3D Distance Fields: A Survey of Techniques and Applications

Mark W. Jones, J. Andreas Bærentzen, and Milos Sramek

Abstract—A distance field is a representation where, at each point within the field, we know the distance from that point to the closest point on any object within the domain. In addition to distance, other properties may be derived from the distance field, such as the direction to the surface, and when the distance field is signed, we may also determine if the point is internal or external to objects within the domain. The distance field has been found to be a useful construction within the areas of computer vision, physics, and computer graphics. This paper serves as an exposition of methods for the production of distance fields, and a review of alternative representations and applications of distance fields. In the course of this paper, we present various methods from all three of the above areas, and we answer pertinent questions such as *How accurate are these methods compared to each other? How simple are they to implement?*, and *What is the complexity and runtime of such methods?*

Index Terms—Distance field, volume, voxel, fast marching method, level-set method, medial axis, cut locus, skeletonization, voxelization, volume data, visualization, distance transform.

✦

1 INTRODUCTION

PERHAPS the earliest appearance of distance fields in the literature is the 1966 image processing paper by Rosenfeld and Pfaltz [97], where they present the application of a chamfer distance transform to an image, and also create a skeleton which is a minimal representation of the original structure. Since then, many authors have improved the accuracy of chamfer distance transforms, and have introduced alternative algorithms such as vector distance transforms, fast marching methods, and level sets. Most of the earlier work concentrated on two-dimensional image processing, but as three-dimensional data sets grew in importance, recently much research has been targeted at processing this and higher dimensional data. The literature seems broadly split between the computer vision community (for image processing), physics community (for wavefront, Eikonal equation solving schemes), and computer graphics community (for object representation and processing). This paper will draw together the literature from these communities and will, for the first time, independently and thoroughly compare the various main algorithms and approaches.

For the purposes of this paper, we are most interested in the application of algorithms using distance fields for the modeling, manipulation, and visualization of objects for computer graphics, and so we shall emphasize methods that enable such processes. Recently, it seems that there is general widespread agreement that distance fields provide the most

suitable antialiased representation of geometric objects for the purposes of *Volume Graphics*. The term *Volume Graphics* was first introduced by Kaufman et al. in 1993 [64], where they presented the advantages of using volumetric models. Although they were working with binary representations which suffered from aliasing, many of the methods they proposed and discussed have adapted well to distance fields. Volume Graphics is now a subject area in its own right, demonstrated by an annual "Volume Graphics" conference series which started at Swansea in 1999 [24]. However, distance fields have recently found many applications unrelated to traditional volume graphics. For instance, they can be used for collision detection, correcting the topology of meshes, or to test whether a simplified mesh is within a given distance threshold of the original.

A distance field representation of an object can be particularly useful in situations where algorithms provide for the fast processing of three-dimensional objects, and so this paper shall concentrate on the methods by which distance fields are produced, and the applications that can use these distance fields to accelerate modeling, manipulation and rendering techniques. Apart from a survey of the literature in those areas, the main contributions of this paper are a summary of some of the very latest results in the production and use of distance fields, a new simplified version of the Fast Marching Method (FMM) and the first thorough comparison of FMM and its variants, Chamfer Distance Transforms (CDTs) and Vector Distance Transforms (VDTs) on both error and speed.

The remainder of this paper is organized as follows: In Section 2, we present some properties of the continuous distance field. Section 3 acquaints the reader with the main approaches to calculating discrete distance fields. Aspects such as computing the distance field directly from the data, and computing the sign for a signed distance field are accounted in Section 3.1. Using a shell boundary condition as a basis it is possible to create distance fields using the vector and chamfer transform methods and the fast marching methods of Section 3.2. Section 3.6 provides the first in-depth comparison of all three approaches—vector,

---

- *M.W. Jones is with the Department of Computer Science, Swansea University, Singleton Park, Swansea, SA2 8PP, United Kingdom. E-mail: m.w.jones@swan.ac.uk.*
- *J.A. Bærentzen is with the Informatics and Mathematical Modelling Department, Technical University of Denmark, Richard Petersens Plads DTU, Building 321, DK-2800 Lyngby, Denmark. E-mail: jab@imm.dtu.dk.*
- *M. Sramek is with the OAW Visualisierung, Donau-City-Strasse 1, A-1220 Vienna, Austria. E-mail: milos.sramek@oeaw.ac.at.*

chamfer, and fast marching methods on a variety of data sets. A thorough analysis of both time and error is given. Section 4 examines alternative representation schemes for distance fields including Adaptive Distance Fields (ADFs), lossless compression schemes, wavelets, and the Complete Distance Field Representation (CDFR). The main application areas using distance fields are briefly examined in Section 5. Finally, we conclude this paper with a discussion in Section 6.

2 CONTINUOUS DISTANCE FIELDS

2.1 The Continuous Signed Distance Function

Assuming that we have a set Σ, we first define the *unsigned distance function* as the function that yields the distance from a point $\mathbf{p}$ to the closest point in Σ:

$$\text{dist}_\Sigma(\mathbf{p}) = \inf_{\mathbf{x} \in \Sigma} \|\mathbf{x} - \mathbf{p}\|. \tag{1}$$

Frequently, we are mainly interested in the *signed distance* function associated with a solid S. The signed distance function returns the distance to the boundary, ∂S, and the sign is used to denote whether we are inside or outside S. Here, we use the convention that the sign is negative inside. This leads to the following formula for the signed distance function corresponding to a solid S

$$\text{d}_S(\mathbf{p}) = \text{sgn}(\mathbf{p}) \inf_{\mathbf{x} \in \partial S} \|\mathbf{x} - \mathbf{p}\|, \tag{2}$$

where

$$\text{sgn}(\mathbf{p}) = \begin{cases} -1 & \text{if } \mathbf{p} \in S \\ 1 & \text{otherwise.} \end{cases}$$

When no ambiguity is possible, we drop the subscript S from d_S.

2.2 Derivatives of the Distance Function

An important property of the signed distance function d is that

$$\|\nabla\text{d}\| = 1 \tag{3}$$

almost everywhere, with the exception being points without a unique closest point (e.g., the center of a sphere, see also Section 2.3). In this case, the gradient is not defined. Otherwise, the gradient at a given point $\mathbf{p}$ is orthogonal to the isosurface passing through $\mathbf{p}$. An isosurface is a set $\{\mathbf{p}|\text{d}(\mathbf{p}) = \tau\}$ where τ is the isovalue.

The second order derivatives contain information about the curvature of isosurfaces of the distance function [50]. For general functions, $f : \mathbb{R}^3 \to \mathbb{R}$, we can also obtain curvature information from the second order derivatives, but the equations become particularly simple for distance functions.

The Hessian, H, of d is the matrix of second order partial derivatives

$$H = \begin{pmatrix} \text{d}_{xx} & \text{d}_{xy} & \text{d}_{xz} \\ \text{d}_{yx} & \text{d}_{yy} & \text{d}_{yz} \\ \text{d}_{zx} & \text{d}_{zy} & \text{d}_{zz} \end{pmatrix}. \tag{4}$$

The mean curvature of the isosurface passing through a given point is simply the trace of H at that point divided by two:

$$\kappa_M = \frac{1}{2}\left(\text{d}_{xx} + \text{d}_{yy} + \text{d}_{zz}\right). \tag{5}$$

The Gaussian curvature is

$$\kappa_G = \begin{vmatrix} \text{d}_{xx} & \text{d}_{xy} \\ \text{d}_{yx} & \text{d}_{yy} \end{vmatrix} + \begin{vmatrix} \text{d}_{xx} & \text{d}_{xz} \\ \text{d}_{zx} & \text{d}_{zz} \end{vmatrix} + \begin{vmatrix} \text{d}_{yy} & \text{d}_{yz} \\ \text{d}_{zy} & \text{d}_{zz} \end{vmatrix} \tag{6}$$

and the principal curvatures are the two nonzero eigenvalues of H. The last eigenvalue is 0 reflecting the fact that d changes linearly in the gradient direction. Monga et al. give a good explanation of how the Hessian is related to curvature [72].

2.3 Continuity and Differentiability

It follows from the triangle inequality that the signed and unsigned distance functions of a given surface are continuous everywhere. However, neither is everywhere differentiable. This raises the question of where the signed distance function *is* differentiable which is a question a number of authors have considered.

