Surface Distance Maps

Avneesh Sud, Naga Govindaraju, Russell Gayle and Dinesh Manocha
Department of Computer Science, Univeristy of North Carolina
Chapel Hill, NC, USA
{sud,naga,rgayle,dm}@cs.unc.edu
http://gamma.cs.unc.edu/gvd/SDM

UNC Computer Science Technical Report TR06-011

Abstract

We present an interactive algorithm to compute surface distance maps for triangulated models. The distance map represents the distance-to-closest-primitive mapping at each point on the boundary of the mesh. We precompute a surface parameterization and use the parameterization to define an affine transformation for each primitive of the mesh. Our algorithm efficiently computes the distance field by applying the affine transformation of the primitive to the distance functions of the sites. We evaluate the distance functions using texture mapping hardware. In practice, our algorithm can compute high resolution surface distance maps at interactive rates and provides tight error bounds on their accuracy. We use surface distance maps to perform interactive collision and distance queries between complex deformable models. As compared to prior approaches, our distance map computation and collision detection algorithms can provide up to one order of magnitude performance improvement.

1 Introduction

Distance fields are scalar fields that represent the closest distances. Given a set of primitives $\mathcal{O}$ in $\mathbb{R}^3$, the distance field at a point equals the distance to the closest point on $\mathcal{O}$. 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 [FPRJ00], skeletonization [BKS01], collision and penetration depth computation [HZLM01], remeshing [KBSS01], motion planning [HCK*99], implicit surface representation [Gib98], non-photorealistic rendering [KSFC92], etc.

In this paper, we consider the problem of computing the distance map on a two-manifold triangulated mesh in $\mathbb{R}^3$. The *surface distance map* computes the distance-to-closest-primitive mapping at each point on the boundary of the mesh. The distance function varies continuously along the surface and the gradient of the distance map yields the direction vector to the closest object. If the primitives $\mathcal{O}$ are closed and orientable, we can also associate a sign with the distance map.

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 [HCK*99, SGGM06, SPG03, FG05]. 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 [SOM04, PS05]. However, these volumetric techniques have many limitations. Their storage overhead and computation time is $O(n^3)$, where n is the resolution along the grid. As a result, current 3D distance field computation algorithms are not fast enough for interactive applications. Moreover, their accuracy can be low as most of the grid vertices do not exactly lie on the mesh boundary.

Main Results: We present a new algorithm to compute surface distance maps of triangulated models. Our algorithm uses a simple texture representation to store a piecewise planar parametrization of the mesh. The parameterization defines an affine transformation for each primitive of the mesh. The 2D texture map is used as a discrete sampling of the mesh for distance map computation.

We apply the affine transformation of the geometric primitive to compute the distance functions of 3D primitives using the texture mapping hardware. We use the stencil test to clip the distance functions to regions correspond-

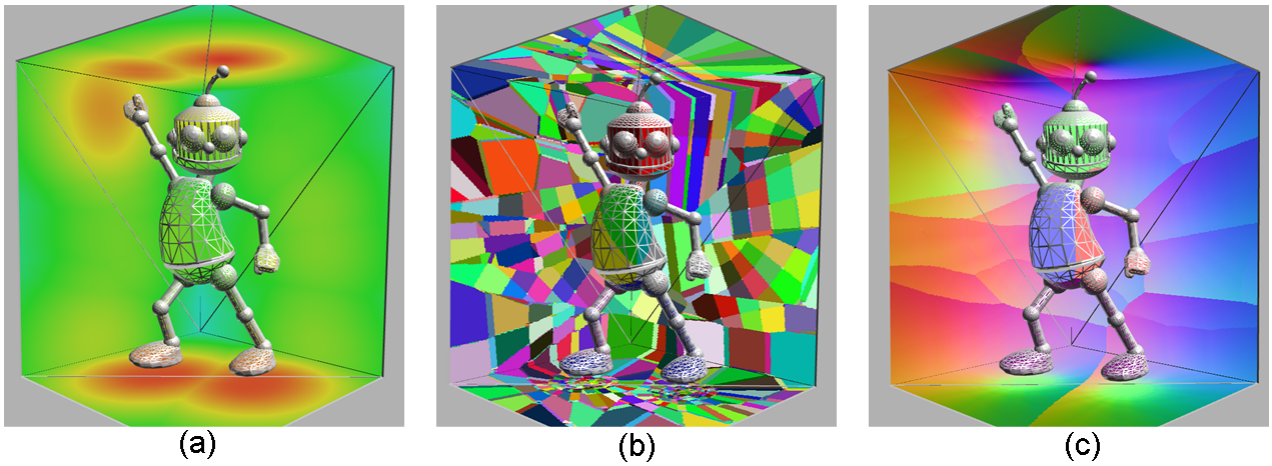

Figure 1: Surface distance map of the Hugo model enclosed in a box: *We show the surface distance map of the Hugo model (*17.2*K polys - in wireframe) within a box (*12 *polys). (a) The surface distance fields of Hugo on the box and of the box on the Hugo model. The distances increases from red to green. (b) The Voronoi diagrams of the Hugo and box that are used to compute the distance maps. Each colored region represents a Voronoi region of a site. (c) The normalized gradient of the distance field. The color of a point on the box encodes a vector representing the direction to its closest point on Hugo (and vice versa). Our algorithm can compute surface distance map of the Hugo and the box in 600ms on a grid of resolution* 256×256.

ing to the geometric primitive in the 2D texture. Our algorithm employs spatial hierarchies to localize the distance field computations and improve the overall performance.

