Eurographics/ IEEE-VGTC Symposium on Visualization 2008
A. Vilanova, A. Telea, G. Scheuermann, and T. Möller
(Guest Editors)

Volume 27 (2008), Number 3

Computing Local Signed Distance Fields for Large Polygonal Models

B. Chang and D. Cha and I. Ihm
Department of Computer Science, Sogang University, Korea

Abstract

The signed distance field for a polygonal model is a useful representation that facilitates efficient computation in many visualization and geometric processing tasks. Often it is more effective to build a local distance field only within a narrow band around the surface that holds local geometric information for the model. In this paper, we present a novel technique to construct a volumetric local signed distance field of a polygonal model. To compute the local field efficiently, exactly those cells that cross the polygonal surface are found first through a new voxelization method, building a list of intersecting triangles for each boundary cell. After their neighboring cells are classified, the triangle lists are exploited to compute the local signed distance field with minimized voxel-to-triangle distance computations. While several efficient methods for computing the distance field, particularly those harnessing the graphics processing unit's (GPU's) processing power, have recently been proposed, we focus on a CPU-based technique, intended to deal flexibly with large polygonal models and high-resolution grids that are often too bulky for GPU computation.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling–Boundary representations; I.3.6 [Computer Graphics]: Methodology and Techniques–Graphics data structures and data types

1. Introduction

1.1. Problem Specification

A distance field is a volumetric dataset that represents the closest distance information. Given a geometric object in n-dimensional space, the distance field defines the shortest distance $d(\mathbf{x})$ from a point $\mathbf{x} \in R^n$ to a point on the object. The distance field has been explored as an important tool in developing efficient algorithms in various fields, such as computer graphics, scientific visualization, image processing, robotics, and computer vision.

When an object is closed and orientable, the distance field may be assigned a proper sign that indicates whether a voxel point is inside or outside the object. The signed distance is very useful because it enhances the efficiency and flexibility in solving many applications. Because a brute-force distance computation is very expensive for even moderate-size geometric models and grids, considerable efforts have been made to design an efficient algorithm to construct the distance, or signed distance, field.

Most earlier studies have been concerned with constructing a global (signed) distance field where the shortest distance is calculated for all voxels in the computational domain. However in many applications, it is often sufficient to build the distance field only close to the surface. For example, when an object is represented in level sets, its geometric properties, such as its normal, curvature, and surface area, may be determined using the signed distance values at voxels within a narrow band close to its surface. Furthermore, once the local signed distance field is created, it may be used as an initial condition for such propagation algorithms as the fast marching method [Set96] and the fast sweeping method [Zha04], which easily compute the signed distances of voxels within a desired band region (or in the entire domain) by solving the Eikonal equation $|\nabla d| = 1$.

1.2. Our Contribution

In this paper, we present a novel approach that efficiently builds a local signed distance field within a narrow band around a three-dimensional triangular mesh. Our method consists of two stages. First, those boundary cells that cross the polygonal surface are located robustly using a new voxelization technique, for which lists of overlapping triangles are built. Then after their neighboring cells are classified according to their proximity to the surface, the triangle lists are explored to compute the local signed distance field with minimum voxel-to-triangle distance computations. In developing our technique, we assume that complicated polygonal models are discretized on high-resolution grids, requiring nontrivial computational time and memory space. Although several efficient methods for constructing the global distance field, particularly those exploring the graphics pro-

cessing unit's (GPU's) processing power, have recently been proposed, this work concentrates on the development of a CPU-based technique that allows a flexible method for a detailed volumetric representation of large polygonal models on high-resolution grids, which are often too heavy a computational load for the GPU computation.

2. Previous Work

The current algorithms that construct the Euclidean distance field for a given triangular mesh can be classified largely into two approaches (for a survey of related works, refer to the recent article [JBS06]). In the first one, a generalized Voronoi diagram or its variant is built from which the distance field is computed as the distance to the respective site. As constructing an exact generalized Voronoi diagram requires a substantial amount of effort for a large, complex polygonal model, several practical methods have been proposed. In stead of the exact Voronoi diagram, Mauch computed a polyhedral bounding volume for each Voronoi site that at least contain all the closest voxel points [Mau03]. Then, the characteristic polyhedra were scan-converted to produce the distance field. This method, called the Characteristics/Scan-Conversion (CSC), can be applied naturally for computing the distance field in a narrow band around the surface.

The second approach is mostly based on the propagation method that numerically solves the Eikonal equation. The fast marching method updates the distance field voxel by voxel in the order of strictly increasing distance [Tsi95, Set96]. The time complexity of the method is $O(N \log N)$ for N voxel points because a heap data structure must be used to implement a priority queue (for an improved implementation of the propagation method, refer to the thesis of Mauch [Mau03]). In discretizing a Constructive Solid Geometry (CSG) model consisting of superellipsoid primitives, the idea of the fast marching method was modified to employ a heuristic rule for propagating the closest-point information [BMW00]. In this method, the local zero set within a narrow band was computed so that it can be used as an initial condition for their iteration scheme. While not solving the Eikonal equation, a contour by contour propagation was also performed in the technique called a complete distance field representation [HLC*01]. Recently, a fast sweeping method was presented by Zhao that uses a nonlinear upwind difference scheme and Gauss-Seidel iterations with alternating sweeping ordering [Zha04]. This sweeping method is optimal in the sense that the Eikonal equation is solved in $O(N)$-time.