In [67], it is demonstrated that for a C^k surface ($k \geq 1$), the signed distance function is also C^k in some neighbourhood of the surface. In his technical report, Wolter [135] presents various theorems regarding the cut locus. The cut locus of a surface is the set of points equally distant from at least two points on the surface. Hence, the cut locus is the same as the union of the interior and exterior medial surfaces. Theorem 2 in [135] pertains directly to the differentiability of the distance function. Specifically, it is shown that the *unsigned* distance function is differentiable, and its gradient Lipschitz continuous at points which neither belong to the surface nor to its cut locus. However, the signed distance is differentiable also on the surface (except at points where the cut locus touches the surface), and the critical points[1] of the signed distance coincide with the cut locus.

In the remainder of this paper, we are mostly concerned with discrete distance fields. Such distance fields may be obtained by sampling a continuous distance function, and in that case it may be important to know whether two grid points straddle the cut locus. If they do, it means that their distance values correspond to surface points that may be very far apart. This can lead to poor reconstruction of the continuous field, its gradient, and other characteristics, since they are typically estimated using finite reconstruction kernels which use a stencil of grid points. In [5], it is shown that if we can touch all points (inside and out) of the surface, ∂S, by rolling a ball with a given radius R_s on the inside and the outside of the surface, then the cut locus, and hence the critical points, does not come closer to the surface than R_s.

This property becomes important in modeling and visualization of objects represented by discrete DFs. In this case, it is necessary to reconstruct the continuous field, and, if shading is required, also its gradient in the vicinity of the object surface. It is, therefore, required that the aforementioned distance R_s between the cut locus and the surface be larger than the characteristic radius of the reconstruction filter. These conditions will be further on referred to as the *DF representability criterion* and an object which fulfils it a *DF representable solid*. Note that the actual size of R_s depends on the grid resolution and for common reconstruction kernels it has values between one and three voxel units [113].

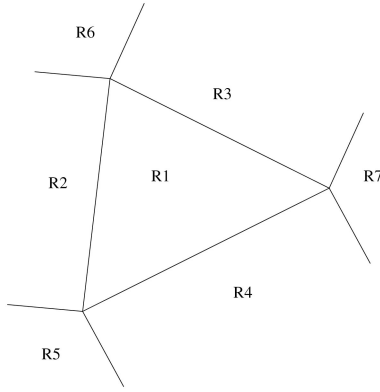1. For instance, where the distance function is not differentiable.

Fig. 1. Calculating distance to a triangle: If $\mathbf{p}$ projects onto $R1$ it is closest to the plane, $R2 - R4$ edge, $R5 - R7$ vertex.

3 COMPUTING DISTANCE FIELDS

A brute-force algorithm for computation of a shortest distance to a set of objects over a 3D discrete grid $\mathcal{V}$ is very simple: For each voxel of $\mathcal{V}$, its distance to all objects is computed and the smallest one is stored. In spite of its simplicity, this approach is impractical since it leads to prohibitively long computational times. Therefore, techniques were developed which 1) keeping the basic scheme discard most of the objects by exploitation of their spatial coherency (computing distances from primitives, Section 3.1) and 2) methods, which in an initialization step evaluate the distances in certain regions in a trivial way (inside the objects or in a thin layer around the surface) and subsequently propagate them through the whole volume (distance transforms, Section 3.2). Since some of these accelerated techniques are only approximate, the role of the computationally demanding but precise brute-force techniques is still unavoidable in algorithm evaluation.

3.1 Distance Computation for Common Surface Representations

3.1.1 Triangle Meshes

The triangle mesh representation is probably the most frequently used representation for 3D geometry. Therefore, it is particularly important that we are able to convert triangle meshes to signed distance fields. We can only generate distance fields from a certain class of triangle meshes, namely, meshes that are closed, orientable 2-manifolds. In practice, we can impose the manifold condition by requiring that [54]:

- The mesh does not contain any self intersections: Triangles may share only edges and vertices and must be otherwise disjoint.
- Every edge must be adjacent to exactly two triangles.
- Triangles incident on a vertex must form a single cycle around that vertex.

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

We will discuss first the basic methods for computing unsigned distance fields and then discuss techniques for generating the sign.

The distance to a triangle is easily computed using a simple case analysis. When a point $\mathbf{p}$ is projected onto the plane containing the triangle, the projected point $\mathbf{p}'$ lies in one of the seven regions shown in Fig. 1. If $\mathbf{p}'$ is projected

onto $R1$ then the distance from the point to the triangle is equal to the distance from the point to the plane containing the triangle. If $\mathbf{p}'$ lies in $R2$, $R3$, or $R4$ a distance to the corresponding line should be calculated. Last, with regions $R5$, $R6$, or $R7$ a distance to the corresponding vertex should be calculated.

3.1.2 Hierarchical Organization

Since the brute force method requires $N \cdot M$ steps, where N is the number of voxels, and M is the number of triangles, it is sensible to use hierarchical data structures to allow $O(\log M)$ access to the triangles.

Payne and Toga have described the basic approach of calculating distance to triangles [92]. They also proposed some optimizations. For instance, one can compute squared distances and then take the square root only when the shortest distance is found. Further, in a more comprehensive algorithm, they utilize the data coherency by storing the triangles in a tree of bounding boxes. From a given point, we can compute the smallest and greatest possible distance to anything inside the box. This can be used to prune branches as we move down the tree.

Quadtrees were used by Strain [123] to speed up computation of distance to an interface in 2D (redistancing, see Section 5.4.2) thus creating an $O(N \log N)$ algorithm (where N is the size of the interface).

Another hierarchical approach is the recent Meshsweeper algorithm proposed by Guéziec [48]. The Meshsweeper algorithm is based on a hierarchical representation of the mesh. At each level, a bounding volume is associated with every triangle. The bounding volume of a given triangle encloses all child-triangles at more detailed levels. A distance interval is computed for each bounding volume. This distance interval gives the shortest and greatest possible distance to any triangle inside the bounding volume. The lower bound is finally used as an index to a priority queue. A salient point is that we can remove a bounding box from the queue if its lower bound is greater than the greater bound of some other bounding box thus removing a branch of the hierarchy from consideration. Guéziec compares his method to an octree based method and to a brute force approach.

3.1.3 Characteristic Methods

Hierarchies are not always needed. If the actual value is only required up to a certain distance from the surface, then the influence of a triangle becomes local. If only distances up to, say, five units are required, we can use a bounding volume around each triangle to ensure that distances are only computed for grid points that are potentially closer than five units. The smaller this max distance, the smaller the need for a hierarchical structure.

For each triangle, we simply compute the distance to that triangle for all grid points inside the corresponding bounding volume. In [34], Dachille et al. propose such a method where the volume is a simple axis aligned bounding box. Their contribution is to perform the case analysis (Fig. 1) using only distance to plane computations. The advantage is that only one addition is needed for each voxel to compute its distance to a given plane—using an incremental approach—making the technique suitable for hardware acceleration.

Many other methods that generate distance fields with distances only up to some maximum value are based on the notion of a characteristic which was introduced by
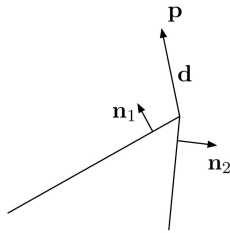
Fig. 2. The mesh feature closest to $\mathbf{p}$ is a vertex, but the dot products $\mathbf{d} \cdot \mathbf{n}_1$ and $\mathbf{d} \cdot \mathbf{n}_2$ do not have the same sign.

Fig. 3. The angle weighted normal $\mathbf{N} = \frac{\sum_i \mathbf{n}_i \alpha_i}{\| \sum_i \mathbf{n}_i \alpha_i \|}$.

Mauch [70], [71]. Each feature in the triangle mesh is converted to a polyhedron—the characteristic—which contains the points closest to that feature. For instance, an edge becomes a wedge, a face becomes a prism, and a vertex becomes a cone with polygonal base. These characteristics contain all points that are closest to their respective feature and within a certain distance of the mesh. Thus, the characteristics can be seen as truncated Voronoi regions. However, in general, they are made to overlap slightly to avoid missing grid points. The characteristics are then scan converted and for each voxel within the characteristic, the distance to the generating feature is computed.

