Surface Distance Maps

Avneesh Sud Naga Govindaraju Russell Gayle Erik Andersen Dinesh Manocha

Dept of Computer Science, University of North Carolina at Chapel Hill
http://gamma.cs.unc.edu/gvd/sdm

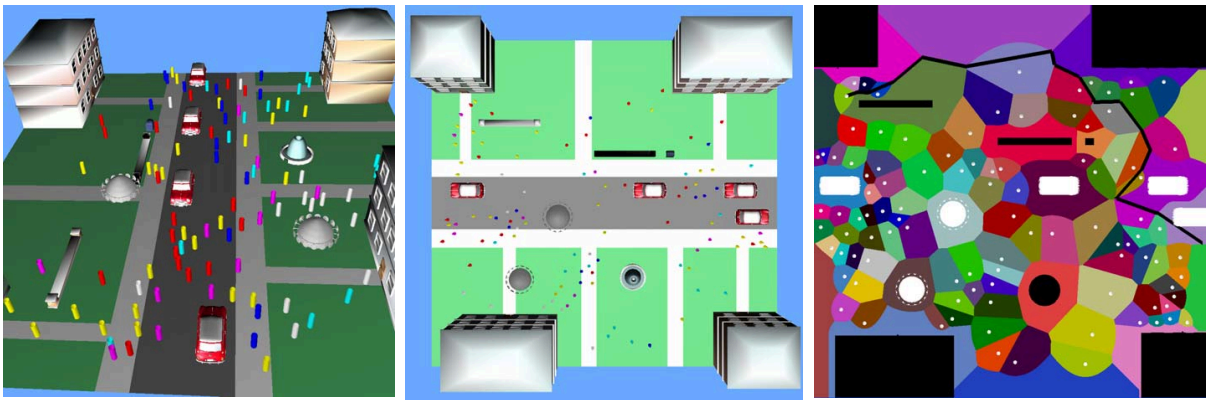

Figure 1: Interactive motion planning of hundreds of agents for crowd simulation: We use our novel surface distance map computation algorithm for interactive path computation and collision detection in a dynamic environment: (left and center) Two views of the environment with dynamic 3D obstacles, including cars and flying drones. Each human agent is represented as a cylinder and colored based on its goal. (right) The nearest neighbor map of the obstacles and agents is computed using surface distance maps. Each colored region is closer to one of the 3D obstacles than to any other. The path for one of the agents is shown using solid black lines. Our algorithm can perform the simulation, including distance computations and path planning, for 100 agents at 10fps on a high-end PC.

ABSTRACT

We present a new parameterized representation called surface distance maps for distance computations on piecewise 2-manifold primitives. Given a set of orientable 2-manifold primitives, the *surface distance map* represents the (non-zero) signed distance-to-closest-primitive mapping at each point on a 2-manifold. The distance mapping is computed from each primitive to the set of remaining primitives. We present an interactive algorithm for computing the surface distance map of triangulated meshes using graphics hardware. We precompute a surface parameterization and use the it to define an affine transformation for each mesh primitive. Our algorithm efficiently computes the distance field by applying this affine transformation to the distance functions of the primitives and evaluating these functions using texture mapping hardware. In practice, our algorithm can compute very high resolution surface distance maps at interactive rates and provides tight error bounds on their accuracy. We use surface distance maps for path planning and proximity query computation among complex models in dynamic environments. Our approach can perform planning and proximity queries in a dynamic environment with hundreds of objects at interactive rates and offer significant speedups over prior algorithms.

CR Categories: I.3.5 [Computing Methodologies]: Computational Geometry and Object Modeling—Geometric algorithms; I.3.7 [Computing Methodologies]: Three-Dimensional Graphics and Realism—Animation, virtual reality

Keywords: distance fields, parameterization, deformable models, collision detection, motion planning

1 INTRODUCTION

Distance fields are scalar fields that represent the closest distances. Given a set of primitives $\mathcal{S}$ in $\mathbb{R}^3$, the distance field at a point equals the distance to the closest point on $\mathcal{S}$. Distance fields are widely studied in computer graphics, computational geometry, computer vision and robotics. They are used for several applications including shape representation and sculpting [15], skeleton computation [2], collision and proximity computations [32], remeshing [21], motion planning [18], implicit surface representation [16], non-photorealistic rendering [20], etc.

Most of the prior techniques compute the distance field along a volumetric grid or a voxelized representation of space. At a broad level, these algorithms can be classified into object space methods that perform direct scan conversion into 3D voxels or image space methods that compute the closest primitive at each grid point. The latter methods can be accelerated by rasterizing the distance functions using the graphics hardware [19, 31, 30, 12]. These algorithms compute the distance field along each slice of a 3D grid and the computation can be accelerated by using spatial bounds on the Voronoi regions of the primitives [31, 27]. However, many applications require distance information on the surface boundary of a mesh. The existing volumetric techniques are inefficient for such computations due to high storage overhead and computation cost. Moreover, their accuracy can be low as most of the grid vertices do not exactly lie on the mesh surface.

In this paper, we consider the problem of computing the distance map on triangulated meshes in $\mathbb{R}^3$. Given a set of 2-manifold primitives $\mathcal{S}$, the *surface distance map* at each point on a primitive o represents the (non-zero) distance-to-closest-primitive from the set

$\mathcal{S} \setminus \{o\}$. The distance function varies continuously along the surface and the distance map yields the direction vector to the closest primitive. As the surface distance map encodes closest primitive mapping, it also provides the Voronoi diagram at each point on the 2-manifold primitives, or the *nearest neighbor map* on the boundary. If the primitives $\mathcal{S}$ are orientable, we can also associate a sign with the distance map.

Main Results: We present a new algorithm to compute surface distance maps of triangulated models. Our algorithm uses a simple texture representation and precomputes a piecewise planar parametrization of each mesh. The parameterization defines an affine transformation for each primitive of a mesh to the plane. The 2D texture map is used as a discrete sampling of the mesh for distance map computation. As a result, the resolution of distance map is limited by the size of the texture memory. We apply the affine transformations to compute the distance functions of 3D primitives using the texture mapping hardware. This formulation is also used to compute the *nearest-neighbor map* in terms of first order and second order Voronoi diagrams of the primitives. Finally, we present tight error bounds on the discretization error in surface distance map and nearest neighbor map computation.

