Interactive Computation of Discrete Generalized Voronoi Diagrams using Range Culling

Avneesh Sud Naga Govindaraju Dinesh Manocha

{sud,naga,dm}@cs.unc.edu
Department of Computer Science
University of North Carolina at Chapel Hill

Abstract

We present an algorithm for fast computation of discrete generalized Voronoi diagrams. Given a set of geometric primitives, and a distance metric, our algorithm efficiently computes the Voronoi diagram along a uniform grid by evaluating distance fields. We use a multipass approach and divide the computation into intervals along each dimension. We present an efficient culling algorithm to reduce the region for distance-field computation and use conservative bounds on Voronoi regions in 2D and 3D to cull away primitives. We have implemented our algorithm on programmable graphics hardware and applied it to compute the discrete Voronoi diagram of complex 2D and 3D datasets. We compute the discrete Voronoi diagram of $20,000$ primitives on a 2D grid of size 1200×1200 in less than 1 second. Our experimental results indicate a two to four times improvement over prior algorithms and implementations.

Keywords: generalized Voronoi diagram, distance field, medial axis, graphics hardware

1 Introduction

Given a set of objects ('sites'), the *Voronoi diagram* under a distance function is a spatial subdivision of the space into regions all points in a region have the same closest site according to the given distance function [Vleugels and Overmars 1998]. Voronoi diagrams can be generalized based on different distance functions, types of sites, and number of closest sites. The Voronoi diagram also provides information about the *distance field* which is defined at each point by the smallest distance from the point to the objects. Moreover, the *medial axis* of a geometric object is the set of interior points that have at-least two closest points on the boundary of the object and is closely related to the Voronoi diagram of the boundary of the closed object.

The Voronoi diagram is a fundamental geometric data structure which has been used for solving a large number of problems in the field of computational geometry [Aurenhammer 1991; Fortune 1992; Okabe et al. 1992]. Voronoi diagrams and medial axes have been used for a number of applications, including collision and proximity queries [Hoff et al. 2001], motion planning and navigation [Foskey et al. 2001; Hoff et al. 2000], point-location [Edelsbrunner et al. 1986], clustering [Schreiber 1991], mesh generation and finite element analysis [Sheffer et al. 1998; Suresh 2003], solid modeling [Blanding et al. 1999], design and interrogation [Patrikalakis and Gürsoy 1990; Wolter 1992], and shape simplification [Tam and Heidrich 2003].

The problem of computing Voronoi diagrams has been well studied in computational geometry and related areas. Good theoretical and practical algorithms are known for computing ordinary Voronoi diagrams of points in any dimensions. However accurate and efficient computation of Voronoi diagrams of higher order primitives, such as lines, polygons or higher-degree algebraic primitives remains a major challenge. The Voronoi boundaries are defined using non-linear algebraic equations and no good practical algorithms are known for computing them robustly.

Many authors have proposed algorithms to compute approximate Voronoi diagrams. At a broad level, they can be classified into *point-based* approximate Voronoi diagram computation and *discretized* Voronoi diagram computation The point-based Voronoi diagram algorithms approximate each higher order site with points and compute an ordinary Voronoi diagram of the point primitives. The discretized Voronoi diagrams compute the closest sites at a finite set of sample points in $\mathbb{R}^3$. Typically the set of points are the vertices of a uniform or an adaptive grid. Moreover, the computation of discretized Voronoi diagrams on a uniform grid can be accelerated using parallel rasterization capabilities of graphics hardware [Hoff et al. 1999; Denny 2003; Sud et al. 2004]. However, the resulting algorithms are not fast enough for interactive applications.

Main Results: We present a fast algorithm to compute discretized generalized Voronoi diagrams under distance metrics that are symmetric, positive definite and satisfy the triangle inequality. The domain of computation is divided into intervals along each dimension, called *ranges*. We evaluate the distance function for each site and Voronoi diagram is computed as the lower envelope of distance functions. We compute a bounded-error approximation of the distance function using graphics hardware. We use properties of Voronoi diagram to cull away sites that do not contribute the to Voronoi diagram in a range. Furthermore, we use compute conservative bounds on the Voronoi regions in 2D and 3D. Some of the novel aspects of our work include:

- A multipass algorithm to compute Voronoi region bounds in 2D and 3D. The underlying approach also extends to higher dimensions.
- A range-based culling technique to reduce the region of distance field computation.
- An efficient implementation on modern programmable graphics hardware, accurate upto IEEE 32-bit floating point precision.

We perform culling tests efficiently using visibility queries commonly found on modern graphics hardware. We use a conservative sampling strategy to account for undersampling errors that can arise within culling tests. We have implemented our algorithm on a 3.2GHz PC with an NVIDIA GeForce 7800 graphics processor and used it to compute the discrete Voronoi diagram of complex models consisting of

thousands of sites. The running time ranges from 60ms for small models (1k sites) to a second for large models (20k) sites, on a grid of size 1200×1200. In 3D, the running time for a model with 6K polygons is less than 1 second on a grid of size $256 \times 110 \times 85$.

Organization: The rest of the paper is organized in the following manner. We give a brief survey of Voronoi diagram computation algorithms in Section 2. We give an overview of our approach in Section 3. Section 4 describes our range-based culling algorithm and Section 5 presents an efficient implementation our algorithm on graphics hardware. We highlight its performance in Section 6 and analyze our algorithm in Section 7.

2 Previous Work

In this section we give a brief overview of previous work on computing Voronoi diagram, distance fields and medial axis. The algorithms for Voronoi diagram computation can be categorized based on different model representations.

Image datasets. Given discrete binary image data, many exact and approximate algorithms for distance field and discrete Voronoi diagram computation have been proposed [Mullikin 1992; Breen et al. 2000; Gibson 1998]. A good overview of these algorithms has been given in [Cuisenaire 1999]. The approximate methods compute the distance field in a local neighborhood of each voxel. Danielsson [1980] uses a scanning approach in 2D based on the assumption that the nearest object pixels are similar. The Fast Marching Method (FMM) [Sethian 1999] propagates a contour to compute the distance transformation from the neighbors. This provides an approximate finite difference solution to the Eikonal Equation $|\nabla u| = 1/f$. A linear time algorithm for computing exact Euclidean distance transform of a 2-D binary image is presented in [Breu et al. 1995]. This is extended to k-D images and other distance metrics [Maurer et al. 2003].

Geometric Models. There is extensive work in computing the exact Voronoi diagram of a set of points as the dual of the Delaunay triangulation of the points. A good survey of these algorithms is given in [Aurenhammer 1991]. For line segments in 2D, a sweep algorithm has been presented [Fortune 1987]. Hanniel et al [2005] present a method for extracting the Voronoi regions of free-form rational planar closed curves based on tracing of the bisector curves.

For geometric models represented using polygonal or higher order surfaces in 3D, many algorithms compute an approximation to the Voronoi diagram by computing distance fields on a uniform grid or an adaptive grid. A key issue in generating discrete distance samples is the underlying sampling rate used for adaptive subdivision. Many adaptive refinement strategies use a Voronoi region based labeling of the sample points to generate an octree spatial decomposition [Vleugels and Overmars 1998; Teichmann and Teller 1997; Etzion and Rappoport 2002]. For closed polyhedral and free-form models, algorithms are based on 3D tracing of the Voronoi edge curves [Milenkovic 1993; Sherbrooke et al. 1996; Reddy and Turkiyyah 1995; Muthuganapathy and Balan 2005]. These algorithms solve systems of algebraic equations to compute the Voronoi vertices and the Voronoi edge curves. Culver et al. [1999] used exact arithmetic to compute the Voronoi skeleton accurately. That algorithm has been used on models composed of hundreds or a few thousand triangles in 3D. All these algorithms have been applied to polyhedral models composed of only a few hundred triangles. The run time of these algorithms is more than $O(n^2)$, where n is the number of faces, or they are susceptible to accuracy and robustness problems.