A recent improvement of this method was proposed by Sigg et al. [119]. They use graphics hardware to scan convert the characteristics. First, the characteristic is sliced, and then the slices are sent to graphics hardware and rasterized in the usual way. The distances are computed in a fragment program [100]. However, slicing is done on the CPU and this may be a bottleneck. To alleviate this problem, only a single characteristic is computed for each triangle. This reduces both the amount of work to be performed by the CPU and the amount of bandwidth required in order to send the slices to the graphics card. The per-triangle characteristics are denoted *prisms*. Details on prism computation are provided in [91].

Another work that involves hardware acceleration is due to Sud et al. [118] who also use observations regarding spatial coherence to limit the number of primitives considered for each slice. Sud et al. claim to be two orders of magnitude faster than a plain software implementation of characteristics scan conversion.

One may construe the characteristics as very tight bounding boxes around the triangles, and herein lies their advantage. The methods are linear in both the number of triangles and voxels [70]. On the other hand, the characteristics must be created and scan converted. It is simpler to bound each triangle by a box aligned with the voxel grid. This can be made more effective by culling voxels that are farther from the plane of the triangle than a given distance.

3.1.4 Computing the Sign

The most obvious method for computing the sign when generating a signed distance field is to use the surface normals. If we have a C^1 smooth surface, the sign of the distance can be found by evaluating the dot product of the normal, $\mathbf{n}$, and a direction vector, $\mathbf{d}$, from the closest point to the point $\mathbf{p}$ where the sign is desired. $\mathbf{d}$ will always point either in the same direction as $\mathbf{n}$ (if we are outside) or the opposite (if we are inside).

The problem with triangle meshes is that they are not C^1: The normal is not defined on edges and vertices. As a simple solution, one could use the normal of the incident
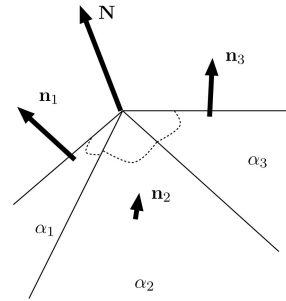
triangle even when the closest feature is an edge or a vertex. Unfortunately, this does not work, because in many cases we have the same distance to two or more triangles but different sign [92]. In particular, this occurs if the closest feature is a vertex, and the corresponding situation in 2D is shown in Fig. 2.

Most authors propose to use scan conversion to generate the sign [58], [92], [123], [70]. Typically, this is done in the following way: For each z-level plane in the grid, we compute the intersection of the mesh and the plane. This produces a 2D contour that we can scan convert [92]. An even simpler approach would be to cast rays along rows of voxels. At voxel locations where the ray has crossed the mesh an uneven number of times, we know we are inside.

The characteristics methods [70], [119], [91] are a bit different. A voxel belongs to precisely one characteristic associated with the closest edge, vertex, or face. This means that the sign problems do not occur in principle, since we can compute the sign for that characteristic. However, it is necessary to dilate the characteristics slightly to ensure that voxels do not "fall between" characteristics. Unfortunately, this overlap means that there are cases where characteristics of opposite sign overlap in areas where the numerical distance is the same, and this can lead to erroneously classified voxels [35]. Of course, this problem will be far worse if one simply uses a bounding box around each triangle.

A plausible approach would be to approximate a normal at each vertex and edge, but it is far from obvious how to do this. A recent method by Aanæs and Bærentzen [1], [4] solves this problem using *pseudonormals* assigned to edges and vertices. The challenge is to define a pseudonormal which is guaranteed to have a positive dot product with the direction vector whenever the point is outside and negative whenever the point is inside. Aanæs and Bærentzen use the angle weighted normal [126] as their choice of pseudonormal. To compute the angle weighted normal at a vertex, one sums the normals of the incident faces, weighting each normal with the angle between the two legs that are incident to the vertex. This is illustrated in Fig. 3.

As shown in [1], [4], the dot product of a direction vector $\mathbf{d}$ from a mesh point $\mathbf{c}$ to a point $\mathbf{p}$ and the angle weighted normal $\mathbf{N}$ at $\mathbf{c}$ is always positive if $\mathbf{p}$ is outside and negative otherwise. This leads to a method for computing signed distance fields that is simply an extension of the method for unsigned distance fields [1]. Details on a practical and efficient method for signed distance computation and a discussion of numerical robustness are provided in [4].

Another advantage of angle weighted normals is that they are independent of tessellation [126]. In other words, as long as the geometry is unchanged, we can change the triangulation of the mesh without affecting the vertex

normals. The notion of angle weighted normals is easily extended to edges—the normals of the two faces adjacent to the edge are simply both weighted by π.

3.1.5 Triangle Soups

Unfortunately, triangle meshes do not always form closed two-manifold surfaces, and then the methods above do not work, since only closed two-manifold surfaces divide space into a well-defined interior, exterior, and boundary. However, in many cases, we have just a slight self-intersection, or a small hole in the surface. In these cases, we might still want to compute an estimated distance field.

It has been shown that a binary volume can be generated from a triangle mesh by projecting the mesh from many directions [82]. From each direction, one generates what can be seen as a run-length coded binary volume representing the original triangle mesh. If the mesh contains holes or other degeneracies these will be reflected by holes in the scan conversion. However, a plausible volume can be reconstructed by voting amongst the scan conversions for each voxel.

The scan conversions can also be used to generate a cloud of points with normal information. From this point cloud, we can estimate distances as discussed in [7]. The points will be missing in areas where the original mesh contains holes, but using a diffusion scheme it is possible to fill in the missing distance values.

3.1.6 Implicit and Parametric Surfaces

The simplest objects to voxelize are implicit surfaces. An implicit surface is really just a function $f : \mathbb{R}^3 \to \mathbb{R}$ which serves as the embedding of a surface B which is a level-set or isosurface of f, i.e.,

$$B = \{\mathbf{p}| f(\mathbf{p}) = \tau\}, \tag{7}$$

where τ is the isovalue. In practice, f should be constrained so that the value of f is always $f > \tau$ on the inside and $f < \tau$ on the outside or vice versa. Often, we require that $\nabla f \neq 0$ for any point $\mathbf{p} \in B$ since this means that τ is a regular value and, hence, that B is a manifold [127]. For an in-depth discussion of implicit surfaces, see [127], [13].

The analytic definitions of a 3D sphere $f(\mathbf{p}) = ||\mathbf{p} - \mathbf{p}_0||$ or hyperplane $f(\mathbf{p}) = (\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{n}$ are good examples of implicit surfaces, and these two are particular because the value of f is also the signed distance to the sphere or hyperplane, respectively. Hence, we can voxelize a sphere or hyperplane simply by sampling f. In general, more work is required. If we can accept some error, it is frequently possible to voxelize the implicit surface by sampling an approximation of the signed distance, typically $(f - \tau)/||\nabla f||$. This method is used in the VXT class library of DF-based voxelization techniques by Sramek [115]. A more precise but also more costly method is to find the foot point numerically: Given a point $\mathbf{p}$ find the closest point $\mathbf{p}_{\text{foot}}$ so that $f(\mathbf{p}_{\text{foot}}) = \tau$. The distance is then $||\mathbf{p} - \mathbf{p}_{\text{foot}}||$ and the sign is trivially computed. Hartmann has designed such a foot point algorithm [50] which accepts a point $\mathbf{p}$ in the vicinity of an implicitly defined surface and produces a foot point. The basic idea is to move in the gradient direction until a point $\mathbf{p}_0$ on the surface is found. The estimate is then iteratively refined until the surface normal points to $\mathbf{p}$.

Certain implicit solids can contain sharp surface details, where the aforementioned methods, which assume sufficient smoothness of the implicit function, fail to produce meaningful distance values. To cope with this, a technique was proposed by Novotny et al. [83], where such areas are identified and the solid is locally modified in order to comply with the DF representability criterion (Section 2).

There are several possibilities for computing distances to parametric surfaces. In a few cases (sphere and double cone) closed form solutions are available. In some other ones (superellipsoids [8] supershapes [44]), it is possible to convert their parametric representation to an implicit one [69] and to use the techniques mentioned above. However, in general, it is necessary to minimize for each grid point $\mathbf{p}$ the expression

$$d(u, v) = ||\mathbf{S}(u, v) - \mathbf{p}||, \tag{8}$$

where $\mathbf{S}(u, v)$ is the surface's parametric representation. For example, gradient descent minimization was used by Breen et al. [16]. In general, minimization of (8) may lead to numerical problems and trapping in local minima.