We have implemented our algorithm on a Pentium IV PC with an NVIDIA GeForce 7800 GTX GPU. We highlight its performance on complex benchmarks composed of thousands of triangles. In practice, our algorithm is able to compute high resolution distance fields in a few hundred milli-seconds. The distance values are computed on a floating point buffer using 32-bit floating point precision. We use our algorithm to perform interactive collision and separation distance queries between 3D deforming objects. As compared to prior techniques, our algorithm offers the following advantages:

- **Generality:** Our algorithm is applicable to all manifold triangulated models. The only requirement is computation of the piecewise 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 techniques based on volumetric approaches could barely compute distance fields 128^3 or 256^3 resolution at interactive rates.
- **Performance:** Our algorithm can handle deformable models with thousands of polygons at interactive rates. We observe $5 - 10$ times speedup over prior distance field computation and collision detection algorithms.

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 benchmarks 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 [Cui99, Aur91, PS05].

The algorithms for image-based data sets perform exact or approximate computations in a local neighborhood of the voxels. [Dan80, Set99, BGKW95, MQR03, GF03]. Exact algorithms for handling 2-D and k-D images have been propose to compute the distance transforms in voxel data in $O(N)$ time, where N is the number of voxels [BGKW95, MQR03].

There is extensive work in computing the exact Voronoi diagram of a set of points [Aur91]. However, exact computation of Voronoi regions of higher order primitives such as lines or triangles is a hard problem due to its algebraic and

combinatorial complexity. As a result, most practical algorithms compute an approximation to the Voronoi diagram by computing distance fields on a uniform grid or an adaptive grid. A key issue is the underlying sampling criterion used for adaptive subdivision [VO98, TT97, ER02, PF01].

The computation of a discrete Voronoi diagram on a uniform grid can be performed efficiently using graphics rasterization hardware. This idea was original proposed for point primitives in [WND97]. Hoff et al. [HCK*99] render a polygonal approximation of the distance function on depth-buffered graphics hardware and computed the generalized Voronoi Diagrams in two and three dimensions. The 3D algorithm computes each slice separately. An efficient extension of the 2-D algorithm for point primitives is proposed in [Den03]. Sud *et al.* [SOM04, SGGM06] present algorithms efficiently compute distance fields of polygonal primitives by using a combination of culling and clamping algorithms and map the computations to the texture mapping hardware. In practice, these algorithms can improve the performance of 3D distance field computation considerably, but are not fast enough for interactive applications. Fischer and Gotsman [FG05] describe techniques to approximate higher order Voronoi diagrams and distance fields using GPUs.

A class of exact distance computation and collision detection algorithms based on external Voronoi diagrams are described in [Lin93]. 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 [Mau03]. 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.* [SPG03] describe an efficient GPU based implementation of the CSC algorithm. Peikert and Sigg [PS05] present algorithms to compute optimized bounding polyhedra of the Voronoi cell for GPU-based distance computation algorithms. Lefohn *et al.* describe an algorithm for interactive deformation and visualization of level set surfaces using graphics hardware [LKHW03].

2.2 Surface Mapping and Parameterization

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 [Cat74], which is used to define the color on the surface; displacement mapping [Coo84], which consists of perturbations of the surface positions; bump mapping [Bli78], which give perturbations to the surface normals; and normal maps [Fou92], which contains the actual normals instead of the perturbations. All these mapping are supported by current graphics hardware.

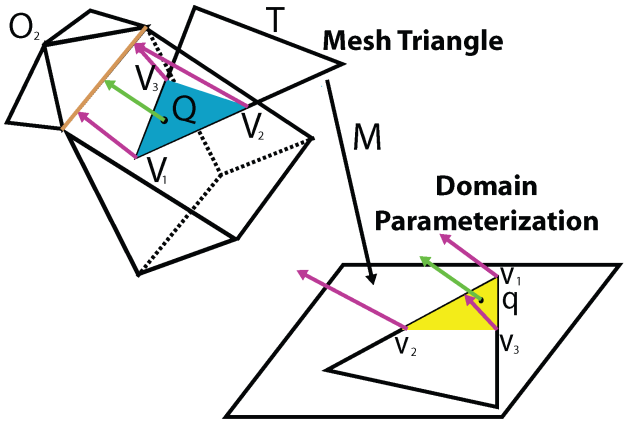

Figure 2: ***Affine map and distance computation:*** We compute the distance map at a point $\mathbf{Q}$ on triangle T (of O_1). The green vector shows the closest site of O_2 to $\mathbf{Q}$. The affine map M maps T to a triangle in the 2D domain.

The problem of computing a parameterization is well studied in the literature. A recent survey of these techniques is given in [FH05]. Given a closed model, these algorithms cut the model into charts such that each chart is homeomorphic to a disk. Each chart is parameterized separately and the final parameterization is an atlas of these chart parameterizations.

3 Surface Distance Maps

In this section, we present surface distance maps and our algorithm to compute them efficiently using texture mapping hardware. We first introduce the notation used in the paper.

3.1 Notation

We use upper case letters to represent objects or triangles in 3D and lower case letter to represent their mapping on a 2D plane. We denote piecewise linear 2-manifold objects or meshes in 3D as O_i. Furthermore, O_i is decomposed into vertices, edges and faces, also known as *sites*. We assume that the model is triangulated and each edge and face represents an open set. A site is denoted as P_i. We use $d(\mathbf{Q}, P_k)$ to denote the distance function of a site P_k at a point $\mathbf{Q} \in \mathbb{R}^3$. The distance function of a site P_i on a triangle T in the 3D mesh represents the closest distance from each point $\mathbf{Q}$ on T to P_i. The closest vector from $\mathbf{Q}$ to P_i is known as the distance vector. The color at $\mathbf{Q}$ is defined as $color(\mathbf{Q})$ and we use this function to denote the closest distance.