Computation of a discrete Voronoi diagram on a uniform grid can be performed efficiently using parallel algorithms implemented on graphics hardware. Hoff et al. [1999] render a polygonal approximation of the distance function on depth-buffered graphics hardware and compute the generalized Voronoi Diagrams in two and three dimensions. This approach works on any geometric model that can be polygonized and is applicable to any distance function that can be rasterized. An efficient extension of the 2-D algorithm for point sites is proposed in [Denny 2003]. It uses precomputed depth textures, and a quadtree to estimate Voronoi region bounds. However, the extension of this approach to higher dimensions or higher order primitives is not presented. Sud et al [2004] present an approach for efficiently computing 3-D Voronoi diagrams of polygonal primitives by culling primitives which do not contribute to the final Voronoi diagram. This culling is performed along a single spatial dimension only.

The medial axis of a solid is closely related to its Voronoi region. Several approaches have been proposed to compute the medial axis of geometric models. These include algorithms that approximate the medial axis as a subset of the Voronoi complex of boundary point samples [Amenta et al. 2001; Dey and Zhao 2002]. Some algorithms for computing the medial axis of objects with higher order sites use Voronoi edge tracing techniques. Attali, Boissonat, and Edelsbrunner [Attali et al. 2004] have surveyed different techniques that generate a stable medial structure. Foskey et al. [2003] used graphics hardware to generate an image-space representation of the gradient of the distance field to the boundary, which can be analyzed to find the medial axis.

3 Overview

In this section we introduce the notation used in the rest of the paper, provide an background on the use of graphics hardware for computing Voronoi diagrams and present an overview of our approach.

3.1 Definitions

Let $\mathbf{q} = (q_1, q_2, \ldots, q_n)$ denote a point in n dimensions. For points in 3-dimensions, we use the standard Cartesian coordinates, $\mathbf{q} = (q_x, q_y, q_z)$. A geometric primitive or an object is called a *site*. Given a site p_i, the scalar distance function $d(\mathbf{q}, p_i)$ denotes the distance from the point $\mathbf{q} \in \mathbb{R}^n$ to the closest point on p_i. Given a set of sites $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$, the *Voronoi region* for p_i is defined as:

$$\mathcal{V}(p_i) = \{\mathbf{q} \mid d(\mathbf{q}, p_i) \leq d(\mathbf{q}, p_j) \forall p_j \in \mathcal{P}, \mathbf{q} \in M\},$$

where $M \subset \mathbb{R}^n$. The minimum distance of $\mathbf{q}$ to a set of sites is represented as $d(\mathbf{q}, \mathcal{P}) = \min_{p_i \in \mathcal{P}}(d(\mathbf{q}, p_i))$. Each point inside the Voronoi region $\mathcal{V}(p_i)$ is closer to the site p_i than

to any other site. The Voronoi diagram is a partition of M into Voronoi regions:

$$\mathrm{VD}_M(\mathcal{P}) = \bigcup_{p_i \in \mathcal{P}} (\mathcal{V}(p_i) \cap M).$$

Voronoi regions are closely related to distance fields. The *distance field* $D_M(\mathcal{P})$, for a domain M, is the scalar field given by the minimum distance function $d(\mathbf{q}, \mathcal{P})$ at all points $\mathbf{q} \in M$. For ease of notation, let $D_M = D_M(\mathcal{P})$. The Voronoi diagram can be represented as the projection of the distance field to the domain M. The *pivot* point of a site is any point lying on the site, and is represented $\pi(p_i)$.

Our goal is to compute a *discrete Voronoi diagram* within a bounded domain M represented as a uniform grid in n-dimensions. Each cell in the grid is sampled at a point, and the closest site is computed at this point. The discrete Voronoi region of site is the (finite) set of cells closer to that site than any other. We require that the domain M is a superset of the bounding box of all sites. This assumption is used to guarantee correctness of the culling algorithm. For convenience we shall refer to the domain M as the half-open unit interval in n-dimensions $(0, 1]^n$, i.e a unit square in 2D and the unit cube in 3D. We shall refer to an n-dimensional hypercube as an n-D *range* (a rectangular tile in 2D, a cube in 3D). The n-D range $(a_0, b_0] \times (a_1, b_1] \times \ldots \times (a_n, b_n]$ is represented as $T_{(a_0,b_0](a_1,b_1]\ldots(a_n,b_n]}$. We define each range to be a half-open set such that the intersection between two ranges is empty, and any point in M belongs to exactly one range. For ease of notation, let $\mathrm{VD}_M(\mathcal{P}) = \mathrm{VD}(\mathcal{P})$. Let $\mathcal{X}^c$, $\partial\mathcal{X}$ and $\mathrm{Int}(\mathcal{X})$ denote the complement, boundary and interior of a set $\mathcal{X}$, respectively. For any domain $N \subseteq M, N^c = M \setminus N$.

3.2 Computation using Graphics Hardware

A brute-force algorithm to compute $\mathrm{VD}(\mathcal{P})$ would evaluate $d(\mathbf{q}, p_i)$ for all sites $p_i \in \mathcal{P}$ and store the minimum at each grid point $\mathbf{q} \in M$. If there are m sites, and the grid has n cells, the time complexity of this algorithm is $O(mn)$. This brute force algorithm can be easily parallelized using depth-buffered graphics rasterization hardware [Hoff et al. 1999]. Graphics hardware is well suited for performing parallel computations on a 2D grid. Computation of the lower envelope is posed as a visibility problem along a view direction that is orthogonal to the 2D grid. In 2D, the resolution of the grid is governed by the image-space resolution of the graphics processors (e.g. 1000×1000). The primitive sites in $\mathcal{P}$ consist of points, edges and triangles.

For 3D grids, the discrete Voronoi diagram is computed as a sequence of 2D computations. The set of voxels with a constant z-value represents a uniform 2D grid and is called a *slice*. A slice s_k is defined as $s_k = \{(x, y, z) | (x, y, z) \in M, z = z_k\}$. A sweep is performed along the Z axis and the Voronoi diagram is computed for each slice. The complexity of this algorithm is linear in m for each slice and the running time can be slow when m is large. The set of sites for which the distance function is computed on a slice can be reduced using culling techniques along the sweep direction [Sud et al. 2004]. However, the distance function for a site is evaluated for all the voxels on a slice, which can be expensive.

3.3 Properties of Voronoi Diagrams

We speed up the Voronoi diagram computation by reducing the number of distance functions that are evaluated for each cell. In this section, we list the properties of Voronoi regions and distance fields that are used by our algorithm to accelerate the computation.

Connectivity: We consider distance metrics that are symmetric, positive definite and satisfy the triangle inequality. Thus, Voronoi regions defined by that distance metric are connected within the domain M. This is true for all L_p norms, including Euclidean distance and max-norm, if all the sites lie inside the domain M [Chew and Drysdale, III 1985]. Note that for higher order sites, like line segments and polygons, each Voronoi region may consist of non-linear boundaries and may not be convex. But each Voronoi region is connected.

Spatial Coherence: The distance values associated with two points in adjacent cells are typically very close to each other. We use the triangle inequality to compute bounds on the maximum change in the distance values between two ranges in the domain.

Our culling algorithm performs two sweeps in each dimension to obtain conservative bounds, along each dimension, of the Voronoi region of a site. The conservative bounds are used to reduce the set of points in the domain at which the distance function of a given site needs to be evaluated. We use the connectivity property and range-based sweeps to compute bounds of a Voronoi region (see figure 3).

3.4 Set Definitions

We introduce a classification of sites used by our algorithm to cull away sites that do not contribute to the distance field of a given range (see figure 1). Using the pivot point of the sites, the *swept set* for a range T is defined as

$$\mathcal{S}(T) = \{p_i \mid \pi(p_i) \in T, p_i \in \mathcal{P}\}.$$

Using bounds on the Voronoi regions of a site p_i, the *intersecting set* of a range T is defined as

$$\mathcal{I}(T) = \{p_i \mid \mathcal{V}(p_i) \cap T \neq \emptyset, p_i \in \mathcal{P}\}.$$

Thus for each point inside a range T, we have to compute the distance values to all sites in the intersecting set $\mathcal{I}(T)$. The *intersecting swept set* for two ranges T_1, T_2 is defined as

$$\mathcal{IS}(T_1, T_2) = \mathcal{I}(T_1) \cap \mathcal{S}(T_2)$$

The intersecting swept set represents the set of sites, which are swept by the second range and their Voronoi regions intersect the first range. Note that the definition is not symmetric. The receding set for a range T is the set of sites with Voronoi regions contained entirely inside T, and is defined as

$$\mathcal{R}(T) = \{p_i \mid \mathcal{V}(p_i) \subset \mathrm{Int}(T), p_i \in \mathcal{P}\}.$$

For two ranges T_i and T_j, if $T_i \subseteq T_j$ then $\mathcal{R}(T_i) \subseteq \mathcal{R}(T_j)$. By computing a receding set for a given range T, we can cull away the sites belonging to the receding set while computing the Voronoi diagram of its complement T^c. Our range based culling algorithm partitions M into a set of ranges, and computes the Voronoi diagram constrained to each range by computing a superset of the intersecting set for each range. The computation of a subset of the receding set is used to compute conservative estimate for each intersecting set.