3.2 Distance Transforms

The principle behind the use of the distance transform (DT) is that a boundary condition close to the surface boundary can be generated (using any of the direct methods of Section 3.1) from which the remaining distances may be evaluated. The boundary condition is discussed in Section 3.2.1.

In a second phase, distances are propagated to the rest of the volume using a DT. As distances away from the boundary condition are not calculated using the exact methods of Section 3.1, some errors may be introduced. This section will examine the errors produced by many of the popular distance transform techniques. DT algorithms can be classified according to how we *estimate* the distance value of a given voxel from the known quantities of its neighbors and how we *propagate* the distances through the volume. The first classification criterion leads us to

- **chamfer DTs**, where the new distance of a voxel is computed from the distances of its neighbors by adding values from a *distance template* (Fig. 4),
- **vector DTs** where each processed voxel stores a vector to its nearest surface point and the vector at an unprocessed voxel is computed from the vectors at its neighbors by means of a *vector template* (Fig. 6), and
- **Eikonal solvers**, where the distance of a voxel is computed by a first or second order estimator from the distances of its neighbors.

According to the second criterion, the distances can be propagated through the volume in a

- **sweeping scheme**, when the propagation starts in one corner of the volume and proceeds in a voxel-by-voxel, row-by-row fashion to the opposite corner, typically requiring several passes in different directions, or in a
- **wavefront scheme**, when the distances are propagating from the initial surface in the order of increasing distances until all voxels are processed.

If required, it is possible to stop the computations in a wavefront scheme as soon as the desired distance isolevel is reached. One can take advantage of this property in numerous applications, as, for example, in morphological operations, surface offsetting and level-set methods.
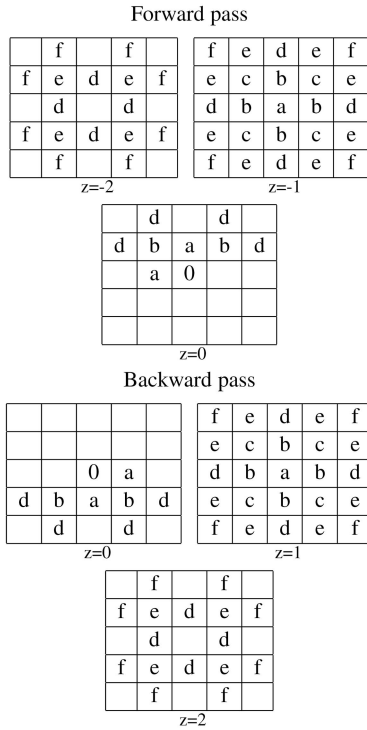
Forward pass

Fig. 4. A distance template. In the forward pass, distances (a)-(f) are added to voxels in the current, z-1 and z-2 slices. In the backward pass, distances are added to voxels in the current, z+1 and z+2 slices.

3.2.1 Initialization

Most distance transformations are reported as operating on discrete binary classified data F which has been obtained by scan-conversion of analytically defined solids or by segmenting volumetric data (e.g., by thresholding) to extract the feature points of a surface:

$$F(\mathbf{p}) = \begin{cases} 0 & \mathbf{p} \text{ is exterior} \\ \infty & \mathbf{p} \text{ is interior,} \end{cases} \quad (9)$$

where voxel $\mathbf{p} \in \mathbb{I}^3$ and $F: \mathbb{I}^3 \to \mathbb{R}$. This classification when operated on by a distance transformation will produce a distance field internal (or external, when reversed) to the object. To produce an unsigned distance field of the type in (1) the following classification could be used:

$$F(\mathbf{p}) = \begin{cases} 0 & \mathbf{p} \text{ is on the discrete surface} \\ \infty & \text{elsewhere.} \end{cases} \quad (10)$$

Here, the numerous voxelization techniques can be used to identify the 0-value voxels [62].

It is recommended that for higher accuracy, a so-called gray-level classification should be used in a shell around the surface [58]:

$$F(\mathbf{p}) = \begin{cases} d_S(\mathbf{p}) & \text{in the shell} \\ \infty & \text{elsewhere.} \end{cases} \quad (11)$$

The boundary condition is said to be minimal if it only consists of inside voxels that are 26-connected to voxels outside (and vice versa) and all voxels that are on the surface. This requires that gray-level voxels known to be close to the surface be calculated using a short cut. Surface mesh data could be scan converted to the voxel grid using many of the triangle-based techniques mentioned in Section 3.1.1, whereas parametric

TABLE 1
Values Used to Produce Each Chamfer Type

Transform	a	b	c	d	e	f
City Block (Manhattan)	1					
Chessboard	1	1				
Quasi-Euclidean $3 \times 3 \times 3$	1	$\sqrt{2}$				
Complete Euclidean $3 \times 3 \times 3$	1	$\sqrt{2}$	$\sqrt{3}$			
$<a,b,c>_{opt} 3 \times 3 \times 3$[102]	0.92644	1.34065	1.65849			
Quasi-Euclidean $5 \times 5 \times 5$	1	$\sqrt{2}$	$\sqrt{3}$	$\sqrt{5}$	$\sqrt{6}$	3

```
/* Forward Pass */
FOR(z = 0; z < f_z; z++)
   FOR(y = 0; y < f_y; y++)
      FOR(x = 0; x < f_x; x++)
         F[x,y,z] =
   inf_{∀i,j,k∈fp}(F[x+i,y+j,z+k]+m[i,j,k])

/* Backward Pass */
FOR(z = f_z-1; z ≥ 0; z--)
   FOR(y = f_y-1; y ≥ 0; y--)
      FOR(x = f_x-1; x ≥ 0; x--)
         F[x,y,z] =
   inf_{∀i,j,k∈bp}(F[x+i,y+j,z+k]+m[i,j,k])
```

Fig. 5. Pseudocode for chamfer distance transform application.

surfaces and curves can be voxelized by splatting [113]. In the case of implicit solids or scalar fields, when the surface is defined by the isovalue τ, the distance should be explicitly computed for all voxels which have a 6-neighbor on the other side of the isovalue using the aforementioned linear approximation $(f - \tau)/||\nabla f||$. In vector DTs, techniques for estimation of the nearest isosurface point listed in Section 3.1.6 can be used.

After classification the distance transformation is applied. A simpler chamfer distance transformation gives poorer results than the vector distance transformation.

3.2.2 Chamfer Distance Transform

In chamfer distance transforms (CDTs) [97], [96], [141], [18], [2], the distance template (Fig. 4) is centered over each voxel, and the distance at the central voxel is computed as the minimum of all of its neighbors' distances with the appropriate component added. Both sweeping (two pass) and wavefront schemes were formulated.

In a sweeping scheme, the distance template is split in two parts: one is used in a forward pass and the other in a backward pass. The forward pass calculates the distances moving from the surface towards the bottom of the data set, and is followed by the backward pass calculating the remaining distances. Fig. 4 is used as the basis with Table 1 giving the appropriate template values for each chamfer method. Note that some values are empty in order to ensure that calculations that yield the identical result are not repeated (if they are filled we have a 5^3 Complete Euclidean DT). The distance transformation is applied using the pseudocode in Fig. 5, where $i, j, k \in \{-2, -1, 0, 1, 2\}$ for a $5 \times 5 \times 5$ transform, fp and bp are the sets of transform positions used in the forward and backward passes, respectively, and checks are made to ensure fp and bp only contain valid voxels at the edges of the data set. Svensson and Borgefors [102] present an analysis of chamfer distance transforms and give numerous examples of distance
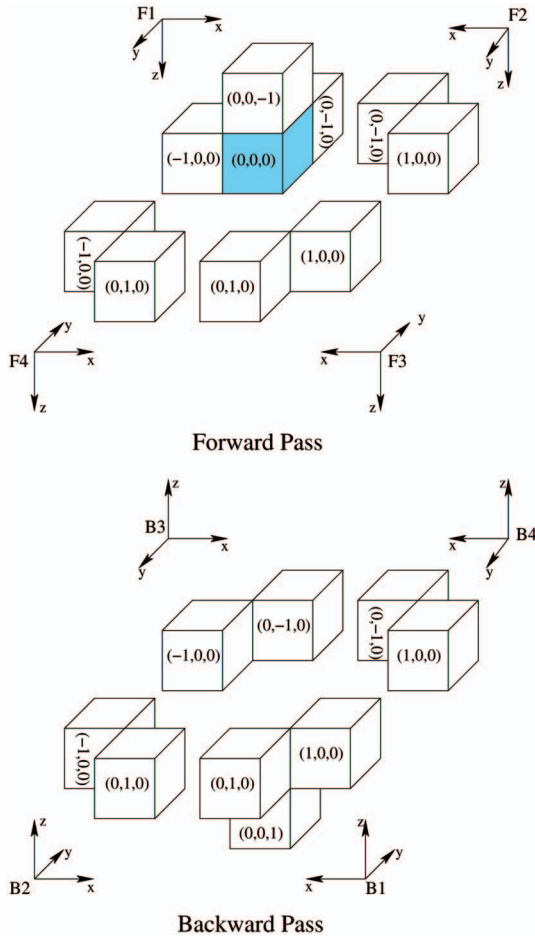
Fig. 6. Vector templates for one pass of the VCVDT technique.

```
FOR(z = 0; z < f_z; z++)
   /* Forward Pass F1*/
   FOR(y = 0; y < f_y; y++)
      FOR(x = 0; x < f_x; x++)
         p=(x,y,z);
         pos=argmin_i ||vec[p + dir_i]+dir_i||
         vec[p]=vec[p + dir_pos]+dir_pos
         F(x,y,z)=||vec[p]||
```