We highlight two interactive applications of surface distance maps and nearest-neighbor maps computed on the surfaces.

1. **Motion planning in dynamic environments:** We use the nearest-neighbor maps to compute a collision free path for robots or multiple agents moving in a dynamic environments. We update the position of all the robots and compute the surface distance map. We use our planner to compute collision free path for human agents in a crowd simulation.

2. **Proximity queries between deformable models:** The surface distance map is used for collision and proximity queries between multiple deformable 3D models. We use the error bounds to perform conservative computation and the resulting algorithm has object-space precision.

We have implemented our algorithm to compute the surface distance map on a 3GHz Pentium D PC with an NVIDIA GeForce 7900 GTX GPU. We highlight its performance on complex benchmarks composed of thousands of triangles. In practice, our algorithm is able to compute 512×512 distance fields on triangulated meshes in a few hundred milli-seconds. The distance values are computed on a floating point buffer using 32-bit floating point precision. We apply our surface distance map computation algorithm to perform proximity queries among deformable models consisting of 6K primitives in 200ms. We use the nearest-neighbor map computation for interactive motion planning of 100 agents in a complex environment at 10 fps.

As compared to prior distance field based approaches, our algorithm offers the following advantages:

- **Generality:** Our algorithm is applicable to all triangulated models. The only requirement is the computation of the piecewise affine parameterization of the mesh.

- **Accuracy:** We can compute very high resolution distance maps, e.g. $1K \times 1K$ at 32-bit floating point precision. On the other hand, previous interactive techniques based on volumetric approaches are typically restricted to lower resolution (64^3 or 128^3) distance fields.

- **Performance:** Our algorithm can compute surface distance fields of deformable models with thousands of polygons at interactive rates. We observe $5-10$ times speedup in the performance of resulting proximity queries and motion planning algorithms over prior approaches.

Organization: The rest of the paper is organized as follows. We briefly survey prior work on distance field computation and surface mapping in Section 2. Section 3 describes our algorithm to compute distance maps for two-manifolds and we present a number of techniques to improve its performance in Section 4. We analyze our algorithm in Section 5 and highlight its performance on different applications in Section 6.

2 Related Work

In this section, we give a brief overview of related work on distance fields and surface mappings.

2.1 Distance Fields

Algorithms to compute distance fields are widely studied. At a broad level, these algorithms can be broadly classified based on the model representations such as images, volumes or polygonal representations. Good surveys of these algorithms are given in [8, 1].

The algorithms for image-based data sets perform exact or approximate computations in a local neighborhood of the voxels [9, 29, 4, 25, 17]. Exact algorithms for handling 2-D and k-D images have been proposed to compute the distance transforms in voxel data in $O(M)$ time, where M is the number of voxels [4, 25].

Many object-space methods based on adaptive subdivision are known for computing approximate Voronoi diagrams of polygonal models [35, 34, 11, 28]. These algorithms can be used to compute distance fields and are limited to static models. The computation of a discrete Voronoi diagram on a uniform grid can be performed efficiently using graphics rasterization hardware [36, 19, 10, 31, 12]. These algorithms compute 3D distance fields along a uniform grid, and can compute them interactively for low resolution grids on current GPUs.

A class of exact distance computation and collision detection algorithms based on external Voronoi diagrams are described in [23]. A scan-conversion method to compute the 3-D Euclidean distance field in a narrow band around manifold triangle meshes (CSC algorithm) is presented by Mauch [24]. 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. Sigg et al. [30] describe an efficient GPU based implementation of the CSC algorithm. Peikert and Sigg [27] present algorithms to compute optimized bounding polyhedra of the Voronoi cell for GPU-based distance computation algorithms. Lefohn et al. [22] describe an algorithm for interactive deformation and visualization of level set surfaces using graphics hardware.

2.2 Surface Mapping

Surface distance maps can be regarded as a *mapping* computed on the surface. In some ways, this problem is related to other surface mapping problems such as texture mapping [5], which is used to define the color on the surface; displacement mapping [7], which consists of perturbations of the surface positions; bump mapping [3], which give perturbations to the surface normals; and normal maps [14], which contains the actual normals instead of the perturbations. All these mapping are supported by current graphics hardware.

3 Surface Distance Maps

In this section, we present our notation and definitions used in the paper. We introduce surface distance maps and list some of their

properties.

3.1 Notation and Definitions

Let $\mathcal{S} = \{o_1, \ldots, o_n\}$ denote the set of n piecewise linear 2-manifold objects or meshes in 3D. Furthermore, each object o_i is decomposed into vertices, open edges and open faces, also known as *sites*. A site is denoted as p_i. Let $\mathcal{T}_i \subset \mathbb{R}^2$ represent the 2D parametric domain for object o_i. The mapping of a primitive to the 2D domain $\mathcal{T}$ is represented with an overline. Vectors and matrices are represented using boldface. For example, a point $\mathbf{q}$ and triangle t in 3D map to $\overline{\mathbf{q}}$ and $\overline{t}$, respectively, on $\mathcal{T}$.

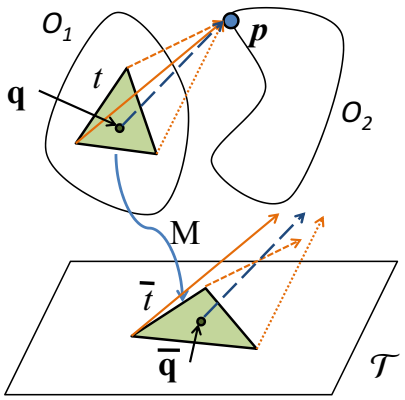

Figure 2: Affine map and surface distance map computation: *The affine map* $\mathbf{M}$ *maps the triangle* t *on object* o_1 *to the triangle* $\overline{t}$ *in domain* $\mathcal{T}$*. The distance vectors from* t *to a point site* p *on object* o_2 *are computed at the vertices of* t*. Then the distance vector of a point* $\overline{\mathbf{q}}$ *on the triangle* $\overline{t}$ *is a convex combination of the distance vectors at the vertices.*

The Euclidean distance function of a site p_i at a point $\mathbf{q} \in \mathbb{R}^3$ is denoted $d(\mathbf{q}, p_i)$. The closest vector from $\mathbf{q}$ to p_i is known as the distance vector, denoted $\vec{\mathbf{d}}(\mathbf{q}, p_i)$. The distance of a point $\mathbf{q}$ to an object o_i is the minimum distance $d(\mathbf{q}, p_j)$ for all sites p_j in o_i.

