Fast Distance Field Computation Using Graphics Hardware

Avneesh Sud and Dinesh Manocha
Department of Computer Science, Univeristy of North Carolina
Chapel Hill, NC, USA
{sud,dm}@cs.unc.edu
http://gamma.cs.unc.edu/DiFi

UNC Computer Science Technical Report TR03-026

Abstract

We present an algorithm for fast computation of discretized 3D distance fields using graphics hardware. Given a set of primitives and a distance metric, our algorithm computes the distance field for each slice of a uniform spatial grid by rasterizing the distance functions of the primitives. We compute bounds on the spatial extent of the Voronoi region of each primitive. These bounds are used to cull and clamp the distance functions rendered for each slice. Our algorithm is applicable to all geometric models and does not make any assumptions about connectivity or a manifold representation. We have used our algorithm to compute distance fields of large models composed of tens of thousands of primitives on high resolution grids. Moreover, we demonstrate its application to medial axis evaluation and proximity computations. As compared to earlier approaches, we are able to achieve an order of magnitude improvement in the running time.

Keywords: Distance fields, Voronoi regions, graphics hardware, proximity computations

1 Introduction

Given a set of objects, a distance field in 3D is defined at each point by the smallest distance from the point to the objects. Each object may be represented as data on a voxel grid or as an explicit surface representation. Moreover, the distances between the point and an object can be specified using different metrics, including Euclidean or max-norm distance. If the primitives are closed or orientable, a sign can be assigned to the distance field.

Distance fields are frequently used in computer graphics, geometric modeling, robotics and scientific visualization. Their applications include shape representation [10, 29, 30], model simplification [14], remeshing [20, 30], morphing [5], CSG operations [1, 2], sculpting [25], swept volume computation [19], path planning and navigation [15, 18], collision and proximity computations [16, 17], etc. These applications use a signed or unsigned distance field along a discrete grid.

Different algorithms have been proposed to compute the distance fields in 2D or 3D for geometric and volumetric models. The computation of a distance field along a uniform grid can be accelerated by using graphics rasterization hardware [15, 8, 28]. These algorithms compute 2D slices of the 3D distance field by rendering the three dimensional distance function for each primitive. However, rendering the distance meshes of all the primitives for each slice may become expensive in terms of transformation and rasterization cost. As a result, current algorithms for 3D distance field computation may be slow and not work well for deformable models or dynamic environments.

Main Contributions: We present a fast algorithm (DiFi) to compute a distance field of complex objects along a 3D grid. We use a combination of novel culling and clamping algorithms that render relatively few distance meshes for each slice. We also exploit spatial coherence between adjacent slices in 3D and perform incremental computations to speed up the overall algorithm.

Our novel site culling algorithm uses properties of the Voronoi diagram to cull away primitives that do not contribute to the distance field of a given slice. We use a two-pass approach and perform culling using occlusion queries. Furthermore, we present a conservative sampling strategy that accounts for sampling errors in the occlusion queries. Our clamping algorithm reduces the rasterization cost of each distance function by rendering it on a portion of each slice.

We have implemented DiFi on a 2.8GHz PC with an NVIDIA GeForce FX 5900 Ultra graphics processor and used it to compute distance fields of complex objects consisting of tens to hundreds of thousands of triangles. The running time ranges from a second for small models to tens

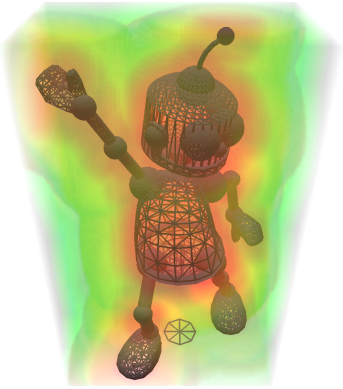

Figure 1. $3D$ Distance Field of Hugo Model*(*$17k$ *polygons): Distance to the surface is color coded, increasing from red to green to blue. Time taken to compute the distance field on* $73 \times 45 \times 128$ *grid using our algorithm is* 4.2 *seconds.*

of seconds for large models. We have used DiFi to compute the simplified medial axis of complex polyhedral models and perform proximity computations in a dynamic environment for path planning. As compared to prior distance field computation algorithms, our approach offers the following advantages:

- **Generality:** No assumption is made with regards to the input models. The objects may be non-orientable or non-manifold surfaces, or may be represented using voxel data.
- **Efficiency:** We show that our approach is significantly faster than previous approaches. The culling techniques provide us with a $3 - 20$ times speedup in distance field computation over previous approaches that can handle generic models. The speedups are higher for complex models with a high number of primitives.
- **Dynamic Models:** Our algorithm involves no preprocessing and can compute distance fields of dynamic objects with a few thousand polygons at almost interactive rates.

Organization: The rest of the paper is organized in the following manner. We give a brief survey of distance field computation algorithms in Section 2 and an overview of our approach in Section 3. Section 4 describes our culling algorithm and Section 5 presents the clamping algorithm. In Section 6, we highlight two applications of our distance field computation algorithm. We describe its performance in Section 7 and analyze it in Section 8.

2 Related Work

The problem of computing a distance field can be broadly classified by the type of input object representation. The object can be specified either as a data on a voxel grid, such as a binary image or as an explicit surface representation, such as a triangulated model.

2.1 Distance Fields of Geometric Models

Many algorithms are known to compute the distance fields of geometric models represented using polygonal or higher order surfaces. These algorithms use either 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 trilinear interpolation or curvature information to generate an octree spatial decomposition [27, 10, 25, 31].

Distance field computation can be accelerated using graphics hardware. The graphics hardware based algorithms compute a 2D slice of the distance field at a time. Hoff et al. [15] render a polygonal approximation of the distance function on the depth-buffer 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 [8]. 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. A class of exact distance transform algorithms is based on computing partial Voronoi diagrams [21]. A scan-conversion method to compute the 3-D Euclidean distance field in a narrow band around manifold triangle meshes is the Characteristics/Scan-Conversion (CSC) algorithm [22]. The CSC algorithm uses the connectivity of the mesh to compute polyhedral bounding volumes for the Voronoi cells. The distance function for each site is evaluated only for the voxels lying inside this polyhedral bounding volume. An efficient GPU based implementation of the CSC algorithm is presented in [28]. The number of polygons sent to the graphics pipeline is reduced and the non-linear distance functions are evaluated using fragment programs.

2.2 Volumetric Models

Given voxel data, many exact and approximate algorithms for distance field computation have been proposed [24, 2, 12]. A good overview of these algorithms has been given in [6]. The approximate methods compute the distance field in a local neighborhood of each voxel. Danielsson [7] uses a scanning approach in 2D based on the assumption that the nearest object pixels are similar. The Fast Marching Method (FMM)[26] 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 [3]. This is extended to k-D images and other

distance metrics [23].

3 Overview and Notation

In this section we introduce the notation used in the paper and give an overview of our approach.