Given two 2-manifold objects O_1 and O_2, the surface distance field $D(O_1)$ of an object O_1 at a point $\mathbf{Q} \in O_1$ is the minimum value of the distance functions of all sites

$P_k \in O_2$ at $\mathbf{Q}$. The surface distance map of O_1 computes the surface distance field at a discrete set of sampled points on O_1. We define an affine mapping $M_k : T_k \rightarrow \mathcal{T}_1$ to transform the sampled points on the triangles $T_k \in O_i$ into a 2D plane $\mathcal{T}_i \subset \mathbb{R}^2$.

3.2 Distance Fields: Background

Distance fields can be computed efficiently on discrete volumetric grids by rasterizing the distance function of each site to the points in the grid. Many algorithms compute the distance functions from each site to the points on the planes swept along the Z-axis of the grid [SOM04, SGGM06, SPG03, PS05]. These algorithms perform the distance field computation using one of these approaches:

1. Evaluate the distance function $d(\mathbf{Q}, P_k)$ at each point $\mathbf{Q}$ in the plane directly by rasterizing the distance functions and use the depth-buffer hardware.

2. Compute the distance vector from $\mathbf{Q}$ to the site and use the magnitude of the distance vector to compute $d(\mathbf{Q}, P_k)$. This computation can be efficiently performed using the bilinear interpolation capabilities of the texture mapping hardware.

In order to accelerate the computations, prior algorithms construct a convex bounding polytope B to represent the region of influence of a site on each plane along the Z-axis. As a result, the distance function is only evaluated at the points inside B. We use similar techniques to accelerate the computation of surface distance maps.

3.3 Planar Parameterization

Given a 3D mesh with triangles $T_k, k = 1, \ldots, n$, our algorithm transforms T_k into a 2D plane t_k by applying an affine mapping M_k (see Fig. 2). M_k is represented as a matrix and ensures the following properties:

- There is a one-to-one mapping from a point $\mathbf{Q} \in T_k$ to the point $M_k\mathbf{Q} \in t_k$. This mapping is computed by performing matrix-vector multiplication $M_k\mathbf{Q}$.

- No two transformed triangles $t_k = M_kT_k$ and $t_l = M_lT_l$ share a common interior point in the 2D plane.

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.

The affine transform for a triangle T_k to a triangle t_k in 2D plane is computed by first rotating the triangle T_k into the plane of t_k and aligning an edge of t_k with the corresponding edge of T_k. Finally, a shear transformation is applied to align the three vertices of t_k with the transformed vertices of T_k. Mathematically, $M_k = \mathbf{A_t}\mathbf{A_s}\mathbf{A_r}$ where $\mathbf{A_t}$ is a translation matrix, $\mathbf{A_s}$ represents a scale and shear matrix in the XY plane and is of the form

$$\mathbf{A_s} = \begin{bmatrix} sx & sh * sy & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

and $\mathbf{A_r}$ is a rotation matrix.

3.4 Surface Distance Computation

Surface distance maps compute the distance-to-closest-primitive in the the scene to the sampled points on the surface of the mesh. We first compute the affine mappings, M_k, for each triangle T_k in the 3D mesh. These affine map defines the sampling on each triangle T_k in 3D space by sampling the projected triangle t_k in the 2D plane of the surface distance map. The surface distance map samples the 2D plane uniformly using a 2D texture. Instead of computing distances using a volumetric grid, our algorithm computes the distance map on each triangle T_k using affine transforms of distance functions to a 2D plane containing t_k.

We present an algorithm to compute distance functions on a set of sampled points on the triangles of the 3D mesh. For each site P_i, we compute a convex bounding polytope B, which acts as a spatial bound on the Voronoi region of P_i. In other words, any point outside B can not lie in the Voronoi region of P_i. We intersect B with the plane of a triangle T_k in 3D mesh. We then use the following lemma to compute the distance vectors on T_k.

Lemma 1: *Given an affine transformation M_k that maps a triangle T_k to a triangle t_k in the 2D texture. Let B be the convex bounding polytope of a site P_i and let $\mathbf{V}_j$, $j = 1, \ldots, l$ denote the vertices of $B \cap T_k$. Let the color at $M_k\mathbf{v}_j \in t_k$ be $color(M_k\mathbf{v}_j) = d(\mathbf{v}_j, P_i)$ and $\mathbf{Q}$ be an interior point on $B \cap T_k$. Then $d(\mathbf{Q}, P_i)$ is equal to $color(M_k\mathbf{Q})$.*

Proof: Let $\mathbf{Q} = \sum_{j=0}^{k} \lambda_j \mathbf{v}_j, \lambda_j \geq 0, 1 \leq j \leq k$. Based on the properties of distance vectors, we can show that $d(\mathbf{Q}, P_i) = \sum_{j=0}^{k} \lambda_j d(\mathbf{v}_j, P_i)$. Furthermore $M_k\mathbf{Q} \in t$ and $M_k\mathbf{Q} = \sum_{j=0}^{k} \lambda_j M_k\mathbf{v}_j$. Since color is linearly interpolated, $color(M_k\mathbf{Q}) = \sum_{j=0}^{k} \lambda_j color(M_k\mathbf{v}_j) = \sum_{j=0}^{k} \lambda_j d(\mathbf{v}_j, P_i)$. Therefore, $color(M_k\mathbf{Q}) = d(\mathbf{Q}, P_i)$. Q.E.D.

Lemma 1 shows that the distance vector computation on $B \cap T_k$ can be performed by assigning the color at each vertex $\mathbf{v}$ to $d(\mathbf{v}, P_i)$ and transform the computation of $B \cap T_k$ to the 2D plane. Then, the distance vector for each point $\mathbf{Q} \in B \cap T_k$ can be computed using the texture mapping at the location $M_k\mathbf{Q}$ in the 2D texture.