Fig. 7. Pseudocode for pass F1 of a vector distance transform.

3.2.3 Vector Distance Transform

CDTs suffer from poor accuracy as the distance from the surface increases (Section 3.6). This problem is overcome by using Vector (or Euclidean) Distance Transforms (VDTs or EDTs) [32], [78], [110] which use a boundary condition of voxels containing a vector to the closest point on the surface, and propagating those vectors according to a pattern (vector template) such as the one given in Fig. 6 (the Vector-City Vector Distance Transform, VCVDT, [110]). Fig. 7 shows the pseudocode for one pass of a vector distance transform (F1 of the VCVDT), where vec is a voxel grid containing actual or estimated vectors to the surface ∂S and $\mathbf{dir} = \{(0, 0, -1), (0, -1, 0), (-1, 0, 0)\}$. Pass F2 would be a loop with increasing y and decreasing x with $\mathbf{dir} = \{(1, 0, 0), (0, -1, 0)\}$. All forward passes are carried out within one single outer loop with increasing z. Backward passes are implemented similarly.

The VCVDT technique requires eight passes through the volume, and in each pass just 6-neighbors of the actual voxel are visited. Of course, different schemes are also possible. Ragnemalm [94] classifies them in *separable* and *nonseparable*. In the former, the passes are mutually independent and can be applied in any order. Thus, they are suitable for parallel implementation, while the latter are more appropriate for sequential implementations. Further, in [94], a separable four pass algorithm with 26-neighborhood vector template is proposed.

Breen et al. [16], [17] implement a wavefront version of a VDT technique by passing closest point information (trivially equivalent to passing vector information) between voxels at a moving wavefront from CSG objects. Breen et al. [17] also demonstrate the passing of color information to create colored offset surfaces.

3.3 The Fast Marching Method

The *fast marching method* (FMM) [125], [106], [107], [55] is a technique for computing the arrival time of a front (which we can think of as, e.g., a balloon) expanding in the normal direction at a set of grid points. This is done by solving the Eikonal equation from a given boundary condition. The Eikonal equation is

$$\|\nabla T\| = \frac{1}{F}, \tag{12}$$

where $F \geq 0$ is the speed of the front, and T is the arrival time of the front. Given a point $\mathbf{p}$, the arrival time $T(\mathbf{p})$ is the time at which the skin of the balloon passed $\mathbf{p}$. The Eikonal equation states the obvious inverse relationship between the speed of the front and the gradient of the arrival time. Since F does not have to be unit or even constant, the FMM is not solely a method for computing distance fields.

templates. Cuisenaire [31] also gives a good review of distance transforms (both Chamfer and Vector).

In wavefront techniques, voxels are processed in the order of their increasing distance [141], [26]. To ensure the correct order, the processed voxels are inserted into a priority queue. In a loop, the voxel from the top of the queue is removed, and distances of its not yet processed voxels are computed according to the distance template and subsequently inserted into the queue. The process continues as long as there are any voxels in the queue.

Asymptotic complexity of the wavefront approaches ($O(N \log N)$, N being total number of processed voxels) is worse than of the sweeping approaches ($O(N)$) due to the priority queue management. However, in special setups, the wavefront approaches were reported to be "considerably faster" [141]. Here, in an algorithm adopted from [129], Zuiderveld et al. take advantage of an observation that the direction of the shortest distance propagation is kept. In other words, knowing a position of the voxel the distance of which was used to compute the distance of the current voxel, it can be predicted, which of its neighbours should be processed. The speed up is thus obtained, in regard to the sweeping schemes, by eliminating computations of distances, which would later be overwritten by lower values.
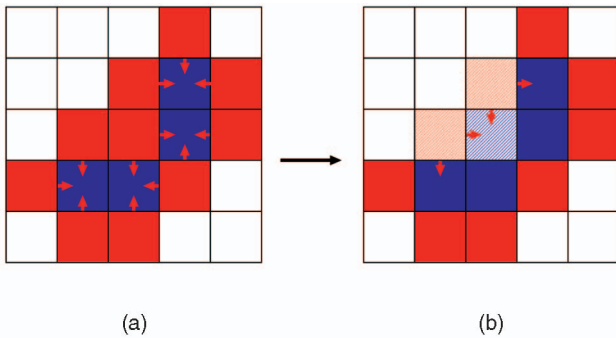
Fig. 8. This figure illustrates the structure of the FMM. The distances are initially known at the boundary condition (blue squares in (a)). The distance is then computed at all adjacent grid points (red squares) and the grid point with the smallest of these distances is frozen (the hatched blue square in (b)). Then, the distance is recomputed at its (hatched red) neighbors. The red arrows indicate which neighbors are used to recompute a distance.

However, if $F = 1$, the front moves at unit speed, and the arrival time at **p** is simply the distance from **p** to the closest point on the front at time 0. Hence, the FMM can be and is frequently used to compute distance fields. The FMM is defined on both 2D and 3D grids [117] and also on surfaces represented as triangle meshes [68], [107]. The FMM was independently proposed first by Tsitsiklis [125] and then Sethian [105] and Helmsen et al. [55].

The FMM is in principle a wavefront scheme which computes the values of T from a set of boundary values, and the structure of the algorithm is almost identical to Dijkstra's single-source shortest path algorithm [CLR]. We say that a grid point with a known arrival time is frozen. In the first step, the distances of all grid points adjacent to a frozen grid point of the boundary condition are computed. Then, we pick the smallest of these distance values and freeze the corresponding grid point. Next, we recompute the distance at all its adjacent grid points (except at those that are already frozen, see Fig. 8). Finally, we loop back and freeze the grid point that *now* contains the smallest distance value. In this way, the set of frozen grid points keeps expanding, and around the frozen set there is a layer of grid points where the distance is computed but not yet frozen. A priority queue implemented as a binary heap is typically used to store these distance values. Whenever, a distance is computed or recomputed, we have to be able to update the heap to reflect this change. This requires that each grid point holds a pointer to its corresponding heap element.

In order to compute the distance at a new grid point, a discretized version of (12) is solved. The discretization is based on a first order accurate finite difference approximation to the gradient which only uses the frozen grid points in the neighborhood of the current grid point. To solve this discretized version of the Eikonal equation, we simply need to find the largest root of a second order polynomial. Unfortunately, the standard FMM is not very precise. This has motivated another version (FMMHA) of the method which is more precise by virtue of the fact that second order finite difference approximations to the partial derivatives are used [106]. Hence, it is necessary to know the distance two grid points away from the grid point where a new distance is computed. In practice, though, it is possible to fall back to the standard FMM if this condition is not met.

For more details on how to implement the FMM and FMMHA, the reader is referred to [6]. In the form described above, the FMM is an $O(N \log N)$ algorithm where N is the

number of grid points. The reason why the FMM is not linear is the fact that at each step, we need to find the smallest distance that is not yet frozen. Typically, the distance values are stored in a heap, and it is a constant time operation to pick the smallest. However, we also need to keep the heap sorted which is a logarithmic operation.