3.1 Distance Fields

A geometric primitive or an object in 3D is called a *site*. Given a site p_i, the scalar distance function $dist(\mathbf{q}, p_i)$ denotes the distance from the point $\mathbf{q} \in \mathbb{R}^3$ to the closest point on p_i. The minimum distance of $\mathbf{q}$ to a set of sites $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$ is represented as $dist(\mathbf{q}, \mathcal{P}) = \min_{p_i \in \mathcal{P}}(dist(\mathbf{q}, p_i))$. The *distance field* $D_M(\mathcal{P})$, for a domain $M \subset \mathbb{R}^3$, is the scalar field given by the minimum distance function $dist(\mathbf{q}, \mathcal{P})$ for all points $\mathbf{q} \in M$. For ease of notation, let $D_M = D_M(\mathcal{P})$. Given a subset, $\mathcal{X} \subset \mathcal{P}$, $dist(\mathbf{q}, \mathcal{X}) \geq dist(\mathbf{q}, \mathcal{P}) \forall \mathbf{q} \in M$.

Distance fields are closely related to Voronoi regions. The Voronoi region for p_i is defined as:

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

Our goal is to compute the distance field within a bounded domain M represented as an axis-aligned uniform 3D grid. Let the size of each voxel in the 3D grid be $\delta_x \times \delta_y \times \delta_z$. In a bounded domain, Voronoi regions are bounded. Let the minimum and maximum bounds of the Voronoi region of a site p_i along Z be $V(p_i).z_{min}$ and $V(p_i).z_{max}$, respectively.

3.2 Computation using Graphics Hardware

A brute-force algorithm to compute D_M would evaluate $dist(\mathbf{q}, p_i)$ 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 [15]. The primitive sites in $\mathcal{P}$ consist of points, edges and triangles. 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 $s_k = \{(x, y, z) | (x, y, z) \in M, z = z_k\}$. In the rest of the paper, we represent the distance field $D_{s_k}(\mathcal{P})$ for a slice s_k as $D_k(\mathcal{P})$. A sweep is performed along the Z axis and the distance field D_k 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.

3.3 Our Approach

We speed up the 3D distance field computation by reducing the number of distance functions that are rasterized for each slice. We exploit the following properties of Voronoi regions and distance fields to accelerate the computation:

1. **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. This is true for all L_p norms, including Euclidean distance and max-norm [4]. 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.

2. **Spatial Coherence:** The distance fields of adjacent slices, s_k and s_{k+1}, can have high spatial coherence. The distance values associated with two points in adjacent voxels on a 3D grid will be very close to each other. We use this coherence to compute bounds on the maximum change in the distance field between adjacent slices.

3. **Monotonicity:** Given a slice, the distance function of a site is a monotonic function. It has a minimum value in the interior of the slice and is maximum on the boundary of the slice.

Our goal is to cull away sites that do not contribute to the final distance field for a particular slice. Furthermore, the distance field for each site should be computed in the region of the slice where it contributes to the final distance field (see Figure 2). Our algorithm utilizes the above mentioned properties and computes conservative bounds on the Voronoi regions. We use these bounds in two steps: to cull the set of sites for each slice (described in Section 4) and clamp the region of computation for each site in the non-culled set (described in Section 5).

3.4 Site Classification

We introduce a classification of the sites used by our algorithm to cull away sites that do not contribute to the distance field for a slice. Let us assume that the sweep direction is along the $+Z$ direction. For a slice s_k at $z = z_k$, we partition the set of sites $\mathcal{P}$ into three subsets depending on the *Voronoi region bounds of each site* along the Z axis (shown in Figure 2):

Intersecting, $\mathcal{I}_k^+ = \{p_i \mid V(p_i).z_{min} \leq z_k \leq V(p_i).z_{max}\}$. Only the distance functions of these sites contributes to the final distance field of slice s_k.

Approaching, $\mathcal{A}_k^+ = \{p_i \mid V(p_i).z_{min} > z_k\}$. The Voronoi region of an approaching site does not intersect with current slice, but could potentially intersect with a slice s_l, where $z_l > z_k$.

Receding, $\mathcal{R}_k^+ = \{p_i \mid V(p_i).z_{max} < z_k\}$. Due to the connectivity property of Voronoi regions, a receding site can never become *intersecting*, hence it can be discarded for any slice s_l, where $z_l > z_k$.

For efficient computation, the algorithm presented in Section 4 performs two passes along $+Z$ and $-Z$ directions and considers only the sites swept up-to the current

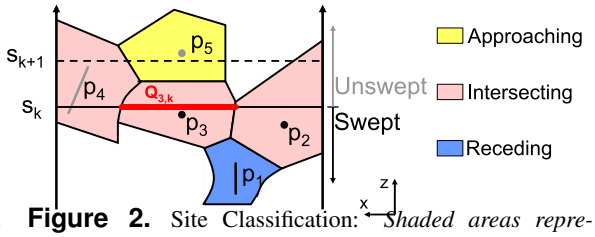

Figure 2. Site Classification: *Shaded areas represent the connected Voronoi regions for a subset of sites* $\{p_1, \ldots, p_5\}$. *Sweep direction is along* $+Z$. *For slice* s_k, *the site sets are: Intersecting* $\mathcal{I}_k^+ = \{p_2, p_3, p_4\}$, *Approaching* $\mathcal{A}_k^+ = \{p_5\}$, *Receding* $\mathcal{R}_k^+ = \{p_1\}$ *and Swept* $\mathcal{S}_k^+ = \{p_1, p_2, p_3\}$, *Unswept* $\mathcal{U}_k^+ = \{p_4, p_5\}$. *Distance functions have to be drawn for set* $\mathcal{I}_k^+$ *only. For site* p_3, *the distance function has be drawn only in the region* $Q_{3,k} = V(p_3) \cap s_k$. *For the next slice* s_{k+1}, p_4 *is moved to* $\mathcal{S}_{k+1}^+$, p_5 *is moved to* $\mathcal{I}_{k+1}^+$ *and* p_3 *is moved to* $\mathcal{R}_{k+1}^+$.

slice. We also partition $\mathcal{P}$ based on the *spatial bounds of each site* along Z axis. Let $p_i.z_{max}$ denote the maximum Z value of a site p_i. Then the set $\mathcal{P}$ is partitioned as (shown in Figure 2):

Swept, $\mathcal{S}_k^+ = \{p_i \mid p_i.z_{max} \leq z_k\}$

Unswept, $\mathcal{U}_k^+ = \{p_i \mid p_i.z_{max} > z_k\}$

The intersecting set $\mathcal{I}_k^+$ can be further partitioned into an *intersecting swept* set $\mathcal{IS}_k^+ = (\mathcal{I}_k^+ \cap \mathcal{S}_k^+)$ and an *intersecting unswept* set $\mathcal{IU}_k^+ = (\mathcal{I}_k^+ \cap \mathcal{U}_k^+)$.

