Section 5.3. We describe the details of subdividing a cell later. This algorithm is based on a generalized version of octrees in which cell geometry and number of children of a node representing a cell varies, but the latter is bounded by 9.

Initially, each node contains a set of labels obtained using the algorithm in the previous section. (This algorithm can also be used independently, in which case, initially, all sites are inserted into the root node, which is labeled with them.) Nodes are subdivided until a predetermined small number of labels remain or (as required by the polygonizing algorithm), the required maximum cell size is reached (the precision of the approximation), or a maximum subdivision level is reached. After subdivision of a node, for each child node, labels of the parent are tested and possibly eliminated using TestLabel.

A bound on the worst case running time would be $O(n^2 \log d)$ where $d$ is the subdivision depth, but this does not take into account the fact that often TestLabel degenerates to a constant time intersection test instead of a linear number of distance tests. This algorithm will be used in the next section after the initial subdivision of section 5.3.

## 6 Adaptive Polygonizing Algorithm

We now wish to compute a polygonal approximation of the Voronoi diagram of $\mathcal{T}$, based on the computation of the previous section. We assume that the cell size obtained from the calculation of Section 5.3 defines the resolution at which we would like the polygonal approximation to be for simple bisectors.

Further subdivision will take place only for complex cells, such as cells containing Voronoi vertices.

Our algorithm is an adaptive version of the algorithm of Bloomenthal [BF95], that is specialized for Voronoi surfaces. It is complete in the sense that using subdivision, we are not limited to a single Voronoi vertex and edge in each initial cell, and correctly identify them, up to a requested resolution. Our goal is to

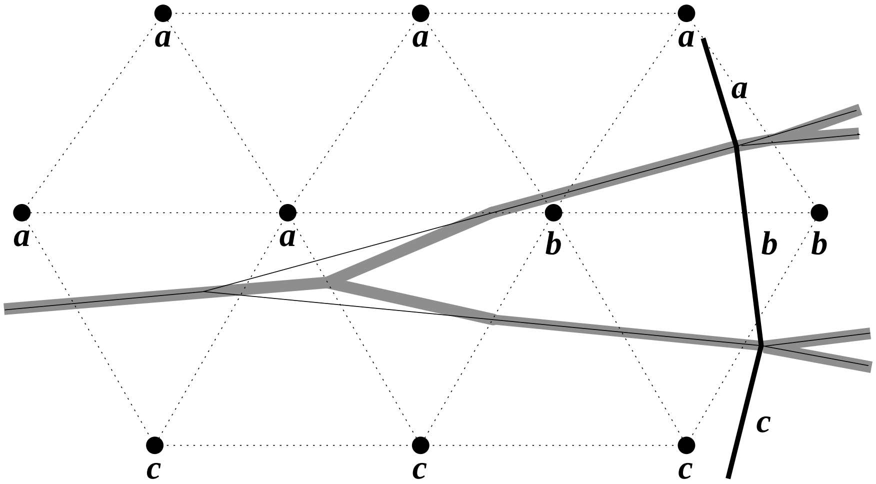


Figure 3: Simple approximation (in grey) of the actual Voronoi diagram (thin lines).

produce a simple algorithm that would satisfactorily treat these cases, yet, for each tetrahedron considered, needs to polygonize only a small number of surfaces.

## 6.1 A simple non-adaptive approximation

Consider a given cell $C$. Recall that $L(C)$ is the set of labels associated with the entire cell $C$. Let $V(C)$ be the set of labels associated with the corners of $C$, The simplest possible set of cases is as follows. We call a case *canonical* if $|L(C)| = |V(C)|$. These cases are easy to polygonize. If $|L(C)|$ is

1, the cell contains no Voronoi surface,

2, the cell contains one bisector,

3, the cell contains an edge which passes through three faces, i.e. two of the cell faces contain an intersection with the edge. The faces involved are two with three labels, see Figure 4a),

4, the cell contains one Voronoi vertex as in Figure 4a).

Of course, there are other cell types: a simple example can be seen on the left in Figure 3 where two Voronoi edges (thin lines) cross a cell without intersecting. However, if we consider only labels at cell corners as in [VO95], it is possible to produce a first approximation, which is consistent in the sense that the boundary of each region produced contains no holes, also shown in with grey lines on the left in Figure 3.

In the case $|V(C) = 2|$, we take the intersection of the cell edges with the bisector whose generators are the cell corner labels. Then we construct the 2 or 3 triangles with vertices at these intersections that approximate the surface, as in [Blo94].

