Generating Signed Distance Fields From Triangle Meshes

IMM-TECHNICAL REPORT-2002-21

J. Andreas Bærentzen and Henrik Aanæs
Informatics and Mathematical Modelling (IMM)
Technical University of Denmark
DK-2800 Kongens Lyngby - Denmark

Abstract

A method for generating a discrete, signed 3D distance field is proposed. Distance fields are used in a number of contexts. In particular the popular level set method is usually initialized by a distance field. The main focus of our work is on simplifying the computation of the sign when generating signed distance fields. Previous methods largely rely on scan conversion to classify voxels as being inside or outside. In contrast, our method relies on defining a pseudo normal, the angle weighted normal, which is used to determine, for each grid point, whether that point is inside or outside. This leads to a method for generating signed distance fields which is a simple and straightforward extension of the method for generating unsigned distance fields. We prove that our choice of pseudo normal leads to a correct technique for computing the sign.

Keywords: Distance Field, Signed Distance Field, Mesh, Triangle Mesh, Level Set.

1 Introduction

The level set method proposed by Osher and Sethian [17, 20] is a technique for tracking deforming interfaces. It was proposed a little more than a decade ago, and it has become very popular in the intervening period. It is extremely adaptable and has found many uses in computer vision and computer graphics. Examples include stereo [8], image segmentation [15], and volume segmentation [24], statistical modelling [10], simulation of water [7], shape metamorphosis [2], and shape modelling [4].

The level set method operates on discrete 3D grids of scalar values. By interpolation, we can obtain a continuous scalar field, and the deforming interface is an isosurface of this scalar field. Almost invariably, the level set method is initialized by a 3D signed distance field of the shape. In other words, from the initial shape we generate a voxel grid where each voxel contains the shortest distance to the surface. The distance is positive if the voxel is outside and negative otherwise.

Unfortunately, when dealing with 3D surfaces, our starting point is often a triangle

mesh (from now on, we simply write *mesh*) and not a signed distance field. Therefore, we need a technique which converts meshes to the distance field representation.

Such methods do exist, but, in general, they involve a scan conversion step which classifies voxels as being either inside or outside. In the case of smooth surfaces such scan conversion is not necessary since we can use the surface normal at the closest point on a surface to determine whether a given point is inside or outside. However, the normal is discontinuous at the edges and vertices of a mesh. To overcome this problem, we introduce a pseudo normal for vertices and edges. The pseudo normal is simply the angle weighted normal which was proposed by Sequin [22] and again by Thürmer et al. [23].

The angle weighted normal at the closest feature is used to determine whether a voxel is inside or outside, if the closest feature is an edge or a vertex. In the case of faces, the face normal is used. In this paper, we prove that the dot product of the angle weighted normal and the vector from the closest point on the mesh to a given point is positive if the point is outside the mesh and negative if it is inside.

To use our method, pseudo normals must be stored for each edge and vertex of the mesh. On the other hand, no scan conversion is required, and the generation of a signed distance field is nearly identical to the generation of an unsigned distance field.

Finally, it should be mentioned that distance fields and in particular signed distance fields have many other applications than the level set method. In general, distance fields are also far more useful than, say, binary volumes as a representation for shapes [9, 5]. We can use signed distance fields to generate iso surfaces, for volumetric CSG [3], and for things like hypertexturing of shapes [19].

This paper is organized as follows; in Section 2 previous work is presented along side a discussion of how this work deviates from the previous work. In Section 3, the proposed method is presented in detail, including a fairly specific guide on implementation. The proposed method for computing the sign of the sign distance field is proven correct in Section 4. Note, that this section can be skipped on a first reading and is not essential for implementing our method. Lastly some experimental results are presented in Section 5 followed by a discussion of the presented work in Section 6.

2 Background

We use the letter M to denote a triangle mesh. A triangle mesh, M, is a union of triangles T_i where $i \in [1, N]$, N being the number of triangles. In other words

$$M = \bigcup_{i \in [1,N]} T_i.$$

We assume that M is a closed, orientable 2–manifold in 3D Euclidean space. This assumption is important because the sign of a distance field tells us whether the voxel is inside or outside, but this is only defined for closed, orientable manifolds. We will use the convention that the normal of a face points toward the exterior of the mesh.

In practice, we can impose the manifold condition by requiring that [12]

- The mesh does not contain any self intersections: Triangles may share only edges

and vertices and must be otherwise disjoint.

- Every edge must be adjacent to exactly two triangles.
- Triangles incident on a vertex must form a single cycle around that vertex.

If the mesh fulfills these conditions, we know that it partitions space into a well–defined interior, exterior, and the mesh itself.