In addition to these solutions, a method to compute a piecewise linear approximation of the signed distance field was presented by Wu and Kobbelt, that is based on an adaptive hierarchical space partition [WK03]. The processing power of graphics hardware has also been harnessed to accelerate the distance computation. Interpolation-based polygon rasterization hardware was utilized to compute generalized 3D Voronoi diagrams that naturally lead to the distance computation [HICK*99]. The CSC technique was modified by Sigg et al. [SPG03] to offer an efficient GPU implementation. To speed up the hardware-assisted distance field computation, Sud et al. proposed to explore a culling algorithm, based on the property of the Voronoi diagram, along with a clamping algorithm [SOM04, SGGM06]. Sud et al. presented a GPU-based algorithm to efficiently compute the surface distance map for a triangular mesh using an affine mapping [SGG*07].

Finally, representing local or global distance fields at very high resolutions is a challenging problem. For quick pointers on this topic, refer to the recent works [HNB*06, NNSM07].

3. Generation of Triangle Lists

Consider a volume space of resolution $n_x \times n_y \times n_z$ in a three-dimensional Euclidean real space Σ that encloses, possibly partially, a given triangular mesh M. In our framework, the unit cubes that partition the rectangular grid space are called *cells* and their respective centers are *voxels*. Furthermore, those cells that cross the polygonal surface are called *boundary cells*.

The first task in building a local signed distance field for M is to locate its boundary cells and build triangle lists such that each list holds triangles that overlap, at least partially, the respective boundary cell region. Then, these lists are used to construct the signed distance field within a local band close to the polygonal surface. To construct such triangle lists, some form of voxelization process must be performed in a robust manner so that those boundary cells are found exactly and efficiently. In this section, we describe the list creation for a given triangular mesh, and then describe the use of the triangle lists to build a local signed distance field while minimizing the expensive voxel-to-triangle distance computations, described in the following section.

3.1. Voxelization of Triangular Mesh

Several voxelization or three-dimensional scan-conversion techniques have been proposed by the computer graphics community to build volumetric representations of polygonal models (for example, refer to [CK90]). In developing our algorithm, we have designed yet another three-dimensional voxelization method specifically for our purpose. Our technique is different from most previous methods in that it is an object-space algorithm, where triangles of the mesh are subdivided into sufficiently small subtriangles to quickly determine which cell, or cells, they overlap in the volume space.

Note that the voxelization method used in this paper finds exactly those cells that intersect the given polygonal model (finding such cells can be also done easily if we have a max-norm distance field as described by Varadhan et al [VKK*03]). In terms of the classification formulated by Huang et al. [HYVK98], the voxelization from our method may not satisfy the minimality condition. However, this criterion is not critical in our method because the purpose of voxelization in our algorithm is to locate exactly the boundary cells from which a local signed distance field propagates, and build the triangle lists for them.

3.1.1. Generation of Microtriangle(s) from a Triangle

Given a triangle $T = T(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2)$ of M, defined in the volume space Σ, consider its three edge vectors $\mathbf{v}_0 = \mathbf{p}_1 - \mathbf{p}_0$, $\mathbf{v}_1 = \mathbf{p}_2 - \mathbf{p}_1$, and $\mathbf{v}_2 = \mathbf{p}_0 - \mathbf{p}_2$. The largest value of the maximum norms $d_T = \max_i ||\mathbf{v}_i||_\infty$ of these three vectors represents the length of the longest edge of the tight axis-aligned bounding box (AABB) surrounding T.

The value of this metric indicates how large a triangle is with respect to the cells in Σ. Note that, if d_T is less than or equal to one, the triangle T can fit into a unit-bounding box in Σ. Our voxelization algorithm first checks whether d_T is greater than one and, if it is, subdivides T into a set of smaller *microtriangles* such that each of these may be bounded by a unit box. In this case, the three edges of T are subdivided respectively into $\mathsf{ceil}(d_T)$ segments that, in turn, are meshed as shown in Figure 1. Once generated from T, the microtriangles are investigated one at a time to find the intersecting cells quickly.

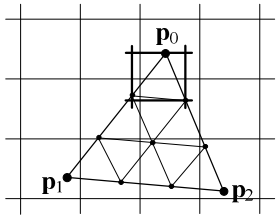


Figure 1: *Generation of microtriangles. Each triangle of a given triangular mesh is subdivided into microtriangles that can fit into a unit cube.*

Before describing the next step, we note that the timing performance of our algorithm for finding the boundary cells is influenced by the total number of processed microtriangles, which greatly depends on the relative size of triangles with respect to the unit cell in Σ. In case a *minification* situation occurs, in other words, when the triangles from M are mostly smaller than a unit cell in size, most are not subdivided. On the other hand, when a *magnification* situation takes place where triangles usually span more than one cell, possibly several microtriangles are generated per triangle, increasing the processing cost. In the boundary situation that we usually target, at most only a few microtriangles are generated per original triangle. Consequently, the boundary cells are usually found quite quickly by our method unless a severe minification situation occurs.