In the case $|V(C) = 3|$, one can find the intersection of the Voronoi edge with the cell, (see Section 6.3) or appropriate points on two faces (with three labels) if none are found. In the case $|V(C) = 4|$, we use the actual vertex if in the cell, or the centroid of the tetrahedron. The construction of approximating triangles is illustrated in Figure 4, and described below. This approximation gives reasonable results for small cell sizes, see Figure 6 for an example. A disadvantage is that estimated vertices can be arbitrarily far from the their exact counterparts.

## 6.2 A better approximation

We can produce a better approximation by subdividing complex cells, i.e. cells with more than four labels, or cells with four corner labels and no Voronoi vertex, or cells with three corner labels and no Voronoi edge, etc. Subdivision is done until either we reach one of the simple cases described in the previous section, or a maximum subdivision level is reached. Once subdivision is terminated, we revert to the simple approximation technique using corner labels mentioned above. Our algorithm also tests for degeneracies, and subdivides accordingly. We summarize our algorithm here:

**Phase 1.** The subdivision of Section 5.3 is performed.

**Phase 2.** The resulting tetrahedral cells are processed to find cells that are easy to polygonize, possibly subdividing them further, as described in Section 5.4. We enforce *subdivision smoothness*, to simplify case analysis during subdivision: whenever a tetrahedron is subdivided, we make sure that its neighbors are at most one level of subdivision away from it, and if necessary trigger cascaded subdivision. Neighbors at a lower subdivision level are triangulated. The tetrahedral cells are subdivided with planes parallel to the cell sides and through edge midpoints as in [Moo92], see Figure 5a), unless degeneracies occur (Section 6.4).

**Phase 3.** In the last phase, triangles approximating the surface are created, as described in Section 6.4.

### 6.3 Primitives

We first describe the additional primitive operations that we will need and their implementation. Let $S_1, \ldots, S_4$ be sites of $\mathcal{T}$.

COMPARESITES$(p, S_1, S_2)$ determines which of the two sites, $S_1$ or $S_2$ is closer to a point $p$.

FINDBISECTOR$(e, S_1, S_2)$ finds where the bisector of sites $S_1$ and $S_2$ intersects the cell edge $e$, or returns failure if none exists.

FINDVEDGEONFACE$(f, S_1, S_2, S_3)$ finds the intersection of a Voronoi edge on triangular cell face $f$ that is equidistant to $S_i$, or returns failure if none found.

FINDVVERTEXINCELL$(C, S_1, S_2, S_3, S_4)$ finds a Voronoi vertex in tetrahedral cell $C$ that is equidistant to $S_i$, or returns failure if none found.

The primitive COMPARESITES is easy to implement. FINDBISECTOR insersects a segment with either a planar or quadratic surface by solving a one-parameter equation. We do binary search along the edge using COMPARESITES, which takes $O(\log b)$ time, where $b$ is the desired precision.

FINDVEDGEONFACE is implemented by a double binary search. An alternative would be to use the algorithm below in a lower dimensional setting. For FINDVVERTEXINCELL, we take three of the bisectors of each pair of sites among the 4 sites. Next, we numerically find their common intersection(s) and test if the result is actually equidistant to the sites. The precision of the answer affects only the precision of the resulting approximating polygonal surface. See also [Mil93] for an alternative.

### 6.4 Processing one tetrahedron

In phase 2 of the algorithm, to polygonize the content of a cell $C$, we distinguish cases according to the number of labels in $L(C)$, and subcases according to $|V(C)$. We create approximating triangles only for cells with $|V(C)| = |L(C)| > 1$ unless there is a degeneracy. Other cases are subdivided.

**One label.** If $|L(C) = 1$, we do nothing.

**Two labels.** If $|L(C)| = 2$, then a Voronoi surface must intersect the cell. If in addition $|V(C)| = 2$, we use FINDBISECTOR on the cell edges having 2 labels to determine where the surface intersects each cell edge and create a two or three triangles, approximating the surface as in [Blo94]. In the case where the surface exits through another face, subdivision occurs. This is easily tested using TESTLABEL on the face. Some of this information is already available since TESTLABEL has been used to find $L(C)$, and is cached.

Finally, if $|V(C)| = 1$, a portion of the surface enters the face without crossing three edges. In this case, we subdivide the cell, as in Figure 5a). Note that since bisector curvature is bounded subdivision will stop eventually [VO95]. Our implementation limits subdivision to a predetermined level.

