Signed Distance Function Evaluation for a Polygonal Mesh

by

Shigeo Noguchi

Faculty of Computer and Information Science
Hosei University
n02k1030@k.hosei.ac.jp

Supervisor: Alexander Pasko

March 2006
Abstract

We explore approaches to the implementation of the signed distance function for polygonal meshes. To make the function, at first, we adopt octree data structure to define points around a 3D object for efficient calculations and saving calculated distance data. To judge whether given points are inside, outside, or just on the surface of the 3D object, we search the nearest point and the nearest mesh triangle. Then, we compute scalar product between the normal vector at the nearest point and the vector from the octree node vertex to the nearest point and judge the function sign from the scalar product value. As a result of this work, we have implemented a prototype of the signed distance function by the combination of the above techniques.
Introduction

One of the most popular ways to describe 3D objects with computer graphics is using triangle meshes (polygons). We can describe various polyhedrons by combination of some triangle meshes. To calculate signed distance to mesh that makes up a 3D object from arbitrary points in 3D space can provide the expansion of shape descriptions in 3D computer graphics. Here, "signed" means that the distance from points that exist inside of the object gets positive values, and the distance from points that exist outside become gets negative values, and it is equal to zero just on the surface defined by the mesh.

We consider that the calculation provides necessary and efficient data for making new primitives and operations (e.g., blending or metamorphosis) in HyperFun, where a 2D or 3D object is described by F-rep (Function Representation). F-rep defines a geometric object by a single real continuous function of point coordinates as

\[ F(x,y,z) = 0 \] when point \((x,y,z)\) is surface of object,
\[ F(x,y,z) > 0 \] when point \((x,y,z)\) is inside of object,
\[ F(x,y,z) < 0 \] when point \((x,y,z)\) is outside of object. [1]

If we can calculate the signed distance to polygonal meshes, we can describe 3D objects by calculated distance data similar to other F-rep functions, and we can extend the HyperFun library by new primitives. In our work, we try to make a prototype of the signed distance function evaluation procedure using existing techniques.

In the following section, the techniques that are needed to make the function are described. First, we describe what kind of file format is used to generate a 3D polygonal object in section 2.1. The algorithm of generating the octree data structure is described in section 2.2. And the algorithm of the nearest point search using the ANN (Approximate Nearest Neighbor) function is shown in section 2.3. We describe in section 2.4 how to judge whether points are inside or outside of the
given 3D polygonal object. Then, our experimental result is shown in section 3. Finally, conclusions and future work discussion are given in section 4.

1.1 Related Work

There are different ways to describe 3D objects using distance. One of the most known is the Blobby model. The metaballs are also known and similar to the blobby model. The metaballs is a type of implicit modeling description. The metaballs are defined as an aggregate of many points that have the same level in the scalar (density) field, where the level of density attributed to the point decreases with distance from the point location. The powerful aspect of metaballs is the way they can be combined. By simply summing the influences of each metaball at the given point, we can get very smooth blending of the spherical influence fields.

Also, there are algorithms that calculate signed distance to a mesh in different ways. The main algorithm to calculate distance is just to compute distance between a query point and the closest vertex of a triangle mesh:

\[
Dist = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}
\]

Differences in methods appear in the phase of choosing the closest mesh vertex and in the distance sign detection. The way to choose mesh vertices is described in section 2.3. In our phase of the sign detection, we compute normal vector of a nearest triangle mesh and judge from scalar product between the normal vector and the vector that direct query point to vertex of a triangle mesh. Since we use 3D polygonal objects that have no holes and no dangling pieces, we can adopt this algorithm for the distance sign detection. In case of using 3D polygonal objects that have holes, the "Angle Weighted Pseudo-normal" [4] allows to discriminate between points that are inside and outside the mesh. The "Angle Weighted Pseudo-normal" is different with our approach in part of computing the normal vector.
2. Distance evaluation procedure

2.1 Input polygonal mesh

At first, to generate 3D polygonal object that is made of a lot of polygonal meshes, we use some known file format that describe the 3D object by point data and facet data. It must not be any hole or any dangling pieces in the object to be defined as a polyhedron in 3D space. It means the object must be a solid boundary that can help us distinguish 3D space subspaces as inside of object and outside of object.

In this research, we use the VRML (Virtual Reality Modeling Language) file. In the VRML format, 3D polygonal meshes are described by combination of point data and polygon (coordIndex) data. Point data shows vertex coordinates of the 3D polygonal mesh. CoordIndex data shows indices of point data making a single facet. Three point data are needed to make a mesh triangle and coordIndex data shows which three points to use for generating the mesh. The applications can read a VRML file, scan point data and coordIndex data from it and generate a 3D polygonal object.