2.1 Computing Unsigned Distance Fields

By *discrete distance field* we understand a 3D grid of points (i.e. a voxel grid) where each voxel contains a scalar whose value is the shortest distance to the mesh.

To generate a discrete distance field (or distance field for short), we need to compute the distance from every point in the grid to the mesh, where the distance from a point $\mathbf{p}$ to a triangle mesh M is defined as[1]

$$d(\mathbf{p}, M) = \inf_{\mathbf{x} \in M} \|\mathbf{p} - \mathbf{x}\|. \tag{1}$$

The point on the mesh which is closest to a given $\mathbf{p}$, we will denote the *closest point* for this $\mathbf{p}$, when it is not ambiguous. Observe that the mesh is simply the union of all its triangles $M = \bigcup_{i \in N} T_i$ where N is the number of triangles and T_i is triangle number i, so clearly

$$d(\mathbf{p}, M) = \inf_{\mathbf{x} \in M} \|\mathbf{p} - \mathbf{x}\| \tag{2}$$

$$= \inf_{i \in [1,N]} \left(\inf_{\mathbf{x} \in T_i} \|\mathbf{p} - \mathbf{x}\| \right) \quad . \tag{3}$$

In other words, we compute the distance from a point to M by computing the distance from that point to every triangle in M and picking the smallest distance to any triangle.

The next question is how to compute the distance from a point to a triangle. This is complicated by the need to know if the closest point to $\mathbf{p}$ is a vertex, lies on an edge, or lies on the face itself. This is resolved by a case analysis, and is discussed further in Section 3.1.

In an early paper on discrete distance fields, Payne and Toga [18] proposed the case analysis approach along with some optimizations. Perhaps the most important is to compute square distances and only take the square root, when the shortest distance is found. Another (obvious) optimization is to keep the transformation matrix used to project points into the plane of the triangle and use it for all points. They also propose to store the triangles in a tree of bounding boxes. From a given point, we can compute the smallest and greatest possible distance to anything inside the box. This can be used to prune branches as we move down the tree.

Mark Jones found that using the above method resulted in a fourfold increase in efficiency when compared to naïvely computing the distance in 3D [14]. Jones is only interested in voxels within a certain distance of the mesh and performs an initial scan conversion to find voxels that are adjacent to the mesh. Distances are only computed for

[1] Infimum is the greatest lower bound of a set, denoted $\inf$.

those voxels. Furthermore, the accurate distance to a given triangle is only computed if the voxel is within a given distance of the triangle plane.

Dachille and Kaufman proposed a method that is suitable for hardware implementation [6]. The idea is to perform the case analysis using only distance to plane computations. The advantage is that, using the proposed incremental approach, only one addition is needed for each voxel (within the bounding box of the polygons) to compute the distance from that voxel to a given plane.

Recently, André Guéziec [11] proposed a new method called the "Meshsweeper" algorithm. This algorithm is based on a hierarchy of simplifications of the mesh. The distance is first computed at a coarse level, and then we proceed to finer levels. The advantage is that at any level, it is possible to prune branches if they cannot contain the shortest distance.

Huang et al. proposed a new distance field representation called CDF (Complete Distance Fields) [13]. The central idea is to divide a volume into cells, and in each cell to store references to all triangles that might contain the point closest to any point inside the cell.

2.2 Sign Computation

Originally, Payne and Toga [18] suggested that the signs should be computed in a separate pass. Thus, we begin by classifying each voxel according to whether it is inside or outside the mesh. Voxels inside are given negative sign, voxels outside are given positive sign (or vice versa).

This classification can be done in a number of ways, Payne and Toga suggested that for each z–level plane in the grid, we compute the intersection of the mesh and the plane. This produces a 2D contour that we can scan convert. An even simpler approach would be to cast rays along rows of voxels. At voxel locations where the ray has crossed the mesh an uneven number of times, we know we are inside.

A completely different approach has been suggested by Sean Mauch [16]. The idea is to create (truncated) Voronoi regions for every face, edge and vertex in the mesh. These Voronoi regions are represented as polyhedra which, in turn, are scan converted. The regions corresponding to edges and faces will be either interior or exterior to the mesh depending on whether the mesh is locally concave or convex. The Voronoi region of a face will straddle the mesh, but if the closest point is on a face there will be no sign ambiguity. Thus, Mauch's method handles sign correctly, although the author does not discuss the issue.

Although scan conversion does the job, it adds complexity to the distance computation, and it seems that there is an alternative: Computing the sign locally at the closest point on the mesh.