The *surface distance map*, $D() : \mathcal{S} \rightarrow \mathbb{R}$, at a point $\mathbf{q}$ on object o_i is defined as the distance to the closest object (excluding itself). The surface distance map is closely related to the nearest neighbor map on the surface, denoted $N() : \mathcal{S} \rightarrow \mathcal{S}$, and is defined as the closest object (excluding o_i) at the point $\mathbf{q}$. Formally,

$$D(\mathbf{q}|\mathcal{S}) = \min_{j \neq i}(d(\mathbf{q}, o_j)), \mathbf{q} \in o_i, o_j \in \mathcal{S}$$
$$N(\mathbf{q}|\mathcal{S}) = \{o_j \mid d(\mathbf{q}, o_j) = D(\mathbf{q})\}$$

For ease of notation, when the surface distance map and the nearest neighbor maps are computed with respect to all the objects, we do not explicitly denote $\mathcal{S}$, i.e. $D(\mathbf{q}) = D(\mathbf{q}|\mathcal{S})$ and $N(\mathbf{q}) = N(\mathbf{q}|\mathcal{S})$. For a set of points, the surface distance map provides the distance field to closest objects (excluding itself).

The nearest neighbor map on the surface is closely related to the Euclidean Voronoi diagram in 3D. Assuming each object o_i to be a Voronoi site, let $\mathrm{VD}^k(\mathcal{S})$ denote the k-th order Euclidean Voronoi diagram of the set of objects, and $\mathrm{Gov}^k(\mathbf{q}, \mathcal{S})$ denote the k-th order governor set of a point $\mathbf{q}$. The k-th order governor set of a point $\mathbf{q}$ is the set of k closest sites at $\mathbf{q}$. (see [26] for standard definitions). Then, the following result relates the nearest neighbor map on the surface and the 3D Voronoi diagram.

Lemma 1. *Let* $\mathbf{q} \in o_i$*. Then (a)* $\mathrm{Gov}^1(\mathbf{q}, \mathcal{S} \setminus \{o_i\}) = N(\mathbf{q})$*, and (b)* $\mathrm{Gov}^2(\mathbf{q}, \mathcal{S}) = \{o_i, N(\mathbf{q})\}$.

Lemma 1 implies that the nearest neighbor map on the surface of o_i is given by the intersection of o_i with the 1st order Voronoi diagram $\mathrm{VD}^1(\mathcal{S} \setminus \{o_i\})$, or equivalently with the 2nd order Voronoi diagram $\mathrm{VD}^2(\mathcal{S})$. Later, we use these properties in order to use surface distance fields for interactive motion planning and proximity computations.

3.2 Surface Parameterization

We define an affine mapping $\mathbf{M_{i,1}} : t_i \rightarrow \mathcal{T}_1$ to transform the sampled points on the triangles t_i on o_1 into the 2D domain $\mathcal{T}_1$. For ease of notation, when the object id j is implicit ($j = 1$ in this case), we shall drop the object id subscript from $\mathbf{M_{i,j}}$ and denote the affine map as $\mathbf{M_i}$. Given a 3D mesh o with triangles $t_k, k = 1, \ldots, n$, our algorithm transforms each triangle t_k into a triangle $\overline{t_k}$ by applying an affine mapping $\mathbf{M_k}$ (see Fig. 2). The matrix $\mathbf{M_k}$ is constructed to satisfy the following constraints:

- There is a one-to-one mapping from a point $\mathbf{q} \in t_k$ to the point $\mathbf{M_k q} \in \overline{t_k}$.
- No two transformed triangles $\overline{t_k} = \mathbf{M_k} t_k$ and $\overline{t_l} = \mathbf{M_l} t_l$ share a common interior point in the 2D domain $\mathcal{T}$.

These constraints are satisfied using piece-wise planar parameterizations of the surface in 3D space and the mapped triangles can be represented in a 2D texture atlas. Since $\mathbf{M}$ is affine, it can be written as a composition of a scale, shear, translation and rotation matrices.

3.3 Linear Interpolation

In this section, we show that the surface distance field at a point in the triangle can be represented as linear interpolant of distance vectors of the vertices of that triangle. We shall present this property for point, *infinite* lines and planes, and then extend it to finite edges and triangles. Given a triangle $t_k \in o_i$ with vertices $\mathbf{x}_a, a = 1, 2, 3$, for a given point, infinite line or plane site p_l on object o_j ($j \neq i$), we use the linear interpolation property of distance vectors [31] to express the distance vector at any point on triangle t_k to site p_l as a convex combination of the distance vectors at the vertices $\mathbf{x}_a$:

$$\vec{\mathbf{d}}(\mathbf{q}, p_l) = \sum_{a=1}^{3} \beta_a \vec{\mathbf{d}}(\mathbf{x}_a, p_l), \mathbf{q} \in t_k, \sum_{a=1}^{3} \beta_a = 1, \beta_a \geq 0 \quad (1)$$

Given that $\overline{t_k} = \mathbf{M_k} t_k$, it implies $D(\overline{t_k}) = D(\mathbf{M_k} t_k)$. Since the convex combination is invariant under affine transform $\mathbf{M_k}$, from equation (1) we have,

$$\Rightarrow \vec{\mathbf{d}}(\overline{\mathbf{q}}, p_l) = \vec{\mathbf{d}}(\mathbf{q}, p_l) = \sum_{a=1}^{3} \beta_a \vec{\mathbf{d}}(\mathbf{x}_a, p_l), \overline{\mathbf{q}} = \mathbf{M_k q} \quad (2)$$

Thus the distance vector at a point $\mathbf{q} \in t_k$ (in object o_i) is computed by performing linear interpolation of distance vectors on the 2D domain $\mathcal{T}$, as shown in equation (2). Thus the surface distance map at point $\mathbf{q}$ is computed as the minimum of the length of distance vectors,

$$D(\mathbf{q}) = D(\mathbf{M_k q}) = \min(d(\overline{\mathbf{q}}, p_l)), \forall p_l \in o_j, j \neq i$$

We now extend the linear interpolation property to finite sites (edge, triangle). For a finite site, the linear interpolation is valid inside a convex region defined by the boundary of the site [31].

