Optimized Bounding Polyhedra For GPU-Based Distance Transform

Ronald Peikert and Christian Sigg

Swiss Federal Institute of Technology, CH-8092 Zürich, Switzerland
peikert@inf.ethz.ch, sigg@inf.ethz.ch

Many problems in areas such as computer graphics, scientific visualization, computational geometry, or image processing require the computation of a distance field. The distance field indicates at each point in space the shortest distance to a given object. Depending on the problem setting, the object is described either by a voxel attribute within a volume data set or by a surface representation such as a triangle mesh. The two cases require separate approaches, and only the case of the triangle mesh is studied in this paper. Often, the distance field is needed as a regular grid of samples. The samples can be computed either in image space or object space, referring to the outer loop of the algorithm, which iterates over all samples or all triangles of the mesh, respectively. Object space methods can be competitive, especially for higher resolutions. An ideal object space method would compute a generalized Voronoi diagram (GVD) of the mesh and then scan convert its cells. At each sample location, the distance to the Voronoi site associated with the cell would yield the field value. A practical method however, avoids the expensive GVD computation and instead works with bounding polyhedra for the Voronoi cells. In this paper, we propose a new type of bounding polyhedra. This reduces the number of polyhedra and simplifies their geometry. The choice of these bounding polyhedra pays off especially if scan conversion is run on graphics hardware.

1 Introduction

For any set S of points in $\mathbb{R}^n$, the distance field u is a unique scalar function defined in $\mathbb{R}^n$. At each point, u equals the distance to the closest point on S. If S is a closed and orientable manifold of dimension $n-1$, the space is divided into inner and outer parts. Therefore, a signed distance field can be defined. A positive sign is chosen outside the surface and a negative sign inside. Thus, the gradient of the distance field on the surface is equivalent to the surface normal.

The type of distance metric which is chosen depends on the application. Common choices are chessboard, chamfer and Euclidean distance [RP68]. We will restrict ourselves to the Euclidean distance, which is probably the most meaningful, but it is also the most expensive to compute.

The signed distance field u is the solution to the Eikonal equation $|\nabla u| = 1$ with boundary condition $u|_S = 0$. The boundary condition shows that the definition of S as a subset of $\mathbb{R}^n$ and the signed distance function are equivalent descriptions. The manifold corresponds to the zero-set of the signed distance function: $S = \{x|u(x) = 0\}$. Therefore, the signed distance transform converts an explicit surface representation to an implicit one.

Signed or unsigned distance fields have many applications in computer graphics, scientific visualization and related areas, such as implicit surface representation [FPRJ00] [Gib98], object metamorphosis [CSL98], collision detection and robotics [FPRJ00], skeletonization [BKS01] [WDK01], accelerated volume raytracing [SK00], camera path planning and image registration [Cui99]. Depending on the application, the distance field is required on a full pixel or voxel grid or only within a band of width d around the objects.

2 Related Work

The problem of computing a 3D Euclidean distance transform exists in two varieties, distinguished by the type of object representation. The object can either be given as data on a voxel grid or in vector representation. The latter is typically a triangle mesh in the case where the object is a surface. Both problems have been studied extensively and fast methods have been developed for both of them. It is reasonable to treat the two problems separately. If the goal is to sample the exact distance to a triangle mesh, the problem cannot be stated in voxel space. Likewise, there is usually no advantage to transform the problem from voxel representation to vector representation. For triangle meshes, time complexity must depend on the number M of surface primitives (faces, edges, and vertices). Therefore, algorithms for the two different problem settings cannot be directly compared.

A method [MQR03] which has been recently presented, finds the distance transform in voxel data in $O(N)$ time, where N is the number of voxels. In the same paper, a good overview of earlier methods is given. Essentially, methods fall into two categories, propagation methods and methods based on Voronoi diagrams.