When the closest point $\mathbf{x}$ to our given point $\mathbf{p}$ has been found, the normal at $\mathbf{x}$ can be determined (except at edges and vertices). Based on this normal it is easy to determine if $\mathbf{p}$ is inside or outside. This is done simply by taking the dot product of the normal and $\mathbf{r} = \mathbf{p} - \mathbf{x}$ which will be denoted the *direction vector* in the following. For details on why this works, see Section 4. Unfortunately, if $\mathbf{x}$ belongs to an edge or a vertex, the surface is only C^0 smooth, and the normal is not defined. The obvious

solution to this problem seems to be to find some approximate normal that will do. In the following, we will call such a normal a *pseudo normal*.

However, this is more difficult than one might think. First of all, it is important to note that if $\mathbf{x}$ is on an edge or a vertex, it is easy to find examples where the dot product with the direction vector is positive for some of the normals incident on the vertex (or edge) that contains $\mathbf{x}$ and negative for others (See Figure 1) [18]. We can compute

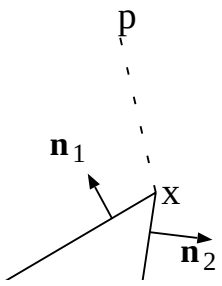


Figure 1: This figure illustrates that we can have the same distance to two triangles (line segments in 2D) but the sign of the dot product with the direction vector is not the same for the two normals.

the average normal, but again, it is possible to find cases where the dot product with the average normal is negative although $\mathbf{p}$ is outside. The counterexample is easy to construct: If we subdivide any one face incident on the vertex and leave the others, the normal of that face will completely dominate.

Huang et al. suggested using the sign of the numerically greatest dot product [13]. In other words, at a vertex or an edge, we take the dot product of the direction vector and the normal of each adjacent face. The sign of the numerically greatest dot product determines the sign of the distance. This seems to work in 2D where triangles are replaced by line segments, but it does not work in 3D. In the case of very sharp corners in the mesh, the greatest dot product may be negative for points outside the mesh. See also Section 5.

As we have seen, it is tricky to determine the sign using the face normals alone or in combination. Thus it might seem that the previously described global methods are more attractive, but in this paper we propose a pseudo normal which does produce the correct sign. This allows us to compute the sign as an integral part of the usual algorithm for distance field generation. We achieve this by defining a pseudo normal for each edge and each vertex. Our main contribution is to prove that our choice of pseudo normal, the *angle weighted normal* (described in detail in the next section), leads to correct sign computations.

In practice, we compute the sign of distances by taking the dot product between the angle weighted normal and the direction vector. This leads to a very simple implementation where no scan conversion is required. What we do need is an initial pass that stores pseudo normals for each edge and vertex of every triangle. Except for this initial pass, the algorithm is almost identical to the algorithm for generating unsigned distance fields. This is the basic idea; in the next section we describe our method in detail.

3 The Proposed Method

As mentioned above, the problem of determining the sign of a given voxel arises when the normal to the mesh is not defined on the point on the mesh closest to the voxel. Since a mesh is a piecewise linear surface, the problem of determining the normal occurs on edges and vertices. In light of this, an approach is proposed which calculates a pseudo normal at these discontinuities, i.e. the edges and vertices.

Without further ado, the pseudo normal that we propose is the *angle weighted normal* originally introduced by [22, 23]

In the case of an edge between faces i and j with normals $\mathbf{n}_i$ and $\mathbf{n}_j$ respectively the *angle weighted normal*, $\mathbf{n}_\alpha$ is given by:

$$\mathbf{n}_\alpha = \pi\mathbf{n}_i + \pi\mathbf{n}_j \quad .$$

This is seen to be a weighted sum of the normals of the incident faces, where the weights correspond to the incident angles of the faces. This generalizes to vertices as well, such that the definition of *angle weighted normal*, $\mathbf{n}_\alpha$, for a given point on the mesh is:

$$\mathbf{n}_\alpha = \sum_i \alpha_i \mathbf{n}_i \quad , \tag{4}$$

where α_i is the incident angle of face i at the point, and $\mathbf{n}_i$ is the corresponding normal. Note, that this generalizes to points on faces, giving a unified frame work, in that $\mathbf{n}_\alpha$ here is the face normal times 2π. It is seen that $\mathbf{n}_\alpha$ is a scaled variant of the average normals in a small ball around the given point. Since only the direction of $\mathbf{n}_\alpha$ is needed for sign computation, the scale is unimportant, and using $\mathbf{n}_\alpha$ thus corresponds to using the normal averaged over a small ball.

In light of this, the proposed method for computing the signed distance field corresponding to a mesh is summarized in the following pseudocode:

1. Calculate $\mathbf{n}_\alpha$ for all vertices.
2. Calculate $\mathbf{n}_\alpha$ for all edges.
3. Initialize the distance grid, by setting all elements to $\inf$.
4. for all faces:
   (a) Calculate bounding box (see below).
   (b) for all voxels in the bounding box:
      i. calculate signed distance to face.
      ii. if absolute value of distance to face is smaller then previous voxel value update the voxel value.