3.1.2. Intersection Test of Microtriangles with Neighboring Cells

The major challenge at this stage of finding the boundary cells is to determine efficiently which neighboring cell or cells overlap a given microtriangle. It is clear that a triangle T intersects with all the cells in which the vertices of the microtriangle mesh exist. Therefore, a quick scan is first made over these vertices, adding T to the triangle lists of the corresponding cells. Most boundary cells crossing T are checked through this simple process because the microtriangles are bounded by an AABB smaller than a unit cube. However, it is possible for a microtriangle to pass a cell or cells to which its three vertices do not belong; this requires an additional checking process.

Consider the three vertices $\mathbf{p}_i = (p_{ix}, p_{iy}, p_{iz})$ $(i = 0,1,2)$ of a microtriangle MT. The integer indices $\mathbf{P}_i = (P_{ix}, P_{iy}, P_{iz})$, are obtained from $\mathbf{p}_i$ by truncating the fractional parts of the three coordinates and denote the cell to which $\mathbf{p}_i$ belongs. If we define three Boolean variables I_α $(\alpha = x, y, z)$ as follows:

$$I_\alpha = \begin{cases} 1, & \text{if } P_{0\alpha} = P_{1\alpha} = P_{2\alpha}, \\ 0, & \text{otherwise,} \end{cases}$$

then the relationship between MT and its surrounding cells may be classified into the four cases according to the value of their sum $I = I_x + I_y + I_z$ (see Figure 2).

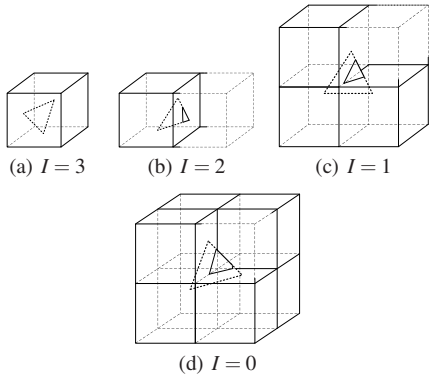


Figure 2: *Four possible cases. Depending upon the spatial relationship between microtriangle and cells in volume space, four different situations may occur. No additional work has to be done for the first two cases (a) and (b), whereas some relatively nontrivial computation must be done in the fourth case (d) to test if the microtriangle overlaps its neighboring cells. This costly case usually occurs, however, with a low frequency, as shown experimentally in Table 2.*

- **Case 1 ($I = 3$):** This case occurs when all three vertices of MT belong to the same cell. In other words, the microtriangle exists entirely within one cell. Because the triangle T, from which MT comes, has been put into the triangle list of this cell through the vertex scan process, nothing more needs to be done in this case.
- **Case 2 ($I = 2$):** This is the case when MT spans two adjacent cells along an axis. As in the first case, no additional work is required as these two cells have been already checked through the vertex scanning process.
- **Case 3 ($I = 1$):** When I is one, four adjacent cells are necessary to enclose MT, as shown in Figure 2(c). Because the three vertices of MT belong to two or three different cells, it is sufficient to check whether the remaining cell(s) is crossed by MT. For efficient computation, we project the vertices onto the two-dimensional base plane and translate them so that the center of the projected cells becomes the origin. Then the remaining cell is crossed by MT if, and only if, the origin is inside the projected MT, which can be easily and efficiently determined by counting how many edges intersect with the positive part of an axis in the projection plane. The cell is intersected by MT only when this is one.
- **Case 4 ($I = 0$):** This last case indicates the most intractable situation in which MT happens to span eight adjacent cells forming a cube, as shown in Figure 2(d). Because two or three of these contain MT's vertices and have been marked already, the remaining cells need to be investigated to find if any of these cross MT. Our experimental test with several algorithms implies that the triangle-box intersection algorithm, proposed by Akenine-Möller [AM01], performs best for this purpose. Based on the separating axis theorem, this algorithm performs three types of tests that check if an AABB, the cell in our case, intersects (1) the triangle's AABB, (2) the plane containing the triangle, and (3) the triangle's three

edges one at a time, terminating as soon as a separating axis is found. In our method, the first group of tests are omitted because the cell is already known to overlap the microtriangle's AABB. Note that some of the five cells to be examined in this fourth case may have been marked already while processing adjacent microtriangles. Therefore, fewer than necessary unmarked cells are usually checked.

3.2. Construction of Triangle Lists

The goal of the first part of our algorithm is to build a triangle list for each boundary cell. It is possible to construct such lists as a given triangular mesh is voxelized. During this process, because we do not know a priori how many triangles are put in each list, the triangle list must be represented using a linked list. This is acceptable if the mesh size and the grid resolution are moderate. However, when complicated models are represented in high resolution, which is the situation that this work targets, a more efficient memory management scheme must be used. In our current implementation, we perform the voxelization step twice. In the first pass, the number of triangles in the list is determined for each local cell, that allows a representation of the triangle list without pointers. Once the necessary memory space is allocated using the information obtained in the first round, the voxelization process is performed again to actually build the lists. Because the voxelization time is trivial compared to the second part of our method, this multipass method only has a slight influence on the entire timing performance.