$$\mathcal{I}_k^+ = \mathcal{IS}_k^+ \cup \mathcal{IU}_k^+ \tag{1}$$

The set of sites, $\mathcal{P}$, is also partitioned into subsets along the $-Z$ sweep direction. The intersecting, swept and unswept subsets are represented as $\mathcal{I}_k^-, \mathcal{S}_k^-, \mathcal{U}_k^-$, and are defined as

$$\begin{aligned}
\mathcal{I}_k^- &= \{p_i \mid V(p_i).z_{min} \leq z_k \leq V(p_i).z_{max}\} \\
\mathcal{S}_k^- &= \{p_i \mid p_i.z_{max} > z_k\} \\
\mathcal{U}_k^- &= \{p_i \mid p_i.z_{max} \leq z_k\}
\end{aligned}$$

Consequently,

$$\mathcal{U}_k^+ = \mathcal{S}_k^- \ , \ \mathcal{I}_k^+ = \mathcal{I}_k^- = \mathcal{I}_k$$

and Eq. (1) reduces to

$$\mathcal{I}_k = \mathcal{IS}_k^+ \cup \mathcal{IS}_k^- \tag{2}$$

The key idea for speedup is that for a large number of sites m and any given slice s_k, the size of $\mathcal{I}_k$ is typically much smaller than m. By computing a conservative estimate of $\mathcal{I}_k$ one can cull away a large number of sites and considerably speed up the distance field computation.

4 Site Culling

In this section, we present our culling algorithm that reduces the number of distance functions that are rasterized for each slice. Our goal is to compute the distance field D_k for each slice s_k. Since only the set $\mathcal{I}_k$ contributes to D_k, we have $D_k = D_k(\mathcal{I}_k)$. Using Eq. (2), D_k can be expressed as:

$$D_k(\mathcal{I}_k) = D_k(\mathcal{IS}_k^+ \cup \mathcal{IS}_k^-) = \min\left(D_k(\mathcal{IS}_k^+), D_k(\mathcal{IS}_k^-)\right)$$

Therefore, the problem is reduced to computing two distance fields $D_k(\mathcal{IS}_k^+)$ and $D_k(\mathcal{IS}_k^-)$ for each slice s_k. We present an algorithm to compute $D_k(\mathcal{IS}_k^+)$ for s_k with a sweep direction along $+Z$. The same algorithm is used to compute $D_k(\mathcal{IS}_k^-)$ by using a sweep direction along $-Z$. In the rest of the paper, we will present our algorithm for $+Z$ sweep direction and drop the $+$ sign to simplify the notation.

We utilize the spatial coherence between successive slices and compute the intersecting swept set $\mathcal{IS}_{k+1}$ by performing incremental computations on $\mathcal{IS}_k$ (see Figure 2). We use the following formulation:

$$(\mathcal{IS}_{k+1}) = \left(\mathcal{IS}_k \cup (\mathcal{S}_{k+1} \setminus \mathcal{S}_k)\right) \setminus (\mathcal{R}_{k+1} \setminus \mathcal{R}_k) \tag{3}$$

where $\setminus$ represents the set-difference operation. The exact computation of $\mathcal{IS}_k$ and $\mathcal{IS}_{k+1}$ is equivalent to exact Voronoi computation. Instead, we conservatively compute a set of *potentially intersecting swept* sites $\widehat{\mathcal{IS}}_k$ using Equation (3), where $\widehat{\mathcal{IS}}_k \supseteq \mathcal{IS}_k$.

Given the sets $\widehat{\mathcal{IS}}_k$ and $\mathcal{R}_k$, the algorithm for computing D_{k+1}, $\widehat{\mathcal{IS}}_{k+1}$ and $\mathcal{R}_{k+1}$ proceeds as follows:

1. **Initialize** $\widehat{\mathcal{IS}}_{k+1} = \widehat{\mathcal{IS}}_k, D_{k+1} = \infty$.
2. **Update** $\widehat{\mathcal{IS}}_{k+1} = \widehat{\mathcal{IS}}_{k+1} \cup (\mathcal{S}_{k+1} \setminus \mathcal{S}_k)$. Add the additional sites swept by slice s_{k+1} to $\widehat{\mathcal{IS}}_{k+1}$.
3. **Compute** D_{k+1}. For each site $\hat{p_i} \in \widehat{\mathcal{IS}}_{k+1}$, compute $D_{k+1}(\hat{p_i})$ in order of increasing i. Each $D_k(\hat{p_i})$ is tested for visibility with respect to $D_{k+1}(\mathcal{X}_{i-1})$, which is the distance field of set $\mathcal{X}_{i-1} = \{\hat{p_1}, \hat{p_2}, \ldots, \hat{p_{i-1}}\}$. If $D_{k+1}(\hat{p_i})$ is not visible along the direction orthogonal to s_{k+1}, then it does not contribute to D_{k+1}.
4. **Compute** $(\mathcal{R}_{k+1} \setminus \mathcal{R}_k)$. All sites $\hat{p_i}$ for which $D_{k+1}(\hat{p_i})$ is not visible can be moved from $\widehat{\mathcal{IS}}_{k+1}$ to $\mathcal{R}_{k+1}$.
5. **Update** $\widehat{\mathcal{IS}}_{k+1} = \widehat{\mathcal{IS}}_{k+1} \setminus (\mathcal{R}_{k+1} \setminus \mathcal{R}_k)$

Initially we set $k = 0, \mathcal{R}_k = \emptyset, \widehat{\mathcal{IS}}_k = \{p_i | p_i.z_{max} = 0\}$. We proceed along the Z-axis and compute the distance field for each slice as described above. Each site p_i is bucketed into a list according to $p_i.z_{max}$. This allows the addition

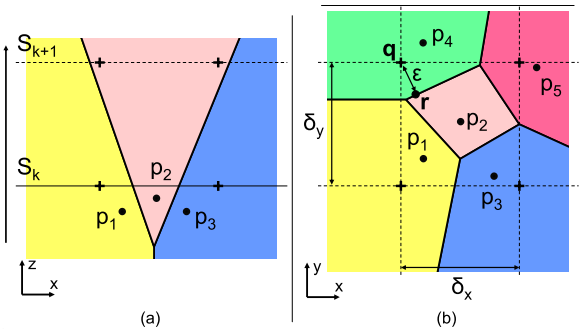

Figure 3. Sampling Error: *(a) The Voronoi region $V(p_2)$ of a swept site p_2 does not lie on any cell (represented by crosses) on slice s_k, but lies on a cell for slice s_{k+1}. (b) The XY intersection of the Voronoi regions with slice s_k. The closest cell $\mathbf{q}$ to $V(p_2)$ is at a distance ϵ.*

of swept sites in Step (2) to be performed in constant time. The distance fields are rasterized approximately in order of increasing distance to the current slice. This results in better culling of the receding sites in Steps (3) and (4) of the algorithm. The complexity of this algorithm for slice s_{k+1} is a linear function of the size $|\ \widehat{\mathcal{IS}}_{k+1}\ |$.