The reason a given face is only 'allowed' to effect the voxels within a bounding box, is computational speed, and memory usage, c.f. [14]. This bounding box is the minimal box that contains the face (only its 3 corners need be considered) extended by a constant k amount along the x, y, and z axes.

The bounding box is only used in the cases where we just want the distance field in a *transition region* on either side of the surface. Thus k should be equal to the width

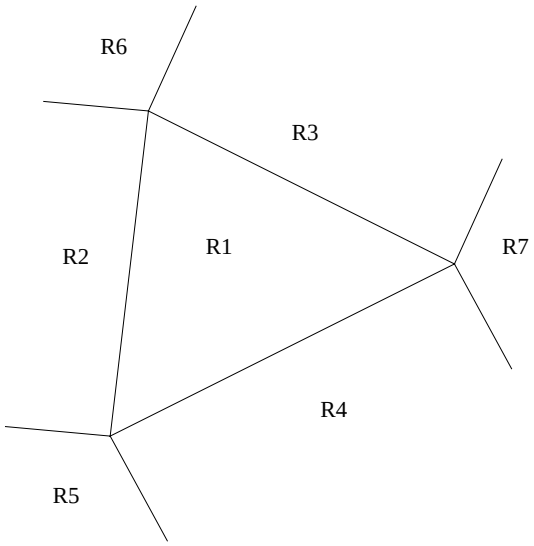


Figure 2: The seven regions in the plane containing a triangle.

of the transition region. Of course, the bounding box can be omitted if we wish a complete distance field, but in this case, it is likely that a more sophisticated method for computing distances should be used (e.g. [11]). This issue is completely orthogonal to the sign issue and will not be discussed further.

The limitations of the distance field should be seen in the context that in most efficient level set implementations narrow banding is used [1]. The narrow band approach in principal consists of only updating the level set in a neighborhood of the front, which is equal to the small distance values. Hence, if a narrow banding is applied the large distance values discarded by using a bounding box are not needed anyway.

3.1 Distance to Triangle

The challenge in calculating the distance from a point to a triangle, consists of determining whether the closest point is on the edge, a vertex, or on the actual face itself. The reason is that in the 3 different cases, a distance to a line, a point or a plane should be calculated, respectively. The situation is shown in Figure 2.

When a point $\mathbf{p}$ is projected onto the plane containing the triangle, the projected point $\mathbf{p}'$ lies in one of the 7 regions shown in Figure 2. If $\mathbf{p}'$ is projected onto $R1$ then the distance from the point to the face is equal to the distance from the point to the plane of the face. If $\mathbf{p}'$ lies in $R2$, $R3$ or $R4$ a distance to the corresponding line should be calculated. Lastly, with regions $R5$, $R6$ or $R7$ a distance to the corresponding vertex should be calculated.

3.2 Implementation

The practical implementation of this method is straightforward. We assume that the mesh is stored in a file that contains a list of vertices and a list of triangles. Each triangle is specified by three vertices. When the file is loaded, a database of edges, vertices, and triangles is built. We loop over all triangles, and for each of the triangle's vertices we add the angle weighted normal to the normal stored with the vertex representation in the database. Similarly, the (unweighted) normal is added to the edge representation in the database.

The generation of the distance field now proceeds. For each triangle, we loop over all voxels within the bounding box. For each voxel, the shortest distance is computed in the way described in the previous section. However, we find not only the shortest distance to the face, but also the actual closest point on each triangle. Since we have to locate the distance feature, this is a trivial modification.

Having obtained the closest point, we can find the direction vector. We then look up the correct angle weighted normal corresponding to the closest feature, and compute the dot product of this normal and the direction vector. The sign of this dot product is the sign of our distance.

The reader may have spotted a way to simplify the algorithm: The dot product of the angle weighted normal and the direction vector is clearly equal to the sum of the inner product of the normal (weighted by angle) and the direction vector.

$$\mathbf{n}_\alpha \cdot \mathbf{r} = \Sigma_i \alpha_i \mathbf{n}_i \cdot \mathbf{r} \ ,$$

so, apparently, we could compute maintain a sum, s, of dot products in a variable for each voxel. When a numerically shorter distance is found, the sum is reinitialized to $s = \alpha_i \mathbf{n}_i \cdot r$. When a new distance that is equal to a previous distance is found, we simply add: $s \leftarrow s + \alpha_i \mathbf{n}_i \cdot r$.

The lure of this approach is that we would not need to precompute the area weighted normals, so the method would be even simpler. Unfortunately, the method has a fundamental problem with numerical precision. Because we rely on floating point computations, we cannot expect the computed quantities to be exact. This means that it is possible that, for the same voxel, we sometimes find that for triangle i the closest point is a vertex, but for triangle j the closest point is a point on the edge between i and j that is just extremely close to the same vertex.