2.2 Generate Octree

2.2.1 Octree

To calculate the signed distance between a point of the mesh and arbitrary points in 3D space, we make the octree data structure. It looks like a lot of cubes of different sizes when it is visualized (Figure 1). The octree has a tree structure, where the root has eight children and each child have also eight nodes as children. Each of them can store data and their vertices can be used as points for calculation of distance to a 3D polygonal object. We can define location and width of each cube and also a number of divisions and the rule for generation of new nodes.
2.2.2 Generation rule

An octree node can divide the corresponding cube into eight children of the same size by halving its own side length. We can define how many times to divide the octree nodes. When the number of divisions is larger, then size of cubes becomes small and the number of their vertices becomes large. This way, we can calculate more detailed data of distance in 3D space around the polygonal object. If the number of subdivisions increases, time to calculate the distance also increases. We apply a rule for the subdivision and generation. The rule is that, when a cube has any point of the polygonal mesh inside, and the level of division has not reached to the level that we defined, the cube is subdivided and we generate new nodes. It is simple, but we can calculate more detail near the given point and skip cubes for the point far from the object (Figure 2).
Figure 2. Octree: (a-c) 6 levels of subdivision; (d) 5 levels of subdivision

2.3 Search a Nearest point and Calculate Distance

There are two algorithms for the calculation of the distance to a 3D polygonal object that we consider to use for the function. One is a pure brute force algorithm that computes the distance from the current point to each triangle from the mesh, and saves the minimum distance.
Another is the nearest search algorithm. We adopt the nearest search algorithm that search nearest point for the 3D polygonal object from the current point faster, then calculate and save distance. If we adopt the pure brute force algorithm to our function, it may take long time to calculate distance when we input a complex shape, especially in the situation when many points in space are tested as in ray-casting or other algorithms.

2.3.1 Approximate Nearest Neighbor

To search the nearest point of the mesh for the given point, we adopt the ANN (Approximate Nearest Neighbor) library to our function. ANN supports the method for nearest point searching with kd-trees. The kd-tree data structure is based on a recursive subdivision of k-dimensional space into disjoint hyper rectangular regions. Each node in the kd-tree is defined by a hyper rectangular region through one of the dimensions that division the set of points into left or right (or up or down) sets, each with half of the points of the parent node. These children are again divided into equal halves (Figure 3).

![Figure 3: kd-tree data structure](image)

At the time when we input all point data, ANN builds the kd-trees data structure with set of points that make up the 3D polygonal mesh. And vertices of the octree cubes also are input. Then, nearest searching is performed with each vertex of the octree cube. The ANN procedure is given a
nonnegative number $k$, an array of point indices, $\text{nn idx}$, and an array of distances, $\text{dists}$. Both arrays are assumed to contain at least $k$ elements. The procedure computes the $k$ nearest neighbors to each octree vertex, and stores the indices of the nearest point. The nearest neighbor can be stored in $\text{nn idx}[0]$, the second nearest in $\text{nn idx}[1]$, and so on. The squared distances to the corresponding points are stored in the array $\text{dists}$.

When first encountering a node of the kd-tree, the algorithm first visits the child that is closest to the query point. On return, if the box containing the other child lies within $1 / (1 + \varepsilon)$ times ($\varepsilon$ is error bound) the distance to the closest point seen so far, then the other child is visited recursively. The distance between a box and the query point is computed exactly, using incremental distance updates. [2][3]

### 2.4 Decision on Sign of Distance

\[
\begin{align*}
\mathbf{N}_x &= (y_2 - y_1)x(z_3 - z_1) - (z_2 - z_1)x(y_3 - y_1) \\
\mathbf{N}_y &= (z_2 - z_1)x(x_3 - x_1) - (x_2 - x_1)x(z_3 - z_1) \\
\mathbf{N}_z &= (x_2 - x_1)x(y_3 - y_1) - (y_2 - y_1)x(x_3 - x_1)
\end{align*}
\]

\[
\hat{\mathbf{N}} = \left( \mathbf{N}_x / |\mathbf{N}| \right) + \left( \mathbf{N}_y / |\mathbf{N}| \right) + \left( \mathbf{N}_z / |\mathbf{N}| \right)
\]

\[
\begin{align*}
\mathbf{N}_x &= \hat{\mathbf{N}} x / |\hat{\mathbf{N}}| \\
\mathbf{N}_y &= \hat{\mathbf{N}} y / |\hat{\mathbf{N}}| \\
\mathbf{N}_z &= \hat{\mathbf{N}} z / |\hat{\mathbf{N}}|
\end{align*}
\]

Figure 4: Normal vector of polygonal mesh
To decide whether the point Q (one of the octree cube vertices) is inside or outside or just on surface of 3D polygonal object, we compute and save each normal vector of the polygonal mesh after the point data have been input from the file (Figure 4).

