Distance Field Transform with an Adaptive Iteration Method

Fan Chen Ye Zhao

Department of Computer Science, Kent State University
Kent, Ohio, USA, Email: [fchen, zhao]@cs.kent.edu

***Abstract*—We propose a novel distance field transform method based on an iterative method adaptively performed on an evolving active band. Our method utilizes a narrow band to store active grid points being computed. Unlike the conventional fast marching method, we do not maintain a priority queue, and instead, perform iterative computing inside the band. This new algorithm alleviates the programming complexity and the data-structure (e.g. a heap) maintenance overhead, and leads to a parallel amenable computational process. During the active band propagating from a starting boundary layer, each grid point stays in the band for a lifespan time, which is determined by analyzing the particular geometric property of the grid structure. In this way, we find the Face-Centered Cubic (FCC) grid is a good 3D structure for distance transform. We further develop a multiple-segment method for the band propagation, achieving the computational complexity of $O(m \cdot N)$ with a segment-related constant m.**

***Keywords*—Distance Transform, GPU, Iteration, Wavefront Propagation, Narrow Band, FCC, Multiple-segment**

1. Introduction

A discrete distance field provides an implicit representation of a geometric shape, which is defined by a collection of sampling points inside an enclosing domain of the shape. Each sampling point stores the smallest distance from itself to the interested shape. Usually, these sampling points are grid points on a rectangular grid in 2D or 3D domain. The distance can be defined in terms of arbitrary metrics, while the Euclidean distance is the most popular model for graphical applications. As an implicit shape modeling scheme, the distance field is widely used in many important applications, such as image segmentation and processing, 3D shape editing, smoothing, morphing, collision detection, topology operations and volume graphics.

Therefore, distance field generation has been an essential research topic in computer vision, graphics and visualization, as well as applied mathematics. A variety of approaches have been proposed to address the problem that can be described as solving an Eikonal equation. Most recent endeavors adopt a strategy based on the distance field transform. Instead of direct computing the closest distance from every point to the shape, only the grid points belonging to a boundary band close to the shape is computed, from which the remaining distances are evaluated by distance propagating to the rest of the volume. The distance propagation algorithms can be categorized into two main strategies:

Domain Sweeping: Distance propagation starts from some corners of the rectangle grid to the whole domain by a predefined sequential order related to the axial directions. For example, for a 3D rectangular domain $[0..NX]\times[0..NY]\times[0..NZ]$, distance information propagates from $(0,0,0)$ to $((NX-1),(NY-1),(NZ-1))$ by traversing all grid points in an order of: first $x-$direction, next $y-$direction, then $z-$direction. Such distance transform does not consider the arbitrary locations of the starting bands providing the basis of propagation. Obviously, one such propagating traversal cannot accomplish the task. Typically several passes of traversal in different directional orders are required.

Front(contour) Propagation: Taking the initial band into consideration, the propagation starts from the grid points inside the band and transfers the known distance information to their neighbors until the whole domain is computed. The adaptive approach guarantees distance transform in an increasing order, which is implemented by exploiting a special priority data structure (e.g. a sorted list or a heap) storing a sorted active band. The priority data structure maintains grid points being used for transform with the discrepancy among their distance values. The scheme only retrieves a grid point with the shortest distance from the band, and propagates the distance to its neighbors not in the band. Thus, it avoids backtracking over previously evaluated grid points and enables fast marching and correct results.

The sweeping methods go through the $N = NX \times NY \times NZ$ grid points with several (a constant value) passes, and thus achieve asymptotic complexity $O(N)$. In comparison, the front propagation methods have the complexity of $O(NlogN)$ due to the heap maintenance efforts. Though it seems the former is advantageous, the latter provides a more flexible approach when a particular limit of distance is required to compute. When answering a query "give me the points with distance smaller than d_l?", it can easily stop computing during propagation and provide correct results. A small d_l is usually imposed in many graphical applications, where

the front propagation methods can provide a faster response than the sweeping approaches that have to complete the whole domain computation. Both strategies have been widely studied and achieved great success, we refer interested readers to a good survey [9] for detailed analysis. A common disadvantage of both strategies is that none of the approaches can be easily parallelized. Because both are established on an algorithmic basis of sequential processing. The sweeping methods compute distance transform on grid points one point to another according to particular traversal sequences. The front spreading methods process the active band with a sorted sequential order.

The distance transform can be performed in a parallel computational scheme. A straightforward idea goes back to an approach utilizing iteration strategy [19]: At a time step T, each grids point asks all its neighbors about their current distance, and then updates the distance of itself by finding the smallest among all the values propagated from these neighbors respectively. The updated distance value is the new distance of each point at time $T + 1$. When all the points have a converged distance value, i.e., the value does not change in consecutive time steps, the distance field of the whole domain is achieved. This strategy may take many steps to reach the convergence status, resulting in relatively slow computational performance. However, in this approach each grid point can be processed concurrently at each time step, which enables parallel computing, which is critical for exerting the computational power of modern multi-core CPUs and other parallel architectures. However, this approach is not adaptive and cannot achieve the early termination for queries with a limited distance value.