4 Interactive Distance Map Computation

In this section, we present our algorithm to efficiently compute surface distance maps. We make use of the bilinear formulation of distance computation and evaluate it efficiently using GPUs. We also describe many techniques to accelerate the computation.

4.1 GPU Based Computation

We compute a discrete approximation of surface distance maps at interactive rates using bilinear vertex attribute interpolators in graphis hardware (for eg. texture coordinate interpolation). Discrete surface distance maps compute the distance-to-closest-object mapping at a finite set of point samples on each 2-manifold mesh.

We first compute the affine mappings, $\mathbf{M_k}$ for each triangle t_k in the 3D mesh. We sample the domain $\mathcal{T}$ uniformly using a 2D texture. This defines a sampling on each triangle t_k in 3D space by sampling the projected triangle $\overline{t_k}$ in the 2D domain $\mathcal{T}$. Instead of computing distances along a volumetric grid, our algorithm computes the distance map on each triangle t_k by computing the distance vectors at the vertices of t_k and computing the distance vectors at the point samples on $\overline{t_k}$ using equation (2).

In order to accelerate distance computations, prior algorithms construct a convex polytope G_i which bounds the Voronoi region of a site p_i and reduces the fill requirements [31, 27]. We use similar techniques to accelerate the computation of surface distance maps. For each site p_i, we compute a convex bounding polytope G_i. To compute the surface distance maps on triangle t_k, we intersect G_i with the triangle t_k in the 3D mesh. The distance vector computations are performed in the texture domain $\mathcal{T}$ only for the points that lie inside the transformed domain $\mathbf{M_k}(t_k \cap G_i)$.

Surface distance maps can be computed on the rasterization hardware by using transformations, clipping and interpolation capabilities of the GPUs. The main stages of the pipeline are as follows:

- **Bound Computation and Intersection:** We compute the bounding polytope G_i of site p_i, and intersect it with the plane π_k containing triangle t_k on the CPU. This gives a convex polygon $g_i = G_i \cap \pi_k$.
- **Distance Vector Computation and Transform:** We compute the distance vectors at each vertex of g_i and project the vertices of g_i to the 2D domain $\mathcal{T}$ using the affine map $\mathbf{M_k}$. This per vertex computation is efficiently performed in parallel using the *vertex processor* on the GPU.
- **Clipping:** We restrict the computation of distance vectors to the domain given by $t_k \cap g_i$. In the 2D domain $\mathcal{T}$, this is equivalent to clipping the projection $\overline{g_i}$ against $\overline{t_k}$. We use the **stencil** functionality of GPUs to perform this clipping.
- **Bilinear Interpolation:** The convex combination of distance vectors is equivalent to linear interpolation of texture coordinates assigned to the vertices of the polygon g_i, and is performed by the *texture unit* of the GPU.
- **Distance Computation:** The distance value at a texel in the texture atlas is the norm of the distance vector and computed using the *fragment processor* of the GPU.
- **Distance Comparison:** The distance value is returned as depth and compared with the current minimum distance value using the depth test functionality in the *raster processor* of GPUs. The minimum distance value is stored in the depth buffer.

The overall algorithm to compute the surface distance map of object o_1 using sites in a set of objects $\mathcal{S}$ is given in Algorithm 1. The algorithm requires computing of intersections between bounding polytopes of sites and the triangles in the 3D mesh, and clipping of polygons in 2D. We present details of stencil-based clipping and hierarchical culling techniques that are used to accelerate the performance of the algorithm.

Clipping: Surface distance maps require an efficient clipping algorithm for each triangle-site pair. Given a site p_i and a triangle t_k, we restrict the computation on the 2D domain to the interior of $\overline{t_k}$ using stencil tests. As a result, each triangle-site pair requires a valid stencil to be set in the region corresponding to $\overline{t_k}$. We first set the stencil value of the triangle to 1 by rendering $\overline{t_k}$ on $\mathcal{T}$. We then compute the distance vectors by rendering g_i onto the portions of the surface distance map where the stencil value is set to 1. We then reset the stencil values by rendering $\overline{t_k}$ and setting the stencil value to 0 on the triangle. For a set of triangles, we reduce the state change overhead by associating a unique stencil id with each triangle. Since the stencil buffer is limited to 8 bit precision, we use a buffer that maintains a set of available stencil ids, which are replaced using a least-recently-used replacement policy.

Input: Object o_1, Set of objects $\mathcal{S}$. Parameterization from o_1 to $\mathcal{T}_1$.
Output: The Surface Distance Map $D(o_1)$ of object o_1.

```
1  Initialize D(o_1) to ∞ for all points q in T_1
2  Update AABB hierarchy of o_1
3  foreach triangle t_k in o_1 do M_{k,1} ← UpdateAffine(t_k)
4  foreach object o_a in S contributing to SDM of o_1 do
5      foreach site p_i in o_a do
6          OBB(G_i) ← ComputeOBB (p_i)
7          Intersect OBB(G_i) against AABB hierarchy of o_1
8          foreach triangle t_k in o_1 intersecting G_i do
9              g_i ← IntersectPolytope(G_i, t_k)
10             foreach vertex x_j in g_i do
11                 Compute distance vector d(x_j, p_i)
12                 Transform x_j to x_j' using M_{k,1}
13                 Assign texture coordinates of x_j' ,
                   (r, s, t) ← d(x_j, p_i)
14             end
15             Draw textured polygon g_i on domain T_1
16         end
17     end
18 end
19 Read-back T_1
20 foreach triangle t_k' in T_1 do
21     Map distance values from t_k' to t_k
22 end
```