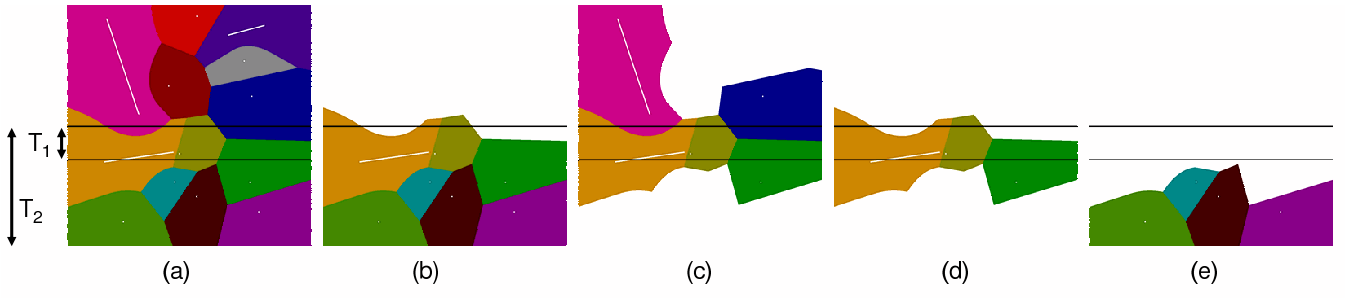

Figure 1: Set Definitions: (a) Voronoi diagram of 10 points and 3 lines and two ranges T_1 and T_2. (b) Swept set $\mathcal{S}(T_2)$ (c) Intersecting Set $\mathcal{I}(T_1)$ (d) Intersecting Swept set $\mathcal{IS}(T_1, T_2) = \mathcal{I}(T_1) \cap \mathcal{S}(T_2)$ (e) Receding set $\mathcal{R}(T_2 \setminus T_1) = \mathcal{S}(T_2) \setminus \mathcal{I}(T_1)$

4 Range Based Culling

In this section we present our algorithm for region-based culling and use it to accelerate Voronoi computations. We first present the idea in 2D and later extend it to higher dimensions.

4.1 2D Culling

In 2D, $M = (0,1] \times (0,1]$ and we have point, line and polygonal sites. The domain is partitioned into a set of rectangular ranges, called *tiles*. Our culling algorithm performs two sweeps along each dimension and incrementally culls away a subset of sites that do not belong to the intersecting swept set of the current tile. Next, we define the tiles, decompose of the Voronoi diagram computation into four sweeps as each tile decomposes the domain into 4 quadrants. Finally we present the update rule for incrementally computing the intersecting swept set in one sweep.

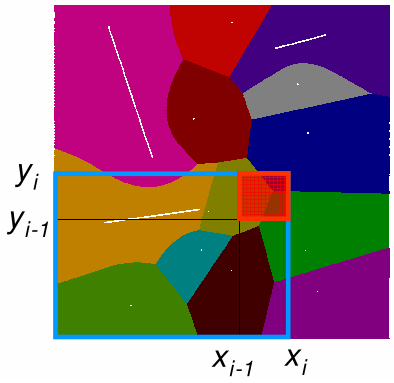

Figure 2: **Ranges in 2D***: The range T_{ij} is shown using a filled red rectangle. The range $T_{i^+j^+}$ is shown using thick blue borders.*

Given a set of $l + k + 2$ real numbers $x_0, x_1, \ldots, x_l, y_0, y_1, \ldots, y_k$ s.t. $x_0 = y_0 = 0, x_l = y_k = 1, x_i \in (0,1], y_j \in (0,1], x_i \geq x_{i-1}, y_j \geq y_{j-1}, 1 \leq i \leq l, 1 \leq j \leq k$. These $l + k + 2$ points partition M into $l \times k$ ranges with $T_{ij} = T_{(x_{i-1}, x_i](y_{j-1}, y_j]}$. Define the ranges $T_{i+j+} = (0, x_i] \times (0, y_j]$, $T_{i-j+} = (x_i, 1) \times (0, y_j]$, $T_{i+j-} = (0, x_i] \times (y_j, 1)$, and $T_{i-j-} = (x_i, 1) \times (y_j, 1)$ (see figure 2). We use the following lemma to compute the Voronoi diagram within the range T_{ij} using the intersecting swept sets.

Lemma 1. *Given $l \times k$ disjoint ranges which partition $(0,1]^2$,*

$$\begin{aligned}\mathrm{VD}_{T_{ij}}(\mathcal{P}) = \mathrm{VD}_{T_{ij}}(&\mathcal{IS}(T_{ij}, T_{i+j+}) \cup \mathcal{IS}(T_{ij}, T_{i-j+}) \cup \\ &\mathcal{IS}(T_{ij}, T_{i+j-}) \cup \mathcal{IS}(T_{ij}, T_{i-j-}))\end{aligned}$$

Proof. By definition, $\mathrm{VD}_{T_{ij}}(\mathcal{P}) = \mathrm{VD}_{T_{ij}}(\mathcal{I}(T_{ij}))$. Also,

$$\begin{aligned}\mathcal{I}(T_{ij}) =& \mathcal{I}(T_{ij}) \cap \mathcal{P} \\ =& \mathcal{I}(T_{ij}) \cap (\mathcal{S}(T_{i+j+}) \cup \mathcal{S}(T_{i-j+}) \cup \\ & \mathcal{S}(T_{i+j-}) \cup \mathcal{S}(T_{i-j-})) \\ =& \mathcal{IS}(T_{ij}, T_{i+j+}) \cup \mathcal{IS}(T_{ij}, T_{i-j+}) \cup \\ & \mathcal{IS}(T_{ij}, T_{i+j-}) \cup \mathcal{IS}(T_{ij}, T_{i-j-})\end{aligned}$$

Thus,

$$\begin{aligned}\mathrm{VD}_{T_{ij}}(\mathcal{P}) = \mathrm{VD}_{T_{ij}}(&\mathcal{IS}(T_{ij}, T_{i+j+}) \cup \mathcal{IS}(T_{ij}, T_{i-j+}) \cup \\ &\mathcal{IS}(T_{ij}, T_{i+j-}) \cup \mathcal{IS}(T_{ij}, T_{i-j-}))\end{aligned}$$

□

As a result of Lemma 1, we compute the Voronoi diagram $\mathrm{VD}(T_{ij})$ by computing four intersecting swept sets. We perform two passes along each axis, sweeping from 0 to 1 and then sweeping from 1 to 0, and compute the intersecting swept sets. Our approach for computing the intersecting swept sets for three other ranges (i^-j^+, i^+j^-, i^-j^-) is similar to the approach for computing the intersecting swept set $\mathcal{IS}(T_{ij}, T_{i^+j^+})$. However, the computation of exact intersecting swept set is equivalent to computing the exact Voronoi diagram. Instead, we present a simple theorem to efficiently compute a superset of the intersecting swept set. This conservative computation does not affect correctness of the algorithm, but influences the level of culling achieved for each range.