The visibility computations are performed using occlusion queries (e.g. GL_NV_occlusion_query) available on current graphics systems. As the distance meshes are scan converted, these queries check for updates to the depth buffer and return the number of pixels that are visible.

4.1 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 classifies a swept site p_i as receding if its Voronoi region $V(p_i)$ does not cover any grid cells, i.e. the occlusion query returns zero visible pixels for the distance field $D_k(p_i)$ in Step (3). This may introduce errors when $V(p_i)$ intersects slice s_k but its intersection with s_k is not sampled by the rasterization hardware. An incorrect classification of p_i as receding can introduce errors in D_l for a subsequent slice s_l, $l > k$. One such case is shown in Figure 3(a), for $i = 2$.

We modify the algorithm for distance field computation to account for these sampling errors. The approach is based on a lemma that states the condition for a Voronoi region to be sampled.

Lemma 1. *Let $V(p_i)$ be a voronoi region for a slice s_k that is undersampled, and the closest cell $\mathbf{q}$ is at a distance ϵ from $V(p_i)$. If we reduce $dist(\mathbf{q}, p_i)$ by ϵ without changing $dist(\mathbf{q}, \mathcal{P} - p_i)$, we ensure that $\mathbf{q} \in V(p_i)$.*

Proof. Let $\mathbf{q}$ belong to the voronoi region $V(p_j)$ of site p_j, and the point in $V(p_i)$ closest to $\mathbf{q}$ be $\mathbf{r}$ (see Figure 3), with $i = 2, j = 4, dist(\mathbf{q}, \mathbf{r}) = \epsilon$. We shall first prove the result for the case when $V(p_i)$ shares a boundary with $V(p_j)$.

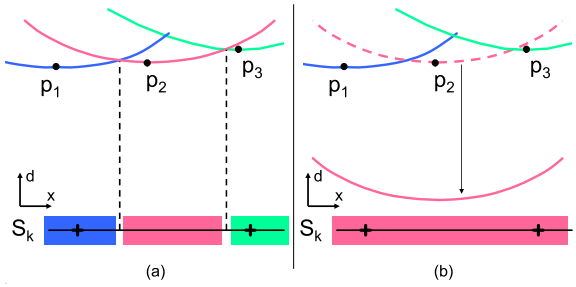

Figure 4. Conservative Sampling: *(a) Distance field $D_k(p_2)$ of site p_2 is occluded at all pixels on s_k. (b) Translating $D_k(p_2)$ by δ_{xy} ensures it is visible at at least one pixel.*

Then using the fact $\mathbf{r} \in V(p_i)$ and $\mathbf{r} \in V(p_j)$, and the triangle inequality, we have

$$\begin{aligned} dist(\mathbf{r}, p_j) &= dist(\mathbf{r}, p_i) \\ dist(\mathbf{q}, p_i) &\leq dist(\mathbf{q}, \mathbf{r}) + dist(\mathbf{r}, p_i) \\ dist(\mathbf{r}, p_j) &\leq dist(\mathbf{r}, \mathbf{q}) + dist(\mathbf{q}, p_j) \\ \Rightarrow dist(\mathbf{q}, p_i) - \epsilon &\leq dist(\mathbf{r}, p_j) < dist(\mathbf{q}, p_j) \end{aligned}$$

Thus, by reducing $dist(\mathbf{q}, p_i)$ by ϵ and keeping $dist(\mathbf{q}, p_j)$ the same, $\mathbf{q}$ will lie in Voronoi region $V(p_i)$. This directly extends to the case when $V(p_i)$ and $V(p_j)$ do not share a boundary, by using a sequence of triangle inequalities across Voronoi boundaries between $V(p_i)$ and $V(p_j)$. □

We apply the result of Lemma 1 in the following manner. In practice, we do not know the point $\mathbf{q}$ or ϵ but use the fact that ϵ is bounded by pixel size, $\epsilon \leq \delta_{xy} = \frac{\sqrt{\delta_x^2 + \delta_y^2}}{2}$. Therefore, we move p_i closer to all the points in slice s_k, by subtracting δ_{xy} from each value of the distance field $D_k(p_i)$. This is equivalent to translating $D_k(p_i)$ along $-Z$ by δ_{xy} and is shown in Figure 4.

Given a slice s_{k+1}, we redraw the translated distance field of each site p_i marked as receding in Step (4) of the algorithm given above (i.e. $p_i \in \mathcal{R}_{k+1} \setminus \mathcal{R}_k$). The redrawn distance field is tested for visibility with respect to D_{k+1}. This redrawing is performed to ensure conservative sampling for site-culling. During this step, updates to the final distance field in the depth buffer are disabled. Moreover, the translated field is clamped to 0 for negative values. For line and triangle sites, the size of the Voronoi region is also limited by the spatial size of the line segment or the triangle. To ensure that the Voronoi region covers at least one of the four neighboring cells, we increase the size of these sites by δ_{xy} in addition to translating the distance field.

5 Distance Function Clamping

In Section 4, we presented an algorithm to cull away the sites that do not contribute to the distance field D_k of slice

s_k. In this section, we present a clamping algorithm to reduce the rasterization cost of the distance function of each potentially intersecting swept site. Given a slice s_k and each site $p_i \in \widehat{\mathcal{IS}}_k$, we compute the distance function $dist(\mathbf{q}, p_i)$ only for the set of points on s_k that lie in the Voronoi region of p_i. In other words, our goal is to evaluate the distance function for the set $Q_{i,k} = \{\mathbf{q} | \mathbf{q} \in V(p_i) \cap s_k\}$. We first present an approach to compute a conservative estimate $\widehat{Q}_{i,k}$ of $Q_{i,k}$ for any arbitrary set of sites. We further improve the performance of the clamping algorithm for manifold surfaces by using the CSC-algorithm [22].

5.1 Conservative Clamping

The connectivity of the Voronoi regions implies that $Q_{i,k}$ is a connected set. We exploit the monotonicity property and compute a superset $\widehat{Q}_{i,k}$. Initially, we assume that we are given the maximum value $\max(D_k(p_i))$ of the distance field $D_k(p_i)$ of site p_i on slice s_k. We compute a set of extreme points on s_k where the value of the distance field $D_k(p_i)$ is equal to the maximum value. By the monotonicity property of distance functions, the set of points whose distance function is less than or equal to $\max(D_k(p_i))$ belong to $\widehat{Q}_{i,k}$. An example is shown in Figure 5.

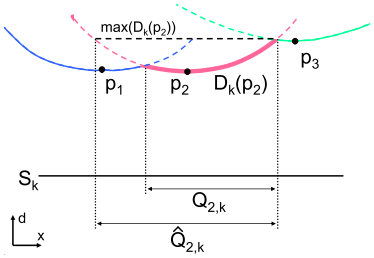

Figure 5. Clamping distance field computation to Voronoi region bounds on a slice. $Q_{2,k} = V(p_i) \cup s_k$. $\widehat{Q}_{2,k} \supseteq Q_{2,k}$ *and is computed from* $\max(D_k(p_2))$.