3.3. An Implementation Issue

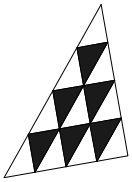


Figure 3: *Black and white microtriangles*

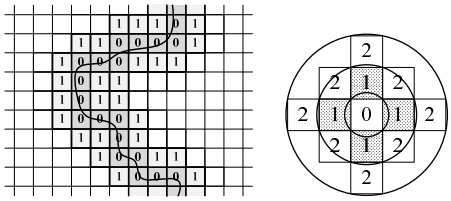
For efficient computation, the generated microtriangles are partitioned into two classes, as illustrated by the two colors in Figure 3, that share vertices and edges. Rather than traverse the microtriangles in scanline order altogether, we first scan the white microtriangles for the intersection test and then repeat the same process for the black microtriangles. The reason for this is that after processing the white microtriangles, there is no need to conduct the third test case ($I = 1$) for the black microtriangles, as the shared edges have been checked already. Therefore, when the black microtriangles are examined in the second round, only the last case ($I = 0$) has to be investigated.

4. Computation of Local Signed Distance

In this work, we only compute the signed distance field at the *local cells*, which are illustrated in Figure 4(a). Here, the boundary cells that cross the polygonal surface are assigned a region number of zero, while the other nonboundary cells, which are the collection of the 6-neighbors of the boundary cells, are allocated as region number one. Note that the region number can be assigned trivially from a simple scan over the boundary cells.

4.1. Computation of the Shortest Distance

In finding the shortest distance of a given local cell, it is critical to minimize the number of candidate triangles that



(a) Local cells (b) Cell layers

Figure 4: *Definition of the local cells and layer level. (a) By the local cells, we denote the boundary cells plus their nonboundary 6-neighbors. (b) Starting at a cell containing a selected voxel, the surrounding cells are piled upon each other like the layers of an onion, approximating to circular bands in voxel space. Refer to Figure 6 to see a three-dimensional illustration of the level-1 and level-2 layers.*

are examined for the distance computation. To locate efficiently the triangles that may contain the shortest point, the neighboring cells around a given local cell are classified as layers, whose level number increases as they are piled upon each other (see Figure 4(b)). These can be viewed as a discrete approximation of circular bands that emanate from the center. See Figure 6 for an illustration of these layers in three-dimensional space.

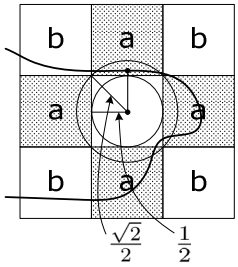


Figure 5: *Cell scan order. The shortest point from a voxel of a boundary cell may exist within the sphere with radius $\frac{\sqrt{3}}{2}$. The 6-neighbor cells, denoted as a, are only examined when the shortest distance obtained using the level-0 cell's triangle list is greater than $\frac{1}{2}$. The remaining level-1 layer cells, denoted as b, don't have to be examined if the shortest distance obtained is less than $\frac{\sqrt{2}}{2}$.*

It can be seen that given a boundary cell, C_0 (with region number 0), the triangle that includes the shortest point from its voxel exists either in the level-0 cell (C_0 itself) or in one of the boundary cells that are in the level-1 layer with respect to C_0. Assuming that d is the shortest distance calculated using only the triangles in C_0's triangle list, then if $d \leq \frac{1}{2}$, there is no need to examine the triangle lists of the remaining level-1 cells of C_0, because the shortest distance from the voxel to any triangle in those lists is longer than $\frac{1}{2}$.

If $d > \frac{1}{2}$, then each level-1 layer cell that is also a boundary cell must be scanned one at a time, calculating the shortest distance using its triangle list and updating the value of d if a shorter distance is found. For efficient computation, we first performed the distance computation over the triangle lists of cells that were the 6-neighbors to C_0 (Figure 6(a)). If the shortest distance calculated from the six cells was shorter than $\frac{\sqrt{2}}{2}$, then there was no need to examine the remaining 12 level-1 layer cells (Figure 6(b)) because the minimum possible distance from the voxel to these 12 cells

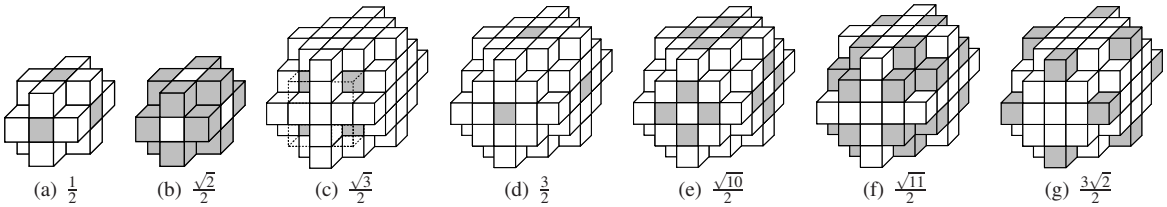


Figure 6: *The minimum distances of layer cells. Figures (a) and (b) illustrate the level-1 layer cells, while the remaining figures display the level-2 layer cells. These layer cells are classified according to their minimum distance to the center cell's voxel. They are used to cull unnecessary voxel-to-triangle computations during the distance computation stage.*

was $\frac{\sqrt{2}}{2}$. Only when the shortest distance found was longer than $\frac{\sqrt{2}}{2}$, which is a relatively rare case, did the distance computation proceed for the remaining cells to search for a possible shorter distance (refer to Figure 5).