Theorem 1. *A superset of the intersecting swept set $\mathcal{IS}(T_{ij}, T_{i^+j^+})$ is given by the relation*

$$\begin{aligned}&\mathcal{IS}(T_{ij}, T_{i^+j^+}) \\ &\quad\subseteq \mathcal{IS}(T_{(i-1)j}, T_{(i-1)^+j^+}) \cup \mathcal{IS}(T_{i(j-1)}, T_{i^+(j-1)^+}) \cup \\ &\quad\quad \mathcal{S}(T_{ij}) \qquad (1)\end{aligned}$$

Proof. Let $\mathcal{X}$ denote the l.h.s of eq (1) and $\mathcal{Y}$ denote the r.h.s of eq (1). Let $p_a \in \mathcal{X} \Rightarrow \mathcal{V}(p_a) \cap T_{ij} \neq \emptyset$ and $\pi(p_a) \in T_{i^+j^+}$. We have two cases.

1. $\pi(p_a) \in T_{ij} \Rightarrow p_a \in \mathcal{S}(T_{ij}) \Rightarrow p_a \in \mathcal{Y}$.

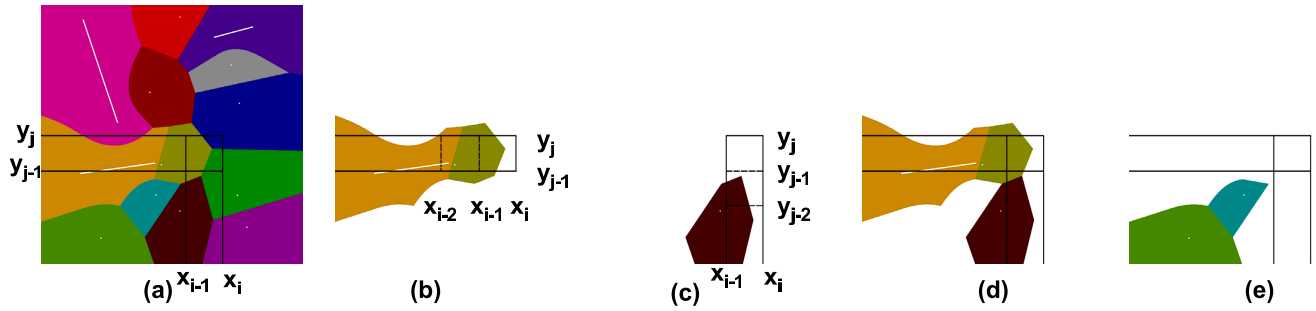

Figure 3: **PIS Computation in 2D***: This image highlights the Voronoi computation in a 2D range* $(x_{i-1}, x_i] \times (y_{j-1}, y_j]$ *based on the sweep along the* $+X$ *and* $+Y$ *axes. Fig. 3(a) shows the 2D Voronoi diagram of a set of points and lines and the 2D range. In Fig. 3(b), we highlight the PIS for the range* $(x_{i-1}, x_i] \times (y_{j-1}, y_j]$ *computed by sweeping along the* $+X$ *direction. Note that the PIS is conservatively computed as the union of PIS for the range* $(x_{i-2}, x_{i-1}] \times (y_{j-1}, y_j]$ *and the set of sites that intersect the range* $(x_{i-1}, x_i] \times (y_{j-1}, y_j]$*. Similarly, in Fig. 3(c), we show the computation of PIS for the range* $(x_{i-1}, x_i] \times (y_{j-1}, y_j]$ *computed using a sweep along the* $+Y$ *axis. The PIS for the sweep along both* $+X$ *and* $+Y$ *directions is shown in Fig. 3(d). The receding set is highlighted in Fig. 3(e). Based on the connectivity property of Voronoi diagrams, the sites in the receding set are ignored in the Voronoi diagram computation for any range beyond* $(0, x_i]$ *in the* $+X$ *direction and beyond* $(0, y_j]$ *in the* $+Y$ *direction.*

2. $\pi(p_a) \in T_{i^+j^+} \setminus T_{ij} \Rightarrow \mathcal{V}(p_a) \cap (T_{i^+j^+} \setminus T_{ij}) \neq \emptyset$. Since $\mathcal{V}(a)$ is connected, $\mathcal{V}(p_a) \cap (T_{i^+j^+} \setminus \partial T_{ij}) \neq \emptyset$. This implies either $\mathcal{V}(p_a) \cap (x_{i-1} \times (y_{j-1}, y_j] \neq \emptyset$ or $\mathcal{V}(p_a) \cap (x_{i-1} \times (y_{j-1}, y_j] \neq \emptyset$. Hence $p_a \in \mathcal{IS}\left(T_{(i-1)j}, T_{(i-1)^+j^+}\right) \cup \mathcal{IS}\left(T_{i(j-1)}, T_{i^+(j-1)^+}\right) \Rightarrow p_a \in \mathcal{Y}$.

□

Similarly, we can conservatively compute $\mathcal{IS}\left(T_{ij}, T_{i^-j^+}\right)$, $\mathcal{IS}\left(T_{ij}, T_{i^+j^-}\right)$, and $\mathcal{IS}\left(T_{ij}, T_{i^-j^-}\right)$. Theorem 1 indicates that the voronoi diagram $\mathrm{VD}_{T_{ij}}(\mathcal{P})$ can be computed incrementally within each range $(i^+j^+, i^+j^-, i^-j^+, i^-j^-)$. For example, in the range i^+j^+, both $\mathcal{IS}\left(T_{(i-1)j}, T_{(i-1)^+j^+}\right)$ and $\mathcal{IS}\left(T_{i(j-1)}, T_{i^+(j-1)^+}\right)$ have already been computed before the sweep reaches T_{ij} and these sets are then used for *incrementally* computing $\mathcal{IS}\left(T_{ij}, T_{i^+j^+}\right)$. The swept set $\mathcal{S}\left(T_{ij}\right)$ is easily computed by binning the sites into ranges using the pivot points. Fig. 3 highlights the incremental computation of the $\mathrm{VD}_{T_{ij}}(\mathcal{P})$ using sweep along +X and +Y directions.

Corollary 1. *Let a site* $p_a \in \mathcal{R}\left(T_{(i-1)^+(j-1)^+}\right)$*. Then* $p_a \notin \mathcal{IS}\left(T_{ij}, T_{i^+j^+}\right)$.

Proof. $p_a \in \mathcal{R}\left(T_{(i-1)^+(j-1)^+}\right)$
$\Rightarrow \mathcal{V}(p_a) \subset \mathrm{Int}(T_{(i-1)^+(j-1)^+}) \Rightarrow \pi(p_a) \in T_{(i-1)^+(j-1)^+}$
$\Rightarrow p_a \notin \mathcal{S}\left(T_{ij}\right)$. Also $\mathcal{V}(p_a) \cap \partial T_{(i-1)^+(j-1)^+} = \emptyset$ and by connectivity of Voronoi regions, $\mathcal{V}(p_a) \cap \mathcal{I}\left(T_{(i-1)j}\right) = \emptyset, \mathcal{V}(p_a) \cap \mathcal{I}\left(T_{i(j-1)}\right) = \emptyset$. Using the result of theorem 1, $p_a \notin \mathcal{IS}\left(T_{ij}, T_{i^+j^+}\right)$. □

A direct consequence of Corollary 1 is that one can check if a site belongs to the receding set of range $T_{i^+j^+}$ and cull it for Voronoi diagram computation in $T^c_{i^+j^+}$. Furthermore, in the three other passes, let $p_a \in \mathcal{R}\left(T_{k^-l^+}\right), \mathcal{R}\left(T_{m^+n^-}\right), \mathcal{R}\left(T_{p^-q^-}\right)$. Then $\mathcal{V}(p_a) \subset (\min(x_k, x_p), \max(x_i, x_m)] \times (\min(y_n, y_q), \max(y_j, y_l)]$, giving us spatial bounds on $\mathcal{V}(p_a)$.