We believe that this problem can be addressed, for instance using robust floating point techniques [21], but we feel that the trivial solution of precomputing the angle weighted normals is preferable.

4 Proof

It is shown here that the proposed method for computing the sign is valid. In this section we will use the following definitions:

- We assume a triangle mesh, M, that is a closed 2–manifold in 3D Euclidean space. Let $\mathcal{M}$ be the closure of the volume enclosed by the mesh. In other

words, $\mathcal{M}$ is a 3–manifold with boundary, and the boundary is the triangle mesh. Note that $M = \partial\mathcal{M}$

- $\mathbf{v}$ is a vertex of the mesh.
- i is the index of a face one of whose vertices is $\mathbf{v}$. Let k faces share vertex $\mathbf{v}$, then $i \in [1, k]$
- $\mathbf{n}_i$ is the normal to a face of the mesh incident on $\mathbf{v}$. $\mathbf{n}_i$ points away from $\mathcal{M}$.
- α_i is the angle of the face at $\mathbf{v}$

The main part of this section is Theorem 1, where the use of the angle weighted normal, $\mathbf{n}_\alpha$, as pseudo normal is proven correct for vertices. The use of $\mathbf{n}_\alpha$ in general then follows straight from this.

Theorem 1 *Let there be given a point* $\mathbf{p}$*, and assume that vertex* $\mathbf{v}$ *is a closest point in* M *so that* $\|\mathbf{v} - \mathbf{p}\| = d$ *where* $d = \inf_{\mathbf{x}\in M} \|\mathbf{p} - \mathbf{x}\|$*. Let* $\mathbf{n}_\alpha$ *be the sum of normals to faces incident on* $\mathbf{v}$*, weighted by the angle of the incident face, i.e.*

$$\mathbf{n}_\alpha = \Sigma \mathbf{n}_i \alpha_i \ . \tag{5}$$

Finally, consider the vector $\mathbf{r} = \mathbf{p} - \mathbf{v}$*. It now holds for*

$$D = \mathbf{n}_\alpha \cdot \mathbf{r} \ , \tag{6}$$

that $D > 0$ *if* $\mathbf{p}$ *is outside the mesh.* $D < 0$ *if* $\mathbf{p}$ *is inside.*

To prove this, we first consider the case where $\mathbf{p}$ is outside the mesh.

Define the volume S as the intersection of the mesh, $\mathcal{M}$, and a ball, B, centered at $\mathbf{v}$. The radius of B is chosen arbitrarily to be 1. If one of the edges incident on $\mathbf{v}$ is shorter than one, we simply scale the mesh. ∂S (the boundary of S) consists of a part coincident with the mesh, ∂S_M, and a part coincident with the ball, $\partial S_B = \partial S - \partial S_M$. Observe that $\partial S = \partial S_M \cup \partial S_B$ and $\partial S_M \cap \partial S_B = \emptyset$.

Introduce a divergence free velocity field, F, where at any point $\mathbf{q}$, $F(\mathbf{q}) = \mathbf{r}$. Then, from the theorem of Gauss we have (F being divergence free)

$$\begin{aligned} \int_{\partial S} F \cdot \mathbf{n}(\tau) d\tau &= 0 \\ = \int_{\partial S_M} \mathbf{r} \cdot \mathbf{n}(\tau) d\tau &+ \int_{\partial S_B} \mathbf{r} \cdot \mathbf{n}(\tau) d\tau \ . \end{aligned} \tag{7}$$

Lemma 1 *For any point* $\mathbf{q} \in S$ *the angle* $\angle(\mathbf{vq}, \mathbf{vp})$ *is greater than or equal to* $\pi/2$*.* $\mathbf{p}$ *is outside* $\mathcal{M}$ *by assumption.*

Proof: By construction $\mathbf{v}$ is a star point in S, i.e. the line segment between v and any point in S lies completely in S. Hence, if there is a $\mathbf{q}$ such that $\angle(\mathbf{vq}, \mathbf{vp}) < \pi/2$, there would be a point on the line between $\mathbf{v}$ and $\mathbf{q}$ which is closer to $\mathbf{p}$ then $\mathbf{v}$. This is easily seen, because if