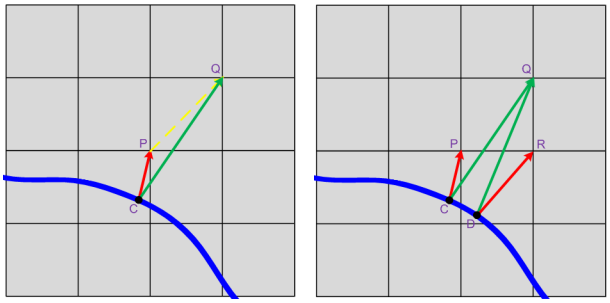
In this paper, we propose a new approach that has (1) the ease of simple programming and the parallel computing capability by an iterative scheme; and (2) the controlling ability to handle particular distance limits of propagating fronts, enabling the query-based computation. In order to provide front propagation features, we enable a similar narrow band to manage the active grid points; however, to apply a parallel computing scheme, these points do not have different priorities as in the fast marching method [21]. Our method utilizes an adaptive iteration on the active band to fulfill the requirements. Without distinguishing points inside the band by a priority-enabled data structure, the algorithm provides correct distance results only if a grid point is active in the narrow band until its distance will no longer have the possibility to be updated. Therefore, each grid point will be assigned a lifespan monitoring its existence inside the band. The lifespan is determined by the structural feature of the domain-decomposition grids. We examine the geometric properties of several grid structures to find the theoretical lifespan of an arbitrary grid point, and then, use it to control how long a grid point should stay in the band. Moreover, we exploit a multiple-segment narrow band propagation algorithm to further reduce the complexity and improve the performance.

2. Related Work

A distance field represents surfaces or curves with implicit representation, which has been extensively used in shape modeling purposes in computer graphics, visualization and computer vision [3], [28], [16], [1], [15]. A distance field can be generated directly by computing from each point to a geometric primitive. An interactive algorithm computes 3D Euclidean distance fields on GPU [23] by rasterizing the distance vectors from the points on the slice to the primitives.

Related to our research, distance transform is a general approach to form the distance field by propagation from a starting set computed by direct geometric and analytic algorithms. Iteration methods are first applied to solve the shape-from-shading problem on the whole domain [14], [19] in numerical computing field. Another strategy is to propagate the distances to neighbors with special templates through the domain. The template can be designed based on chamfer distance [10], or more precisely, on vector distance [20]. Fast marching methods [24], [2], [21] are proposed to compute the arrival time of a wavefront expanding in the normal direction at an active band of grid points, which actually solve the Eikonal equation from a given boundary condition. Zhao et al. [26], [18] present a sweeping method to solve the Eikonal equation by Gauss-Seidel iterations with alternating sweeping ordering on rectangle grid or meshes. Fast sweeping method is applied for static convex Hamilton-Jacobi equations [17]. Frisken et al. [6] propose a hierarchical computation for distance field generation. For more details, we refer the interested readers to good surveys on 2D [5] and 3D distance transform [9].

Except the whole-domain iteration methods, these basic algorithms cannot be easily parallelized due to their computational schemes. Several approaches employ specially-designed GPU algorithms, such as tile-based updating scheme of fast marching [8], domain division of sweeping method [27], and delicate narrow band packing [11]. Working on polygon meshes, a method based on scan-conversion of the mesh is proposed by Mauch [12], [13]. Sigg et al. improve their algorithm with hardware implementation [22]. Another approach is to construct a Voronoi diagram, which leads to a distance field generation of 2D and 3D polygons [7]. Weber et al. present a parallel algorithm for approximation of geodesic distances on geometry images [25]. These methods process the geometric elements independently, and thus can utilize the parallel nature of graphics hardware to achieve good performance and success in distance generation. In comparison, our method does not rely on the meshes to represent geometric shapes. In our approach, we utilize the inherent parallel scheme in iteration methods, which provide a wavefront scheme to further improve the performance and flexibility.



(a) (b)

Fig. 1. Vector based distance propagation.

3. Distance Field Transform

3.1 Definition

A distance field represents surfaces or curves with implicit representation, which has been broadly used in shape modeling purposes in computer graphics. A distance field is defined as a scalar field that specifies a distance to a shape, where the distance is usually signed to distinguish between the inside and outside of the shape. Data set X representing a distance field to surface S is defined as: $X : R^3 \rightarrow R$ and for $p \in R^3$,

$$X(p) = sgn(p) \cdot min\{|p - q| : q \in S\} \qquad (1)$$

where $sgn(p) = 1(-1)$ if p is inside (outside) of S, and $||$ is the Euclidean norm.

The distance transform computes the distance field of all the points, $p \in R^3$, from an initial starting set with known distance values. The starting set stores the distance of points in a boundary layer of S, which is computed by geometric or analytical algorithms.

3.2 Vector-Based Distance Transform

We compute the distance transform from one point to its neighbors by a vector propagation method, in which the distance is represented as a vector from a grid point to the closest point on the surface. As illustrated in Figure 1a, a known distance of P is represented as a vector $\overrightarrow{CP}$, where C is the closest point on the surface to P. When P propagates to its neighbor Q, the distance of Q will be computed by

$$\overrightarrow{CQ} = \overrightarrow{CP} + \overrightarrow{PQ}. \qquad (2)$$

Here, $\overrightarrow{PQ}$ is the constant vector between two neighbors.