3.5 Mapping to GPUs

Surface distance maps can be computed by the rasterization hardware by using the transformation, clipping and interpolation capabilities of the GPUs. We use Lemma 1 to design an efficient pipeline for surface distance map computation using GPUs:

- **Vertex Engines:** We first compute the bounding polytope on the CPU, and transform the vertices of the bounding polytope onto the 2D plane of surface distance map. The transformation operation is defined using a projective matrix and is implemented using the vertex processors on a GPU.
- **Stencil Test:** $B \cap T$ corresponds to the region of B projecting inside the triangle t (which is a mapping of T in the 2D domain) in the texture atlas. We use the stencil functionality of GPUs to clip the projection of B to the region inside t.
- **Texture Mapping:** The linear interpolation of color is equivalent to the interpolation of texture co-ordinates assigned to the vertices of the triangle. This functionality computes the distance vectors in the texture atlas.
- **Fragment processing:** The distance value at the fragment is the norm of the distance vector and computed using the fragment processor.
- **Depth Test:** The distance value is stored in the depth and compared with the current minimum distance value using the depth test functionality of GPUs. The minimum distance value is stored in the depth buffer.

Input: Two objects O_1, O_2. Parameterization $t(O_1)$ from O_1 to $\mathcal{T}_1$.
Output: The SDF $D(O_1)$ of object O_1.

```
1  Initialize D(O1) to ∞ for all points Q in T1
2  Update AABB hierarchy of O1
3  foreach face fi^w in O1 do UpdateAffine (fi^w, t(O1))
4  foreach site Pj in O2 do
5      Gj ← ComputeOBB(Pj)
6      Intersect Gj against AABB hierarchy of O1
7      foreach face fi^w in O1 intersecting Gj do
8          gj ← ClipPolytope(Gj, fi^w)
9          foreach vertex xk^w in gj do
10             Compute distance vector d(xk^w, Pj)
11             Transform xk^w to xk^s
12             Assign texture coordinates of xk^s,
               (r, s, t) ← d(xk^w, Pj)
13         end
14         Draw textured polygon gj on domain T1
15     end
16 end
17 Read-back T1
18 foreach face fi^s in T1 do
19     Map distance values from fi^s to fi^w
20 end
```

Algorithm 1: *Pseudo-code to compute the surface distance map of O_1 using sites in O_2. We initialize the distance values in the surface distance map $D(O_1)$ to ∞ (line 1). We then update the hierarchy and the affine transforms of triangles in O_1 using a linear-time algorithm (lines 2–3). Next, we update the surface distance map of O_1 using the sites in O_2 (lines 4 – 13). For each site, we compute its bounding polytope and intersect the OBB of bounding polytope with the AABB hierarchy (lines 5–6). For each intersecting polytope, we compute the surface distance map using stencil tests (lines 7–13).*

4 Interactive Distance Map Computation

In the previous section, we presented our algorithm to compute the distance map using the GPUs. The algorithm requires computing of intersections between bounding polytopes of sites and the triangles in the 3D mesh. In this section, we present culling and clipping techniques to accelerate the performance of the algorithm.

4.1 Clipping

Surface distance maps require an efficient clipping algorithm for each triangle-site pair (see Lemma 1). Given a site P_i and a triangle T_k, we restrict the computation on the 2D plane to the interior of t_k using stencil. As a result, each triangle-site pair requires a valid stencil portion in t_k and the stencil has to be set in the region corresponding to t_k. We first describe an algorithm to perform clipping using a single valid stencil value, and present a more efficient stencil caching algorithm that uses multiple valid stencil values to perform clipping.

The algorithm proceeds as follows. For each site P_i and a triangle T_k in a plane P_k in 3D space, we first compute its bounding polytope B and compute $B \cap P_k$. Next, we clip the transformed primitive $M_k(B \cap P_k)$ to t_k in the 2D plane. We use the stencil test functionality to perform the clipping operation. We first set the stencil value of the triangle to 1 by rendering t_k. We then render $M_k(B \cap P_k)$ onto the portions of the surface distance map with the stencil value set to 1. We then render t_k by setting the stencil value to 0 on the triangle.

For every two consecutive triangle-site pairs (T_k, P_i), $(T_l, P_j), k \neq l$, our algorithm resets stencil on regions corresponding to t_k and sets the stencil on regions corresponding to t_l. The reset and set stencil operations can become fill-bound. We improve the performance of our clipping al-

Figure 3: Collision and separation distance computation on deforming alphabets "EG": *Deforming dynamic simulation on two alphabets, (*3.7*K triangles total). (a)-(b) Two frames from the simulation. (c) The gradient of surface distance maps of each alphabet shows the direction of the closest point on the other alphabet. Our algorithm can compute the global distance maps for both bunnies in* 100*ms at a grid of resolution* 512×512*. The proximity queries involve readback and scanning and takes about* 10*ms on top of distance map computation.*

gorithm using a cache that maintains multiple stencil values. Initially, all the stencil values are unassigned. As the distance computations are performed on the triangles, the cache sets the unassigned stencil values to the triangles and to the newly assigned stencil value used for clipping operation on a triangle. We use a simple replacement policy if no value is available.

In order to compute the valid stencil value for T_k, we first test if the stencil is set on regions corresponding to t_k. If the stencil is set, we simply use that value for the clipping operations. On the other hand, if the stencil value is not set, we need to assign a valid stencil value to T_k. In order to assign a valid stencil value, we check if any of the stencil values in the cache are unassigned. If an unassigned value is available, we assign that value to T_k. If no valid stencil value is available, the cache uses the least recently used (LRU) replacement policy to determine the stencil value to be allocated to T_k. In this case, we first reset the stencil on the triangle whose stencil is least recently used and then allocate that stencil value to T_k.