3.4   Variations of the FMM

A problem in the implementation of the FMM is that voxels in the priority queue may be recomputed. If they are recomputed they should be moved in the priority queue which entails that we need to store handles to priority queue elements in the voxels. However, two observations can be used to motivate a simplified FMM which does not require these handles:

- It is our experience that errors increase if one allows the values of priority queue voxels to increase as a result of a recomputation. This means that we do not need handles into the priority queue, instead we accept having multiple values in the priority queue for a single voxel. The smallest value will emerge first, and then we freeze the voxel. If a second value for a given voxel is popped from the priority queue, it is simply disregarded.
- Errors also increase if nonfrozen voxels are used in the computation of priority queue voxels. Hence, it is not necessary to assign distance values to voxels before they are frozen. This, in turn, means that we know a voxel has been frozen simply because it has been assigned a value.

The advantage of this scheme is that it is simpler, and we can use a standard priority queue. The following pseudo-code illustrates the simplified FMM loop:

```
Extract voxel with smallest value.
If voxel is not frozen,
   freeze voxel
   compute unfrozen neighbors
   and insert them in priority queue
```

We will refer to this simplified fast marching method as SFMMHA.

Tsai [124] proposes a hybrid algorithm. For each voxel with three known neighbors, the distance is calculated to the closest of two points which are the intersection of spheres centerd on each neighboring voxel with radius equal to the distance at that neighboring voxel. If only one neighbor is known a fixed amount h is added to the distance of that neighbor. If the intersection of the spheres is ambiguous, then the distance is calculated using the Godunov Hamiltonian. Adding the fixed amount h is the same as using a Chamfer Distance Transform, with h set to be an appropriate value from Table 1. The sphere intersection part of the algorithm produces at best the same result as a Vector Transform (although, relies on less storage, but requires more complex calculation to solve the intersection of the spheres). The Godunov Hamiltonian calculates the distance from a wave front propagating through the data. The algorithm combines all three methods (FMM, CDT, and VDT). It seems to produce accurate results from point data, and from piecewise linear objects when they are oriented so their normals are integer multiples of $\pi/4$ (the combined use of the CDT and Godunov Hamiltonian makes that restriction for accurate results). For arbitrary data as we test here, the VDT is superior in accuracy, speed and simplicity of implementation.

Kim [65] proposed the group marching method (GMM) where a group of voxels on the wavefront is used to calculate the distances for their neighboring voxels. The group is determined as those voxels that are within a certain distance of the wavefront and are chosen so that they do not affect the travel time calculation to each other. Due to the fact that their neighboring voxels could be affected by several members within the group, iterations in two different directions are carried out. GMM is tested on simple artificial problems for which similar errors are generated (to FMM). Although GMM is $O(N)$, it has a high overhead in the form of keeping track of the group, and determining which members of the group are to be updated.

Zhao [140] uses a sweeping method to solve the Eikonal equation. The volume is swept in forward-and-backward directions in a similar manner to Chamfer and Vector Transforms. At each voxel, the Godunov discretization of the Eikonal equation is calculated, rather than the chamfer matrix multiplication of the Chamfer Transform, or the vector additions of the Vector Transforms. This results in an $O(N)$ method which produces a similar result to the FMM at a similar speed to $O(N)$ Chamfer and Vector Transforms. He also proves convergence for sweeping methods using the Godunov Hamiltonian to solve the Eikonal equation. Going a bit further, Kao et al. proposed a fast sweeping method for more general hamiltonians based on Lax-Friederichs rather than Godunov's scheme [66].

Hysing and Turek [121] compare and evaluate the FMM method with various methods including the methods of Tsai and Zhao.

Yatziv et al. [137] create an $O(N)$ implementation by replacing the heap with an array of linked lists. The arrival times are discretized and placed at the end of their appropriate linked list ($O(1)$ insertion). By keeping track of which array represents the least time, the head of the list is used as the next grid point to compute ($O(1)$ removal). As the lists are not sorted, errors are introduced, but these were found to be acceptable when compared to the time saving.

3.5 Reinitialization of Distance Fields

The goal of reinitialization is to convert an arbitrary scalar field, $\Phi : \mathbb{R}^3 \to \mathbb{R}$, whose 0-level isosurface (or 0-level set) represents some interface, to a distance field in such a way that the 0-level isosurface is unchanged. Reinitialization is often required as a part of the *level set method* (LSM) [86], [107], [84] (c.f. Section 5.4.2), and most work on reinitialization has been carried out in the context of the LSM.

There is a simple solution to reinitialization based on the methods discussed in the previous sections. If we assume that the grid points immediately adjacent to the 0-level isosurface contain correct distances, FMM can be used to rebuild the distance field up to the required value. Unfortunately, this condition may be violated, which was addressed by Chopp [23]. Here, in each cell (group of eight grid points) intersected by the 0-level set of Φ, a cubic polynomial which interpolates the values at the corners is constructed. The value of each corner grid point is, subsequently, replaced by the distance to the 0-level set of this polynomial.

While the FMM is often employed for reinitialization, other distance transforms could also be used, and, in fact, there is an entire class of methods based on the reinitialization equation,

$$\Phi_t + s(\Phi_0)(\|\nabla\Phi\| - 1) = 0, \tag{13}$$

which was introduced by Sussman et al. [122] extending work by Rouy and Tourin [99]. In (13), Φ is a function that is typically almost but not quite a distance function. $s(\Phi_0)$ is the sign of the original function which must be known for all grid points in advance. Most authors use a "smeared" sign function which is very small near the interface to avoid instability. Sussman et al. proposed

$$s(\Phi_0) = \frac{\Phi_0}{\sqrt{\Phi_0^2 + \epsilon^2}}, \tag{14}$$

where ϵ is a constant often chosen to be about the size of a cell in the grid [90]. A different sign function more adapted to steep gradients was proposed in [90]. $\|\nabla\Phi\|$ must be computed in an upwind fashion, i.e., the derivatives should be one sided which means that for a given grid point they should look in the direction to the 0-level set [84].

Evolving (13) forward in time will cause the value of Φ to iteratively increase or decrease in order to attain a gradient length of unity. When a steady state has been reached, Φ is a distance function. Thus, methods based on (13) can be used as distance transforms provided the sign is given for each grid point. However, this is not efficient since many time steps are needed.

Most authors use a small, constant number of steps (e.g., one or two) to correct a field that is already close to a distance field. In this scenario, the schemes are $O(N)$, where N is the total number of grid points while FMM is $O(N \log N)$. However, in practice, FMM might be faster. We surmise that (13) is best if small frequent corrections are needed while FMM could be better for infrequent, large corrections. It is also easy to stop the FMM when all distance up to a given maximum have been computed. However, when evolving (13), the distance information flows outward from the interface just like in the fast marching method. This indicates that a similar wavefront scheme should be feasible as mentioned in [98].

A concern with the methods based on (13) is that the 0-level set may move slightly. As a countermeasure Sussman et al. [108] proposed a volumetric constraint, and Russo et al. [98] an upwind method which does not accidentally look beyond the 0-level set. Finally, it should be noted that (13) can be discretized in a variety of ways both with regard to time derivatives and spatial derivatives. In [84], the interested reader will find an overview with more details about this.