**Three labels.** Refer to first row of Figure 4. If $|L(C)| = 3$, and two faces have 3 different vertex labels (3 label canonical case), we attempt to find the intersections of a Voronoi edge with them using FINDVEDGEONFACE on each cell face. If successful, and if the edge does not stray out of the cell (as verified by FINDVEDGEONFACE), approximating triangles are created according to case a) in the Figure. Subdivision occurs otherwise.

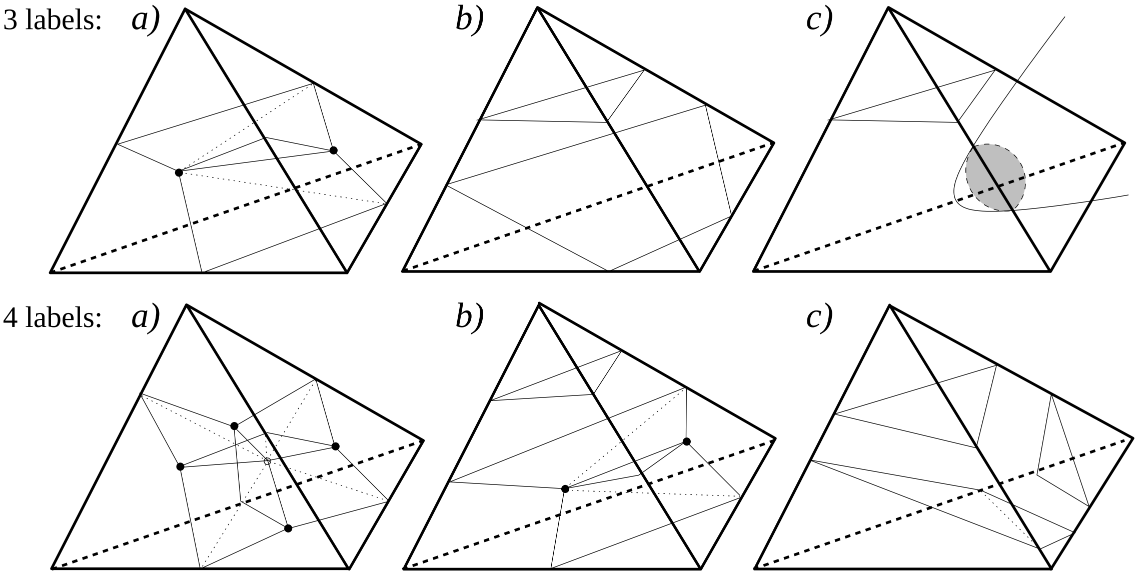


Figure 4: Polygonal approximations are shown for typical cases. Only a) cases get polygonized.

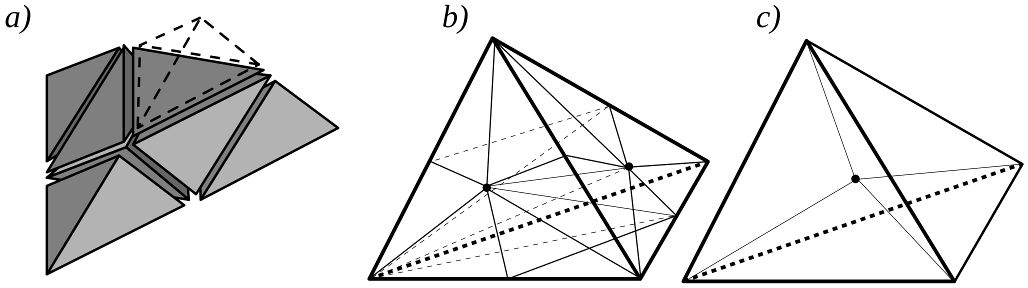


Figure 5: Tetrahedron subdivision cases. Case a) is the Kuhn simplex.

**Four labels.** Here $|L(C)| = 4$. The corresponding canonical case occurs if FINDVVERTEXINCELL locates a vertex, and if $|V(C)| = 4$, we create triangles as in [BF95], see the second row of Figure 4a). If $|V(C)| \neq 4$, we do a subdivision with the vertex as apex of 4 sub-tetrahedra, as in Figure 5c). Otherwise, we test for a degenerate Voronoi edge using three of the labels and FINDVEDGEONFACE, and if it exists, and if the fourth site is also equidistant to the both points returned by FINDVEDGEONFACE, we assume degenerate Voronoi edge. To reduce to the canonical cases, we then subdivide the cell along the edge by doing a (precomputed) constrained triangulation of the cell, and, the cell is subdivided. See Figure 5b). If the above tests fail, standard cell subdivision takes place.