The problem of distance function clamping reduces to computing $\max(D_k(p_i))$ for each site p_i in $\widehat{\mathcal{IS}}_k$ for a slice s_k. We use the following lemma to compute an upper bound on $\max(D_k(p_i))$.

Lemma 2. *Let* $\max(D_k(\mathcal{S}_k))$ *denote the maximum value of the distance fields* $D_k(\mathcal{S}_k)$ *of set* $\mathcal{S}_k$ *on a slice* s_k *and* $\max(D_{k+1}(\mathcal{S}_{k+1}))$ *be defined similarly. Let the distance between* s_{k+1} *and* s_k *be* $|z_{k+1} - z_k| = \delta_z$. *Then*

$$\max(D_{k+1}(\mathcal{S}_{k+1})) \leq \max(D_k(\mathcal{S}_k)) + \delta_z \qquad (4)$$

Proof. Given two points $\mathbf{q}_k(x, y, z_k) \in s_k$ and $\mathbf{q}_{k+1}(x, y, z_{k+1}) \in s_{k+1}$ that lie in the Voronoi regions of some two sites. Then

$$|dist(\mathbf{q}_{k+1}, \mathcal{P}) - dist(\mathbf{q}_k, \mathcal{P})| \leq \delta_z. \qquad (5)$$

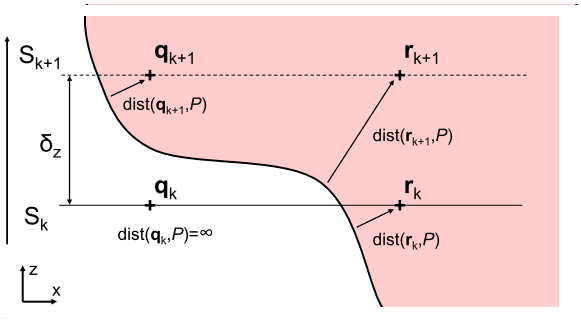

Figure 6. Change in distance field for signed distance computation

This follows directly from the triangle inequality, and the definition of the distance function $dist(\mathbf{q}, \mathcal{P})$. Moreover, $\max(D_k(\mathcal{X})) = \max_{\mathbf{q} \in s_k}(dist(\mathbf{q}, \mathcal{X}))$. This implies that

$$\max(D_{k+1}(\mathcal{X})) \leq \max(D_k(\mathcal{X})) + \delta_z \qquad (6)$$

Moreover, for a slice s_k and any two sets of sites $\mathcal{X}_1$ and $\mathcal{X}_2$, $\mathcal{X}_1 \subseteq \mathcal{X}_2 \Rightarrow D_k(\mathcal{X}_2) \leq D_k(\mathcal{X}_1)$. We know $\mathcal{S}_k \subseteq \mathcal{S}_{k+1}$. This combined with Eq. (6), where $\mathcal{X} = \mathcal{S}_{k+1}$, leads to the result in Eq. (4). □

Given the maximum value $\max(D_k(\mathcal{S}_k))$ of the distance field for slice s_k, we use Eq. (4) to compute the maximum value $\max(D_{k+1}(\mathcal{S}_{k+1}))$ of the distance field for slice s_{k+1}. This also gives a conservative bound on maximum value of the distance function for each site p_i on slice s_{k+1}, $\max(D_{k+1}(\mathcal{S}_{k+1})) \geq \max(D_{k+1}(p_i)) \; \forall \; p_i \in \mathcal{S}_{k+1}$. We use it to compute a conservative bound on the set of points $Q_{i,k+1}$ on slice s_{k+1} and use this bound for clamping.

Note that the maximum distance value, $\max(D_k(\mathcal{S}_k))$, may be infinity, if one is computing the distance field in a narrow band at a distance d_{max} [22], or if one is computing the signed distance field for a closed manifold. For the first case we define $\max(D_k(\mathcal{S}_k))$ to be the maximum *finite* value of the distance field, and set the update rule to be $\max(D_{k+1}(\mathcal{S}_{k+1})) = \min(d_{max}, \max(D_k(\mathcal{S}_k)) + \delta_z)$.

For the second case, if $\mathbf{q}_k$ does not lie in region where the signed distance field is computed, and $\mathbf{q}_{k+1}$ does, then the manifold surface lies between $\mathbf{q}_k$ and $\mathbf{q}_{k+1}$ and $\max(dist(\mathbf{q}_{k+1}, \mathcal{S}_{k+1})) \leq \delta_z$. This is shown in Figure 6.

5.2 Manifold Surfaces

In many cases, the primitives lie on manifold surfaces and we have the connectivity information In these cases, we use the CSC algorithm [22] to further refine $\widehat{Q}_{i,k+1}$ for signed Euclidean distance fields. A site is marked as *CSC-valid* if it lies on the interior of a convex or concave manifold surface patch. Boundary sites, hyperbolic points (which are neither convex or concave) and non-manifold sites are marked as *CSC-invalid*. For each CSC-valid site p_i, a convex polyhedron bounding $V(p_i)$ is computed. This polyhedron is

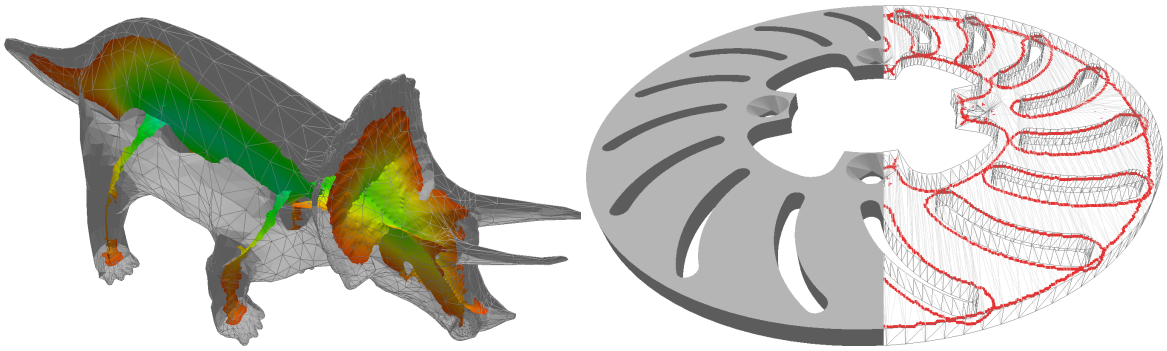

Figure 7. Medial Axis Transform: ***Left****: Triceratops model (5.6k polygons, Grid Size=*$128 \times 56 \times 42$*, Computation Time=*$0.79s$*) The medial surface is color coded by the distance from the boundary.* ***Right****: Brake rotor model (4.7k polygons, Grid Size=*$4 \times 128 \times 128$*, Computation Time=*$0.61s$ *). The medial seam curves are shown in red.*

intersected with s_{k+1} to compute a convex polygon $G_{i,k+1}$. In this case, $G_{i,k+1} \cap \widehat{Q}_{i,k+1}$ results in a tighter approximation of $Q_{i,k+1}$.