After the nearest point search, we must search the nearest mesh triangle that has the nearest point as one of its vertices. If 3D the polygonal object is defined as a polyhedron in 3D space, there must be some triangle that has nearest point as its vertex (Figure 5). To calculate signed distance between the point in 3D space and point of 3D polygonal mesh correctly, we must decide the nearest polygonal mesh triangle and the appropriate normal vector correctly. So after the nearest point search, we also need to search the nearest polygonal mesh triangle. [4]

Using the nearest search algorithm described in section 2.3, we can find the nearest point of the mesh. Next, we calculate distances to some points that are defined as set of triangles including nearest point. Then a point that has the minimum distance in some point set without a nearest point would be found. That means, we can find a nearest line of mesh. But there must be two triangles
including the line yet. So we calculate again the same way. After this search, we can find the nearest triangle and calculate the appropriate normal vector N.

Then, we calculate the vector V from point Q to vertex P of the mesh and calculate scalar product of the normal N and the vector V. If it is positive, we judge that the point Q is inside the object and the distance sign function becomes ’+1’. If the scalar product is negative, we judge that the point Q is outside object and the sign function becomes ‘-1’.

Finally, we multiply the sign function and the calculated distance between the point P and the point Q, and we can get signed distance to the 3D polygonal object.

\[
\text{Signed dist}_{PQ} = \text{sign} \times \sqrt{(x_P - x_Q)^2 + (y_P - y_Q)^2 + (z_P - z_Q)^2}
\]

In this formula, sign is computed according to the above step, and it must be ’+1’ or ‘-1’. Squared distance between P and Q is computed by the ANN procedure. So, in this step, we just put the square distance in the root. Then, we can get signed distance between P and Q.

The signed distance data are saved in the octree data structure. One cube of octree can save eight signed distance values.
3. Experimental results

As the result, we combine all techniques discussed in section 2, and can make a prototype of the signed distance function evaluation procedure. The prototype program can input VRML files that describe 3D polygonal objects made of some triangle meshes. Then, we create the octree data structure: define its location in 3D space, width of first cube, and the number of subdivisions. After these data a specified, the procedure operates automatically. The nearest point search starts with each vertex of the generated octree. We calculate and define the sign of the distance function, and then save the signed distance data to the octree data structure. Each cube can save eight distance data values, the same as the number of its vertices (Figure 6).

![Figure 6(a): Result of nearest point search with 3 levels of subdivision](image)

(defined width of octree cube is 4 at first, so minimum width of cube is 0.5 in.); each line connects most nearest point of the mesh to each node vertex of octree cubes.)
See in Figure 6(a), the lines just connect each octree vertex and the nearest point from each vertex. We can see those octree points are connected with nearest point. Figure 6(b) is just a little part of the calculation result printout. Query point shows coordinates of the octree vertex. In Figure (b), only node vertices of the biggest cube that can see in Figure 6(a) are shown. NN shows 0, it means that ANN has found the nearest point. Index shows the index of the found nearest point. Sign and Distance are results of the calculation discussed in sections 2.3 and 2.4.

| Query point: (-2, -2, 2) | NN:  
Index | Sign | Distance |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>128</td>
</tr>
</tbody>
</table>
| Query point: (-2, 2, 2) | NN:  
Index | Sign | Distance |
| 0                       | 23   | -1       | -1.2286   |
| Query point: (2, 2, 2)  | NN:  
Index | Sign | Distance |
| 0                       | 87   | -1       | -1.2286   |
| Query point: (2, -2, 2) | NN:  
Index | Sign | Distance |
| 0                       | 204  | -1       | -2.64743  |
| Query point: (-2, -2, -2)| NN:  
Index | Sign | Distance |
| 0                       | 113  | -1       | -2.3882   |
4. Conclusion and Future work

In this research, we studied and have implemented the signed distance function evaluation, which can be used to add new primitives to the HyperFun library. We succeeded in making the prototype of the function evaluation and it works well.

There are some problems in the function evaluation implementation. One is calculation time, we adopted the nearest search algorithm in order to cut surplus time, but it still takes long time. Another is some overlapping in making tree data structures. We made octree data structure at first, next made kd-tree data structure for the nearest point searching. It may be possible to remove one of those. Another problem sometimes occurs when we compute the normal vector of a triangle. When we input point data, the order of three points that need to generate a triangle mesh is defined by coordIndex data, and the order also relate with computing the normal vector. Every normal vector should be directed to outside of 3D polygonal object. But if the order of coordIndex changed, direction of the normal vector may be also changed.

The main purpose of this research is to make a new primitive for HyperFun using the distance function we made and to test it. It remains most needed in future work.
Acknowledgments

The author and the supervisor are thankful to Pierre-Alain Fayolle for the fruitful discussions of the distance function evaluation issues.

References