$$\angle(\mathbf{vq}, \mathbf{vp}) < \pi/2 \ ,$$

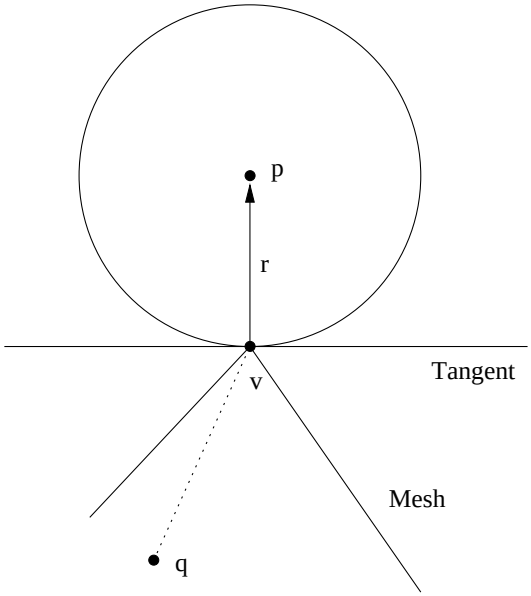


Figure 3: It is seen that $\angle(\mathbf{vq}, \mathbf{vp}) \geq \pi/2$, since $\mathbf{v}$ is the point in M closest to $\mathbf{p}$.

the line segment from $\mathbf{v}$ to $\mathbf{q}$ must pass through the interior of a closed ball of radius r centered at $\mathbf{p}$, and any point in the interior of this ball will be closer to $\mathbf{p}$ than $\mathbf{v}$. Finally, since $S \subset \mathcal{M}$, this contradicts our requirement that $\mathbf{v}$ is the point in M closest to $\mathbf{p}$. See Figure 3. □

For all points $\mathbf{q} \in \partial S_B$ it is seen that the normal, $\mathbf{n}(\mathbf{q})$, is given by $\mathbf{vq}$, since B is the unit sphere centered at $\mathbf{v}$. So, by Lemma 1, $\mathbf{n}(\mathbf{q}) \cdot \mathbf{r} \leq 0$ for all $\mathbf{n}_B$. Therefore, we have that

$$\int_{\partial S_B} \mathbf{r} \cdot \mathbf{n}(\tau) d\tau < 0 \ . \tag{8}$$

The inequality in (8) is strict because the left hand side is only zero if the area of ∂S_B is zero, and this, in turn, would require the mesh to collapse breaking our manifold assumption.

From (7) it now follows that

$$\int_{\partial S_M} \mathbf{r} \cdot \mathbf{n}(\tau) d\tau > 0 \ . \tag{9}$$

It is seen that the intersection of face i and S has an area equal to α_i, implying that the flux of F through this intersection is given by $\mathbf{r} \cdot \mathbf{n}_i \alpha_i$. So

$$\int_{\partial S_M} \mathbf{r} \cdot \mathbf{n}(\tau) d\tau = \Sigma \mathbf{r} \cdot \mathbf{n}_i \alpha_i = \mathbf{r} \cdot \mathbf{n}_\alpha = D > 0 \ . \tag{10}$$

Proving the theorem for $\mathbf{p}$ outside the mesh. If $\mathbf{p}$ is inside the mesh, the situation is essentially the same, except for the fact that the direction of the involved normals point the other way. This means that the integral over ∂S_B changes sign. Thus, D becomes negative which concludes our proof □

Note that we do not assume that the closest point is unique. The proof requires only that $\mathbf{v}$ is *a* closest point. This means that Theorem 1 also holds in the case where $\mathbf{p}$ lies on the medial axis.

For the sake of completeness, we now extend Theorem 1 to edges and faces. Recall that on an edge

$$\mathbf{n}_\alpha = \pi\mathbf{n}_i + \pi\mathbf{n}_j \ ,$$

where i and j are the faces incident on the edge. Similarly, on a face i

$$\mathbf{n}_\alpha = 2\pi\mathbf{n}_i \ .$$

Corollary 2 *Let there be given a point* $\mathbf{p}$*, and assume that* $\mathbf{c}$ *is a point in* M *closest to* $\mathbf{p}$*.*

$$\mathbf{n}_\alpha = \Sigma\mathbf{n}_i\alpha_i \ . \tag{11}$$

Finally, consider the vector $\mathbf{r} = \mathbf{p} - \mathbf{c}$*. It now holds for*

$$D = \mathbf{n}_\alpha \cdot \mathbf{r} \ , \tag{12}$$

that $D > 0$ *if* $\mathbf{p}$ *is outside the mesh.* $D < 0$ *if* $\mathbf{p}$ *is inside.*

The proof is trivial since we can transform any edge point or face point to a vertex point by inserting a vertex. This does not change the geometry of M, but the case is now handled by Theorem 1 □

5 Results

We have used two simple models, a tetrahedron and a pyramid, to verify our own approach and to demonstrate the problems with the two other local methods, we have discussed previously. The results are shown in Figure 4. Except for the mesh all the figures have been generated using texture mapped volume visualization. Positive distances are dark and transparent, negative distances are white and opaque. This make it easy to spot exterior areas containing voxels that have erroneous sign.