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. Lemma 1 indicates that the distance functions are computed on a triangle T_k in 3D mesh only when $B \cap T_k$ is not empty. We use an AABB-hierarchy of each object to quickly cull away sites whose bounding polytopes B do not overlap with the triangles in the 3D mesh T_k.

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. We use the initially constructed AABB hierarchy and update the bounding boxes of the hierarchy nodes in a bottom-up manner. The update cost of a hierarchy is linear to the number of leaves in the AABB-hierarchy and is usually fast. For each site P_i, we construct a bounding polytope B and compute a tight-fitting oriented bounding box $OBB(B)$ that encloses B. We perform overlap tests between $OBB(B)$ and the AABBs that correspond to the nodes of the AABB hierarchy. For each leaf with triangle T_k that overlaps with OBB(B), we perform distance computations on $B \cap T_k$ as described in Section 3. The OBBs are constructed only once for each site, and therefore, the time taken to update the OBBs is linear to the number of sites in the scene.

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 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 B of the site with T_k and compute the distance vectors at the vertices of $B \cap T_k$. We then perform distance function computation on $B \cap T_k$.

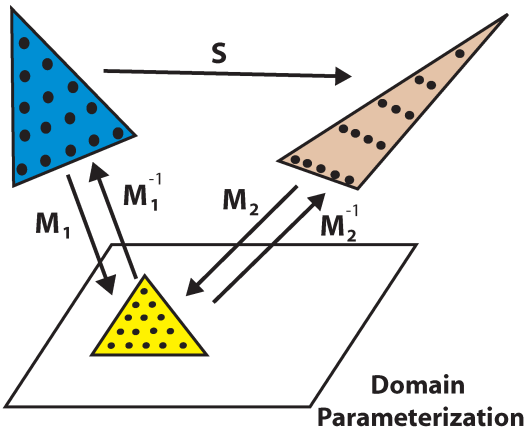

Figure 4: ***Distance map computation for a deforming triangle:*** The triangle undergoes a non-rigid deformation (S) in terms of shear and scale. We compute a new affine mapping for the triangle (M_2) and use it to compute the distance map on the triangle. The sample locations are shown as dots in the 2D domain and the triangles.

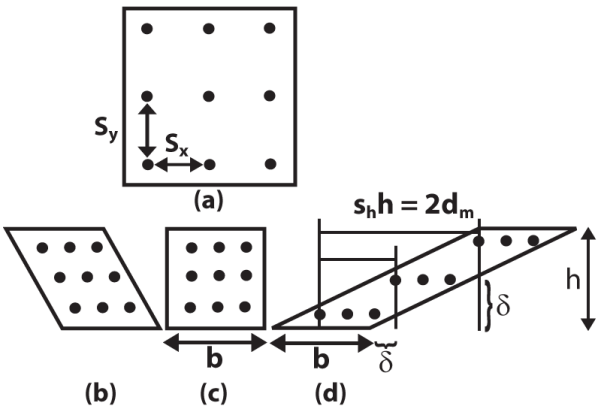

Figure 5: This figure highlights the distance between adjacent samples in the 2D plane when a rectangular planar primitive undergoes a scale (Fig. 5(a) and Fig. 5(c))or a shear transformation(Figs. 5(b),(c) and Figs. 5 (c), (d)).

5 Analysis

In this section, we analyze the accuracy of our algorithm. We show that our algorithm can be used to compute a distance map up to a desired precision. We also consider the case when the triangles undergo non-rigid deformations and highlight the accuracy of distance maps based on the affine transformations.

5.1 Error Analysis

The algorithm presented in section 4 computes an accurate surface distance map at the sample points on the boundary of the objects. Its accuracy 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-linear deformations. Given a sampling on the texture domain, we derive a function to compute the sampling density on the surface in 3D using the inverse of the affine map. Given the sampling density in 3D, we compute bounds on the distance. 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 distance field.

Given two points $\mathbf{p}^w$ and $\mathbf{q}^w$ on an object O_1 and the surface distance map of O_1 w.r.t. object O_2, the change in the value of surface distance map from $\mathbf{p}$ to $\mathbf{q}$ is bounded by the distance between $\mathbf{p}^w$ and $\mathbf{q}^w$ [SOM04]:

$$\| d(\mathbf{p}^w, O_2) - d(\mathbf{q}^w, O_2) \| \leq \| \mathbf{p}^w - \mathbf{q}^w \| .$$

In order to bound the error in computed distances, we bound the distance between a given point and the closest sample from the surface distance map. This is bounded by the maximum distance between four adjacent samples in the surface distance map.

Let $\mathbf{p}^w$ and $\mathbf{q}^w$ be adjacent points on the surface of O_1. The corresponding points $\mathbf{p}^s$ and $\mathbf{q}^s$ on the texture domain $\mathcal{T}_1$ are given by t. The affine transform is defined using a combination of scaling, translation and rotations. The function t is invertible since the scaling used to compute the affine transforms are non-zero. The distances are preserved under translation and rotation, as the corresponding matrices used to define the affine transformation do not change the distance between adjacent samples. Only the scale and shear change the distance between four adjacent samples and we derive error bounds under shear and scaling.

We assume the mapping $t(O_1)$ from initial position of O_1 to the texture atlas $\mathcal{T}_1$ has unit scale and shear, and the spacing between two adjacent samples along each axis in $\mathcal{T}_1$ is δ. We provide a function $f(\delta)$ which bounds the distance between two adjacent samples in O_1.

In the initial position of O_1, since $s_x = 1, s_y = 1, sh = 0$, the spacing between two samples is bounded by $f(\delta) \leq \sqrt{2}\delta$.