3.6 Comparison

Each tested distance transformation was executed on several test data sets, and the results are presented in Table 2. The Point data set is a single point in a 256^3 voxel grid, the RotCube data set is a voxelized cube rotated by 30° on both the x and y-axes (again 256^3). Hyd is a distance field to the 64^3 AVS Hydrogen data set (measured to $\tau = 127.2$) and CT is a distance field to the bone ($\tau = 400$) of the UNC CThead ($256 \times 256 \times 113$, Fig. 9). In the latter two cases, the distance field was measured to the triangles created by the tiling tetrahedra algorithm [92] using a threshold of τ. The boundary condition consists of internal voxels with an external 26-neighbor, and external voxels with an internal 26-neighbor. The vector transform requires the vector to the closest point whereas the other transforms just require the distance to the surface. Each method is compared to a ground truth distance field that has been computed using a direct method. Note that a 5^3 Complete Euclidean exists which creates an equivalent result to the 5^3 Quasi Euclidean

TABLE 2
Each Distance Transformation Method was Tested with Each Data Set

Transform	Point (1)			RotCube (386755)			Hyd (11056)			CT (507240)		
	avg.	max	secs	avg.	max	secs	avg.	max	secs	avg.	max	secs
VCVDT	0	0	6.39	0.0034	0.089	8.10	0.01	0.13	0.1	0.01	0.19	3.35
City Block	67.96	159.76	10.7	15.51	91.28	8.74	4.45	20.28	0.14	9.04	74.16	3.78
Chessboard	18.53	52.19	11.99	9.44	31.76	9.81	2.18	8.77	0.16	4.63	26.94	4.18
3^3 Q-Euclidean	17.29	49.21	11.99	4.50	26.96	9.80	1.13	5.31	0.15	2.67	18.64	4.18
3^3 C-Euclidean	9.57	18.54	14.80	2.49	14.96	12.45	0.74	2.84	0.19	2.01	13.88	5.39
$3^3 <a,b,c>_{opt}$	3.39	9.27	14.59	1.00	7.82	12.51	0.31	1.54	0.2	0.76	6.97	5.29
5^3 Q-Euclidean	2.40	6.55	37.04	0.50	3.40	34.95	0.22	0.98	0.54	0.60	4.46	15.30
FMM	1.76	2.78	167.72	0.23	2.07	183.67	0.31	1.00	1.41	0.40	1.84	77.93
FMMHA	0.40	0.62	170.43	0.03	0.27	184.15	0.04	0.33	1.45	0.06	0.95	78.42
SFMMHA	0.40	0.62	118.99	0.03	0.27	115.17	0.04	0.33	0.74	0.06	0.95	46.77

The average error, maximum error, and runtime are given for each (2.6Ghz P4). The number in brackets indicates the amount of voxels in the boundary condition.

(Table 1), and so it is not reported here. The following conclusions may be drawn:

3.6.1 Precision

- The VDTs (represented here by the VCVDT) are the fastest to execute and have the lowest error.
- VDTs produce accurate results for cases where distances are measured to point data sets. The FMM is accurate in the case of planar surfaces.
- The greater precision of the VDT reflects, to some extent, that more information from the boundary condition is used: VDT requires that a vector to the closest point is stored in each boundary condition voxel.
- VDTs produce the least error for an offset surface n for any n (Table 3).

Fig. 9. True 3D distance closure of 20, 10, and 5 voxels of UNC CThead data set (bottom right).

- VDTs are the only methods where the error diminishes as a function of distance (Table 3).
- Larger CDT kernels give more directional possibilities for the source of the shortest distance, and are therefore more accurate, but they increase computational time.
- The $<a,b,c>_{opt}$ method is the best 3^3 CDT as it has been optimally designed to limit the distance error [102].
- The max error of the CDTs rise significantly as a function of distance (compared to FMM and VDTs).
- The FMMHA is significantly more accurate than the original FMM.
- If the original analytic representation (in addition to the boundary data) is available, the result of a VDT can be used to measure distances to the original data to improve accuracy further [16], [109].

3.6.2 Speed

- CDTs and VDTs are $O(N)$ methods, whereas FFMs are $O(N \log N)$. This is reflected in the computational times.
- Wavefront methods should be faster than the other techniques when requesting only distances up to a level, say τ, at least for small τ. We have tried to verify this with the SFMMHA. Indeed, for $\tau \leq 3$, the SFMMHA outperforms all other methods, but only narrowly in the case of the CT head.

3.6.3 Ease of implementation

- Arguably, VDTs and CDTs are easier to implement than FMMs. However, the simplified FMM is easier to implement than the method proposed by Sethian [107].
- FMM, being an Eikonal solver is more general and can also compute arrival times for non unit-speed fronts.

4   REPRESENTATION OF DISTANCE FIELDS

Discrete distance fields are usually stored in voxel grids due to the great simplicity of this representation. However, there are some disadvantages to regular voxel grids. In order to
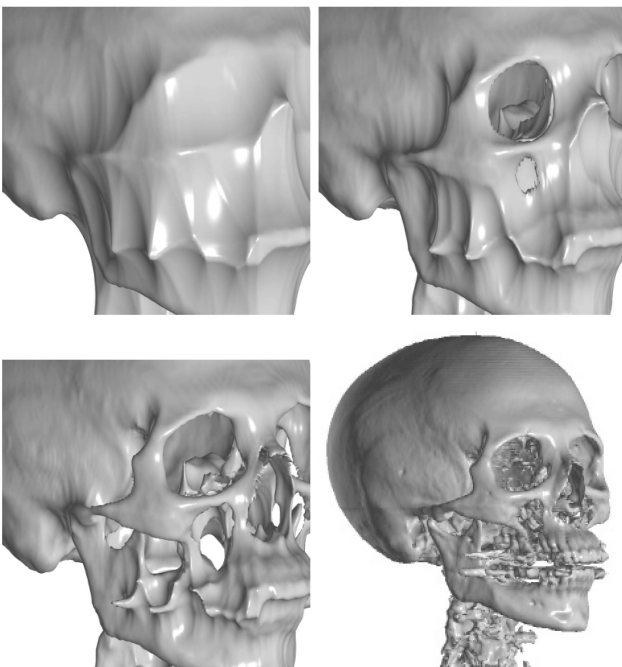
TABLE 3
The Average and Maximum Error for the UNC CThead Using Several Methods

Average (UNC CThead)							
	2	3	4	5	10	20	50
FMM	0.065	0.075	0.085	0.096	0.141	0.219	0.364
FMMHA	0.037	0.039	0.041	0.042	0.047	0.054	0.062
SFMMHA	0.037	0.039	0.041	0.042	0.047	0.054	0.062
VCVDT	0.025	0.024	0.022	0.020	0.017	0.014	0.012
$3^3 <a,b,c>_{opt}$	0.046	0.056	0.067	0.078	0.130	0.248	0.544
5^3 Q-Euclidean	0.062	0.060	0.063	0.073	0.115	0.202	0.425

Max (UNC CThead)							
	2	3	4	5	10	20	50
FMM	1.187	1.187	1.187	1.187	1.333	1.517	1.760
FMMHA	0.475	0.475	0.560	0.679	0.799	0.851	0.891
SFMMHA	0.475	0.475	0.560	0.679	0.799	0.851	0.891
VCVDT	0.168	0.191	0.191	0.191	0.191	0.191	0.191
$3^3 <a,b,c>_{opt}$	0.229	0.336	0.359	0.409	0.702	1.398	3.612
5^3 Q-Euclidean	0.302	0.313	0.326	0.368	0.598	1.065	2.508

For each method, the column n indicates the errors for all voxels up to distance n from the surface (that are not in the boundary condition).

capture tiny details, a high resolution grid must be used, but large grids are extremely memory consuming. Hence, there is a great incentive to develop more parsimonious representations which adapt better.

A very simple, effective improvement over a regular grid is to use either a hierarchical or run-length encoded grid. Both of these representations are useful in cases where the distances are clamped beyond a maximum value, as, for example, in object modeling (Section 5.4.1) or in LSM methods (Section 5.4.2). However, even for other applications this need not be a limitation, since the the clamped distances can be extended to a full distance field easily (Section 3.2). A hierarchical grid is simply a grid of cells where each cell contains a small voxel grid (e.g., $16 \times 16 \times 16$ voxels). In this case, we often have large homogeneous regions that can be represented with a single value for the entire cell. In the run-length approach, voxels of each data row are classified either as inside, outside and transitional [80]. A row of voxels is then represented as a linked list of spans of voxels of the same category. The transitional spans are represented fully, while only length of the other spans is stored. In both cases, reading and writing are efficient, and if there are large homogeneous regions in the data set (which is often the case), the memory efficiency is very good, reaching compression values at the level of a few per cent of the full volume representation. However, if the distance field contains features at very diverse scales, the *Adaptive Distance Fields* technique, proposed by Frisken et al. [45] is a better choice for its representation, at the cost of more complex storage and retrieval. The basic idea is to subdivide space using an octree data structure [74], [101]. The distance values are sampled from a continuous distance field at the vertices of each cell in the octree, and a cell is split recursively until the interpolated distance field within closely matches the continuous field.

ADFs are useful for compactly representing complex distance fields. Frisken et al. also demonstrate how their ADFs can be manipulated using CSG operations. In more recent work, Perry and Frisken [88] improve on some of

their results. Especially, techniques for fast rendering using points and techniques for triangulation are proposed. The triangulation is an extension of surface nets [43] from regular to adaptive grids. In recent work by Ju et al. [57], the method has been extended further to adaptive grids where precise edge intersections and normals are known. Finally, a new, faster tile-based generation of ADFs is proposed.