4.2 Culling in 3D and Higher Dimensions

Our approach for range-based culling extends directly to higher dimensions. In n-D, let $M = (0, 1]^n$. As in section 4.1, let there be k_i ranges along each dimension, giving a total of $\prod_{i=1}^{n} k_i$ ranges. Let range $T_{i_1 i_2 \ldots i_n} = (x_{i_1 - 1}, x_{i_1}] \times (x_{i_2 - 1}, x_{i_2}] \times \ldots \times (x_{i_n - 1}, x_{i_n}]$, where $1 \leq i_j \leq k_j \ \forall\ 1 \leq j \leq n$, and x_{i_k} is the i^th coordinate in k^th dimension. Also, $T_{i_1^+ i_2^+ \ldots i_n^+} = (0, x_{i_1}] \times (0, x_{i_2}] \times \ldots \times (0, x_{i_n}]$, and the symmetric ranges along other sweep directions are defined similarly. In particular, range $T_{i_1 i_2 \ldots i_n}$ partitions M into 2^n swept ranges. Thus the intersecting set $\mathcal{I}\left(T_{i_1 i_2 \ldots i_n}\right)$ is partitioned into 2^n intersecting swept sets. We present a theorem that is used to compute a superset of the intersecting swept set:

Theorem 2. *A superset of the intersecting swept set* $\mathcal{IS}\left(T_{i_1 i_2 \ldots i_n}, T_{i_1^+ i_2^+ \ldots i_n^+}\right)$ *is given by the relation*

$$\begin{aligned} &\mathcal{IS}\left(T_{i_1 i_2 \ldots i_n}, T_{i_1^+ i_2^+ \ldots i_n^+}\right) \\ &\subseteq \ \mathcal{IS}\left(T_{(i_1-1) i_2 \ldots i_n}, T_{(i_1-1)^+ i_2^+ \ldots i_n^+}\right) \cup \\ &\quad \mathcal{IS}\left(T_{i_1 (i_2-1) \ldots i_n}, T_{i_1^+ (i_2-1)^+ \ldots i_n^+}\right) \cup \\ &\quad \ldots \\ &\quad \mathcal{IS}\left(T_{i_1 i_2 \ldots (i_n-1)}, T_{i_1^+ i_2^+ \ldots (i_n-1)^+}\right) \cup \\ &\quad \mathcal{S}\left(T_{i_1 i_2 \ldots i_n}\right) \end{aligned} \tag{2}$$

The proof is similar to that of Theorem 1 and uses the connectivity property to ensure that any Voronoi region intersecting the range $T_{i_1 i_2 \ldots i_n}$ must intersect one of its adjacent ranges, or the site must lie inside the range $T_{i_1 i_2 \ldots i_n}$. The following corollary gives a similar relation between the receding set $\mathcal{R}\left(T_{(i_1-1)^+(i_2-1)^+ \ldots (i_n-1)^+}\right)$ and the intersecting swept set $\mathcal{IS}\left(T_{i_1 i_2 \ldots i_n}, T_{i_1^+ i_2^+ \ldots i_n^+}\right)$.

Corollary 2. *Let a site* $p_a \in \mathcal{R}\left(T_{(i_1-1)^+(i_2-1)^+ \ldots (i_n-1)^+}\right)$*. Then* $p_a \notin \mathcal{IS}\left(T_{i_1 i_2 \ldots i_n}, T_{i_1^+ i_2^+ \ldots i_n^+}\right)$.

As in 2D, Corollary 2 provides conservative bounds on the spatial bounds of the Voronoi region of a site.

5 GPU Based Algorithm

In this section, we present our algorithm which uses the graphics hardware to efficiently compute the discrete generalized Voronoi diagram. Computation of the exact intersecting swept set $\mathcal{IS}(T_1, T_2)$ is equivalent to exact Voronoi computation. Instead we compute a set of *potentially intersecting swept* (PIS) sites, $\widehat{\mathcal{IS}}(T_1, T_{1'})$ which is a superset of the intersecting swept set $\mathcal{IS}(T_1, T_{1'})$. We use Corollary 1 to check if a site belongs to the receding set and use it to cull receding sites from the potentially intersecting swept set. To check for the membership in the receding set, we maintain conservative bounds $\widehat{\mathcal{V}}(p_a)$ on the Voronoi region $\mathcal{V}(p_a)$ of each site p_a, where $\widehat{\mathcal{V}}(p_a) \supseteq \mathcal{V}(p_a)$. The bounds are maintained at the resolution of a range T, i.e. $T \subseteq \widehat{\mathcal{V}}(p_a)$ if $\mathcal{V}(p_a) \cap T \neq \emptyset$. The key operation is to test if a Voronoi region $\mathcal{V}(p_a)$ intersects a given range T. A Voronoi region $\mathcal{V}(p_a)$ intersects a given range T if and only if the distance field of the site p_a $D_T(p_a)$ contributes to the final distance field $D_T(\mathcal{P})$. This computation is performed by testing the distance field $D_T(p_a)$ for visibility. The visibility computations are performed using occlusion queries (e.g. GL_NV_occlusion_query) available on current graphics systems. As the distance values are written to the depth buffer, these queries check for updates to the depth buffer and return the number of pixels that are visible.

We first describe the algorithm for computing 2D discrete Voronoi diagrams and then extend it to 3D discrete Voronoi diagrams.

5.1 2D Culling

In 2D, the domain is divided into $k \times l$ rectangular ranges, each range called a tile. All tiles with the same X limits form a row. The Voronoi diagram for the domain is computed by performing two sweeps across all rows. Within a row, we perform two sweeps across all tiles and compute the Voronoi diagram for the tile. The algorithm for computing the Voronoi diagram for the domain is given in Algorithm 1.

The function ComputeTile(T_{ij},$\widehat{\mathcal{IS}}(T_1, T_{1'})$ $\widehat{\mathcal{IS}}(T_2, T_{2'})$) computes the Voronoi diagram in the range T_{ij} using our incremental culling algorithm, where T_1, T_2 are adjacent to T_{ij}, and $T_{1'}$, $T_{2'}$ are the corresponding swept sets. It returns the updated potential intersecting swept set $\widehat{\mathcal{IS}}(T_{ij}, T_{i'j'})$ for T_{ij}. The details are given in Algorithm 2.

Based on Corollary 1, we need to check if the Voronoi region $\widehat{\mathcal{V}}(p_a)$ is a subset of the *interior* of the range T_{ij}, or equivalently if $\widehat{\mathcal{V}}(p_a)$ does not intersect the boundary of T_{ij}. The intersection test is performed with the entire range T_{ij} using visibility queries. To test if $\widehat{\mathcal{V}}(p_a)$ intersects the boundary of T_{ij}, we compute the intersection with the adjacent ranges $T_{(i+1)j}$, $T_{(i-1)j}$, $T_{i(j-1)}$, $T_{i(j+1)}$. The function *UpdateBounds*(p_a,T_{ij}) in Algorithm 2 updates the Voronoi region bound $\widehat{\mathcal{V}}(p_a)$ by adding T_{ij}. Thus *UpdateBounds*(p_a,T_{ij}) adds the adjacent ranges to the Voronoi region bounds $\mathcal{V}(p_a)$.

Input: Domain M, site set $\mathcal{P}$, num tiles k, l
Output: Voronoi Diagram $\mathrm{VD}_M(\mathcal{P})$