5.3 Complete Algorithm

Given $\widehat{\mathcal{IS}}_k$, $\mathcal{R}_k$ and D_k, the algorithm for computing D_{k+1} as presented in Section 4 is refined to perform clamping as follows:

1. **Compute** $\max(D_k)$ by using multiple occlusion queries as described in [13]. Compute $\max(D_{k+1}) = \min(d_{max}, \max(D_k) + \delta)$.

2. **Initialize** $\widehat{\mathcal{IS}}_{k+1} = \widehat{\mathcal{IS}}_k, D_{k+1} = \infty$.

3. **Update** $\widehat{\mathcal{IS}}_{k+1} = \widehat{\mathcal{IS}}_{k+1} \cup (\mathcal{S}_{k+1} \setminus \mathcal{S}_k)$.

41. **Compute** $\widehat{Q}_{i,k+1}$. For each site $p_i \in \widehat{\mathcal{IS}}_{k+1}$, compute $\widehat{Q}_{i,k+1}$ from $\max(D_{k+1})$

42. **Refine** $\widehat{Q}_{i,k+1}$. For each *CSC-valid* site $p_i \in \widehat{\mathcal{IS}}_{k+1}$, compute the convex polygon $G_{i,k+1}$. Refine $\widehat{Q}_{i,k+1} = \widehat{Q}_{i,k+1} \cap G_{i,k+1}$.

43. **Compute** D_{k+1}. For each site $p_i \in \widehat{\mathcal{IS}}_{k+1}$, compute $D_{\widehat{Q}_{i,k+1}}(p_i)$ and test for visibility as before.

44. **Perform Conservative Sampling** Disable distance field updates. For each site $p_i \in \widehat{\mathcal{IS}}_{k+1}$ which is marked as occluded, expand the site and compute $D_{\widehat{Q}_{i,k+1}}(p_i) - \delta_{xy}$. Test for visibility against the computed distance field D_{k+1} as before. Enable distance field updates.

5. **Compute** $(\mathcal{R}_{k+1} \setminus \mathcal{R}_k)$ from the results of the visibility tests of Step 3.4.

6. **Update** $\widehat{\mathcal{IS}}_{k+1} = \widehat{\mathcal{IS}}_{k+1} \setminus (\mathcal{R}_{k+1} \setminus \mathcal{R}_k)$.

Given a 3D grid with $N+1$ slices and a Z range $[z_{min}, z_{max}]$, we make 2 passes. In the first pass, we increment k from 0 to N. Initially, $\mathcal{R}_0^+ = \emptyset, \widehat{\mathcal{IS}}_0^+ = \{p_i | p_i.z_{max} = z_{min}\}$. In the second pass, k is decremented from N down to 0. Initially, $\mathcal{R}_N^- = \emptyset, \widehat{\mathcal{IS}}_N^- = \{p_i | p_i.z_{max} = z_{max}\}$. The final distance field for each slice is the lower envelope of both.

6 Applications

We have applied our distance field algorithm to compute the medial axis transform of polyhedral models and path planning. These applications require global distance field computation along a 3D grid.

Simplified Medial Axis Computation: We compute a simplification of the Blum medial axis, called the θ-*simplified medial axis* (θ-SMA) [9]. The θ-SMA provides a good approximation of the stable subset of the medial axis. The algorithm for computing the θ-SMA of an object X is based on computing the vector field called the *neighbor direction field* of the object X and denoted by $N(X)$. $N(X)$ is the negated gradient of the distance field defined by the boundary of X. Given $N(X)$, a separation criterion is defined using the separation angle θ. The criterion is used to check whether a line segment connecting the centers of adjacent voxels of a grid crosses a sheet of the medial axis. When a pair of points passes the separation criterion, we add the facet between them to the approximation of θ-SMA and compute a polygonal approximation of the medial axis. In some cases a discrete voxel representation of the θ-SMA is desirable. A voxel is added to the medial axis if it lies on one side of a facet on the medial axis, which is determined as above. This selection operation can be efficiently performed on modern programmable graphics hardware using fragment programs. The gradient field is stored on graphics card texture memory. This avoids the costly readbacks of the entire distance field to the CPU.

Interactive Path Planning in Dynamic Environments: We have used our distance field computation algorithm within a constraint-based path planner [11]. The path planning problem is reduced to simulating a constrained dynamic system, and computes an approximation of the generalized Voronoi diagrams (GVD) of the robot and obstacles in the environment. Each robot is subject to virtual forces introduced by geometric and mechanical constraints, such as making the robot follow an estimated path computed using the GVD and linking the rigid objects together to represent an articulated robot. The distance field is used to compute an approximate GVD and a Voronoi graph. The distance field is also used to perform proximity tests between the robot and the obstacles and maintain a minimum clearance. Given a pair of objects, R_1 and R_2, the distance field of R_2 is drawn in a potentially overlapping region. The surface of R_1 is sampled at points inside the overlapping region, and a force is generated at each sample point $\mathbf{q}_i$. The force is in the direction of the gradient of the distance field and proportional to the distance between $\mathbf{q}_i$ and the surface of R_2. As the obstacles in the environment undergo motion, our algorithm recomputes the distance field and uses it for path computation. We have used this path planner for virtual prototyping applications.

7 Implementation and Results

In this section we describe the implementation of our distance field computation algorithm and highlight its performance on complex polygonal models.

We have implemented our algorithm in Microsoft Visual C++ and use OpenGL as the graphics API. The distance function for each primitive is approximated as a polygonal mesh based on techniques presented in [15]. The visibility test is performed using the OpenGL occlusion query extension GL_NV_occlusion_query. We exploit the parallelism of this query by batching together occlusion queries for an entire set of potentially intersecting sites. The CSC algorithm [22] is used for clamping the region of distance field computation for CSC-valid sites. We have integrated the CSC algorithm with the distance field computation algorithm presented in [15]. We clamp the approximate distance mesh of each CSC-valid site with the bounding convex polyhedra. The bounding convex polyhedra are computed at run time. Our implementation involves no precomputation and is directly applicable to deformable models.

We generate the gradient vector field along with the distance field to compute the θ-SMA. The gradient vector is encoded in the color values of each vertex of the distance mesh. The voxel representation of the θ-SMA is computed directly on the graphics processor using OpenGL's ARB_fragment_program extension. The medial axis is rendered directly from the GPU as a volume grid.

7.1 Performance

We have applied our algorithm to 3D polygonal models. These include scanned models and CAD models. Some of them are non-manifold.

Distance Field Computation: All the timings reported in this paper were generated on a Pentium4 2.8GHz PC with 2GB RAM and an NVIDIA GeForce FX 5900 Ultra graphics card, running Windows XP. We have compared the performance of our distance field computation algorithm (DiFi) with the algorithm presented by Hoff et al. [15](called HAVOC), a software implementation of CSC algorithm [22], and an implementation that combines HAVOC with CSC. The timings are presented in Table 1. In our benchmarks, DiFi obtains more than two orders of magnitude over a software implementation of the CSC algorithm and more than one order of magnitude performance improvement over an implementation combining HAVOC and CSC for manifold objects. For non-manifold models, we obtain $4 - 20$ times speedup over HAVOC.