For a nonboundary local cell, C_1 (with region number 1), the distance computation was similar, except that the boundary cells that were in the level-1 and level-2 layers with respect to C_1 were examined in the order illustrated in Figure 6. Again, each group of cells was associated with a minimum possible distance, where the triangle list of a cell was tested only if the shortest distance calculated before its group was longer than its minimum distance.

Conceptually, our algorithm consists of two stages: the shortest distance is computed for the boundary cells in the first round and then for the nonboundary local cells in the second round. However, in terms of implementation, it was more efficient to interleave the two stages, computing the shortest distances in parallel, which is summarized as follows.

- **[Initialization]** Set each local voxel's distance to the maximum value.
- **[First pass]** Traverse each boundary cell, C_b, performing the following actions. Find each local cell C_l whose region number, i ($i = 0$ or 1), matches its layer level with respect to C_b. If C_l's current distance is longer than the minimum distance from C_l's voxel to C_b, then calculate the shortest distance using C_b's triangle list. If C_l's current distance is longer than the calculated distance, then replace it with the new distance value.
- **[Second pass]** Traverse each boundary cell, C_b, again, performing the following actions. Find each local cell C_l, whose region number i ($i = 0$ or 1) is one less than the layer level with respect to C_b. If C_l's current distance is longer than the minimum distance from C_l's voxel to C_b, then calculate the shortest distance using C_b's triangle list. If C_l's current distance is longer than the calculated distance, then replace it with the new distance value.

4.2. Assignment of a Sign to Local Voxels

After the local distance field was constructed, the local voxels were classified as being inside or outside the object and were assigned the appropriate sign. To determine the sign of the voxels, we employed the angle-weighted pseudonormal technique, which is known to offer robust and numerically stable decisions [BA05].

5. Experimental Results

To verify the effectiveness of our ideas, we implemented our algorithm on a 3.2 GHz Intel Xeon CPU and tested the implementation using several polygonal models with different complexities from The Stanford 3D Scanning Repository. To obtain the experimental results presented in Table 1 to 4, we applied three levels of grid resolution to each model, as follows: A: $107 \times 250 \times 107$ (1x), B: $210 \times 500 \times 210$ (2x), C: $312 \times 750 \times 313$ (3x) for Happy Buddha, D: $169 \times 250 \times 12$ (1x), E: $337 \times 500 \times 18$ (2x), F: $504 \times 750 \times 23$ (3x) for Vellum Manuscript, G: $250 \times 142 \times 168$ (1x), H: $500 \times 281 \times 335$ (2x), I: $750 \times 420 \times 501$ (3x) for Asian Dragon, and J: $151 \times 250 \times 132$ (1x), K: $300 \times 500 \times 260$ (2x), L: $448 \times 750 \times 388$ (3x) for Thai Statue.

5.1. Analysis of the Voxelization Method

Table 1 presents the experimental results of our voxelization technique, where the timing (in milliseconds) of our method was only the time taken when determining the boundary cells, excluding the time required for building the triangle lists. Our object space voxelization algorithm repeatedly called for the expensive float-to-int type casting operation to determine which cell contained a given vertex. During the experiments, we set up the option "Fast floating-point model" in Microsoft Visual C++ 2005, which accelerated the casting operation markedly without leading to any precision errors.

For a more precise evaluation, our voxelization method was compared with a typical scanline method [Kau87]. As expected, the scanline method was found to be faster than our method. However, the numbers of cells found by both methods can be compared in the 'B-cell' and 'Cell' columns. As emphasized in this paper, our method exactly located the boundary cells that crossed the triangular surfaces. On the other hand, the simple scanline algorithm, while faster, obviously failed to mark all the boundary cells. This weakness became more severe when the relative size of the triangle over the cell increased, i.e., the grid resolution increased, (for example, when the grid resolution was $321 \times 750 \times 313$, the scanline method located approximately 70% of the boundary cells for the Happy Buddha data).

The voxelization time increased as the volume resolution increased, because the larger triangles ranging over multiple cells generated more microtriangles and more complicated situations that were difficult to handle. This is experimentally demonstrated in Table 2, where the ratio of the most ex-

(Time unit: ms)

Object	Res.	Ours		Scanline	
		Time	B-cell	Time	Cell
Happy Buddha (1,087,716)	A	253.51	117,855	181.25	103,514
	B	501.49	488,720	227.84	377,874
	C	825.21	1,111,966	285.23	791,105
Vellum Manuscript (4,305,679)	D	584.93	41,556	446.06	41,100
	E	615.33	171,373	494.14	168,128
	F	667.71	392,312	512.96	380,435
Asian Dragon (7,218,906)	G	1,087.45	84,424	787.89	81,606
	H	1,318.79	351,597	869.83	326,886
	I	1,419.21	802,734	956.23	717,455
Thai Statue (10,000,000)	J	1,746.99	119,727	1,396.31	115,660
	K	1,943.07	505,870	1,484.36	468,206
	L	2,288.25	1,156,530	1,613.67	1,025,354