Note that the length of $\overrightarrow{CQ}$ is used as the distance of Q propagated from P, though theoretically C is the closest point to P not Q. This is the assumption of all distance transform methods, which can lead to computational errors related to the grid resolution.

The grid point Q will be able to compute its distance from other neighbors in the same way as from P. For example, in Figure 1b, another neighbor R also provides Q's distance represented by $\overrightarrow{DQ}$. The actual distance of Q for next step is one of these distance vectors with the smallest length.

3.3 Our Computational Scheme

The domain sweeping methods control the directions and orders of propagation. That is, Q is only allowed to compute the distance vector from particular neighbors in each pass, so that after a few different passes, every point achieves its shortest distance to surfaces. In comparison, the front marching methods only allow one existing point with the smallest distance to find its inactive neighbors and propagate. As a result, each neighboring point guarantees to acquire its shortest distance (see [21] for proof). In the conventional iteration method, each point Q of the whole domain computes many temporary distance vectors from all the neighbors and choose the smallest for the next time step. In an arbitrary time step, Q may not have achieved its final distance value. After many iteration steps (usually proportional to O(N)), eventually, all the grid points will approach their correct closest distance. Such convergency is achieved when all the points no longer change (in reality, within a very small error tolerance) their distance value in consecutive steps.

In our approach, the iteration computation applies only on an active narrow band, a small portion of the whole domain, to improve performance. Furthermore, we propose a new algorithm that does not maintain the priority queue inside the narrow band. Therefore, our method provides an adaptive iteration method for distance transform. In the next section, we describe the details of our active band propagation algorithm and the iteration strategy based on a point's lifespan.

4. Active Band Scheme

4.1 Propagation Procedure

At the beginning, grid points in a boundary layer closely enclosing the interested shape obtain their distance vectors by direct geometrical computing. This starting set initiates the active narrow band, NB. This band stores all the grid points, to whom the evolving distance transform front has been propagated and whose distance has the probability to be updated in future steps. The band will evolve along time steps by adding new points, and by removing points having achieved their final distance.

Assuming the distance d_P^T of each point P inside the band NB is known at a time step T, each of its neighbors, Q, is considered. Next, we use the method described in Section III-B to compute a temporary distance of each Q. In this way, each Q will have a group of temporary distances and it will use the one with the smallest magnitude as its distance d_Q^{T+1} at next time step $T + 1$. Q can be a point newly added to NB in $T + 1$. It can also be a point already existing in NB before this time step. This illustrates the difference between our method and the fast marching methods.

We are facing a challenge of how to remove a point from NB. Due to the use of the the narrow band, the iteration convergence rule, which examines

whether points no longer update their distances, cannot be applied. Because the rule is only valid when all points in the whole domain are computed together. Instead, we seek a solution by investigating the structural property of the grid to define the lifespan of a point (i.e. how long a point should stay in NB). In other words, we want to find the time step when a point has achieved the correct final distance with no chance to be updated again, and then remove it from NB.

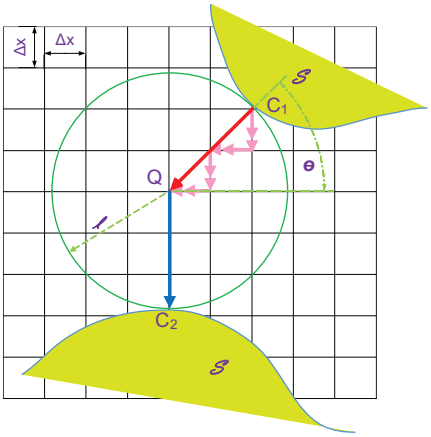


Fig. 2. Distance transform on a rectangular grid.

4.2 Lifespan of a Point

The lifespan of a point can be determined from the geometry of a grid. As illustrated in Figure 2, for a point Q with distance l to the shape S, all the possible closest points on S to Q must reside on a virtual circle (in 3D, a sphere). Considering two possible locations of the closest point, $C_1 \in S$ and $C_2 \in S$, each of them transfers the closest distance information to Q however in different time steps with respect to the grid structure. When we only allow the distance transform to happen between axial neighbors (neighbors along x and y axes) for each time step, C_2 will use $t(C_2) = \frac{l}{\Delta x}$ steps to reach Q. However, C_1 will use $t(C_1) = \frac{l(cos(\frac{\pi}{4})+sin(\frac{\pi}{4}))}{\Delta x}$ steps following the pink path. Theoretically, for any possible closest point $C(\theta) \in S$ with an angle θ, with respect to the symmetry, the time step used to transform the distance information to Q is defined as

$$\begin{aligned} t(C(\theta)) &= \frac{l(cos\theta + sin\theta)}{\Delta x} \\ &= \frac{l(\sqrt{2}cos(\theta - \frac{\pi}{4}))}{\Delta x}, \theta \in (0, \frac{\pi}{4}). \end{aligned} \quad (3)$$

From this equation, the minimum and maximum of possible time steps are computed as $t^Q_{min} = t(C(\theta = 0)) = t(C_2)$ and $t^Q_{max} = t(C(\theta = \frac{\pi}{4})) = t(C_1)$.