Two sides in the tetrahedron have been subdivided. As seen from the figure, this does not cause problems for our method, but, if we use unweighted sum, regions of spurious negative distances appear around the vertices where many triangles meet.

A similar problem occurs in the case of the pyramid. If we use the maximum of the dot products to determine sign, three regions of spurious negative distances appear around the top vertex. Again, our method handles this case correctly.

To demonstrate our method in conjunction with a slightly larger mesh, we have voxelized a model containing 11492 triangles. To illustrate an application of distance fields, a series of offset surfaces have been generated from the model. These offset surfaces have also been visualized using texture based volume rendering, but this time diffuse and opaque isosurfaces have been generated. The results are shown in Figure 5.

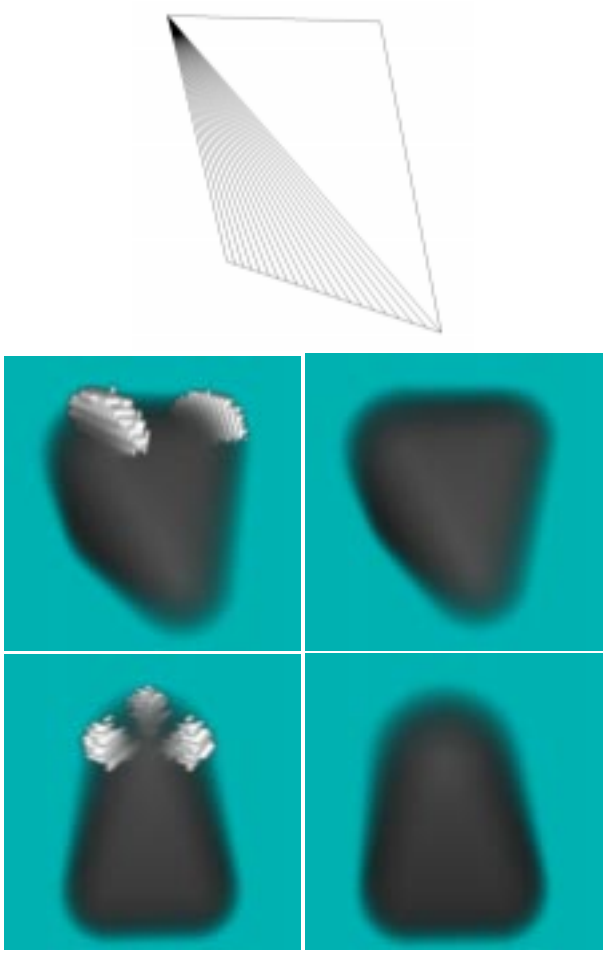


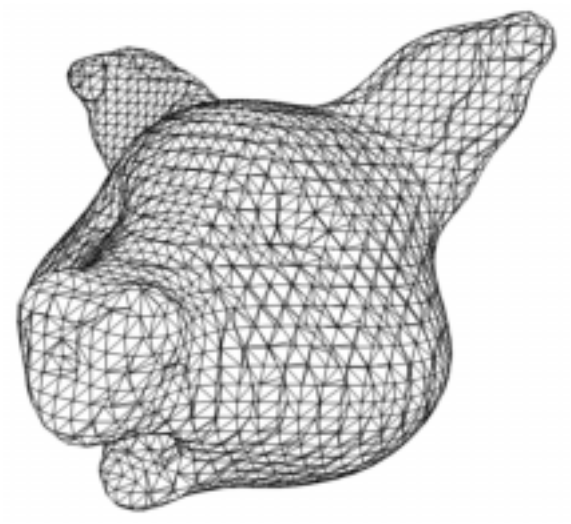
Figure 4: Tetrahedron with two subdivided sides (top). Tetrahedron voxelized using unweighted sum of normals (middle left). Tetrahedron voxelized using our method (middle right). Pyramid voxelized using maximum dot product (bottom left). Pyramid voxelized using our method (bottom right).

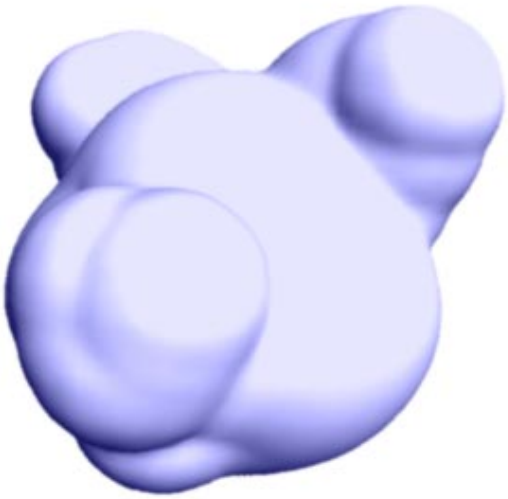
6 Discussion

In this paper, we have proposed a new and simple method for generating signed distance fields from triangle meshes. Our method does not rely on scan conversion but is a simple extension to the well known method for generating unsigned distance fields.