Table 1: *Comparison of our voxelization technique with the standard scanline method [Kau87]. The figures in parentheses indicate the number of polygons of the respective models. Our method, while slower than the scanline method, robustly marked all the boundary cells. One can compare the actual number of boundary cells (B-cell column) with the number of cells found by the scanline method (Cell column).*

Object	Res.	M-tri	C1,2	C3	C4
Happy Buddha (1,087,716)	A	1,241,251	0.78	0.19	0.03
	B	2,498,768	0.58	0.35	0.07
	C	3,860,249	0.43	0.43	0.14
Vellum Manuscript (4,305,679)	D	4,307,312	0.98	0.02	0.00
	E	4,311,413	0.92	0.08	0.00
	F	4,318,054	0.82	0.17	0.01
Asian Dragon (7,218,906)	G	7,220,368	0.97	0.03	0.00
	H	7,225,657	0.89	0.10	0.01
	I	7,265,321	0.77	0.20	0.03
Thai Statue (10,000,000)	J	10,277,117	0.96	0.03	0.01
	K	11,145,247	0.86	0.13	0.01
	L	13,051,184	0.74	0.23	0.03

Table 2: *The frequencies of the four cases occurring during voxelization. The figures in the 'M-tri' column represent the number of microtriangles produced. As shown in the table, the first two cases (C1 and C2), which incur no additional cost, occur frequently, whereas the most expensive case (C4) occurs with a low frequency.*

(Time unit: sec.)

Object	Res.	L-cell	Time		
			Ours	CPT	BBox
Happy Buddha (1,087K)	A (1x)	258K	5.11(1.20)	19.01	11.55
	B (2x)	1,086K	9.93(2.69)	20.25	12.87
	C (3x)	2,475K	16.10(4.89)	22.12	14.64
Vellum Manuscript (4,305K)	D (1x)	117K	8.4(1.92)	76.65	35.59
	E (2x)	478K	10.92(2.22)	77.11	36.04
	F (3x)	1,086K	13.81(2.67)	77.19	36.77
Asian Dragon (7,218K)	G (1x)	176K	15.29(3.96)	116.71	66.69
	H (2x)	741K	22.29(5.28)	119.51	68.93
	I (3x)	1,695K	30.88(7.52)	123.72	71.38
Thai Statue (10,000K)	J (1x)	257K	25.35(6.70)	166.31	99.74
	K (2x)	1,111K	35.48(8.59)	169.81	101.81
	L (3x)	2,560K	47.77(11.69)	176.25	105.28

Table 3: *Timing results for building the local signed distance field. Here, the figures in parentheses in the column 'Ours' denote the time taken for voxelizing the triangular mesh and then constructing the triangle lists for all the boundary cells. As expected, it was observed that the number of local cells found (in thousands) shown in the 'L-cell' column increased quadratically with respect to the grid resolution.*

pensive fourth case (C4) is shown to increase with the number of microtriangles created (M-tri), with increasing relative triangle size. Our voxelization method can be viewed as a tessellating, if necessary, of triangles into the 'tractable' (the first three cases) and 'intractable' (the fourth case) regions, treating the latter case more carefully. Considering that this costly fourth case is rare, particularly when the models are very large, and that the number of boundary cells found (the B-cell column in Table 1) increases faster than the voxelization time (Time) (in other words, the time complexity of the voxelization computation is, at worst, linear with respect to the number of located boundary cells), then we can say that the voxelization process of building the precise volumetric boundary representation was well optimized for very large models.

5.2. Analysis of the Method of Building the Local Signed Distance Field

Table 3 shows the performance statistics collected during the process of building the local signed distance field, where the values in the 'Ours' column represent the total times (in seconds) taken using our method. As can be seen in the 'L-cell' column, the number of local cells around the two-dimensional manifolds increased quadratically. When the local signed distance field was constructed, the distance calculation was the most expensive elementary operation. For a fixed grid resolution, it appears that the total computation time is linearly proportional to the total number of local cells. However, this is not the case, as the average size of the triangle list per boundary cell decreases with increasing resolution, reducing the overall frequency of the distance computation. Experimentally, a much slower increase was observed.

We also tested two prospective methods for a comparison with other relevant methods. The first method we examined was the Closest Point Transform (CPT), developed by Mauch [Mau03], which is appropriate for computing the signed distance field of polygonal models of moderate size up to a given maximum distance. In this method, special polyhedra serving as truncated Voronoi cells are constructed, and then, these are scan converted one at a time to build the local distance field. This algorithm is efficient, in that its time complexity is linear in both the number of voxels for which the distance is computed and in the number of triangles used.

The second method we investigated was a simple bounding-box method, which computed the local distances as follows. *For each triangle of the polygonal model, a tight axis-aligned bounding box is built and expanded to a proper*

width. Then, examine all the voxels inside the bounding box, immediately rejecting those farther from the triangle than the width. For each remaining voxel, calculate the distance to the triangle and store this value to the voxel if a shorter distance is found.