In propagation methods, the distance information is carried over to neighbor voxels, either by sweeping in all grid dimensions, or by propagating a contour. A well-known example of the latter is the Fast Marching Method (FMM) [Set96], an upwind scheme which can solve the Eikonal Equation $|\nabla u| = 1/f$ in a single iteration and in $O(N \log N)$ operations. A signed distance field is obtained by using a constant propagation function f. However, due to the finite difference scheme, FMM is not an exact method.

Besides the distance, additional information can be stored in the distance field. Such information can be the vector pointing to the nearest object point, known as the vector distance transform [Mul92]. Alternatively, the index of the nearest surface primitive can be attributed to each point, the resulting field is called a complete distance field representation [HLC+01]. By propagating this type of additional information, FMM and similar propagation methods can be turned into exact distance transform algorithms [BMW00] [Egg98] [Tsa00].

For the second type of problem setting where the distance field of a polyline or triangle mesh is sought, a brute force algorithm would compute the distance of each grid point to each primitive. If the triangle mesh consists of a large number of triangles and the sampling grid is large, this approach is impractical. For an efficient algorithm, one needs to reduce the number of distances calculated per grid point or alternatively, per primitive.

To achieve this goal, a spatial data structure such as a BSP tree can be used for storing the primitives. When computing the distance field value for a given sample, a primitive can be excluded from the calculation if it is known that a closer primitive exists. By using this data structure, one can quickly find the closest primitive to a point: While the tree is scanned for the closest primitive, one can give an upper limit of the final distance. At the same time, a lower bound of the distance can be computed for any subtree. If the lower bound of a subtree is larger than the current upper bound of the final distance, the subtree can be excluded from the search. This leads to an algorithm logarithmic in the number of primitives of the input mesh.

An alternative to such an image space approach are object space methods, i.e. methods based on scan conversion. Here, the distance field is obtained by scan converting a number of polyhedra related to the triangle mesh and by conditionally overwriting the computed voxel values. The advantage of object space methods is their sub-pixel accuracy. However, it is obvious that the relative performance degrades if the average size of the polyhedra shrinks to the size of a single voxel. It has been shown that for distance fields of triangle meshes, methods based on scan conversion are competitive.

Optimally, only distances to grid points contained in the Voronoi cell of the corresponding primitive are calculated. If the primitives are not restricted to points, but include edges and triangles, a generalized Voronoi diagramm (GVD) is required. Once a GVD is computed, the distance field can easily be caluclated as the distance to the respective site. However, the computation of Voronoi diagrams is not easier than the computation of distance fields. The time for generating a diagram with M sites is $O(M \log M)$. Generalized Voronoi diagrams can be computed on graphics hardware [HCK+99] by rendering local distance fields as $n+1$-dimensional function graphs and using the z-buffer for minimization. A disadvantage of this method is that it requires accurate rendering of curved surfaces, requiring tesselations in the order of 100 triangles for a cone. The 3D version even requires doubly curved surfaces which strongly limits the number of primitives that can be handled.

Nevertheless, if a point is known to lie outside of a Voronoi cell, the distance to its base primitive does not need to be calculated. This led to the idea of using polyhedra bounding a Voronoi cell instead of the Voronoi cells themselves. The first such algorithm was presented by Mauch [Mau03]. It will be shortly explained in Section 3.3. It is possible to run the scan conversion part of that algorithm on graphics hardware. Although the scan conversion runs faster on the graphics hardware, the overall speed up gained is minimal because of the large amount of geometry that has to be send to the graphics card. In Section 3.4, we present an optimized type of polyhedra [SPG03] giving significantly better performance on the graphics hardware.

3 Distance field methods for triangle meshes

For the rest of the paper, we focus our attention to the distance field computation for a triangle mesh. We shortly describe the more theoretical method based on scan conversion of the generalized Voronoi cells. We also outline Mauch's method of using bounding polyhedra. In Section 3.4, we describe how the method can be made more suited to be run on a programmable graphics card by using an optimized type of bounding polyhedron.

3.1 Vertex classification

The vertices of a closed and oriented triangle mesh can be classified into convex, concave and saddle vertices, depending on their incident edges. If all of them are convex (concave), the vertex is convex (concave), if both types occur, it is a saddle. Because convex edges become concave and vice-versa when we flip the orientation of the surface, we only distinguish convex/concave vertices and saddle vertices.