Algorithm 1: *Pseudo-code to compute the surface distance map of o_1 using sites in $\mathcal{S}$. Line 1 initializes the depth values in the texture atlas for the triangles in o_1. We compute the affine transform for each triangle in o_1 (Line 3). We first perform a broad-phase culling to compute objects that contribute to SDM of o_1. For each site in these objects, we compute the potentially overlapping triangles in o_1 with the distance function of site p_i (Lines 6-8). We then clip the bounding polytope enclosing the distance function of p_i to the overlapping triangles in o_1 (Line 9). Each clipped polygon corresponds to a triangle in o_1 and we map the clipped polygons onto the texture atlas using the affine transformations of the corresponding triangles (Lines 10-15).*

4.2 Hierarchical Culling

We use a hierarchical distance culling algorithm to reduce the number of triangle-site pairs in the surface distance map computation. The distance functions are computed from a site p_i to a triangle t_k in 3D mesh only when the intersection of the bounding polytope with the triangle is non empty (i.e. $G_i \cap t_k \neq \emptyset$). We use an axis-aligned bounding box (AABB)-hierarchy of each object to quickly

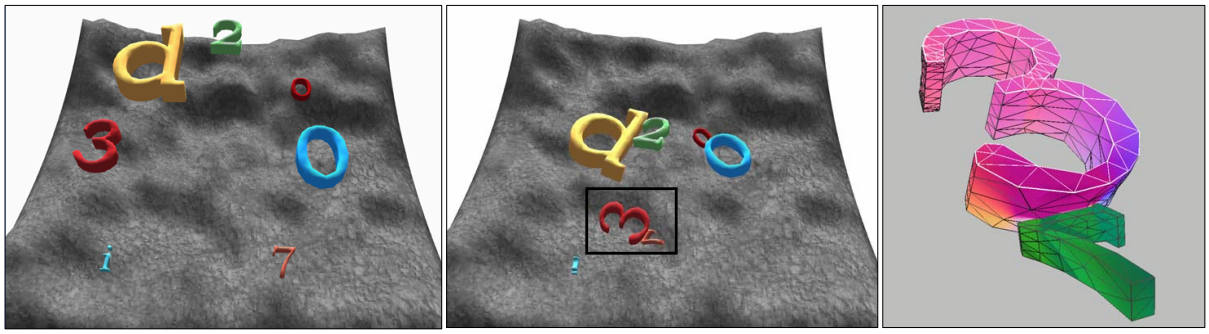

Figure 3: Surface Distance map computation on deforming letters "3d2007i": *Deforming dynamic simulation on 7 letters falling on an uneven terrain, (6K triangles total). (a)-(b) Two frames from the simulation. (c) The surface distance maps between two letters show the direction of the closest point on the other letter. Our algorithm can perform proximity queries using high resolution surface distance maps of resolution* 512×512 *in* $100 - 200$ *ms per frame.*

cull away sites whose bounding polytopes do not overlap with the triangles in the 3D mesh.

Our algorithm initially constructs an AABB hierarchy for each object. Each leaf of the hierarchy stores a triangle of the object. At run-time, we update the AABB-hierarchy and use it for culling bounding polytopes that do not intersect with the AABB-hierarchy. The hierarchy nodes are updated in a bottom-up manner. The update cost of a hierarchy is linear in the number of leaves in the AABB-hierarchy. For each site p_i, we compute a bounding polytope G_i and compute a tight-fitting oriented bounding box $OBB(G_i)$ that encloses G_i. We perform overlap tests between $OBB(G_i)$ and the nodes of the AABB hierarchy. For each leaf with triangle t_k that overlaps with $OBB(G_i)$, we perform distance computations on $G_i \cap t_k$ as described in Section 4. The OBBs are constructed only once per frame for each site, and therefore, the time taken to update the OBBs is linear in the number of sites in the scene. The AABB hierarchy and OBBs are updated each frame for deforming models.

We further improve the performance of our surface distance map algorithm by reducing the number of distance function rasterization operations using distance bounds computed using the AABB hierarchy. For each node in the AABB hierarchy, we maintain a lower bound on the maximum distance from the AABB of a triangle t_k to the AABB of the sites. Initially, the maximum distance bound of each node in the hierarchy is set to ∞. We do not perform distance evaluation of a site p_i for triangle t_k if the distance bound stored for a node in the hierarchy is less than the minimum distance from the AABB of the node to the AABB of p_i. This culling test based on distance bounds is used to reject sites whose distance functions do not contribute to the distance map on t_k, as there exists some other sites that are closer to T_k.

If a site is not culled away, we intersect the bounding polytope G_i of the site with t_k and compute the distance vectors at the vertices of $G_i \cap t_k$. We then perform distance function computation on $G_i \cap t_k$.

5 Analysis

In this section, we analyze the time complexity of our algorithm. We also derive error bounds on the distance computation as a function of 2D grid resolution in the parametric domain.

Time Complexity: Let there be m sites in each object. The cost of performing hierarchical culling is $O(m \log m)$. The rasterization cost of computing the surface distance map of resolution $M \times M$ is $O(rM^2)$, where $1 \leq r \leq m$. The value of r depends on the culling efficiency achieved by our hierarchical culling algorithm and is typically close to 1, especially when we perform distance computations in a localized region or narrow bands. Hence the total computation cost of computing the surface distance map for each object is $O(rM^2 + m \log m)$.

Error Bounds: Our algorithm, described in Section 4, computes a discrete surface distance map at the sample points on the mesh. The accuracy at these samples is governed by the precision of the texture mapping hardware that performs bilinear interpolation. Current GPUs offer 32-bit floating arithmetic to perform these computations. We also present an error bound on the computed distance for any point on the surface, as the object undergoes non-rigid affine transformations, including scaling and shearing. Given a sampling on the parametric domain, we first derive a function that computes the sampling density on the surface in 3D using the inverse of the affine map. The 3D sampling density is used to compute discretization error bounds on the surface distance map.

The affine transform for each triangle can be decomposed into a combination of scale, shear, translation and rotation transforms. The distances are preserved under rigid transformations. In this case, scaling and shear transforms change the distance values between adjacent samples. Next, we present a lemma that relates the sample density on the parametric domain with that on the 3D mesh.