In Table 3, the timing performances of these two methods are shown in columns 'CPT' and 'BBox', respectively, where the CPT code, cordially provided by Mauch [Mau07], was run for timing the CPT algorithm. As clearly indicated by these results, our method compares very favorably with the two methods tested. While our method was competitive for large polygonal models, all three methods showed different characteristics in their timing performance. First, our method and the simple bounding-box method performed better than the CPT approach when the distance of the local cells within three layers, as illustrated in Figure 4(a), was computed. This is because the times for constructing the polyhedra and for preparing the scan conversion dominated the time for computing the distance of a relatively small number of cells. However, the efficiency of the CPT method increases relatively as the local distance is computed for a thicker layer, as the setup times are amortized over more cells. The setup time also becomes less severe for either less complex polygonal models or for higher-resolution grids.

Compared with the bounding-box approach, our method was similar, in that both methods attempted to build the local distance field by locating the neighboring cells first and then calculating the distance from them. The major difference is that our method scanned the local cells more cleverly, so that any unnecessary distance computation between voxels and triangles was culled as much as possible. Because the distance calculation is the most expensive operation in building the local distance field, this effort to minimize the number of distance calculations has a marked influence on the timing performance.

5.3. Our Method as an Initializer for the Propagation Method

In many applications in computer graphics, the distance fields must often be constructed in the entire volume domain or within a band region with an arbitrary thickness. One of the most competitive solutions for this problem is the fast sweeping method, proposed by Zhao [Zha04]. This propagation method solves the Eikonal equation $|\nabla d(\mathbf{x})| = f(\mathbf{x}), \mathbf{x} \in R^n$, with the boundary condition $d(\mathbf{x}) = \phi(\mathbf{x}), \mathbf{x} \in \Gamma \subset R^n$. The first-order hyperbolic partial differential equation may be applied to building the distance field by setting $f(\mathbf{x}) = 1$ and $\phi(\mathbf{x}) = 0$. The easy-to-implement sweeping method offers an optimum linear-time complexity with respect to the number of voxels for which the distance is evaluated, and the only difficult part in using this method is enforcing the boundary condition, $d(\mathbf{x}) = 0, \mathbf{x} \in \Gamma \subset R^n$. In fact, our local distance computation method was very well suited as an initializer for the propagation method.

Table 4 shows some selective timing results (in seconds), obtained when the local signed distance field, built using our method, was used as an initial condition for the fast sweeping method to build the signed distance field within a thick band region with a width = 60. When the sweeping method was applied, we first assigned a proper sign to each voxel in the band region before its Gauss–Seidel iterations began for fast sweeping.

(Time unit: sec.)

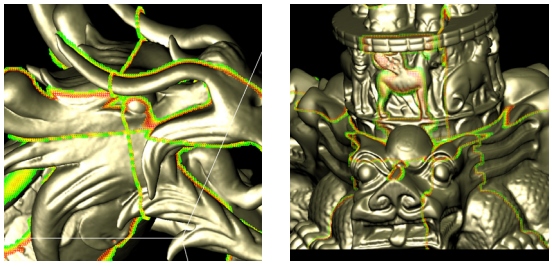
Object	Res.	Ours+Sweeping		CPT
		Local	Sweep	
Happy Buddha (1,087,716)	A	5.11	1.40	562.25
		(2,622,959)		(2,620,213)
	B	9.93	9.79	701.40
		(14,797,764)		(14,628,477)
	C	16.10	29.67	757.38
		(38,064,117)		(37,489,486)
Asian Dragon (7,218,906)	G	15.29	2.63	1,770.46
		(3,272,140)		(3,203,092)
	H	22.29	17.49	1,866.44
		(13,425,491)		(12,911,713)
	I	30.88	56.75	1,898.10
		(30,208,238)		(29,181,039)
Thai Statue (10,000,000)	J	25.35	2.20	3,552.08
		(2,918,966)		(2,905,718)
	K	35.48	15.17	3,773.54
		(13,599,690)		(13,488,835)
	L	47.77	48.78	3,860.48
		(34,085,470)		(33,761,899)

Table 4: *Our method as an initializer for a propagation method. Our method (timed in the 'Local' column) was combined with the fast sweeping method [Zha04] (timed in the 'Sweeping' column) to build a signed distance field within a band region with a width 60 times as large as the cell width. The figures in parentheses indicate the actual number of cells processed, where our method found slightly more cells than the CPT method did.*

The computation times were also compared with those obtained using the CPT method, which is also a linear-time algorithm. As clearly implied in the results, the combination of our initializer and the fast sweeping method was a very efficient tool for computing the global signed distance field of large polygonal models. The timing difference between the two computational schemes increased rapidly when either a thicker band region was considered or when the number of triangles increased. Note that, unlike the CPT algorithm, the fast sweeping method does not compute the exact Euclidean distance. It only constructs a signed distance field having a unit gradient vector, i.e., $|\nabla d(\mathbf{x})| = 1$. While not a geometric distance field, the field produced by the sweeping method is sufficient for many computer graphics applications.