A point Q in the square grid should be kept in the active band NB from t^Q_{min} to t^Q_{max} time steps, since it will have the possibility to be updated to a smaller distance by the iteration between the time range. From Equation 3, we get

$$\begin{aligned} t^Q_{max} &= \sqrt{2} \cdot t^Q_{min} \\ &= t^Q_{min} + (\sqrt{2} - 1) \cdot t^Q_{min}. \end{aligned} \quad (4)$$

Notice that t^Q_{min} is the first time Q is added to NB, and therefore, we can remove Q from NB after $((\sqrt{2} - 1) \cdot t^Q_{min})$ time steps. In this way, we guarantee that a point has approached its final distance upon leaving NB. So finally, when NB is empty with no new points added and all points being removed, the distance field generation completes.

It is obvious that the propagation pattern, i.e., how a point propagates to its neighbors, can affect the lifespan. In a 2D square grid (Figure 2), if we allow the distance propagation between diagonal neighbors and axial neighbors. In this case, the time steps from C_1 to Q is $t(C_1) = \frac{lcos(\frac{\pi}{4})}{\Delta x} = \frac{l}{\sqrt{2} \cdot \Delta x}$, which is now smaller than that from C_2: $t(C_2) = \frac{l}{\Delta x}$. Here, we find $t^Q_{min} = t(C_1)$ and $t^Q_{max} = t(C_2) = \sqrt{2} \cdot t(C_1)$. Therefore, Equation 4 is still valid for the grid. However, with propagation in the diagonal directions, the speed of the distance transform is improved, since a point can be added to the active band faster with a smaller t^Q_{min}.

From the analysis above, we find that the lifespan of a point in NB is between t_{min} and t_{max}. Such lifespan is determined by the grid structure. Please note that when a general Eikonal equation is considered, a link between neighboring points has a weight value defined by a particular function. Therefore, we cannot find a uniform grid scale Δx or Δy, which requires further intensive study and is on our future research agenda.

4.3 Grid Structures and Lifespan Coefficient

We have shown that a point Q should stay in NB for $((\sqrt{2} - 1) \cdot t_f) \approx (0.414 \cdot t_f)$ time steps from Equation 4. $t_f = t^Q_{min}$ is the time step it is first added to NB and is proportional to the length l as described in Figure 2. A lifespan coefficient can be defined as $\alpha \approx 0.414$, which determines the lifespan $(\alpha \cdot t_f)$. For a 3D domain, the lifespan coefficient is defined in the similar way as $\alpha = (\sqrt{3} - 1) \approx 0.732$. If l has a large value, then the point will stay in NB for a significant time $(0.732 \cdot t_f)$. In the implementation, the lifespan of time steps will need to use an integer value, $Ceiling(\alpha \cdot t_f)$. We can reduce the lifespan of a point by changing the coefficient α to a smaller value, by using a different grid structure.

For example, we show a 2D triangular grid in Figure 3. Applying the similar geometric analysis, we can also describe the maximum and minimum for this grid structure. We get $t_{min} = \frac{l}{\Delta x}$ and $t_{max} = \frac{l}{cos(\frac{\pi}{12})\Delta x}$, and then the lifespan coefficient $\alpha \approx 0.04$.

For 3D distance transform, we investigate the Face-Centered Cubic (FCC) grid structure, which has better sampling efficiency compared with the cubic grid. An FCC grid consists of simple Cubic Cartesian (CC) cubic cells with additional sampling points located at the center of each cell face. Each point in the FCC grid has

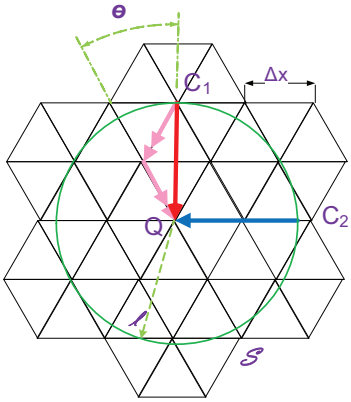


Fig. 3. Distance transform on a triangle grid.

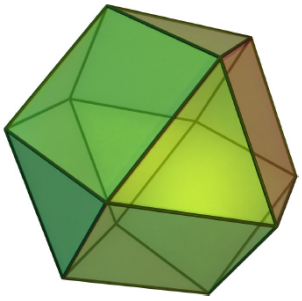
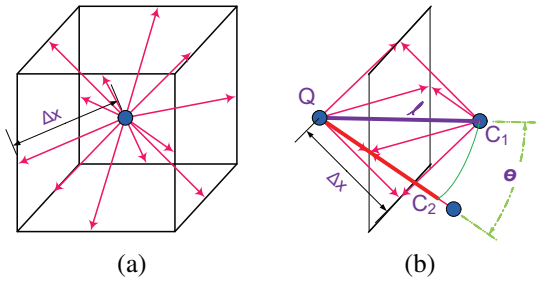


Fig. 4. The 12 neighbors of an FCC lattice site forms a cuboctahedron (Image courtesy of Dr. Feng Qiu).