Let the maximum motion of a vertex in 3D, modulo any rigid body transformations, space be bounded by d_m. This gives a bound on the maximum deformation of a face on O_1.

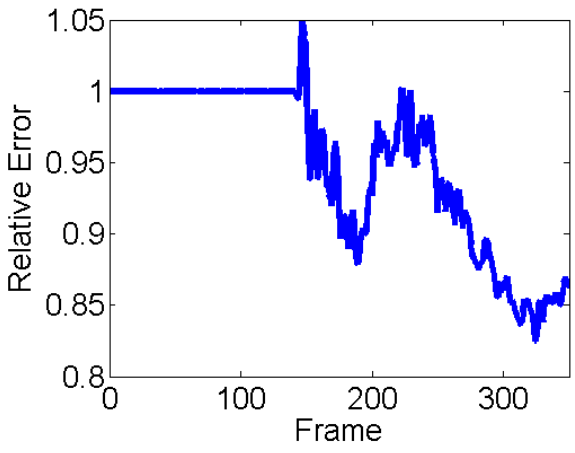

Figure 6: Relative error in distance map computation for a deformable model: *The relative error measures the ratio of maximum distance between adjacent samples on the distance map for all the triangles in a frame to the maximum distance between adjacent samples measured at the beginning of the simulation. The graph highlights the relative error on a deformable simulation using a resolution of* 512×512.

An upper bound on the scaling is given by $(s_x^2+s_y^2) \leq 2d_m$. Maintaining the sample spacing in $\mathcal{T}_1$ turns out to be δ^s, the maximum distance between two adjacent samples in O_1 is bounded by $f(\delta) \leq \sqrt{(s_x\delta)^2 + (s_y\delta)^2} \leq 2d_m\delta$.

We now show that the distance between two adjacent samples changes when the shearing exceeds a threshold, and derive the bounds. Consider a rectangular face in 2D with width b along X, and height h. Let the shear along Y be s_h. Assuming that the motion only produces shear (see Fig. 4),

$$s_h = \frac{2d_m}{h} \tag{1}$$

Distance between two adjacent samples increases by more than $\sqrt{2}\delta$ only if the first sample in row $y+\delta$ moves past the last sample in row y, $s_h\delta > b + \delta$. Replacing from equation (1) we get

$$d_m > \frac{bh}{2\delta} + 1.$$

Thus, if $d_m \leq \frac{bh}{2\delta}+1$, then there is no additional error due to shear. If $d_m > \frac{bh}{2\delta}+1$, then the effective increase in spacing along X axis between two rows of adjacent samples is $d_x = \max(s_h - (\frac{b}{\delta}+1), 0)$. In presence of scaling, the spacing along each axis is replaced by $s_x\delta$ and $s_y\delta$ respectively. We make the simplifying assumption that $s_x = s_y$. Then the increase in spacing along X is given by $(s_x + d_x)$, where $d_x = \max(s_h - (\frac{b}{s_x\delta}+1), 0)$, and the total error bound in the distance is $f(\delta) \leq \delta\sqrt{(s_x+d_x)^2 + s_y^2}$.

6 Implementation and Performance

In this section we describe the implementation of our algorithm to compute surface distance maps and its application to proximity queries 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 2.4Ghz Opteron 280 CPU, 2GB of memory and an NVIDIA 7800 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 for implementing 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 $1K \times 1k$) surface distance map of objects with tens of thousands of polygons in fraction of a second. We highlight the performance of our algorithm on scenes with varying polygon counts is highlighted in the graph. 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 $5-10$ times faster.

6.2 Proximity Queries

We use our algorithm to compute proximity information among 3D deformable models. This includes separation distance, collision detection, penetration depth and contact normal computation [HZLM01]. We first localize the region of overlap between two objects O_1 and O_2, and compute the surface distance map for all triangles of each object that lie inside the localized region. The separation distance between two objects is computed using minimum Euclidean distance from points on one object to points on the other object. We read back the surface distance maps of O_1 and O_2, and scan the pixels to determine the minimum distance. Collision detection is performed by checking for pixels with zero distance. In order to compute local penetration depth, we assign a sign to the distance values based on the orientation of the surface. In particular, all points of O_2 that are inside O_1 are assigned negative distance values. We then compute the maximum of these values to approximate the local penetration depth.

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

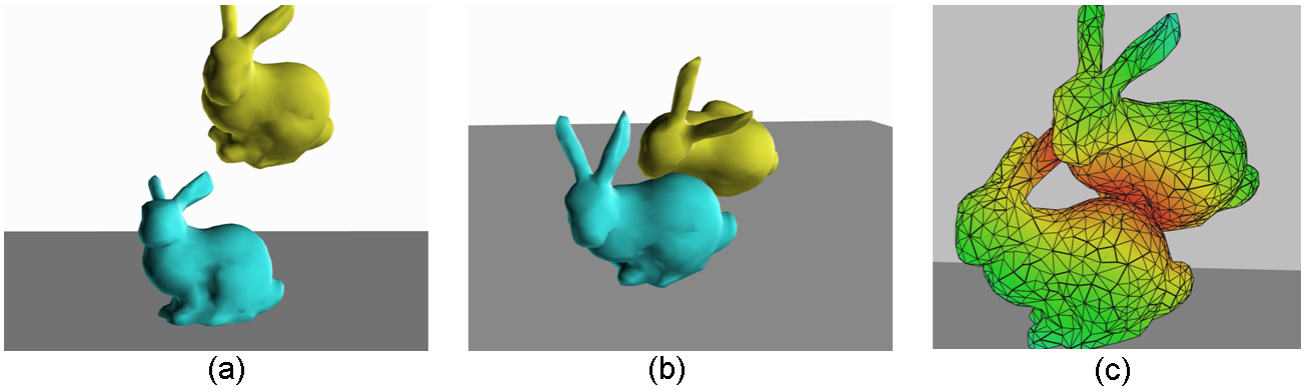

Figure 7: Proximity computation on deformable models: *Dynamic simulation of two deforming bunnies, each with* $2K$ *triangles . (a)-(b) Two frames from the simulation. (c) The surface distance map of both the bunnies that shows the distance field on the boundary. The distance increases from red to green. Our algorithm can compute the global distance maps and proximity queries in* $300 - 320$*ms at a resolution of* 512×512*.*