Besides the topological consistency, we assume also a geometric regularity requirement for mesh: At saddle points, all incident faces must keep their orientation when viewed from the normal direction. The normal direction in a vertex is defined simply by averaging all incident face normals. Failure of this assumption would indicate a poor triangulation. It can be fixed by subdividing triangles.

3.2 Voronoi diagrams of triangle meshes

The Voronoi diagram of a finite set of points is a partitioning of space into cells. Each site (i.e. point) is associated with a cell containing all points for which this site is the nearest one. Points on cell boundaries have more than one nearest site.

A straightforward extension is to allow sites to be manifolds than just points, leading to generalized Voronoi diagrams. For our purpose, sites will

be restricted to the points, edges and faces of a closed and oriented triangle mesh.

If the GVD of such a mesh was known, it would be a simple task to compute the distance field. For a given sample point, one would first identify the cell in which it is contained and then calculate the distance to the associated site. If a full grid of samples is needed, one would use an object-space approach, i.e. loop over the cells, which is known as scan conversion.

In computer graphics, scan conversion is a key operation in the rendering pipeline and is efficiently performed by standard graphics cards. By reading back the frame buffer data, the computing power of graphics cards becomes available for more purposes than just rendering. In recent years, the programmability of graphics cards made it possible to adapt the scan conversion operation. In particular, nonlinear interpolation functions can be programmed.

3.3 GVD-based bounding polyhedra

To actually compute a GVD is not only expensive, it can also produce arbitrarily complex polyhedra. Therefore, Mauch [Mau03] replaced the cells by bounding volumes. As bounding volumes he used polyhedra which are possibly larger but of simpler geometric shape than the cells. The distance field can again be calculated by looping over the polyhedra. To correctly treat regions where two or more polyhedra overlap it is sufficient to take the minimum of all computed values.

For reasons of efficiency, only local information is used for constructing bounding polyhedra. This requires the introduction of a maximal distance d up to which the distance field is computed on either side of the surface. The following types of bounding polyhedra are used, depending on the type of site

- three-sided orthogonal prism for faces ("tower" of height $2d$ extruded from the triangle in both directions),
- three-sided orthogonal prism for edges ("wedge" of height d, filling the space between towers)
- n-sided pyramid for convex/concave vertices of degree n (of height d, filling the space left by towers and wedges, see Fig. 1).

The case of the saddle vertex is not mentioned in [Mau03]. However, a possible solution would be to construct an n-sided pyramid in the same way as for a convex/concave vertex, but on both sides of the surface, and then taking the convex hull of each pyramid.

3.4 Optimized bounding polyhedra

While these bounding polyhedra work well for scan conversion done purely in software, the large number of polyhedra is not ideal for a hardware-based scan conversion method. The reason is that the overhead per polyhedron is

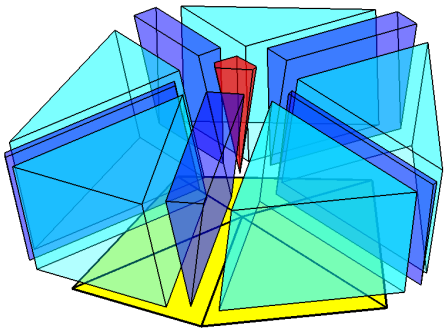

Fig. 1. Polyhedra constructed on one side of a (yellow) one-ring of the mesh: (cyan) towers, (blue) wedges, and a (red) cone. The polyhedra are moved away from the surface for better visibility.

larger for the hardware-based method because the geometry data has to be sent to the graphics card.

In order to reduce the number of polyhedra, our approach uses a different type of bounding polyhedra for the faces, such that their union completely covers space. This eliminates the need for wedges and pyramids. The price to pay is a slightly more complicated distance field computation: Each polyhedron no more represents a single site, but seven sites, namely a face, its three boundary edges, and its three vertices. In principle, the minimum of the distances to the seven sites must be calculated. However, we showed in [SPG03] that this can be done quite efficiently, requiring little more operations than for a single distance calculation.