Medial Axis Computation: We have applied the distance field to compute the simplified medial axis of polyhedral models. The simplified medial axis for two models is shown in Figure 7. Our algorithm takes less than a second to compute the medial axis of polyhedral models consisting of thousands of polygons.

Path Planning: We have applied the path planning algorithm to an assembly environment (shown in Figure 9). The environment consists of an articulated robot arm with 6 degrees of freedom placed in the middle of a complicated piping structure. The robot arm reaches for a part moving on a conveyor belt and avoids collision with obstacles. Various links on the robot arm come in close proximity with the piping structures. We are able to dynamically compute the path at interactive rates using our fast distance field computation algorithm.

8 Analysis and Limitation

In this section we analyze the performance of our algorithm. We highlight its computational complexity and the errors in distance computation. We also compare its performance with earlier algorithms.

We approximate each non-linear distance function with a polygonal distance mesh. This introduces a tessellation error [15]. The tessellation error is bounded by a user defined $\epsilon > 0$. We set $\epsilon = \frac{\sqrt{\delta_x^2+\delta_y^2+\delta_z^2}}{2}$ so that the error in the distance field is no more than half the diagonal length of a grid cell. As a result, the main source of discretization error is grid resolution. Current graphics processors support 24-bit depth buffers, so the error in depth computation and comparisons is relatively small.

Given a model with m sites and a 3D grid of size $n = N \times N \times N$, the cost of computing the distance field is proportional to the number of processed cells over

Model	Polys	Resolution	CSC	HAVOC	HAVOC+CSC	DiFi
Rotor	4736	4x128x128	59.22	6.33	3.98	0.61
Rotor	4736	8x254x254	424.89	18.73	12.12	1.16
Triceratops	5660	128x56x42	127.81	2.11	1.10	0.79
Triceratops	5660	254x111x84	990.48	6.33	3.65	1.92
Hugo	17000	73x45x128	X	30.55	19.24	4.22
Hugo	17000	145x90x254	X	108.84	75.85	8.63
Head	21764	78x105x128	201.12	37.89	16.76	4.98
Shell	22598	254x252x252	X	162.97	95.31	7.79
Cassini	93234	186x254x188	X	356.03	298.55	29.86
Dragon	108926	57x90x128	X	171.13	95.69	24.76

Table 1. Distance Field Computation: *Times (in seconds) to compute the global distance fields using approaches by Mauch [22] (CSC), Hoff et al. [15](HAVOC), an implementation combining CSC with HAVOC on graphics hardware (HAVOC+CSC), and our algorithm (DiFi). For the entries marked X, CSC algorithm fails as the model contains CSC-invalid sites.*

Figure 8. Cassini Model: *A volume rendering of the distance field of the Cassini with 93K polygons. The distance to the surface is color coded, increasing from red to green to blue.*

which the distance function is evaluated. The optimal cost for computing the 3D distance field is $O(N^3) = O(n)$. For a slice s_k, the *optimal* number of processed cells is $\sum_{i=1}^{|\mathcal{I}_k|} |Q_{i,k}| = N^2$. The *actual* number of processed cells is $\sum_{i=1}^{|\widehat{\mathcal{I}}_k|} |\widehat{Q}_{i,k}|$. We define the following average number of cells covered by one site:

optimal = $\langle |Q_{i,k}| \rangle = \frac{\sum_{i=1}^{|\mathcal{I}_k|} |Q_{i,k}|}{|\mathcal{I}_k|}$, actual = $\langle |\widehat{Q}_{i,k}| \rangle = \frac{\sum_{i=1}^{|\widehat{\mathcal{I}}_k|} |\widehat{Q}_{i,k}|}{|\widehat{\mathcal{I}}_k|}$

The per-slice efficiency of our algorithm can be measured by two ratios: the *clamping efficiency*, $e_{1k} = \frac{\langle |Q_{i,k}| \rangle}{\langle |\widehat{Q}_{i,k}| \rangle}$ and *culling efficiency*, $e_{2k} = \frac{|\mathcal{I}_k|}{|\widehat{\mathcal{I}}_k|}$. The *average efficiency* per slice can be defined as $\langle e \rangle = \frac{1}{N} \sum_{k=1}^{N} e_{1k} \times e_{2k}$. The total cost of the algorithm is $O(n/\langle e \rangle)$, and is bounded between $O(n)$ and $O(mn)$. For CSC-invalid sites, the clamping efficiency e_{1k} approaches 1 as the sites are uniformly distributed on the 3D grid. For CSC-valid sites, the complexity is similar to that of the CSC algorithm, i.e. $O(m + rn)$. However, our algorithm obtains tighter bounds on the parameter r, $r = 1/\langle e \rangle$. In practice, $e_{1k} \approx 1$, thus $r = \frac{1}{\langle e_{2k} \rangle}$.

We sample and render some of the distance functions twice in order to overcome the sampling errors introduced by occlusion queries. Moreover, to detect under-sampling errors, the distance field is offset by a larger value of cell size δ_{xy}, leading to a more conservative estimate of the potentially intersecting set. This extra computation is performed only when a site is marked as receding due to under-sampling, and becomes smaller at higher grid resolutions.

8.1 Comparison

We now provide a comparison of our algorithm, DiFi, with two previous algorithms for computing 3D distance fields using graphics hardware: HAVOC, and the algorithm by Sigg et al. [28].

HAVOC: DiFi is restricted in the distance functions handled compared to HAVOC, since it assumes that the Voronoi regions are connected. It can however handle a wide range of distance functions, including all the L_p metrics, while giving more than one order of magnitude speedup. Like HAVOC, it is applicable to generic models without connectivity information, and has the same error bounds.

Sigg et al.: The algorithm by Sigg et al. [28] is applicable only to manifold surfaces and has the same asymptotic complexity as the CSC algorithm [22], i.e. $O(m+rn)$. It is particularly efficient for computing the distance field in narrow bands around manifold surfaces. For small band sizes, the parameter r is close to unity. However, for computing the global distance field of complex environments with multiple manifold surfaces and high depth-complexity, r can be $O(m)$. Distances computed by this algorithm are exact up to GPU floating texture precision. The culling and clamping techniques presented in DiFi are complementary to those presented in [22, 28]. In fact, the approach presented in [28] can be used for distance field computation of manifold sites inside DiFi instead of HAVOC. This would give significant speedups over [28] for computing global distance fields in complex environments. In particular, we have demonstrated that DiFi provides significant speedups over HAVOC and CSC combined.