We have proven that the method really works, i.e. that the dot product of the angle weighted normal and the direction vector is always positive if we are outside and negative if we are inside.

We believe that this method will prove useful for generating signed distance fields which seem to be becoming increasingly popular as volumetric techniques in general and the level set method in particular become more and more important.





References

[1] David Adalsteinsson and James A. Sethian. A fast level set method for propagating interfaces. *Journal of Computational Physics*, 118(2):269–77, 1995.

[2] David E. Breen and Ross T. Whitaker. A level-set approach for the metamorphosis of solid models. *Visualization and Computer Graphics, IEEE Transactions on*, 7(2):173–192, 2001.

[3] Andreas Bærentzen and Niels Jørgen Christensen. A technique for volumetric csg based on morphology. In *Proceedings of International Workshop on Volume Graphics*, 2001.

[4] Andreas Bærentzen and Niels Jørgen Christensen. Volume sculpting using the level-set method. In *Proceedings of Shape Modelling International 2002*, 2002.

[5] Andreas Bærentzen, Miloš Šrámek, and Niels Jørgen Christensen. A morphological approach to the voxelization of solids. In Vaclav Skala, editor, *Proceedings of WSCG 2000*, volume I, February 2000.

[6] Frank Dachille and Arie Kaufman. Incremental triangle voxelization. In *Proceedings of Graphics Interface 2000*, pages 205–212, 2000.

[7] Douglas Enright, Steve Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. *ACM Transactions on Graphics*, 21(3):736–44, 2002.

[8] Olivier D. Faugeras and Renaud Keriven. Variational principles, surface evolution, pdes, level set methods, and the stereo problem. *Image Processing, IEEE Transactions on*, 7(3):336 –344, 1998.

[9] Sarah F.F. Gibson. Using distance maps for accurate surface representation in sampled volumes. In Stephen Spencer, editor, *Proceedings of IEEE Symposium on Volume Visualization*, October 1998.

[10] José Gomes and Aleksandra Mojsilovic. A variational approach to recovering a manifold from sample points. *Computer Vision - ECCV 2002. 7th European Conference on Computer Vision. Proceedings (Lecture Notes in Computer Science Vol.2351)*, pages 3–17, 2002.

[11] André Guéziec. "meshsweeper": Dynamic point–to–polygonal mesh distance and applications. *IEEE Transactions on Visualization and Computer Graphics*, 7(1):47–60, January–March 2001.

[12] Christoph M. Hoffmann. *Geometric and Solid Modeling*. Morgan Kaufmann, 1989.

[13] Jian Huang, Yan Li, R. Crawfis, Shao-Chiung Lu, and Shuh-Yuan Liou. A complete distance field representation. *Visualization, 2001. VIS '01. Proceedings*, pages 247–254, 2001.

[14] Mark W. Jones. The production of volume data from triangular meshes using voxelisation. *Computer Graphics Forum*, 15(5):311–318, 1996.

[15] Ravi Malladi, James A. Sethian, and Baba C. Vemuri. Shape modeling with front propagation: a level set approach. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(2):158–175, 1995.

[16] Sean Mauch. A fast algorithm for computing the closest point and distance transform. Technical report, Applied and Computational Mathematics, California Institute of Technology, 2000.

[17] Stanley Osher and James A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[18] Bradley A. Payne and Arthur W. Toga. Distance field manipulation of surface models. *Computer Graphics and Applications*, 12(1), 1992.

[19] Richard Satherley and Mark W. Jones. Extending hypertextures to non-geometrically definable volume data. *Proceedings of the International Workshop on Volume Graphics*, pages 77–88 vol.1, 1999.

[20] James A. Sethian. *Level Set Methods and Fast Marching Methods Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, 1999.

[21] Jonathan Richard Shewchuk. Robust Adaptive Floating-Point Geometric Predicates. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, pages 141–150. Association for Computing Machinery, May 1996.

[22] Carlo H. Séquin. Procedural spline interpolation in unicubix. *Proceedings of the 3rd USENIX Computer Graphics Workshop*, pages 63–83, 1986.

[23] Grith Thürmer and C.A. Wüthrich. Computing vertex normals from polygonal facets. *Journal of Graphics Tools*, 3(1):43–6, 1998.

[24] Ross T. Whitaker, David E. Breen, Ken Museth, and N. Soni. Segmentation of biological volume datasets using a level set framework. pages 249–263, 2001.