foreach *site* $p_a \in \mathcal{P}$ **do**
  Find tile T_{ij} s.t. $\pi(p_a) \in T_{ij}$
  Initialize $\widehat{\mathcal{V}}(p_a) \leftarrow T_{ij}$
end
for *j=1* to *l* **do**
  for *i=1* to *k* **do**
    $\left(\mathrm{VD}_{T_{ij}}(\mathcal{P}), \widehat{\mathcal{IS}}(T_{ij}, T_{i^+j^+})\right) \leftarrow$
    ComputeTile($T_{ij}, \widehat{\mathcal{IS}}(T_{(i-1)j}, T_{(i-1)^+j^+})$,
    $\widehat{\mathcal{IS}}(T_{i(j-1)}, T_{i^+(j-1)^+})$)
    $\mathrm{VD}_M(\mathcal{P}) \leftarrow \mathrm{VD}_M(\mathcal{P}) \cup \mathrm{VD}_{T_{ij}}(\mathcal{P})$
  end
  for *i = k* downto *1* **do**
    $\left(\mathrm{VD}_{T_{ij}}(\mathcal{P}), \widehat{\mathcal{IS}}(T_{ij}, T_{i^-j^+})\right) \leftarrow$
    ComputeTile($T_{ij}, \widehat{\mathcal{IS}}(T_{(i+1)j}, T_{(i+1)^-j^+})$,
    $\widehat{\mathcal{IS}}(T_{i(j-1)}, T_{i^-(j-1)^+})$)
    $\mathrm{VD}_M(\mathcal{P}) \leftarrow \mathrm{VD}_M(\mathcal{P}) \cup \mathrm{VD}_{T_{ij}}(\mathcal{P})$
  end
end
for *j=l* downto *1* **do**
  for *i=1* to *k* **do**
    $\left(\mathrm{VD}_{T_{ij}}(\mathcal{P}), \widehat{\mathcal{IS}}(T_{ij}, T_{i^+j^-})\right) \leftarrow$
    ComputeTile($T_{ij}, \widehat{\mathcal{IS}}(T_{(i-1)j}, T_{(i-1)^+j^-})$,
    $\widehat{\mathcal{IS}}(T_{i(j+1)}, T_{i^+(j+1)^-})$)
    $\mathrm{VD}_M(\mathcal{P}) \leftarrow \mathrm{VD}_M(\mathcal{P}) \cup \mathrm{VD}_{T_{ij}}(\mathcal{P})$
  end
  for *i = k* downto *1* **do**
    $\left(\mathrm{VD}_{T_{ij}}(\mathcal{P}), \widehat{\mathcal{IS}}(T_{ij}, T_{i^-j^-})\right) \leftarrow$
    ComputeTile($T_{ij}, \widehat{\mathcal{IS}}(T_{(i+1)j}, T_{(i+1)^-j^-})$,
    $\widehat{\mathcal{IS}}(T_{i(j+1)}, T_{i^-(j+1)^-})$)
    $\mathrm{VD}_M(\mathcal{P}) \leftarrow \mathrm{VD}_M(\mathcal{P}) \cup \mathrm{VD}_{T_{ij}}(\mathcal{P})$
  end
end

Algorithm 1: Compute2D(M, $\mathcal{P}$, k, l)

Input: Tile T_{ij}, PIS $\widehat{\mathcal{IS}}(T_1, T_{1'})$, PIS $\widehat{\mathcal{IS}}(T_2, T_{2'})$
  where T_1, T_2 are adj to T_{ij}
Output: Voronoi Diagram $\mathrm{VD}_{T_{ij}}(\mathcal{P})$, PIS
  $\widehat{\mathcal{IS}}(T_{ij}, T_{i'j'})$

Update $\widehat{\mathcal{IS}}(T_{ij}, T_{i'j'}) \leftarrow \widehat{\mathcal{IS}}(T_1, T_{1'}) \cup \widehat{\mathcal{IS}}(T_2, T_{2'}) \cup \mathcal{S}(T_{ij})$
foreach *site* $p_a \in \widehat{\mathcal{IS}}(T_{ij}, T_{i'j'})$ **do**
  if $\widehat{\mathcal{V}}(p_a) \cap T_{ij} = \emptyset$ **then**
    $\widehat{\mathcal{IS}}(T_{ij}, T_{i'j'}) \leftarrow \widehat{\mathcal{IS}}(T_{ij}, T_{i'j'}) \setminus \{p_a\}$
  Compute distance field $D_{T_{ij}}(p_a)$
  Update $\mathrm{VD}_{T_{ij}}(\widehat{\mathcal{IS}}(T_{ij}, T_{i'j'}))$
  Check $D_{T_{ij}}(p_a)$ for visibility
  if $D_{T_{ij}}(p_a)$ *is visible* **then**
    UpdateBounds(p_a,T_{ij})
end

Algorithm 2: ComputeTile(T_{ij},$\widehat{\mathcal{IS}}(T_1, T_{1'})$)

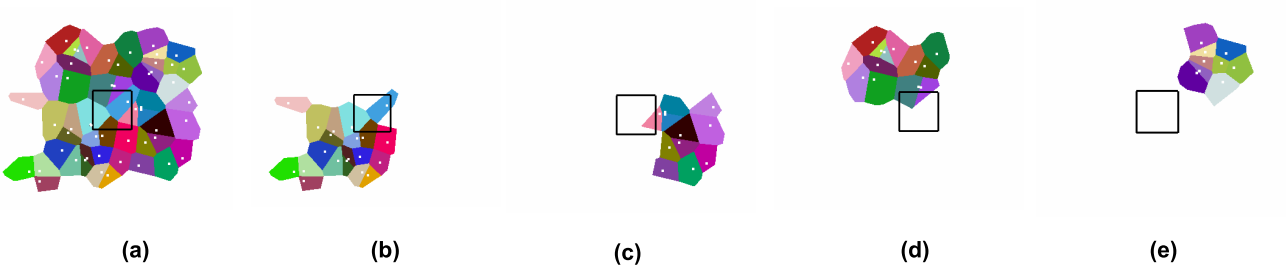

Figure 4: **PIS Computation in 2D***: This image highlights the PIS sets and the Voronoi regions computed for the tile T_{ij} shown in black during each of the four sweeps in 2D (a) The final potential intersecting set $\mathcal{I}(T_{ij})$ (b) PIS $\widehat{\mathcal{IS}}(T_{ij}, T_{i^+j^+})$ (c) PIS $\widehat{\mathcal{IS}}(T_{ij}, T_{i^-j^+})$ (d) PIS $\widehat{\mathcal{IS}}(T_{ij}, T_{i^+j^-})$ (e) PIS $\widehat{\mathcal{IS}}(T_{ij}, T_{i^-j^-})$*

5.2 3D Culling

In 3D, the domain is divided into $k \times l \times m$ cubical ranges. The set of ranges with the same Z coordinate forms a 2D domain called a slice. A slice is further divided into $k \times l$ rectangular tiles. We compute the 3D Voronoi diagram by computing m slices. Computation of a 2D slice is done as shown in Algorithm 1.

5.3 Conservative Sampling

The occlusion queries sample the visibility at fixed locations in each pixel and can result in sampling errors. In particular, the algorithm presented above may incorrectly classify a site p_a as receding if its Voronoi region $\mathcal{V}(p_a)$ does not cover any grid cells, i.e. the occlusion query returns zero visible pixels for the distance field $D_{T_{ij}}(p_a)$. This may introduce errors when $\mathcal{V}(p_a)$ intersects the range T_{ij} but its intersection with T_{ij} is not sampled by the rasterization hardware. An incorrect classification of p_a as receding can lead to errors in the Voronoi diagram of subsequent ranges.

We account for these sampling errors using a conservative sampling approach presented in [Sud et al. 2004]. This involves expanding the Voronoi region of each site by the size of a grid cell and again testing for visibility. Updates to the depth and color buffers are disabled during this computation.

6 Implementation and Results

In this section we describe the implementation of our discrete generalized Voronoi diagram computation algorithm and highlight its performance on different benchmarks.

We have implemented our algorithm using Microsoft Visual C++ and OpenGL graphics API. The distance functions for each primitive are computed at each grid cell on programmable graphics hardware using the OpenGL's ARB_fragment_program extension. The visibility test is performed using the OpenGL occlusion query extension GL_NV_occlusion_query. We efficiently utilize the parallelism on a GPU by batching together the occlusion queries for an entire set of potentially intersecting sites. Our implementation involves no pre-computation and is directly applicable to deformable models.

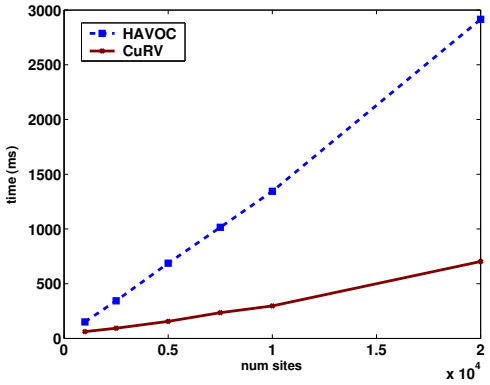

Figure 5: **Timing Comparison***: Growth of time to compute the 2D discrete Voronoi diagram with number of random sites, using HAVOC(Hoff et al. 99) and CuRV (our algorithm). We have used a high grid resolution of* 1200×1200 *for the computation of discrete Voronoi diagram and used a tile size equal to* 75×75*.*

In 2D, our sweep based algorithm computes the Voronoi diagram for all tiles in row i before computing the Voronoi diagram for tiles in row $(i+1)$. We store the Voronoi region bounds as intervals along X and Y axes. In particular, we need to store the interval along X for the current and previous rows only, giving tighter bounds compared to an AABB. In 3D, we store the Voronoi region bounds along X and Y for the current and previous slices.