8.2 Limitations

Our algorithm has certain limitations. Our distance field computation is performed on a uniform grid and its accuracy is governed by grid resolution. Current graphics processors provide up to $4K \times 4K$ pixel resolution and this imposes an upper bound on the grid resolution. Even though

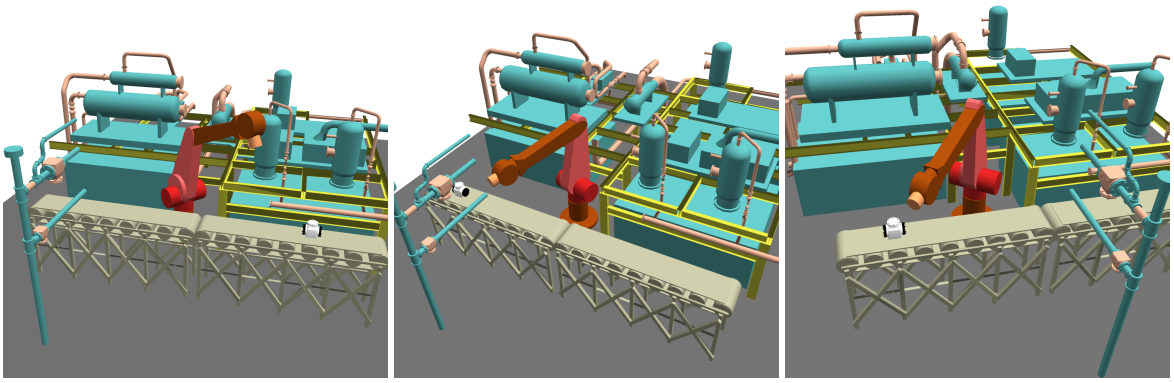

Figure 9. Planning in an assembly environment: *Constraint based planning in a dynamic environment consisting of* $26.9k$ *polygons using distance fields. The robot arm tracks a moving part on a conveyor belt, while avoiding contact with other obstacles in the environment. Our algorithm computes the distance field at interactive rates and uses the distance field to compute a collision free path.*

we use culling and clamping algorithms, the performance of the algorithm is still bounded by the rasterization cost or the fill-rate. Moreover, some applications require reading back the distance field to the CPU and readbacks can be slow on the PCI bus. Our algorithm is best suited for computation on uniform grids and may not result in any speedups for adaptive distance fields [10].

9 Conclusions and Future Work

We have presented an algorithm for fast computation of 3D discretized distance fields using graphics hardware. Our algorithm uses a combination of culling and clamping techniques to reduce the number and size of distance functions that are rendered for each slice. We use occlusion queries to speed up the computation and have presented a conservative scheme to overcome sampling errors. We have used our algorithm to compute distance fields of complex 3D models. The distance fields are used for computing the simplified medial axis and for path planning in a dynamic environment. We achieve one to two orders of magnitude improvement over prior algorithms and implementations.

There are many avenues for future work. We would like to further improve the performance by utilizing temporal coherence between successive frames for dynamic or deformable models. We would also like to use our algorithm for other applications, including dynamic simulation, morphing and proximity computations.

Acknowledgments

This research is supported in part by ARO Contract DAAD 19-02-1-0390, NSF Awards ACI-9876914, ACR-0118743, ONR Contract N00014-01-1-0067, and Intel Corporation. The models are courtesy of the Stanford University Computer Graphics Laboratory, the Georgia Tech Large Models Archive, Alpha_1 Project at University of Utah, Laurence Boisseux at INRIA and Stephen Wall, Gary Cough and Don Jacob at NASA-JPL. We thank Mark Foskey for θ-SMA code, Luv Kohli for help with cPlan, Mark Harris and Greg Coombe for help with GPU programming and the UNC GAMMA group for many useful discussions and support. We are also grateful to the reviewers for their feedback.

References

[1] J. Bloomenthal, editor. *Introduction to Implicit Surfaces*, volume 391. Morgan-Kaufmann, 1997.

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

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

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

[5] D. Cohen-Or, D. Levin, and A. Solomovici. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 1997.

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

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

[8] M. Denny. Solving geometric optimization problems using graphics hardware. *Computer Graphics Forum*, 22(3), 2003.

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

[10] S. Frisken, R. Perry, A. Rockwood, and R. Jones. Adaptively sampled distance fields: A general representation of shapes for computer graphics. In *Proc. of ACM SIGGRAPH*, pages 249–254, 2000.

[11] M. Garber and M. Lin. Constraint-based motion planning using voronoi diagrams. *Proc. Fifth International Workshop on Algorithmic Foundations of Robotics*, 2002.

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

[13] N. Govindaraju, B. Lloyd, W. Wang, M. Lin, and D. Manocha. Fast computation of database operations using graphics processors. *Proc. of ACM SIGMOD*, 2004.

[14] T. He, L. Hong, A. Varshney, and S. Wang. Controlled topology simplification. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):171–184, 1996.

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

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

[17] K. Hoff, A. Zaferakis, M. Lin, and D. Manocha. Fast 3d geometric proximity queries between rigid and deformable models using graphics hardware acceleration. Technical Report TR02-004, Department of Computer Science, University of North Carolina, 2002.

[18] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He. Virtual voyage: Interactive navigation in the human colon. *Proc. of ACM SIGGRAPH*, pages 27–34, 1997.

[19] Y. Kim, G. Varadhan, M. Lin, and D. Manocha. Efficient swept volume approximation of complex polyhedral models. *Proc. of ACM Symposium on Solid Modeling and Applications*, pages 11–22, 2003.

[20] L. Kobbelt, M. Botsch, U. Schwanecke, and H. P. Seidel. Feature-sensitive surface extraction from volume data. In *Proc. of ACM SIGGRAPH*, pages 57–66, 2001.

[21] M. Lin. *Efficient Collision Detection for Animation and Robotics*. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, December 1993.

[22] S. Mauch. *Efficient Algorithms for Solving Static Hamilton-Jacobi Equations*. PhD thesis, Californa Institute of Technology, 4 2003.

[23] C. Maurer, R. Qi, and V. Raghavan. 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):265–270, February 2003.

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

[25] R. Perry and S. Frisken. Kizamu: A system for sculpting digital characters. In *Proc. of ACM SIGGRAPH*, pages 47–56, 2001.

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

[27] R. Shekhar, E. Fayyad, R. Yagel, and F. Cornhill. Octree-based decimation of marching cubes surfaces. *Proc. of IEEE Visualization*, pages 335–342, 1996.

[28] C. Sigg, R. Peikert, and M. Gross. Signed distance transform using graphics hardware. In *Proceedings of IEEE Visualization*, 2003.

[29] G. Varadhan, S. Krishnan, Y. Kim, and D. Manocha. Feature-sensitive subdivision and isosurface reconstruction. *Proc. of IEEE Visualization*, 2003.

[30] G. Varadhan, S. Krishnan, T. V. N. Sriram, and D. Manocha. Topology preserving surface extraction using adaptive subdivision. Technical report, Department of Computer Science, University of North Carolina, 2004.

[31] J. Vleugels and M. Overmars. Approximating Voronoi diagrams of convex sites in any dimension. *International Journal of Computational Geometry and Applications*, 8:201–222, 1997.