direct links to a total number of 12 nearest neighbors, in contrast to 6 in the cubic grid. This is the best angular discretization rate that any 3D regular grid can achieve, since according to [4], in R^3 the maximum number of sphere of radius 1 that can simultaneously touch the unit sphere (kissing number) is 12. This unique feature has important implications for sampling and interpolation. The details can be found in [4]. For example, the 12 links of an FCC grid point are symmetric under rotation and reflection, which presents a relatively simple geometric structure leading to a smaller coefficient α. Figure 4 shows the cuboctahedron defined by the 12 closest neighbors of an FCC site. The FCC grid can be easily



(a) (b)

Fig. 5. Distance transform on an FCC grid.

created from a cubic grid. As shown in Figure 5a, for each grid point, only the links to its 12 neighbors with link length $\sqrt{2}$ are allowed and thus we can construct an FCC grid. We use the geometric analysis to find lifespan coefficient. Now, for a point Q with distance l to the S, the closest point on S to Q must reside on a sphere. Figure 5b illustrates two possible locations, C_1 and C_2, by looking in the neighborhood of Q, where the pink links represent available transform routes. In this case, Q, C_1 and C_2 are on the same plane. We use the similar analysis as for Figure 2, $t_{max} = t(C_2) = (\sqrt{2} \cdot t(C_1))$. Note that C_1 and C_2 could be on other locations which has $\alpha < 0.414$. However, since $(0.414 \cdot t_f)$ is the maximum possible lifespan, Q will have to stay in the active band for the lifespan $(0.414 \cdot t_f)$, thus leading to an improvement of α from 0.732 of cubic grid to 0.414 of FCC grid.

5. Computational Procedure

In recapitulation, our distance transform algorithm is

1. Initialize the active narrow band NB by analytically computing distance of a starting set of grid points to the shape S. The points in the starting set belong to an enclosing boundary layer of S. Set other grid points to a status *UNTOUCHED*. Initialize time step $T = 0$.
2. At time $T+1$, for each point P in NB, find each of its neighbor, G, whose status is not *FINISHED*.
3. Compute the temporary distance d_G^{temp} of N from P by Equation 2. Use the smallest distance for G, $d_G^{temp} = min(d_G^{temp}, d_G^T)$.
4. If G is *UNTOUCHED*, set it to *TOUCHED* and add it to NB. Assign a lifespan value of G as $f_G = Ceiling(\alpha \cdot (T + 1))$.
5. Go back to Step 2, until all the points P in NB are processed.
6. For each point P in NB, assign $d_P^{T+1} = d_P^{temp}$. If $T + 1 > f_P$, remove it from NB and set it to *FINISHED*.
7. If NB is empty, exit; else $T = T + 1$, go back to 2;

In our method, each point in NB can be processed concurrently. Therefore, we propose a parallel distance transform approach based on the active band.

6. Multiple-Segment Distance Transform

We have described the basic algorithm of our adaptive iteration method. Next, we analyze the complexity of the method and then propose further improvement based on the analysis.

For a 3D domain with NX, NY and NZ points in the three dimensions, we have $N = NX \times NY \times NZ$ grid points totally, each point P will stay in NB for a lifespan $(\alpha \cdot t_f)$, where t_f is proportional to the real distance l. For convenience, we represent the lifespan as $(\alpha \cdot t_f) = (\beta \cdot l)$, where β is a new constant coefficient. For the whole propagation procedure, all the

points totally will be visited $\sum_{i=0}^{N}(\beta \cdot l_i)$ steps. Since $0 \leq l_i \leq l_{max}$, the total computation time is

$$\sum_{i=0}^{N}(\beta \cdot l_i) < \sum_{i=0}^{N}(\beta \cdot l_{max}) = N \cdot \beta \cdot l_{max}. \quad (5)$$

Because $l_{max} \propto O(N)$, the complexity of the aforementioned algorithm is approximately $O(N^2)$.

Here, l_{max} is the largest possible distance in the domain. Our method can stop when reaching a user-defined l_{max}. If only a small l_{max} is required, as in many applications, our algorithm can achieve a very fast performance. On the other hand, our algorithm is parallelizable inside the narrow band. If using a parallel computing scheme, its performance could be further improved. Next, we improve the method from another perspective.

The computational performance of our method is related to the distance value. If there are n_d grid points with an arbitrary distance l_d, in adaptive iteration, each point stays in NB for $(\beta \cdot l_d)$ steps. For a large number of n_d points with a large l_d value, the computational speed is slow. For instance, for the distance field of one center point in 3D space, the result consists of multiple spheres centered at the point. The distance transform is very fast at the beginning and becomes slower as the radius increases to a very large l_{max}. From the observation, we have endeavored to further improve the performance by restricting the maximal value of l_d, i.e. l_{max}, to a fixed value. This leads to a new multiple segment propagation method for distance transform.

In general, we decompose the whole 3D domain to a number of segments, each of which represents a group of grid points with a specific range of distance values. We only allow the active band, NB, to propagate inside one segment before all the grid points inside the band achieve their final distance. It works as following:

First, we define a limited value l_1 for the first segment. When the active band propagates to a new point with distance larger than a threshold l_{max}^{seg}, this neighbor point will not be added to NB. Only after all the points with distance smaller than l_{max}^{seg} are computed, we set the boundary layer of these points as the new starting set of NB, and then start the distance transform for the next segment. The threshold of the second segment is $(2 \cdot l_{max}^{seg})$. For the ith segment, the threshold is $(i \cdot l_{max}^{seg})$. Repeating this procedure for all segments, the new algorithm outputs the distance field for the whole domain.