Lemma 2. *Let the affine transform* $\mathbf{M_k}$ *map triangle* t_k *on object* o *to triangle* $\overline{t_k}$ *in texture domain* $\mathcal{T}$*. Let* s_x, s_y *and* s_h*, respectively, be the scale along* X*-axis, scale along* Y*-axis and shear induced by the inverse* $\mathbf{M_k^{-1}}$*. Further, let the base length of the triangle* $\overline{t_k}$ *be* b *and spacing between adjacent samples in* $\mathcal{T}$ *be* δ*. Then the maximum distance between two adjacent samples on* t_k *is given by*

$$f_k(\delta) \leq \delta \sqrt{(s_x + d_x)^2 + s_y^2} \tag{3}$$

where $d_x = \max(|s_h| - (\frac{b}{s_x \delta} + 1), 0)$.

It has been shown that the error introduced by a distance distance field is bounded by the sample density [32]. Hence from Lemma 2, the maximum error in the continuous surface distance map computed on object o is given by $e(\delta) = \max_k(f_k(\delta))$, for all triangles $t_k \in o$. Furthermore, it follows from equation (3) as $\delta \rightarrow 0$, $d_x \rightarrow 0$, and the maximum error, $e(\delta) \rightarrow 0$ as expected.

Moreover, the error in the surface distance map is bounded as the object o undergoes bounded deformations. We assume the initial mapping $\mathbf{M_k}$ has unit scale and zero shear. Then, we can derive the error bound as the object undergoes deformations by the following lemma:

Lemma 3. *Let the maximum motion of a vertex on triangle* t_k*, modulo any rigid body transformation, be bounded by* d_m*, and the*

base and height of triangle $\overline{t_k}$ on domain $\mathcal{T}$ be b and h, respectively. Then the discretization error in surface distance map is bounded by

$$g(\delta) \leq 2d_m\delta + \max\left[d_m - \frac{h}{2}\left(\frac{b}{\delta}+1\right), 0\right]$$

Surface Distance Map Resolution: By definition, the transform $\mathbf{M_k}$ is always invertible. Conversely, one can also use the inverse of the function to compute the sampling required in the texture domain to achieve a desired precision in the computed surface distance map. In particular, let e_m denote the maximum error in the surface distance map. Then the sampling density δ on the texture domain $\mathcal{T}$ is given by $\min_k(f_k^{-1}(e_m))$, for all $t_k \in o$, where $f_k^{-1}x$ is given by the scale and shear transforms of $\mathbf{M_k}$. The resolution of the texture used to sample $\mathcal{T}$ is $M \times M$, where $M = \frac{1}{\delta}$. We use these error bounds to perform proximity computations at object-space precision in Section 6.2

6 Implementation and Performance

In this section we briefly describe our implementation of surface distance map computation and highlight its application to motion planning and proximity computations between deformable models. We also compare our algorithm with prior distance field computation algorithms.

6.1 Implementation

We have implemented our algorithm on a PC with a 3.0GHz Pentium D CPU, 2GB of memory and an NVIDIA 7900 GTX GPU connected via a PCI-Express bus, running Windows XP operating system. We used OpenGL as the graphics API and the Cg programming language to implement the fragment programs. The initial mapping from the manifold objects to the texture atlas is computed using NVIDIA's Melody [1] software. The surface distance map of each object is computed on a floating point buffer using 32-bit floating point precision. The distance vectors are passed as texture parameters to the fragment program.

Our algorithm can compute high-resolution (512×512 to $1K\times1K$) surface distance map of objects with tens of thousands of polygons in a fraction of a second. We also compute the gradient of the distance field which gives the direction to the closest primitive for a point on the surface of an object. As compared to prior approaches based on volumetric techniques, our surface distance map computation algorithm is about $4-10$ times faster.

6.2 Comparison with Prior Distance Field Computation Algorithms

We compare the features and performance of our surface distance map algorithm with prior approaches that compute the distance field on a uniform volumetric grid using GPUs. These include linear factorization [31] and efficient GPU implementations of CSC algorithm [30, 27]. All the prior approaches compute the distance field along a uniform 3D grid. Since the GPU computes the distance field along one slice, these algorithm perform the computations along different slices and exploit spatial coherence between the slices to speed up the computation.

The precision of the distance field computed using a volumetric approach is governed by the cell size in the grid. Let the number of cells in the grid be $M \times M \times M$, and storage overhead is $O(M^3)$.

[1] http://developer.nvidia.com/object/melody_home.html

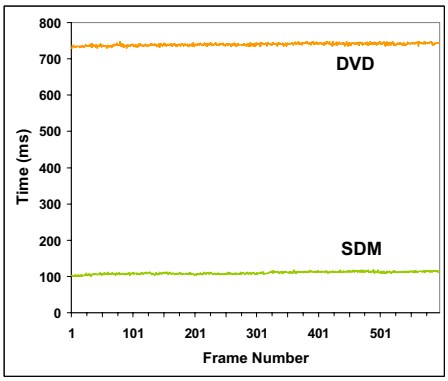

Figure 4: Timing comparison for proximity computation between our algorithm (labeled **SDM**) and a GPU-based volumetric distance field algorithm [Sud et al. 06b] (labeled **DVD**), respectively: *Our algorithm is able to achieve 5–10 times speedup in proximity computation between two deforming alphabets. The scene is composed of 6K polygons. The surface distance field is computed at a resolution of 512×512 and the volumetric distance field is computed at $250 \times 42 \times 250$. Our algorithm is able to obtain higher accuracy in distance field computation on the surface and achieves an interactive performance of 5–10 frames per second.*

Then the discretization error in the distance field is $\frac{\sqrt{3}}{2M}$. In comparison, for a surface distance map of size $M \times M$, the storage cost is $O(M^2)$, and the error in the distance field is $\frac{\sqrt{2}}{2M}$ in absence of any scale and shear. As the model undergoes deformation, the error bound for surface distance map is given by the function $f(\frac{1}{M})$, presented in Section 5. Typically, the maximum amount of deformation d_m in interactive applications is small, and the error in the distance field is given as $O(\frac{1}{M})$. As a result, surface distance maps provide a more compact representation of the distance field with tighter error bounds. Conversely, our approach results in higher resolution distance fields. Current GPUs have 512MB or 1GB of video memory. It may not be possible to store or compute a volumetric distance field at a high resolution (e.g. $(1K)^3$) as it would require 8GB of memory. Furthermore, the cost of reading back a 3D distance field of $(1K)^3$ and scanning could be rather high, i.e. about 16 seconds using a readback bandwidth of 500MB/sec. As a result, prior applications of interactive distance field computations are limited to low resolution distance fields. On the other hand, we can compute surface distance maps at high resolution on current GPUs.