Now, in order to construct the new bounding polyhedra, we must divide wedges and pyramids among the neighbor polyhedra. Two observations can be made:

- Any way of splitting a wedge is allowed because the Voronoi site which it represents is also represented by the new polyhedra.
- On the convex side of the surface, i.e. opposite the wedge, the original towers overlap. This overlap can be eliminated by using the bisector plane of the dihedral angle as a divider. This is exactly where the two Voronoi cells meet.

Taking both observations into account, we can now use the angle bisector planes as the three lateral boundaries of the new polyhedron. Adding two planes parallel to the face at distances $\pm d$ (the limiting distance used for the distance field computation), a three-sided pyramid frustum is obtained (see Fig. 2). This bounding polyhedron has the advantage of having a single topological type and only five faces, all of them planar.

While the bounding polyhedra match along the edges of the mesh, this is not true near the mesh vertices in general. Near mesh vertices, the polyhedra

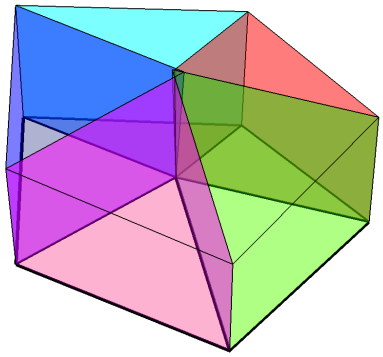

Fig. 2. Optimized bounding polyhedra for a one-ring of the mesh. The polyhedra extend to the other side of the surface, too, which is not shown in this figure. Some non-adajacent pairs of polyhedra are seen to overlap.

can overlap. This is not a problem, it just leads to repeated distance calculations. But the polyhedra can also leave a gap. An example is shown in Fig. 3. In such cases, the gap must be closed by making some of the polyhedra slightly larger. We show in the appendix, that gaps can occur only for saddle

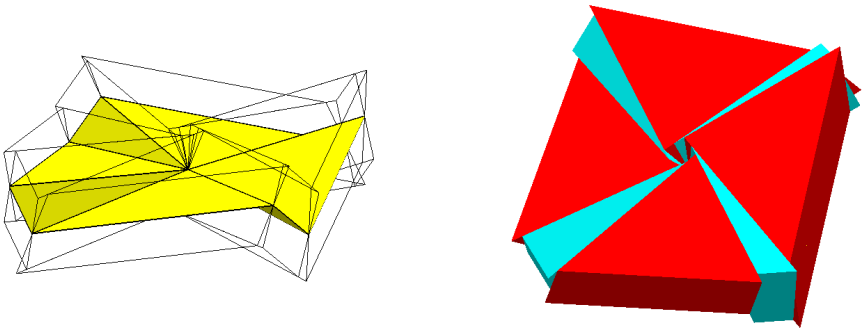

Fig. 3. Example of a saddle vertex with eight incident triangles. Bounding polyhedra are outlined (left) and filled (right). A gap in the shape of an eight-sided double-pyramid is visible in the center.

In order to study the situation near a mesh vertex, we introduce a few notations, see Fig. 4. Let c denote the vertex, $x_0, \cdots, x_{n-1}$ its neighbor vertices, $F_i =< c, x_i, x_{i+1} >$ the incident faces (all indices are meant modulo n), and P_i the polyhedron constructed for the face F_i. That means that P_{i-1} and P_i are separated by the angle bisector plane of F_{i-1} and F_i which we denote by A_i. We denote by F the union of the F_i and by P the union of the P_i (for $i = 0, \cdots, n-1$).