For each segment, the lifespan of the points is computed based on the initial band from the boundary of last segment, instead of the original shape. Then maximal lifespan value of all the points in each segment is $(\beta \cdot l_{max}^{seg})$, no matter what segment it is. As a result, with the number of points, N_i, of each segment i, and with s segments in total, the computational time is

$$\beta \cdot (\sum_{i=0}^{N_0}(l_{max}^{seg}) + \sum_{i=0}^{N_1}(l_{max}^{seg}) + ... + \sum_{i=0}^{N_{s-1}}(l_{max}^{seg})) =$$

$$\beta \cdot l_{max}^{seg}(N_0 + N_1 + ... + N_{seg}) = \beta \cdot l_{max}^{seg} \cdot N. \quad (6)$$

Comparing with Equation 5, this l_{max}^{seg} is user defined and smaller than l_{max}, in this way, we achieve great speed improvement from $O(N^2)$ to $O(m \cdot N)$ with a constant $m = (\beta \cdot l_{max}^{seg})$. We report the effect of the value m to our distance transform performance in the next section.

7. Results and Discussion

We examine our method with many data sets on an ordinary PC with an Intel Core2 CPU 6300 at 1.86 GHz and 2 GB of RAM. To show our computational performance, we use the adaptive iteration method for distance transform on multiple data sets. The examples include artificial data sets with one staring point (i.e. we compute distance field in the whole domain to one existing point) and two starting points, respectively. Moreover, we also compute the distance field of several polygonal models: a fandisk, the Stanford bunny, armadillo and teapot.

We report the performance in three levels of volume size for each model in Table 1, which are $64 \times 64 \times 64$, $128 \times 128 \times 128$ and $256 \times 256 \times 256$, respectively. In our method, we use the FCC grid, so actually we only need to perform the computation in a half size of the regular cubic grid, since the FCC grid is constructed from cubic grid by only allowing the links to its 12 neighbors with link length $\sqrt{2}$ to be included. This straightforward implementation uses a coarse sampling in space. However, it has been shown in previous literature that the FCC sampling scheme requires 29.3% fewer samples compared to CC lattice in 3D domain. In practice, our method achieves good accuracy. In the largest volume size we tested, our method has the maximum error of 0.276 and average error of 0.001 for the one point model in comparison with the analytical result. For the two points model, the the maximum error is 0.547 and average error is 0.001.

In Table 1, for each model of different volume sizes, we compare the computational time between the multiple-segment method with a segment size $l_{max}^{seg} = 20$ (see Section VI), and the method with no segment. It shows that using the multiple-segment method can improve the speed. This improvement is larger for one point and two point models, which have substantial increase on the number of grid points along with the radius increase, and thus very suitable for multiple-segment acceleration. Please note that for polygonal models, the performance depends on the location and relative size of the model inside the 3D domain, which defines the distance value distribution.

In Figure 6, we show the computational time for distance transform on different distance ranges, for three data sets. For example, the bunny model uses 9.36 seconds to compute grid points with distance smaller than 20, and uses 16.578 seconds to compute grid points with distance ranging from 20 to 40, and so on. It shows

TABLE 1
PERFORMANCE OF OUR DISTANCE TRANSFORM METHOD ON MULTIPLE DATA SETS. FOR EACH MODEL IN DIFFERENT GRID SIZE, WE COMPARE THE COMPUTATIONAL TIME BETWEEN MULTIPLE-SEGMENT METHOD WITH A SEGMENT SIZE $l_{max}^{seg} = 20$ (SEE SECTION VI), AND THE METHOD WITH NO SEGMENT. COMPUTATIONAL TIME IS MEASURED IN SECONDS.

Volume Size	$64 \times 64 \times 64$		$128 \times 128 \times 128$		$256 \times 256 \times 256$	
Example	Time with Segment	Time without Segment	Time with Segment	Time without Segment	Time with Segment	Time without Segment
One Point	0.813	1.458	8.157	23.219	97.642	391.423
Two Points	1.078	1.938	11.109	22.766	124.485	409.234
Fandisk	0.952	0.953	9.391	11.797	107.518	171.532
Bunny	0.984	0.969	9.829	11.391	122.298	166.189
Armadillo	1.078	1.094	9.485	11.843	110.313	177.408
Teapot	0.812	0.859	8.937	12.689	117.827	202.907

that our speed mainly depends on the distance value, thus for many applications requiring a limited distance size, we can achieve very fast response. The downtrend after reaching the top is because the number of grid points decreases when the active band approaches to the domain boundary.

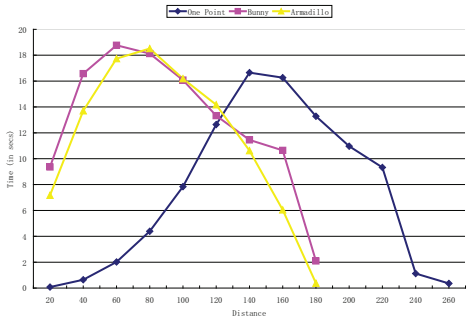


Fig. 6. Computational time for distance transform on different distance ranges.