Let there be m sites in each object. Then the computation cost to compute the global distance field using a volumetric approach varies between $O(mM^3)$ and $O(M^3)$. For *narrow bands*, the cost is $O(m + rM^3)$ where r depends on the relative configuration of sites. On the other hand, the cost of computing the global surface distance map on the GPU is $O(rM^2 + m\log m)$. For *narrow bands*, the cost is close to $O(M^2 + m\log m)$. A quantitative comparison of average time to compute the distance fields and perform proximity queries on deformable models is shown in Figure 4.

6.3 Proximity Queries between Deformable Models

We use surface distance maps and nearest-neighbor maps to perform different proximity queries among 3D deformable models. The set of queries include separation distance, collision detection, contact normal and local penetration depth computation. As shown in Lemma 1, the nearest neighbor map also provides the 2nd-order governor set, and we use the algorithm presented in [32]. We first use AABB hierarchies to localize the computation by computing the region of overlap or compute a potentially neighboring

set (PNS) for the primitives. Next, we compute the surface distance map for all triangles of each object that lie inside the localized region or the PNS. We use the error bounds on distance fields, presented in Section 5, to perform conservative computations and reduce the size of PNS based on properties of nearest-neighbor maps. Finally, we perform exact collision, distance or penetration queries between the primitives lying in the PNS.

We used our algorithm for proximity queries on a scenarios consisting of deforming objects. The first is a sequence of 7 deforming alphabets falling on a bumpy terrain as shown in figure 3. The letters are at different scales allowing us to vary resolution of surface distance map. The entire environment consists of 6K polygons. At each frame, we compute a surface distance map on each letter at a resolution 512×512. The average time to perform all proximity queries varies between $100 - 200$ms. As compared to [32], our surface distance algorithm results in a speedup of 8 times. Since we are computing the distance map at a much higher resolution, the image-space error using our algorithm is much lower as compared to prior approaches, giving us significantly smaller error bounds.

The use of surface distance maps and nearest-neighbor maps considerably improve the performance of the proximity query algorithm. There are two main reasons.

- **Faster computation:** The underlying distance field computation algorithm is much faster. This is due to the fact that we are only computing distance fields on the boundary of the objects (i.e. a 2D manifold) as opposed to a 3D volumetric grid.
- **Higher accuracy :** We compute the surface distance fields at a higher resolution (e.g. 512^2 or $1K \times 1K$) as opposed to volumetric approaches, which would typically compute at 64^3 or 128^3 grid resolutions. As a result, our distance error bounds are much tighter and the Voronoi-based culling results in a smaller PNS and we perform significantly fewer exact tests in the primitives.
- **Adaptive resolution :** Using surface distance maps, we can select a unique resolution for each object, providing us with adaptive resolution in large environments with variation in scale.

6.4 Motion Planning in Dynamic Environments

We use our distance field computation algorithm for interactive motion planning of multiple 3D agents or robots moving along a 2D manifold, but the environment consists of 3D obstacles. Some of the driving applications include crowd simulation in urban environments or architectural models, vehicles moving along a terrain, etc. We represent the environment as a set of objects, $\mathcal{S}$, and partition it into two disjoint sets: a set of obstacles $\mathcal{S}_o$ and a set of 'ground' surfaces $\mathcal{S}_g$. A ground surface is a 2-manifold along which robots are constrained to move.

Our goal is to perform motion planning for multiple moving robots or agents with no assumptions about their motion. In this case, each robot is treated as a dynamic obstacle for the other robots. As a result, prior motion planning algorithms based on sampling-based techniques and pre-computation of a roadmap are not directly applicable.

Voronoi diagrams have been widely used for motion planning, including roadmap computation [6], sample generation [13] or combined with potential field methods for 2D robots [18]. It is well known that the Voronoi diagram of the obstacles $\mathrm{VD}(\mathcal{S}_o)$ represents the connectivity of the free space of the robot, and can provide paths of maximal clearance between the obstacles. The 3D Voronoi diagram consists of 2D Voronoi faces, 1D Voronoi edges and Voronoi vertices. Since the motion of the robots is constrained to $\mathcal{S}_g$, we compute the intersection of the 3D Voronoi diagram $\mathrm{VD}(\mathcal{S}_o)$ with the ground surfaces $\mathcal{S}_g$. In this case, $\mathrm{VD}(\mathcal{S}_o) \cap \mathcal{S}_g$ gives paths of maximal clearance along $\mathcal{S}_g$.

For each ground surface $o_g \in \mathcal{S}_g$, we compute the surface distance map $D(o_g|\mathcal{S}_o)$ and the nearest neighbor map $N(o_g|\mathcal{S}_o)$. Since $\mathcal{S}_o$ and $\mathcal{S}_g$ are disjoint, it follows from Lemma 1 that the nearest neighbor map $N(o_g|\mathcal{S}_o)$ is equivalent to $\mathrm{VD}(\mathcal{S}_o) \cap o_g$. We extract the discrete Voronoi graph from $N(o_g|\mathcal{S}_o)$ and assign the edge weights based on edge length and maximum clearance along the edge, as described in [18]. The Voronoi vertices closest to the robot and the goal position are classified as source and destination, respectively, and the minimum weight path is computed using Dijkstra's shortest-path algorithm.

We combine the roadmap computed from the nearest-neighbor map with a potential field planner for local planning. For each robot, the potential field planner takes into account the proximity to the nearest obstacle. Hence, for each robot object $o_r \in \mathcal{S}_o$, we compute the surface distance map $D(o_r|\mathcal{S}_o)$ and use that map to compute the proximity information. The surface distance map is sampled at a finite set of points and we use that sampling to compute an average force and torque to simulate the robot dynamics.