If P completely covers a neighborhood of c, this means that any test point y near c is contained in at least one of the P_i. The point y is contained in P_i if

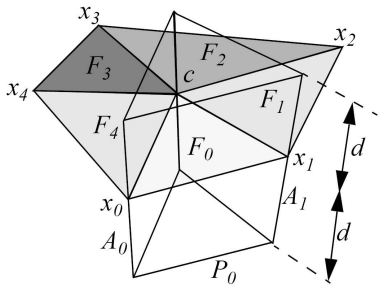

Fig. 4. A vertex c with neighbor vertices x_i, faces F_i, angle bisector planes A_i, and polyhedra P_i.

it lies on the right hand side of A_i and on the left hand side of A_{i+1}. We also observe that in the reverse case (i.e. left of A_i and right of A_{i+1}), the antipodal point $2c-y$ is contained in P_i. Because in the cycle $A_0, A_1, \cdots, A_n = A_0$ there are as many left-right transitions as right left transition, it follows, perhaps surprisingly, that the covering is point-symmetric w.r.t. the center c. The point symmetry holds for the multiplicity of the covering, not for each single polyhedron.

Therefore, if we can verify, that y lies neither on the left hand side of all A_i nor on the right hand side of all A_i, it follows that both y and its antipodal point are covered by P. For the practical test, it is sufficient to use one point on each intersection line $A_i \cap A_{i+1}$. Each test point must lie on the left of at least one A_j with $j \neq i$ and on the right of at least one such. Points lying exactly on a plane should pass the test, too. Also, it has to be noted that full planes can be used for the test, thus there is no need to bother with half-planes.

If the test fails for some of the test points, this means that the corresponding polyhedra must be made larger to avoid a gap. A possible way to do this is to take the centroid of the test points. Polyhedra must be enlarged just as much that they contain this centroid and its antipodal point. We want the polyhedra to remain pyramid frusta, therefore we restrict the modifications to parallel shifts of edges.

4 Results

In Section 4.1, the amount of vertices where the bounding polyhedra leave holes is analysed. In Section 4.2, the performance results of out algorithm are presented.

4.1 Saddle vertices and gaps

By looking at a few typical triangle meshes, it can be noticed that there are often more saddle vertices than convex/concave vertices. This can be caused

by the geometry itself, but also by the triangulation. Especially, if a quadrilateral mesh is subdivided to atriangle mesh, the diagonals can turn a convex vertex into a saddle vertex. This is why the torus mesh has more than the expected 50 % of saddle vertices.

Fig. 5. Datasets used for experiment.

As mentioned, saddle vertices can lead to gaps between the bounding polyhedra and the extra effort to fill them. However, our experiment showed that gaps occur only for some of the saddle vertices. Depending on the mesh characteristics, the percentage of saddle vertices leading to gaps can be quite small (see Table 1).

Table 1. Number of vertices, saddle vertices and vertices with incomplete covering by the unmodified bounding polyhedra.

mesh	vertices	saddles	gaps
sphere6	16386	0	0
torus	3000	1788	0
knot	1440	1378	674
seashell	915	843	148
bunny	34834	30561	516

4.2 Performance of the hardware-based distance transform

The way to make use of the GPU's computing power is to work on layers of sample locations. Each layer defines a slicing plane which is intersected with the bounding polyhedra. A list of active edges is used to avoid empty intersections. The resulting slices are sent to the GPU for scan conversion, where a fragment program is used to compute the local distance field. The local distance field is the distance field of only the seven sites (one triangle, three edges, three vertices) associated with the bounding polyhedron. The final distance field is obtained at each fragment by taking the minimum of the computed local distance field values. An outline of the fragment program can be found in [SPG03].

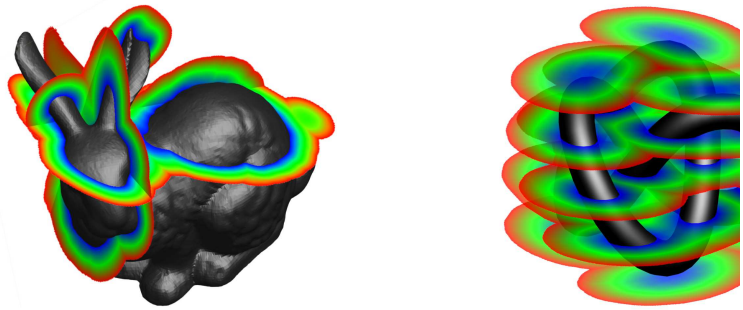

Fig. 6. Slices of the distance fields of bunny and knot data set, computed by the HW-based algorithm.