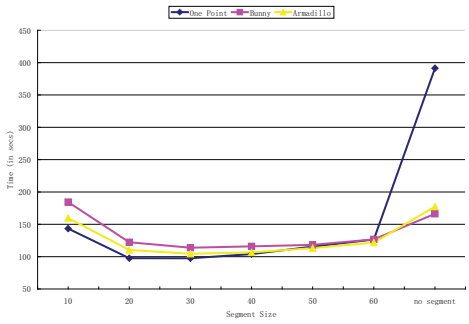


Fig. 7. Performance of using different segment size.

The performance of our segmented method is related to the segment size as described in Section VI. Figure 7 shows the performance of using different segment sizes (i.e. l_{max}^{seg}) as well as no segment for three data sets. It depicts that generally using a small segment size will make the computation faster. However, a very small segment size (i.e. $l_{max}^{seg} = 10$) could slow the computation, because the overload of resetting the narrow band between consecutive segments will compromise the speed achievement in this case.

We use a Marching Cubes method to generate isosurfaces with different distance values for visualization

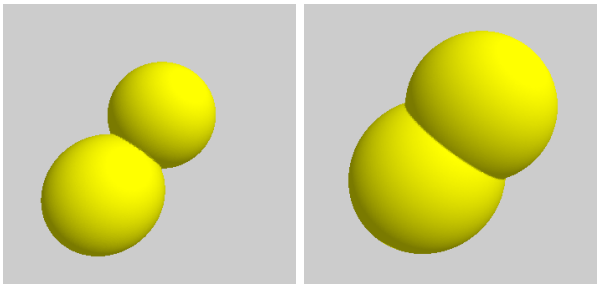


Fig. 8. Distance field of the two points is rendered as isosurfaces with different distance values.

of the results. In Figure 8, the distance field from two starting points consists two propagating spheres. Figure 9 shows the results of the Armadillo in the $256 \times 256 \times 256$ volume. Figure 10 is the results of the bunny model.

8. CONCLUSION

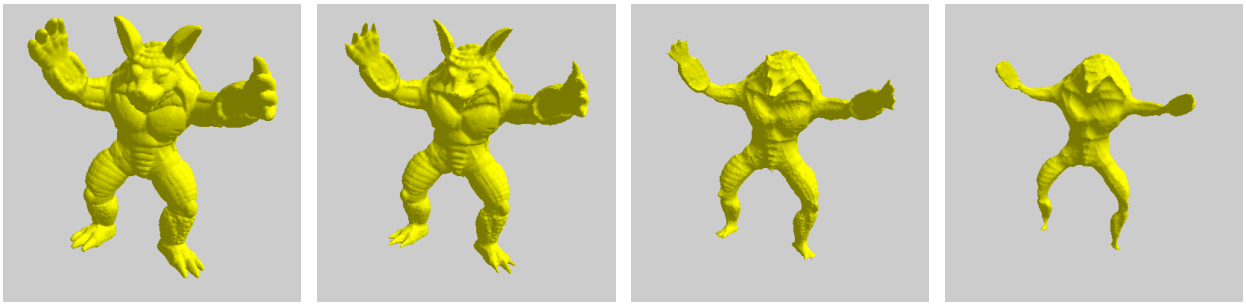
We have proposed a new adaptive method to compute distance transform based on a narrow band. The method performs iteration on the band to avoid maintenance of a priority queue. This iteration scheme is made valid by defining a lifespan value of each point, which determines how long a point should stay in the band to get its correct distance. The lifespan is computed by analyzing the grid structure. We find using an FCC grid can provide a fast computational speed than cubic grid. By applying a multiple segment propagation scheme, our method further reduces the computational complexity. Our method is inherently parallel for iteration inside the band at each time step. In the future, we will extend the method to GPU computing for further speed enhancements.

ACKNOWLEDGMENTS

The work is partially supported by the Research Council of the Kent State University. We thank Dr. Feodor Dragan for helpful discussions. Geometric models are courtesy of the Computer Graphics Laboratory, Stanford University.

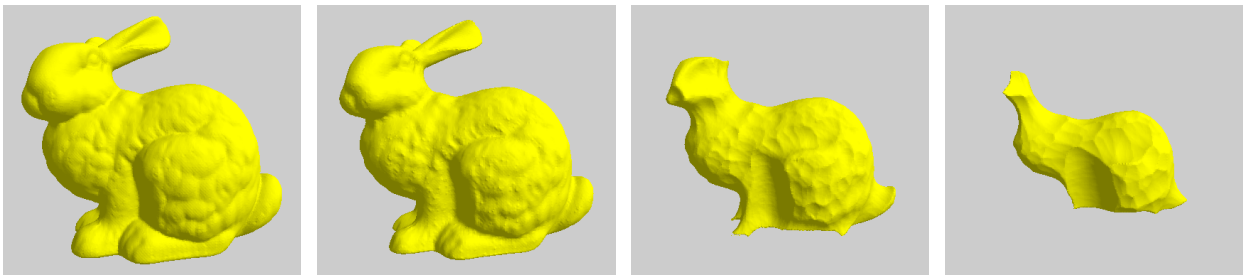
REFERENCES

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



(a) Origin (b) Distance=2.5 (c) Distance=6 (d) Distance=8.5

Fig. 9. Distance field of the Armadillo is rendered as isosurfaces with different distance values.