6.1 Performance

We have applied our algorithm to several 2D models of points and lines. The sites are distributed randomly across the domain. We have tested the performance of our algorithm on a Pentium4 3.2GHz PC with 2GB RAM and an NVIDIA GeForce 7800 GTX graphics card, running Windows XP. We have compared the performance of our Voronoi diagram computation algorithm (called CuRV) with the algorithm presented by Hoff et al. [1999] (called HAVOC). We have used efficient techniques to compute 3D distance fields on graphics hardware using linear factorization [Sud et al. 2005].

We have measured the performance of our algorithm with varying number of sites. Fig. 5 highlights the performance on upto 20K sites. We observe that the Voronoi computation time scales linearly with the number of sites. Furthermore, the computation time scales better than HAVOC. We have

applied our algorithm to compute 3D distance field and discrete generalized Voronoi diagram of polyhedral models (see figure 7). The computation cost of the Voronoi diagram is directly proportional to the *fill rate*. The fill rate is the number of sample points (pixels) where the distance function computation is evaluated. Fig. 6 shows the fill rate requirements of CuRV and HAVOC.

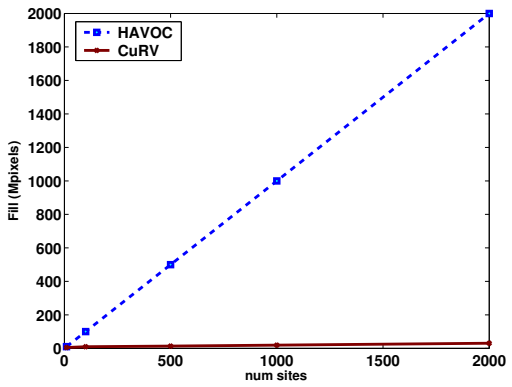

Figure 6: **Fill Rate***: Number of pixels where distance function is computed, using HAVOC(Hoff et al. 99) and CuRV (our algorithm). We have used a grid resolution of* 1024×1024*, and a tile size of* 32×32 *to compute the Voronoi diagram. Our results indicate upto two orders of magnitude reduction in fill over HAVOC.*

Our experimental results indicate up to 5 times performance improvement over HAVOC and approximately two orders of magnitude reduction in the overall fill rate.

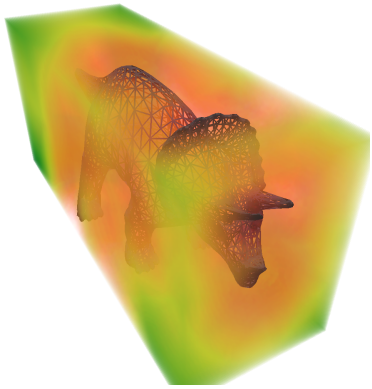

Figure 7: **3D Distance Field***: Computation of 3D distance field and discrete Voronoi diagram of Triceratops model (5660 polygons). Distance increases from red to green. Grid size* $= 255 \times 111 \times 84$*, Computation time* $= 520ms$*.*

7 Discussion

In this section we analyze the computational complexity, space requirements and the accuracy of our algorithm. We also compare our algorithm with some existing discrete Voronoi diagram computation algorithms.

7.1 Analysis

Let the 2D domain M contain $k \times l$ tiles, each covering m grid cells (pixels). We introduce the notion of *average PIS size* which gives the average number of sites for which the distance field is computed per tile, and is defined as

$$\langle \widehat{\mathcal{IS}} \rangle = \frac{1}{kl} \sum_{i=1}^{k} \sum_{j=1}^{l} (|\widehat{\mathcal{IS}}\left(T_{ij}, T_{i^+j^+}\right)| + |\widehat{\mathcal{IS}}\left(T_{ij}, T_{i^-j^+}\right)| + |\widehat{\mathcal{IS}}\left(T_{ij}, T_{i^+j^-}\right)| + |\widehat{\mathcal{IS}}\left(T_{ij}, T_{i^-j^-}\right)|)$$

In higher dimensions, $\langle \widehat{\mathcal{IS}} \rangle$is similarly defined. Then the cost of updating the PIS in algorithm 2 is $O(\langle \widehat{\mathcal{IS}} \rangle \log \langle \widehat{\mathcal{IS}} \rangle)$. The cost of computing the distance field for each site is proportional to the number of grid cells (pixels) inside the tile, $O(m)$. Thus the total cost of one call to algorithm 2 is $O(\langle \widehat{\mathcal{IS}} \rangle \log(\langle \widehat{\mathcal{IS}} \rangle) + \langle \widehat{\mathcal{IS}} \rangle m)$, and the cost of algorithm 1 is $O\left(2^2 kl(\langle \widehat{\mathcal{IS}} \rangle \log \langle \widehat{\mathcal{IS}} \rangle + m\langle \widehat{\mathcal{IS}} \rangle)\right)$. Similarly in n dimensions, the computational cost is $O\left(2^n \prod_{i=1}^{n} k_i(\langle \widehat{\mathcal{IS}} \rangle \log \langle \widehat{\mathcal{IS}} \rangle + m\langle \widehat{\mathcal{IS}} \rangle)\right)$. Note that $\prod_{i=1}^{n} k_i \times m$ gives the total number of grid cells in domain M.

In terms of storage cost, we have to store the PIS for each tile. Thus the storage cost increases by $O(kl\langle \widehat{\mathcal{IS}} \rangle)$ in 2D and $O(\prod_{i=1}^{n} k_i \langle \widehat{\mathcal{IS}} \rangle)$ in n dimensions.

7.2 Comparison

We now compare our algorithm with some previous approaches to compute discrete generalized Voronoi diagrams. HAVOC ([Hoff et al. 1999]) computes discrete generalized Voronoi diagrams in 2 and 3 dimensions, under any distance function. However the computational complexity of HAVOC is $O(MN)$, where M is number of sites in $\mathcal{P}$ and N is number of grid cells in M. This approach does not scale well with large number of sites and in higher dimensions. Furthermore, the discrete Voronoi diagram computed by HAVOC can have significant errors in computed Voronoi regions and can deviate from the exact Voronoi region by more than a single cell (see Fig. 8). This error is caused due to the tessellation error in the distance functions used by HAVOC. In comparison, CuRV provides bounds on the region of distance computation for each site, and the approach is extensible to n dimensions, and scales well to large number of sites. Also, the computed distance field is accurate to 32-bit floating point precision, hence the discrete Voronoi region deviates from the exact boundary by at most one cell (see figure 8). However, in order to compute the Voronoi diagram efficiently, CuRV utilizes the connectivity property of Voronoi diagrams. Therefore, it is applicable for only distance functions which are metrics.

Denny et al.[2003] present an efficient approach for computing 2D discrete Voronoi diagrams for points under Euclidean distance. This approach increases the amount of tessellation to bound errors in the Voronoi region boundaries to 1 cell size. However, the approach is not directly extensible to 3D and higher order sites, and is sensitive to the order of computation of the distance functions. In comparison, our algorithm is simple and applicable to both higher order sites and dimensions.

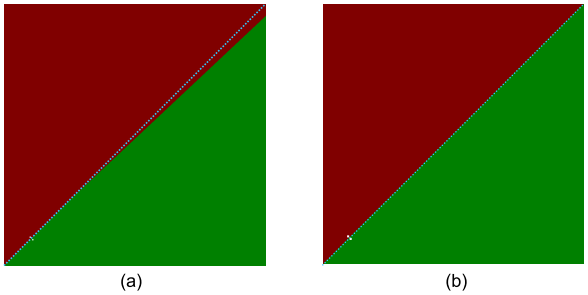

Figure 8: **Voronoi Diagram Accuracy:** *Error in Voronoi region computation in (a) HAVOC [Hoff et al 99] and (b) CuRV (our algorithm). There are two point sites close to the diagonal. The exact boundary is indicated using a dotted blue line. With HAVOC the error can be several pixels, whereas it is at most 1 pixel with CuRV*

8 Conclusions and Future Work

We have presented an algorithm for fast computation of discrete generalized Voronoi diagrams. Our algorithm uses a culling technique to reduce the number of distance computations performed inside a range. We also compute conservative Voronoi region bounds in 2D and 3D and the approach extends to higher dimensions. We have described an efficient implementation of our algorithm on modern programmable graphics hardware and used to compute the discrete Voronoi diagram of geometric primitives. We achieve two to four times improvement over prior algorithms and implementations.