**More than four labels.** In this case, we test for a vertex degeneracy, and subdivide if necessary. By selecting four labels, we test for a Voronoi vertex using four of the labels, as above. If a vertex is found, we check that *one* other label is also equidistant to the vertex. If so, we assume a degenerate Voronoi vertex, and subdivide the cell by creating four new sub-cells each with one apex at the vertex (Figure 5c)). This method is also used when several input triangles share a vertex in a cell, with each apex at the input vertex. Otherwise, the cell is subdivided in the standard way. A similar method is used for testing for a degenerate Voronoi edge.

If we reach a maximum preset level of subdivision without reaching one of the cases above, we fall back on the simple method of polygonizing using only cell vertex labels, ignoring the others. As is done in most polygonizing algorithms, we cache intersections on edges, faces, but also in cells. The last one is used when subdividing in some of the degenerate cases.

## 7 Results

We have currently implemented a non-adaptive version of the propagation algorithm, and give some running times. For an input of 2032 triangles, 189546 tetrahedral cells were traversed producing 333801 approximating triangles, and our implementation took 1.9 minutes on an SGI Indy with 133 Mhz R4600 Processor. Tetrahedral cell corners were on a $50 \times 50 \times 50$ grid dividing the bounding box of the input, with an extra 5 on each side. The priority queue had at most 4703 elements during propagation. The sample image in Figure 6 took 9.1 seconds to produce and contains 7633 triangles in 20334 tetrahedral cells with corners on a $20 \times 20 \times 20$ grid. Finally, Figure 7 shows the surface equidistant to two triangulated tori. This was done by omitting Voronoi surface between triangles of the same torus.

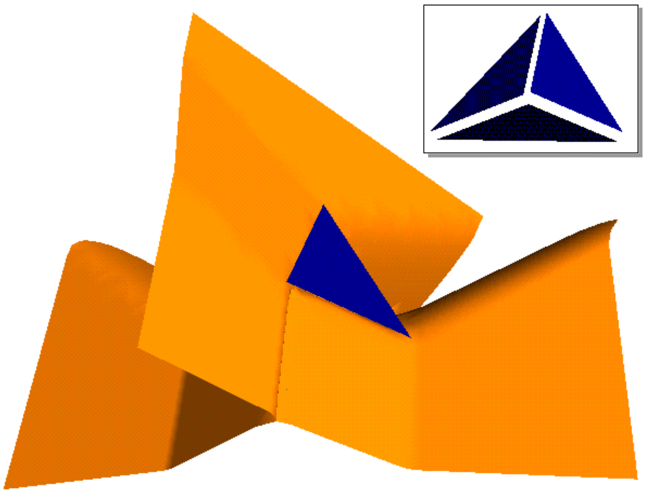

Figure 6: Voronoi diagram of three triangles (bisectors involving vertices or edges are not shown). The input triangles, loosely forming a cone, are shown in the inset.

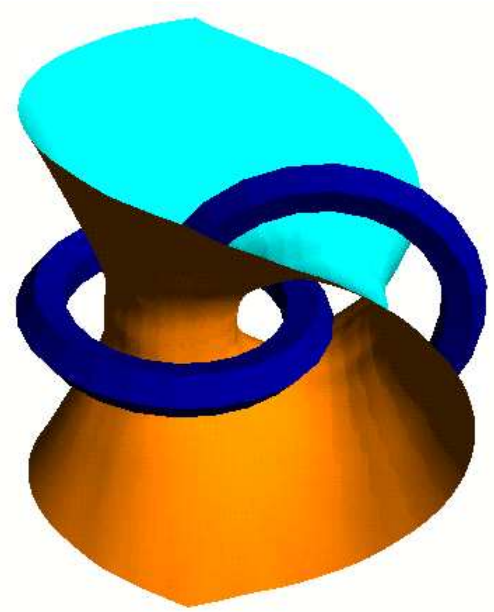

Figure 7: Surface equidistant to two tori.

## 8 An exact algorithm for convex polyhedra

In this section, we present a linear time reduction from the problem of computing the Voronoi Diagram for convex polyhedra in dimension three, which is identical to its Medial Axis, to the problem of computing the convex hull in dimension four. This reduction can also be easily generalized to higher dimensions.