As a basis for comparison of performance, we used the software scan conversion algorithm, which we downloaded from the URL [Mau00]. We then re-implemented this algorithm such that the scanconversion part was done on the GPU. The machine at our disposition was a 2.4 GHz Pentium 4 equipped with 2 GB of RAM and an ATI Radeon 9700 PRO graphics card. It turned out only a negligible speedup could be obtained by this hardware-based program. In addition, the range of parameters (resolution, width of computational band) where a speedup could be measured, was rather narrow. This performance problem could be tracked down to the overhead caused by rendering too many small polygons.

When using our optimized bounding polyhedra, the speedup delivered on the same machine was significant for a wide range of resolutions and widths. When choosing a band of 10 % of the model extent and a resolution of 256^3 samples, we measured an average speedup close to 5 for the sphere6, knot and bunny models. For higher resolution as well as for wider bands, the speedup improved. But also for extremely low sample grid resolutions, the hardware-assisted program performed well. For instance, in the case of a mesh with 131072 triangles of average area less then 2 on the voxel scale, we measured a speedup of 3.30. However, it is obvious that the the scan conversion approach, with or without hardware support, is no more an efficient strategy if sampling density is further decreased. For such problems, an image space method combined with a spatial data hierarchy would obviously be more adequate.

The advantage of the scan conversion approach also degrades when the narrow band is large in comparison to the volume the surface encloses. Because the bounding polyhedra for one triangle is computed using its neighboring triangles only, the bounding volumes tend to overlap on the convex side of the surface. The amount of overlap grows superlinearly with the thickness of the narrow band. In order to compute the distance transform in a dense volume around the surface, the fastest solution would be a combination the CSC and the FMM approach. While the CSC algorithm is faster in computing the distance in the narrow band, the FMM algorithm can then be used to compute the distance in regions further away from the surface.

5 Conclusion

We have shown that today's graphics hardware is suitable for supporting the signed distance field computation. A GPU implementation has a larger overhead per polyhedron while sampling the distance field using scan conversion is faster. By reducing the amount of polyhedra to approximatley one third, we were able to get a significant speed up in comparison to the CPU implementation. It was proven that the polyhedra cover the area around triangles, edges and convex or concave vertices up to a user definable distance. However, the polyhedra can leave a hole in special configurations at saddle vertices. These holes are filled by shifting the sides of the polyhedra outward until they cover the normal of the saddle vertex. This procedure increases the amount of overlap and therefore introduces a certain overhead. But it was shown that gaps don't appear very often for common meshes. For an implementation using graphics hardware, the speed up gained by the reduced amount of geometry outweighs the extra cost of additional distance samples.

Acknowledgment

This work was partially funded by Schlumberger Cambridge Research.

References

[BKS01] Ingmar Bitter, Arie E. Kaufman, and Mie Sato. Penalized-distance volumetric skeleton algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):195–206, 2001.

[BMW00] D. Breen, S. Mauch, and R. Whitaker. 3D scan conversion of csg models into distance, closest-point and colour volumes. In M. Chen, A.E. Kaufman, and R. Yagel, editors, *Volume Graphics*, pages 135–158, 2000.

[CSL98] Daniel Cohen-Or, Amira Solomovici, and Levin Levin. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 17(2):116–141, April 1998. ISSN 0730-0301.

[Cui99] Olivier Cuisenaire. *Distance Transformations: Fast Algorithms and Applications to Medical Image Processing.* PhD thesis, Université Catholique de Louvain, Louvain-La-Neuve, Belgium, January 1999.

[Egg98] Hinnik Eggers. Two fast Euclidean distance transformations in Z^2 based on sufficient propagation. *Computer Vision and Image Understanding: CVIU*, 69(1):106–116, January 1998.