We used our algorithm for proximity query on 2 scenarios consisting of deforming objects. The first is a sequence of two deforming alphabets as shown in figure 3. The alphabet 'E' consists of 2.1K polygons, while the object 'G' consists of 1.6K polygons. At each frame, we compute a surface distance map at a resolution 512×512. The average time to perform all proximity queries is 110ms. As compared to [HZLM01, SGGM06], our surface distance algorithm results in speedup of 8 times. All these GPU-based algorithms are image-space algorithms. 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. We also perform the proximity computation on a sequence of two deforming bunnies. Each bunny consists of 2K polygons. At each frame, we compute a surface distance map at a resolution 512×512. The distance map computation and proximity queries take about $300 - 320$ ms per frame.

6.3 Comparison

In this section, 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 DiFi [SOM04], linear factorization [SGGM06] and efficient GPU implementations of CSC algorithm [SPG03, PS05]. All the prior approaches com pute 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 uniform grid. Let the number of cells in the grid $n \times n \times n$, and storage overhead is $O(n^3)$. Then the error of the distance field is $\frac{\sqrt{3}}{2m}$. In comparison, for a surface distance map of size $n \times n$, the storage cost is $O(n^2)$, and the error in the distance field is $\frac{\sqrt{2}}{2n}$ 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 is small, and the error in the distance field is $O(\frac{1}{m})$. As a result, our approach results in higher resolution distance fields. Current GPUs have $512MB$ or $1GB$ of video memory. It may not even be possible to store a volumetric distance at a very high memory (e.g. $(1K)^3$) on current GPUs, as it would require 8GB of memory. Furthermore, the cost of reading back a 3D distance field of $(1K)^3$ and scanning is rather high, i.e. about 16 seconds using a readback bandwidth of 500MB/sec. On the other hand, we restrict the distance field computation to the surface of a mesh and can compute a high resolution mesh at interactive rates.

Let there be m sites in each object. Then the computation cost to compute the global distance field using a volumetric approach varies between $O(mn^3)$ and $O(n^3)$. For narrow bands, the cost is $O(m + n_1)$ where n_1 is the number of pixels near the surface. On the other hand, the rasterization cost of computing the global surface distance map on the GPU varies between $O(mn^2)$ and $O(n^2)$. For narrow bands, the cost is close to $O(n^2)$ - as all n^2 pixels lie on the surface.

A quantitative comparison of average time to compute the distance fields on deformable models is shown in figure 8.

7 Limitations

Our approach has many 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

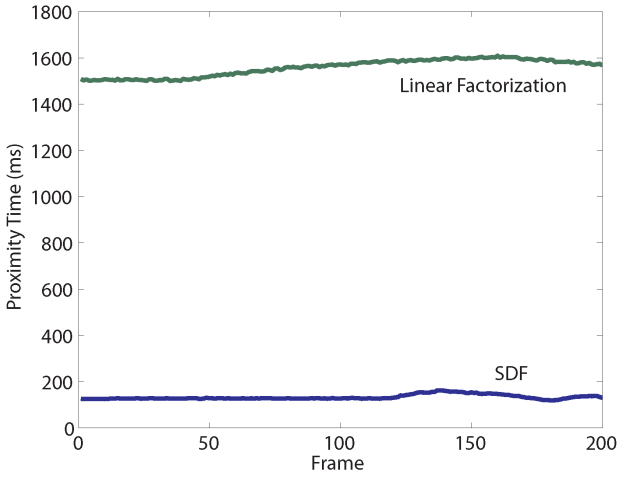

Figure 8: Timing comparison between our algorithm and a GPU-based volumetric distance field algorithm [SGGM06] labeled as **SDF** and **Linear Factorization** respectively: *Our algorithm is able to achieve 5–10 times speedup in proximity computation between two deforming alphabets. The scene is composed of 3.7K polygons. The surface distance field is computed at a resolution of* 512×512 *and the volumetric distance field is computed at* $180 \times 150 \times 256$*. 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.*

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. One possibility is to combine our algorithm with conservative distance bounds[SGG*06] and perform the proximity computations at object-space precision in a conservative manner. Furthermore, our current algorithm can only perform proximity computations between a pair of objects, and does not perform N-body computations.

8 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 mapping 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 transformations. 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 [SKALP05].

References

[Aur91] AURENHAMMER F.: Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv. 23*, 3 (Sept. 1991), 345–405.

[BGKW95] BREU H., GIL J., KIRKPATRICK D., WERMAN M.: Linear time Euclidean distance transform and Voronoi diagram algorithms. *IEEE Trans. Pattern Anal. Mach. Intell. 17* (1995), 529–533.

[BKS01] BITTER I., KAUFMANN A., SATO M.: penalized-distance volumetric skeleton algorithm. *IEEE Trans. on Visualization and Computer Graphics 7*, 3 (2001).

[Bli78] BLINN J. F.: Simulation of wrinkled surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)* (1978), pp. 286–292.

[Cat74] CATMULL E.: *A subdivision algorithm for computer display of curved surfaces*. PhD thesis, University of Utah, 1974.