Let $h_1, \ldots, h_n$ be the set of planes bounding the input convex polyhedron $\mathcal{P}$, and let the equation of $h_i$ be $a_i x = b_i$, with unit outward pointing normal $a_i$. Now, the plane $g_i$ at distance $t$ from $h_i$ towards the interior of $\mathcal{P}$ is $a_i x + t = b_i$ since $a_i$ is unit. Let $g_i^+$ be the halfspace $\{(x,t) \in \mathbb{R}^4 : a_i x + t > b_i\}$. Let $\pi$ be the projection from $\mathbb{R}^3 \times \mathbb{R}$ to $\mathbb{R}^3$ defined by $\pi(x,t) = x$. Then the Medial Axis of $\mathcal{P}$ is the projection by $\pi$ of the $k$-faces ($k < 4$) of $\bigcap_i g_i^+$. This can be seen as follows. Take a point $(y,s) \in \mathbb{R}^3 \times \mathbb{R}$ that is on the boundary of $\bigcap_i g_i^+$. This point lies on a set, say I, (possibly of size 1) of the $g_i$'s. This implies that $y$ is at a distance $s$ from $h_i$, $i \in I$, and $s > b_i - a_i y$ for all $i \notin I$, i.e. $y$ is further from all other $h_i$'s. Hence $y$ is on a $(\min(4, 4 - |I|))$-face of the Medial Axis of $\mathcal{P}$ (barring a degeneracy).

The computation of the intersection can be done by taking the polar dual of the planes $g_i$ and computing the convex hull of the resulting points [Ede87]. This gives an algorithm of time complexity $O(n^2)$ [Ede87] for computing the Medial Axis of an $n$-sided polytope.

## 9 Concluding remarks

We have presented a simple algorithm at the intersection of Computational Geometry and Visualization for approximating the Voronoi diagram of a set of triangles or polyhedra. The current non-adaptive implementation already produces usable results for Visualization, and we are currently implementing the adaptive version.

The algorithm can easily be extended to construct the diagram of polygons or polyhedral objects instead of triangles: we simply do not create the approximation of the bisectors between triangles belonging to the same polygon or object.

Future work includes a generalization of this method to arbitrary non-manifold surfaces. In this case, TestLabel will either be substantially more complex, or only an approximation.

**Acknowledgements.** We would like to thank Bud Mishra and Chee Yap for reading an earlier version of this paper. The first author also wishes to thank Hans Køhling Pedersen for providing an initial impetus for thinking about this project, for interesting discussions, and for some useful code.

## References

[Aur91] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23:345–405, 1991.

[BF95] Jules Bloomenthal and Keith Ferguson. Polygonization of Non-Manifold implicit surfaces. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 309–316. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[Blo94] Jules Bloomenthal. An implicit surface polygonizer. In Paul Heckbert, editor, *Graphics Gems IV*, pages 324–349. Academic Press, Boston, 1994.

[CHL91] C.S. Chiang, C. M. Hoffman, and R. E. Lynch. How to compute offsets without self-intersection. In *Curves and Surfaces in Computer Vision and Graphics II*, volume 1610, pages 76–87. SPIE, 1991.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987.

[GO97] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press LLC, 1997.

[GP92] H. N. Gürsoy and N. M. Patrikalakis. An automatic coarse and fine surface mesh generation scheme based on medial axis transform: Part I algorithm. *Engineering with Computers*, 8:121–137, 1992.

[Hel91] M. Held. *On the Computational Geometry of Pocket Machining*, volume 500 of *Lecture Notes Comput. Sci.* Springer-Verlag, June 1991.

[Hel94] M. Held. On computing Voronoi diagrams of convex polyhedra by means of wavefront propagation. In *Proc. 6th Canad. Conf. Comput. Geom.*, pages 128–133, 1994.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–169, July 1987.

[Mil93] V. Milenkovic. Robust construction of the Voronoi diagram of a polyhedron. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 473–478, 1993.

[Moo92] Douglas Moore. Subdividing simplices. In David Kirk, editor, *Graphics Gems III*. Academic Press, New York, 1992.

[Sam90] H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, Reading, MA, 1990.

[SP95] E. C. Sherbrooke and N. M. Patrikalakis. Computation of medial axis transforms of 3d polyhedra. In J. R. Rossignac and C. M. Hoffmann, editors, *Proc. Third ACM Solid Modeling Conference*, 1995.

[STG+97] D.W. Storti, G. M. Turkiyyah, M. A. Ganter, C. T. Lim, and D. M. Stal. Skeleton-based modeling operations on solids. In *Proc. Third ACM Solid Modeling Conference*, 1997.

[VO95] J. Vleugels and M. Overmars. Approximating generalized Voronoi diagrams in any dimension. Report UU-CS-95-14, Dept. Comput. Sci., Utrecht Univ., Utrecht, Netherlands, 1995.