We demonstrate the application of surface distance maps for interactive motion planning of a large number of human agents in an urban environment with dynamic 3D obstacles (see Figure 1). The set of obstacles consists of buildings, cars, flying drones and humans. The set of ground surfaces consists of roads, sidewalks and lawns. The humans enter the scene from the buildings and exit through another building or the sidewalks. Each human is an individual robot and has an independent goal. The cars and drones, along with other humans, are treated as dynamic obstacles, while the buildings, benches, fountains are treated as static obstacles. In our implementation, a higher weight is assigned to Voronoi edges corresponding to the cars and flying drones. As the flying drones approach the ground, the humans update the paths to evade them. The nearest-neighbor map is computed on a grid of resolution $1K \times 1K$ pixels. The environment has 15 static obstacles, and up to 8 dynamic obstacles, and 100 dynamically moving human agents. The complete motion planning takes 120ms per frame, which includes cost of computing the nearest neighbor map, extracting the discrete Voronoi graph and performing graph search. The time spent on nearest neighbor map computation is approximately 30ms per frame.

7 Limitations and Conclusions

Our approach has certain limitations. We compute a 2D domain triangle for each triangle in the 3D mesh. We pack all these 2D domain triangles in the texture atlas and our current packing algorithm may not be optimal. Our current approach is limited to deforming triangles with fixed connectivity. If the underlying simulation consists of objects with changing topologies, we may need to update the planar parameterization and recompute the spatial hierarchies. The accuracy of our proximity computation algorithm is governed by the resolution of the distance map. It is possible to compute higher resolution distance fields using GPUs, but then the cost of readback to CPU goes up, especially for interactive applications.

7.1 Conclusions and Future Work

We present a new algorithm to compute surface distance maps for triangulated models using the texture mapping hardware. We compute a planar parameterization of the mesh and use the affine map-

ping to efficient evaluate the distance maps. We also present culling and clipping techniques to speed up the computations. We highlight the performance of our algorithm on complex models and use it to perform interactive proximity queries between deformable models.

There are many avenues for future work. We could further improve the performance of our algorithm by using spatial and temporal coherence between successive frames. It may be possible to extend our algorithm to objects with changing topologies, where we incrementally recompute the affine transformation to the parametric domain. We would like to use proximity computation algorithm to perform self-proximity queries including self-collisions or self-penetrations in cloth simulation. Surface distance maps could also be useful to accelerate ray tracing dynamic scenes [33].

ACKNOWLEDGMENTS

This research is supported in part by ARO Contracts DAAD19-02-1-0390, and W911NF-04-1-0088, NSF Awards 0400134, 0118743, ONR Contract N00014-01-1-0496, DARPA RDECOM Contract N61339-04-C-0043 and Intel Corporation. We would like to acknowledge members of UNC GAMMA and the reviewers useful comments and feedback.

REFERENCES

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

[2] I. Bitter, A. Kaufmann, and M. Sato. Penalized-distance volumetric skeleton algorithm. *IEEE Trans. on Visualization and Computer Graphics*, 7(3), 2001.

[3] J. F. Blinn. Simulation of wrinkled surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, pages 286–292, 1978.

[4] 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.

[5] E. Catmull. *A subdivision algorithm for computer display of curved surfaces*. PhD thesis, University of Utah, 1974.

[6] H. Choset and J. Burdick. Sensor based motion planning: The hierarchical generalized Voronoi graph. In *Algorithms for Robot Motion and Manipulation*, pages 47–61. A K Peters, 1996.

[7] Robert L. Cook. Shade trees. In Hank Christiansen, editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 223–231, July 1984.

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

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

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

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

[12] I. Fischer and C. Gotsman. Fast approximation of high order Voronoi diagrams and distance transforms on the GPU. Technical report CS TR-07-05, Harvard University, 2005.

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

[14] Alain Fournier. Normal distribution functions and multiple surfaces. In *Graphics Interface '92 Workshop on Local Illumination*, pages 45–52, May 1992.

[15] 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.

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

[17] J. Gomes and O. Faugeras. The vector distance functions. *Int. Journal of Computer Vision*, 52(2):161–187, 2003.

[18] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Interactive motion planning using hardware accelerated computation of generalized voronoi diagrams. *Proceedings of IEEE Conference of Robotics and Automation*, 2000.

[19] Kenneth E. Hoff, III, Tim Culver, John Keyser, Ming Lin, and Dinesh Manocha. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Computer Graphics Annual Conference Series (SIGGRAPH '99)*, pages 277–286, 1999.

[20] A. Klein, P. J. Sloan, A. Finkelstein, and M. Cohen. Stylized video cubes. *Symposium on Computer Animation*, 1992.

[21] 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.

[22] A. Lefohn, J. Kniss, C.D. Hansen, and R. Whitaker. Interactive deformation and visualization of level set surfaces using graphics hardware. In *Proceedings of IEEE Visualization*, page To Appear, 2003.

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

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

[25] C.R. 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.

[26] Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Chichester, UK, 1992.

[27] R. Peikert and C. Sigg. Optimized bounding polyhedra for gpu-based distance transform. In *Scientific Visualization: The visual extraction of knowledge from data*, 2005.

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

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

[30] C. Sigg, R. Peikert, and M. Gross. Signed distance transform using graphics hardware. In *Proceedings of IEEE Visualization*, pages 83–90, 2003.

[31] A. Sud, N. Govindaraju, R. Gayle, and D. Manocha. Interactive 3d distance field computation using linear factorization. In *Proc. ACM Symposium on Interactive 3D Graphics and Games*, pages 117–124, 2006.

[32] Avneesh Sud, Naga Govindaraju, Russell Gayle, Ilknur Kabul, and Dinesh Manocha. Fast proximity computation among deformable models using discrete voronoi diagrams. *ACM Trans. Graph. (Proc ACM SIGGRAPH)*, 25(3):1144–1153, 2006.

[33] L. Szirmay-Kalos, B. Aszodi, I. Lazanyi, and M. Premecz. Approximate ray-tracing on the gpu with distance impostor. *Proc. of Eurographics*, 2005.

[34] M. Teichmann and S. Teller. Polygonal approximation of Voronoi diagrams of a set of triangles in three dimensions. Technical Report 766, Laboratory of Computer Science, MIT, 1997.

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

[36] M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide, Second Edition*. Addison Wesley, 1997.