[Coo84] COOK R. L.: Shade trees. In *Computer Graphics (SIGGRAPH '84 Proceedings)* (July 1984), Christiansen H., (Ed.), vol. 18, pp. 223–231.

[Cui99] CUISENAIRE O.: *Distance Transformations: Fast Algorithms and Applications to Medical Image Processing*. PhD thesis, Universite Catholique de Louvain, 1999.

[Dan80] DANIELSSON P. E.: Euclidean distance mapping. *Computer Graphics and Image Processing 14* (1980), 227–248.

[Den03] DENNY M.: Solving geometric optimization problems using graphics hardware. *Computer Graphics Forum 22*, 3 (2003).

[ER02] ETZION M., RAPPOPORT A.: Computing Voronoi skeletons of a 3-d polyhedron by space subdivision. *Computational Geometry: Theory and Applications 21*, 3 (March 2002), 87–120.

[FG05] FISCHER I., GOTSMAN C.: *Fast Approximation of High Order Voronoi Diagrams and Distance Transforms on the GPU*. Technical report CS TR-07-05, Harvard University, 2005.

[FH05] FLOATER M., HORMANN K.: *Surface parameterization:: A tutorial and survey*. Tech. rep., 2005.

[Fou92] FOURNIER A.: Normal distribution functions and multiple surfaces. In *Graphics Interface '92 Workshop on Local Illumination* (May 1992), pp. 45–52.

[FPRJ00] FRISKEN S., PERRY R., ROCKWOOD A., JONES R.: Adaptively sampled distance fields: A general representation of shapes for computer graphics. In *Proc. of ACM SIGGRAPH* (2000), pp. 249–254.

[GF03] GOMES J., FAUGERAS O.: The vector distance functions. *Int. Journal of Computer Vision 52*, 2 (2003), 161–187.

[Gib98] GIBSON S.: Using distance maps for smooth representation in sampled volumes. In *Proc. of IEEE Volume Visualization Symposium* (1998), pp. 23–30.

[HCK*99] HOFF III K. E., CULVER T., KEYSER J., LIN M., MANOCHA D.: Fast computation of generalized Voronoi diagrams using graphics hardware. In *Computer Graphics Annual Conference Series (SIGGRAPH '99)* (1999), pp. 277–286.

[HZLM01] HOFF K., ZAFERAKIS A., LIN M., MANOCHA D.: Fast and simple 2d geometric proximity queries using graphics hardware. *Proc. of ACM Symposium on Interactive 3D Graphics* (2001), 145–148.

[KBSS01] KOBBELT L., BOTSCH M., SCHWANECKE U., SEIDEL H. P.: Feature-sensitive surface extraction from volume data. In *Proc. of ACM SIGGRAPH* (2001), pp. 57–66.

[KSFC92] KLEIN A., SLOAN P. J., FINKELSTEIN A., COHEN. M.: Stylized video cubes. *Symposium on Computer Animation* (1992).

[Lin93] LIN M.: *Efficient Collision Detection for Animation and Robotics*. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, December 1993.

[LKHW03] LEFOHN A., KNISS J., HANSES C., WHITAKER R.: Interactive deformation and visualization of level set surfaces using graphics hardware. In *Proceedings of IEEE Visualization* (2003), p. To Appear.

[Mau03] MAUCH S.: *Efficient Algorithms for Solving Static Hamilton-Jacobi Equations*. PhD thesis, Californa Institute of Technology, 4 2003.

[MQR03] MAURER C., QI R., RAGHAVAN V.: A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitary dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence 25*, 2 (February 2003), 265–270.

[PF01] PERRY R., FRISKEN S.: Kizamu: A system for sculpting digital characters. In *Proc. of ACM SIGGRAPH* (2001), pp. 47–56.

[PS05] PEIKERT R., SIGG C.: Optimized bounding polyhedra for gpu-based distance transform. In *Scientific Visualization: The visual extraction of knowledge from data* (2005).

[Set99] SETHIAN J. A.: *Level set methods and fast marching methods*. Cambridge, 1999.

[SGG*06] SUD A., GOVINDARAJU N., GAYLE R., KABUL I., MANOCHA D.: Fast proximity computation among deformable models using discrete voronoi diagrams. *ACM Transactions on Graphics (SIGGRAPH 2006 Proceedings)* (2006).

[SGGM06] SUD A., GOVINDARAJU N., GAYLE R., MANOCHA D.: Interactive 3d distance field computation using linear factorization. In *Proc. ACM Symposium on Interactive 3D Graphics and Games* (2006), pp. 117–124.

[SKALP05] SZIRMAY-KALOS L., ASZODI B., LAZANYI I., PREMECZ M.: Approximate ray-tracing on the gpu with distance impostor. *Proc. of Eurographics* (2005).

[SOM04] SUD A., OTADUY M. A., MANOCHA D.: DiFi: Fast 3D distance field computation using graphics hardware. *Computer Graphics Forum (Proc. Eurographics) 23*, 3 (2004), 557–566.

[SPG03] SIGG C., PEIKERT R., GROSS M.: Signed distance transform using graphics hardware. In *Proceedings of IEEE Visualization* (2003), pp. 83–90.

[TT97] TEICHMANN M., TELLER S.: *Polygonal Approximation of Voronoi Diagrams of a Set of Triangles in Three Dimensions*. Tech. Rep. 766, Laboratory of Computer Science, MIT, 1997.

[VO98] VLEUGELS J., OVERMARS M. H.: Approximating Voronoi diagrams of convex sites in any dimension. *International Journal of Computational Geometry and Applications 8* (1998), 201–222.

[WND97] WOO M., NEIDER J., DAVIS T.: *OpenGL Programming Guide, Second Edition*. Addison Wesley, 1997.