(a) Origin (b) Distance=2.5 (c) Distance=10.5 (d) Distance=21

Fig. 10. Distance field of the bunny is rendered as isosurfaces with different distance values.

[2] D. E. Breen, S. Mauch, and R. T. Whitaker. 3D scan conversion of csg models into distance volumes. *Proceedings of Symposium on Volume Visualization, ACM SIGGRAPH*, pages 7–14, 1998.

[3] D. Cohen-Or, D. Levin, and A. Solomovici. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 17(2):116–141, 1998.

[4] J. H. Conway, N. J. A. Sloane, and E. Bannai. *Sphere-packings, lattices, and groups*. Springer-Verlag, 1987.

[5] R. Fabbri, L. D. F. Costa, J. C. Torelli, and O. M. Bruno. 2d euclidean distance transform algorithms: A comparative survey. *ACM Comput. Surv.*, 40(1):1–44, 2008.

[6] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. *Proceedings of SIGGRAPH 2000*, pages 249–254, 2000.

[7] K. Hoff, J. Keyser, M. Lin, D. Manocha, and T. Culver. Fast computation of generalized voronoi diagrams using graphics hardware. *Proceedings of SIGGRAPH*, pages 277–286, 1999.

[8] W.-K. Jeong and R. Whitaker. Fast Eikonal equation solver for parallel systems. *SIAM conference on Computational Science and Engineering*, 2007.

[9] M. W. Jones, J. A. Baerentzen, and M. Sramek. 3d distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):581–599, 2006.

[10] M. W. Jones and R. Satherley. Using distance fields for object representation and rendering. *Proceedings of Eurographics*, pages 37–44, 2001.

[11] A. E. Lefohn, J. M. Kniss, C. D. Hansen, and R. T. Whitaker. A streaming narrow-band algorithm: Interactive computation and visualization of level sets. *IEEE Transactions on Visualization and Computer Graphics*, 10:422–433, 2004.

[12] S. Mauch. *Efficient Algorithms for Solving Static Hamilton-Jacobi Equations*. PhD thesis, California Institute of Technology, Pasadena, California, 2003.

[13] K. Museth, D. Breen, R. Whitaker, S. Mauch, and D. Johnson. Algorithms for interactive editing of level set models. *Computer Graphics Forum*, 24(4):821–841, 2005.

[14] S. Osher and L. Rudin. Rapid convergence of approximate solutions to shape-from-shading problem. *Technical report, Cognitech Inc.*, 1993.

[15] B. A. Payne and A. W. Toga. Distance field manipulation of surface models. *IEEE Computer Graphics and Applications*, 12(1):65–71, 1992.

[16] R. N. Perry and S. F. Frisken. Kizamu: A system for sculpting digital characters. *Proceedings of SIGGRAPH*, pages 47–56, 2001.

[17] J. Qian, Y.-T. Zhang, and H.-K. Zhao. A fast sweeping method for static convex Hamilton-Jacobi equations. *J. Sci. Comput.*, 31(1-2):237–271, 2007.

[18] J. Qian, Y.-T. Zhang, and H.-K. Zhao. Fast sweeping methods for Eikonal equations on triangular meshes. *SIAM J. Numer. Anal.*, 45(1):83–107, 2007.

[19] E. Rouy and A. Tourin. A viscosity solutions approach to shape-from-shading. *SIAM Journal on Numerical Analysis*, 29(3):867–884, 1992.

[20] R. Satherley and M. Jones. Vector-city vector distance transform. *Computer Vision and Image Understanding*, 82(3):238–254, 2001.

[21] J. Sethian. *Level Set Methods and Fast Marching Methods, second ed.* Cambridge University Press, 1999.

[22] C. Sigg, R. Peikert, and M. Gross. Signed distance transform using graphics hardware. *Proceedings of the 14th IEEE Visualization 2003*, pages 83–90, 2003.

[23] A. Sud, N. Govindaraju, R. Gayle, and D. Manocha. Interactive 3d distance field computation using linear factorization. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 117–124, New York, NY, USA, 2006. ACM.

[24] J. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Trans. Automatic Control*, 40(9):1528–1538, 1995.

[25] O. Weber, Y. S. Devir, A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Parallel algorithms for approximation of distance maps on parametric surfaces. *ACM Trans. Graph.*, 27(4):1–16, 2008.

[26] H. Zhao. Fast sweeping method for Eikonal equations. *Math. of Computation*, 74:603–627, 2004.

[27] W. Zhao. The fast sweeping method of Eikonal equations and its parallelism. *Master Thesis, INRIA France and Royal Institute of Technology, Sweden*, 2003.

[28] Y. Zhou and A. W. Toga. Efficient skeletonization of volumetric objects. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):196–209, 1999.