There are many avenues for future work. We would like to further exploit spatial coherence in 3D for further culling. We would also like to incorporate hierarchical techniques and also extend the approach to dynamic inputs. Finally, we would also like to use our algorithm for other applications, including dynamic simulation, motion planning and proximity computations.

Acknowledgments

This research is supported in part by ARO Contract DAAD 19-02-1-0390, and W911NF-04-1-0088, NSF Awards 0400134, 0118743, DARPA and RDECOM Contract N61339-04-C-0043, DOD Prostate Cancer Research Program DAMD17-03-1-0134 and Intel Corporation. We thank the UNC GAMMA group for many useful discussions and support. We are also grateful to the reviewers for their feedback.

References

Amenta, N., Choi, S., and Kolluri, R. K. 2001. The power crust. In *Proc. ACM Symposium on Solid Modeling and Applications*, 249–260.

Attali, D., Boissonat, J.-D., and Edelsbrunner, H. 2004. Stability and computation of the medial axis. In *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*. Springer-Verlag.

Aurenhammer, F. 1991. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv. 23*, 3 (Sept.), 345–405.

Blanding, R., Brooking, C., Ganter, M., and Storti, D. 1999. A skeletal-based solid editor. In *Proc. ACM Symposium on Solid Modeling and Applications*, 141–150.

Breen, D., Mauch, S., and Whitaker, R. 2000. 3d scan conversion of csg models into distance, closest-point and color volumes. *Proc. of Volume Graphics*, 135–158.

Breu, H., Gil, J., Kirkpatrick, D., and Werman, M. 1995. Linear time Euclidean distance transform and Voronoi diagram algorithms. *IEEE Trans. Pattern Anal. Mach. Intell. 17*, 529–533.

Chew, L. P., and Drysdale, III, R. L. 1985. Voronoi diagrams based on convex distance functions. In *ACM Symposium on Computational Geometry*, 235–244.

Cuisenaire, O. 1999. *Distance Transformations: Fast Algorithms and Applications to Medical Image Processing*. PhD thesis, Universite Catholique de Louvain.

Culver, T., Keyser, J., and Manocha, D. 1999. Accurate computation of the medial axis of a polyhedron. In *Proc. ACM Symposium on Solid Modeling and Applications*, 179–190.

Danielsson, P. E. 1980. Euclidean distance mapping. *Computer Graphics and Image Processing 14*, 227–248.

Denny, M. 2003. Solving geometric optimization problems using graphics hardware. *Computer Graphics Forum 22*, 3.

Dey, T. K., and Zhao, W. 2002. Approximate medial axis as a Voronoi subcomplex. In *Proc. ACM Symposium on Solid Modeling and Applications*, 356–366.

Edelsbrunner, H., Guibas, L. J., and Stolfi, J. 1986. Optimal point location in a monotone subdivision. *SIAM J. Comput. 15*, 2, 317–340.

Etzion, M., and Rappoport, A. 2002. Computing Voronoi skeletons of a 3-d polyhedron by space subdivision. *Computational Geometry: Theory and Applications 21*, 3 (March), 87–120.

Fortune, S. J. 1987. A sweepline algorithm for Voronoi diagrams. *Algorithmica 2*, 153–174.

Fortune, S. 1992. Voronoi diagrams and Delaunay triangulations. In *Computing in Euclidean Geometry*, D.-Z. Du and F. K. Hwang, Eds., 1st ed., vol. 1 of *Lecture Notes Series on Computing*. World Scientific, Singapore, 193–233.

Foskey, M., Garber, M., Lin, M., and Manocha, D. 2001. A voronoi-based hybrid planner. *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*.

Foskey, M., Lin, M., and Manocha, D. 2003. Efficient computation of a simplified medial axis. *Proc. of ACM Solid Modeling*, 96–107.

Gibson, S. 1998. Using distance maps for smooth representation in sampled volumes. In *Proc. of IEEE Volume Visualization Symposium*, 23–30.

Hanniel, I., Ramanathan, M., Elber, G., and Kim, M. S. 2005. Voronoi region extract of free-form rational

planar closed curves. In *Symposium on Solid and Physical Modeling*.

Hoff, K., Culver, T., Keyser, J., Lin, M., and Manocha, D. 1999. Fast computation of generalized voronoi diagrams using graphics hardware. *Proceedings of ACM SIGGRAPH 1999*, 277–286.

Hoff, K., Culver, T., Keyser, J., Lin, M., and Manocha, D. 2000. Interactive motion planning using hardware accelerated computation of generalized voronoi diagrams. *IEEE Conference on Robotics and Automation*, pp. 2931–2937.

Hoff, K., Zaferakis, A., Lin, M., and Manocha, D. 2001. Fast and simple 2d geometric proximity queries using graphics hardware. *Proc. of ACM Symposium on Interactive 3D Graphics*, 145–148.

Maurer, C., Qi, R., and Raghavan, V. 2003. A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitary dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence 25*, 2 (February), 265–270.

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

Mullikin, J. C. 1992. The vector distance transform in two and three dimensions. *CVGIP: Graphical Models and Image Processing 54*, 6 (Nov.), 526–535.

Muthuganapathy, R., and Balan, G. 2005. A tracing algorithm for constructing medial axis transform of 3d objects bound by free-form surfaces. In *Proc. International Conference on Shape Modeling and Applications*.

Okabe, A., Boots, B., and Sugihara, K. 1992. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Chichester, UK.

Patrikalakis, N. M., and Gürsoy, H. N. 1990. Shape interrogation by medial axis transform. In *Proc. 16th ASME Design Automation Conference*.

Reddy, J., and Turkiyyah, G. 1995. Computation of 3D skeletons using a generalized Delaunay triangulation technique. *Comput. Aided Design 27*, 9, 677–694.

Schreiber, T. 1991. A Voronoi diagram based adaptive k-means-type clustering algorithm for multidimensional weighted data. In *Proc. Computational Geometry: Methods, Algorithms and Applications*, Springer-Verlag, vol. 553 of *Lect. Notes in Comp. Sci.*, 265–275.

Sethian, J. A. 1999. *Level set methods and fast marching methods*. Cambridge.

Sheffer, A., Etzion, M., Rappoport, A., and Bercovier, M. 1998. Hexahedral mesh generation using the embedded voronoi graph. *7th International Meshing Roundtable*, 347–364.

Sherbrooke, E. C., Patrikalakis, N. M., and Brisson, E. 1996. An algorithm for the medial axis transform of 3d polyhedral solids. *IEEE Trans. Visualizat. Comput. Graph. 2*, 1 (Mar.), 45–61.

Sud, A., Otaduy, M. A., and Manocha, D. 2004. DiFi: Fast 3D distance field computation using graphics hardware. *Computer Graphics Forum (Proc. Eurographics) 23*, 3, 557–566.

Sud, A., Govindaraju, N., and Manocha, D. 2005. Interactive 3D distance field computation using linear factorization. Tech. Rep. TR05-020, Dept. of Computer Science, University of North Carolina at Chapel Hill.

Suresh, K. 2003. Automating the CAD/CAE dimensional reduction process. In *Proc. ACM Symposium on Solid Modeling and Applications*, 76–85.

Tam, R., and Heidrich, W. 2003. Shape simplification based on the medial axis transform. *IEEE Visualization*.

Teichmann, M., and Teller, S. 1997. Polygonal approximation of Voronoi diagrams of a set of triangles in three dimensions. Tech. Rep. 766, Laboratory of Computer Science, MIT.

Vleugels, J., and Overmars, M. H. 1998. Approximating Voronoi diagrams of convex sites in any dimension. *International Journal of Computational Geometry and Applications 8*, 201–222.

Wolter, F. E. 1992. Cut locus and medial axis in global shape interrogation and representation. Tech. Rep. 92-2, MIT, Dept. Ocean Engg., Design Lab, Cambridge, MA 02139, USA, Jan.