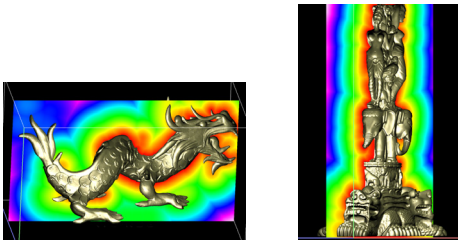
6. Conclusions

We have presented an efficient method for the computation of the local signed distance field for large polygonal models and have analyzed its performance using several complicated models and high-resolution grids. As a side effect, we obtained a robust voxelization scheme, capable of finding all the cells that intersect a given polygonal model, including those missed by the scanline method, for a small additional cost. The local signed distance fields built using our method



(a) Asian Dragon (b) Thai Statue

Figure 7: *Local signed distance field. Three axis-aligned slices, illustrating the computed local signed distance field, are superimposed on the sample meshes.*



(a) Asian Dragon (b) Thai Statue

Figure 8: *Global signed distance field. The global signed distance function $d = d(\boldsymbol{x})$, discretized by the sweeping method combined with our method, is color-coded, increasing from red to green to blue to white.*

can themselves be used effectively in applications where up to the first-order geometric information, i.e., tangential information, of polygonal models is sufficient (as three layers of cells are guaranteed in each principal direction, the second-order information, i.e., curvature information, may be approximated at the voxels of boundary cells if the forward difference is applied). Just as importantly, our method can also serve as an initial condition for a propagation method that builds the global signed distance field efficiently. Our test results demonstrate that our method offers an easy-to-implement and robust computational scheme for building a signed distance field, which becomes more effective in cases where the polygonal models are large.

Acknowledgements: This work was supported by the Korea Science and Engineering Foundation grant funded by the Korea government (MOST) (No. R01-2007-000-21057-0).

References

[AM01] AKENINE-MÖLLER T.: Fast 3D triangle-box overlap testing. *Journal of Graphics Tools 6*, 1 (2001), 29–33.

[BA05] BÆRENTZEN J., AANÆS H.: Signed distance computation using the angle weighted pseudo-normal. *IEEE Transactions on Visualization and Computer Graphics 11*, 3 (2005), 243–253.

[BMW00] BREEN D., MAUCH S., WHITAKER R.: 3D scan conversion of CSG models into distance, closest-point and colour volumes. In *Volume Graphics*, Chen M., Kaufman A., Yagel R., (Eds.). Springer, London, 2000, ch. 8, pp. 135–158.

[CK90] COHEN D., KAUFMAN A.: Scan-conversion algorithms for linear and quadratic objects. In *Volume Visualization*, Kaufman A., (Ed.). IEEE Computer Society Press, 1990, pp. 280–301.

[HICK*99] HOFF III K., CULVER T., KEYSER J., LIN M., MANOCHA D.: Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proceedings of ACM SIGGRAPH 1999* (1999), pp. 277–285.

[HLC*01] HUANG J., LI Y., CRAWFIS R., LU S., LIOU S.: A complete distance field representation. In *Proceedings of IEEE Visualization 2001* (2001), pp. 247–254.

[HNB*06] HOUSTON B., NIELSEN M., BATTY C., NILSSON O., MUSETH K.: Hierarchical RLE level set: a compact and versatile deformable surface representation. *ACM Transactions on Graphics 25*, 1 (2006), 151–175.

[HYVK98] HUANG J., YAGEL R., V. F., KURZION Y.: An accurate method for voxelizing polygon meshes. In *Proceedings of the 1998 Symposium on Volume Visualization* (1998), pp. 119–126.

[JBS06] JONES M. W., BÆRENTZEN J. A., SRAMEK M.: 3D distance fields: a survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics 12*, 4 (2006), 581–599.

[Kau87] KAUFMAN A.: An algorithm for 3D scan-conversion of polygons. In *Proceedings of Eurographics 1987* (1987), pp. 197–208.

[Mau03] MAUCH S.: *Efficient algorithms for solving static Hamilton-Jacobi equations*. PhD thesis, California Institute of Technology, Pasadena, California, 2003.

[Mau07] MAUCH S.: Sean's Templete Library. `http://www.cacr.caltech.edu/~sean/projects/stlib/stlib.html`, September 2007.

[NNSM07] NIELSEN M., NILSSON O., SÖDERSTRÖM A., MUSETH K.: Out-of-core and compressed level set methods. *ACM Transactions on Graphics 26*, 4 (2007), 16.

[Set96] SETHIAN J.: A fast marching level set method for monotonically advancing fronts. In *Proceedings of Nat. Acad. Sci.* (1996), vol. 93, pp. 1591–1595.

[SGG*07] SUD A., GOVINDARAJU N., GAYLE R., ANDERSON E., MANOCHA D.: Surface distance maps. In *Proceedings of Graphics Interface 2007* (2007), pp. 35–42.

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

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

[SPG03] SIGG C., PEIKERT R., GROSS M.: Signed distance transform using graphics hardware. In *Proceedings of IEEE Visualization 2003* (2003), pp. 19–24.

[Tsi95] TSITSIKLIS J.: Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control 40*, 9 (1995), 1528–1538.

[VKK*03] VARADHAN G., KRISHNAN S., KIM Y., DIGGAVI S., MANOCHA D.: Efficient max-norm distance computation and reliable voxelization. In *Proceedings of Eurographics Symposium on Geometry Processing 2003* (2003), pp. 116–126.

[WK03] WU J., KOBBELT L.: Piecewise linear approximation of signed distance fields. In *Proceedings of Vision, Modeling, and Visualization 2003* (2003), pp. 513–520.

[Zha04] ZHAO H.: A fast sweeping method for Eikonal equations. *Mathematics of Computation 74*, 250 (2004), 603–627.