[FPRJ00] Sarah F. Frisken, Ronald N. Perry, Alyn Rockwood, and Thouis R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In Kurt Akeley, editor, *Siggraph 2000 Proceedings*, pages 249–254. ACM SIGGRAPH, 2000.

[Gib98] Sarah F. F. Gibson. Using distance maps for accurate surface reconstruction in sampled volumes. In *Proceedings of the 1998 Symposium on Volume Visualization (VOLVIS-98)*, pages 23–30, New York, October 19–20 1998. ACM Press.

[HCK+99] Kenneth Hoff, Tim Culver, John Keyser, Ming Lin, and Dinesh Manocha. Fast computation of generalized voronoi diagrams using graphics hardware. In Alyn Rockwood, editor, *Siggraph 99 Proceedings*, pages 277–286, N.Y., August8–13 1999. ACM SIGGRAPH.

[HLC+01] Jian Huang, Yan Li, Roger Crawfis, S.C. Lu, and Shu Liou. A complete distance field representation. In Thomas Ertl, Ken Joy, and Amitabh Varshney, editors, *Proceedings Visualization 2001*, pages 247–254. IEEE Computer Society Technical Committee on Visualization and Graphics Executive Committee, 2001.

[Mau00] Sean Mauch. A fast algorithm for computing the closest point and distance transform, 2000. `http://www.acm.caltech.edu/~seanm/projects/cpt/cpt.html`.

[Mau03] Sean Mauch. *Efficient Algorithms for Solving Static Hamilton-Jacobi Equations*. PhD thesis, Caltech, Pasadena CA, April 2003.

[MQR03] Calvin R. Maurer, Jr., Rensheng Qi, and Vijay Raghavan. A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(2):265–270, 2003.

[Mul92] James C. Mullikin. The vector distance transform in two and three dimensions. *Computer Vision, Graphics, and Image Processing. Graphical Models and Image Processing*, 54(6):526–535, November 1992.

[RP68] A. Rosenfeld and J. L. Pfalz. Distance functions on digital pictures. *Pattern Recognition*, 1:33–61, 1968.

[Set96] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proc. Nat. Acad. Sci.*, 94:1591–1595, 1996.

[SK00] M. Sramek and A. Kaufman. Fast ray-tracing of rectilinear volume data using distance transforms. In Hans Hagen, editor, *IEEE Transactions on Visualization and Computer Graphics*, volume 6 (3), pages 236–252. IEEE Computer Society, 2000.

[SPG03] Christian Sigg, Ronald Peikert, and Markus Gross. Signed distance transform using graphics hardware. In R. Moorhead, G. Turk, and J. van Wijk, editors, *Proceedings of IEEE Visualization '03*, pages 83–90. IEEE Computer Society Press, October 2003.

[Tsa00] Yen-hsi Richard Tsai. Rapid and accurate computation of the distance function using grids. Technical report, Dept. of Mathematics, University of California, Los Angeles, 2000.

[WDK01] Ming Wan, Frank Dachille, and Arie Kaufman. Distance-field based skeletons for virtual navigation. In Thomas Ertl, Ken Joy, and Amitabh Varshney, editors, *Proceedings of the Conference on Visualization 2001 (VIS-01)*, pages 239–246, Piscataway, NJ, October 21–26 2001. IEEE Computer Society.

Appendix: Proof for complete covering around convex/concave vertices

Let c be a convex/concave vertex of a closed and oriented triangle mesh. Using the notation of Section 3.4, we want to show that a small neighborhood of c is completely covered by the union P of polyhedra. This can be seen best by taking intersections with a small sphere S centered at c (see Fig. 7). We use the overbar symbol to denote the intersection with S. It follows that $\bar{F}_i$ are great circles, and their union $\bar{F}$ is a convex spherical polygon. Also the $\bar{A}_i$ are great circles, which can be oriented consistently towards the interior of $\bar{F}$. Finally, $\bar{P}_i$ are spherical lunes, because we can assume that the diameter of S is smaller than all edges.

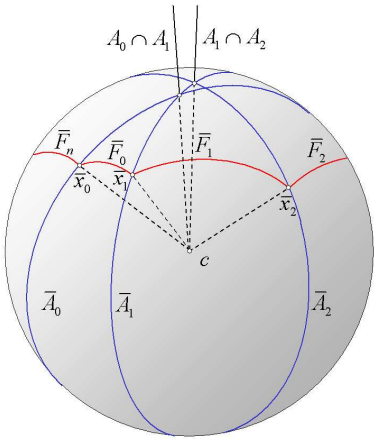

Fig. 7. Intersections of faces and angle bisector planes with the sphere.

The conjecture is now that S is completely covered by $\bar{P}$, the union of the lunes. By the argument given in Section 3.4, it is sufficient to show that the northern hemisphere is covered.

Let y be a test point on S and on the convex side of the surface, i.e. an interior point of $\bar{F}$. We choose coordinates in such a way that y is the north pole of the sphere. By connecting the vertices of the spherical polygon with the north pole, we get n spherical triangles which add up to a full 2π angle at the north pole. For the i-th triangle, let γ_i be the angle at the north pole, and α_i and β_i the angles to the meridians (see Fig. 8).

Let us now assume that the north pole lies on the left of all $\bar{A}_i$ which can be expressed as

$$\alpha_{i+1} > \beta_i \tag{1}$$

Convexity implies that

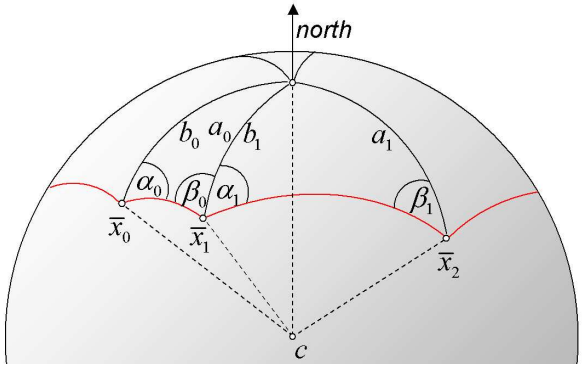

Fig. 8. Spherical triangles above mesh triangles.

$$\alpha_{i+1} + \beta_i \leq \pi \tag{2}$$

From Eq. 1, Eq. 2 and $0 < \alpha_{i+1}, \beta_i < \pi$ follows that

$$0 < \frac{\sin \beta_i}{\sin \alpha_{i+1}} < 1 \tag{3}$$

Taking the product yields

$$\prod_{i=0}^{n-1} \frac{\sin \beta_i}{\sin \alpha_i} = \prod_{i=0}^{n-1} \frac{\sin \beta_{i+1}}{\sin \alpha_i} < 1 \tag{4}$$

On the other hand, we can derive

$$\prod_{i=0}^{n-1} \frac{\sin \beta_i}{\sin \alpha_i} = \prod_{i=0}^{n-1} \frac{\sin b_i}{\sin a_i} = \prod_{i=0}^{n-1} \frac{\sin b_i}{\sin b_{i+1}} = 1 \tag{5}$$

making use of the spherical law of sines, the fact that $a_i = b_{i+1}$ because triangles fit together, and finally $b_n = b_0$.

From this contradiction follows that the test point is covered by $\bar{P}$, and so the interior of the spherical polygon $\bar{F}$.

Because of convexity, it is possible to choose an interior point of $\bar{F}$ as the north pole such that all of $\bar{F}$ lies in the northern hemisphere. It remains to show that $\bar{P}$ not only covers $\bar{F}$ but the whole hemisphere. Any spherical lune must have one of its end points below the equator, and because of convexity, this is the one on the concave side. But this means that along the equator, the sequence of lunes $\bar{P}_0, \cdots, \bar{P_{n-1}}, \bar{P}_0$ can't have any gaps, and therefore the hemisphere is completely covered, which was the conjecture.