An ADF scheme was also proposed by Bærentzen [5]. The data structure and the CSG operations resemble the work by Frisken and Perry. However, Bærentzen proposes a simple decoupling of the space subdivision and the representation of distances at cell corners. An octree is used to represent the space subdivision whereas a 3D hash table is used to represent points. This decoupling is important because cells share corners. Thus (without a separate point data structure), each cell must have either separate copies of the distance values at its corners or pointers to these values. With a 3D hash table, the corner position is simply used to look up distance values.

In [53], a representation called a *Complete Distance Field Representation* (CDFR) is proposed. A CDFR is a regular subdivision of space into a 3D grid of cells where each cell contains a set of triangles. These are the triangles which might affect the distance field within the cell. To build this data structure, triangles are initially stored in the cells they intersect. Subsequently, the triangles are pushed to neighboring cells. A neighboring cell tests whether the triangle might influence its distance field and stores the triangle if this is the case. When no cells contain triangles that influence neighboring cells, the process stops. From the triangles stored in each cell, we can compute the exact shortest distance from a point in the cell to the surface.

Another approach which was recently proposed by Wu et al. is to construct a BSP tree from a triangle mesh [133]. Within each cell, the distance field is approximated by a simple linear function (plane distance). The triangle model is passed to a function that recursively splits the model into smaller pieces. For each piece of the model, a distance field approximation is generated and the approximation error is estimated. If the error is above a given tolerance the model

is split again. Medial axis information is used to split the model approximately along high curvature ridges. The method leads to a very compact representation. It is not even C^0, but the authors argue that this is less important as long as the maximum approximation error can be controlled. As an example of an application, the method is used to guarantee a bound on the error during mesh decimation.

The techniques proposed by Kobbelt et al. [63] and Qu et al. [93] are aimed at representation and subsequent reconstruction of surfaces with sharp edges by triangulation techniques. This is, referring to the sampling theory, a task which is in general not possible to accomplish by means of a regular sampled field. Therefore, additional information should be provided. In the first case, the *enhanced DF representation* [63], for each voxel its directed distances along the x, y, and z axes to the surface are stored. Thus, more precise information about the surface shape is provided. In the latter technique, the *offset DF* (ODF) [93], the distance field is sampled in a semiregular pattern, which iteratively adapts itself to the actual geometry. Further on, the authors propose a *unified DF* (UDF), which combines the aforementioned offset DF with representations proposed in [63], [53], [57], together with the plain DF storing just the minimal distances. Their motivation is that none of these techniques can itself successfully capture all the possible variations of surface details and therefore the most suitable one should always be chosen.

These representations have very different properties: CDFR, ODF, UDF, and Wu's BSP tree representation can be construed as static data structures for accelerating distance computations (almost like Meshsweeper [48]). Hierarchical and run-length encoded grids and ADFs on the other hand allow for modifications of the distance field, the latter being the most suitable in situations where very small features in the distance field are of interest.

Nielsen and Museth [79] propose *Dynamic Tubular Grids* (DT-grids) as a data structure for representing the evolving interface during PDE simulation. DT-grids are a very compact data structure just representing the interface, and are therefore dependent upon the size of the interface rather than the size of the domain of the simulation (as grid methods such as ADFs are). The advantages of this scheme include being able to track the interface "out of the box"—the interface is not restricted to a finite grid as is the case in the other representations. They give an example simulation where their method requires just 64MB compared to a hierarchical method which would require 5GB of grid storage, and offers better time performance. The data is represented by 1D columns which are runs of connected data. Additional structures store (or imply) the coordinates of voxels within the run, and store the voxel values. Algorithms are given for accessing the grid randomly, via neighboring voxels in the coordinate direction and within a stencil.

Jones [61] gives a study into compressing distance fields. He derives a predictor for calculating distance based upon the vector transform (Section 3.2.3). If the predictor is successful in one of 13 previously visited directions a direction value indicating the direction can be stored. Otherwise, the full distance value is stored. This is followed by entropy encoding on the directions, and bit-plane encoding followed by entropy encoding on the distance values. He shows that this lossless Vector Transform predictor gives a compression similar to the lossy wavelet transform where around 75 percent of the coefficients have been set to zero (i.e., files sizes are 25 percent of the original). Jones [61] gives further analysis.

5 APPLICATIONS

5.1 Object Skeletons, Centerlines, and Medial Axes

An object skeleton (medial axis, centerline [14]) in a plane is a locus of points having equal distance to at least two distinct points on the object boundary. In 3D space this leads to the notion of a medial surface and, in order to obtain a 1D centerline, the condition must be strengthened to at least three closest surface points. Unfortunately, such a set of points can be discontinuous and subsequent processing ensuring continuity is necessary.

Skeletons are a highly compact representation of objects, serving for their description, recognition, collision detection, navigation, animation, etc. This multitude of purposes leads to different requirements on the precision, with which a skeleton should describe the given object. For further analysis and eventual recreation [32], [26] they should represent the object with high fidelity, keeping all its topological properties. At the other extreme, when used as navigation paths, just the most important features should be kept [143], [12]. Therefore, a large variety of specialized skeletonization and centerline detection approaches exist, many of which depend on DF analysis [77] and detection of its C^1 discontinuities—ridges. For example, Blum [14] defined the skeleton as a locus of those DF points, where it is not possible to define a tangent plane. Gagvani and Silver detect skeletal points taking advantage of an observation that the distance of a ridge point is larger than average of its neighbors [46]. This, by specifying a threshold of this difference, enables them to control the skeleton complexity. For a similar purpose, in order to enable the level-of-detail representation and also to decrease skeleton sensitivity on surface noise, Malandain and Fernández-Vidal [75] use the fact that in vector DTs the position of the closest surface point is registered and that two mutually close points, which straddle the medial axis, have their corresponding closest surface points far apart (Section 2.3). Observing that magnitude of the angle defined by these surface points and the medial axis location reflects importance of the given voxel as a skeletal one, they propose adjustable parameters for their classification. The skeleton detection approach proposed by Siddiqi et al. [103] arises from the conservation of energy principle, which is violated at the ridge (shock) locations of a vector field. First, a Euclidean distance field to a continuous approximation of the object surface is computed. In a second step, its gradient vector field is obtained, followed by local computation of the field's net outward flux. The flux field is zero everywhere, except for the shock (skeleton) points, where it is negative due to local energy loss. Thus, the skeletal points are obtained by thresholding of the flux field. This algorithm can lead to a skeleton topology which is different with respect to the topology of the original object. Therefore, in a sequel [20], the technique was extended to preserve topology by introduction of a homotopy preserving thinning technique.

In navigation, a centerline, i.e., a central path through the often elongated and winding object is required [143], [12], [130]. It is usually a simple path connecting the start and end points. If the goal is navigation in a tree structure, the path can be represented by a tree structure of simple segments. In this sense the centerline should be connected and should not contain segments, which represent unimportant windings, bulges or even noisy artifacts. Since neither of these requirements is ensured by the general

purpose skeletonization algorithms, special ones were developed.

Typically, two distance fields are employed. One, which similarly as in the aforementioned skeletonization approaches ensures centering of the path within the object and is computed with respect to the object boundary, and one which is computed with respect to the path starting point and provides the necessary topological and connectivity information. Zhou and Toga [143] use a 3-4-5 distance metric for the first field and a simpler 1-2-3 metric for the second one. Areas with constant values (codes) of the second field are denoted clusters. Since a cluster is approximately planar and perpendicular to the object axis, it usually contains a single voxel with a maximum value of the first field. Thus, sequentially inspecting clusters in the order of their increasing code, centerline voxels are extracted with ensured connectivity and centering.

The penalized volumetric skeleton algorithm proposed by Bitter [21], [12] builds upon Dijkstra's minimum cost path algorithm [25]. Here, the cost is given as a sum of the piecewise Euclidean lengths of path segments (segments are on the voxel level, so the possible distances are 1, $\sqrt{2}$, and $\sqrt{3}$) and voxel penalties, which increase with decreasing distance to the object boundary. Thus, the centerline is the simple path of maximum length among all lowest cost paths. Further, object skeletons may be built as the tree of maximum length lowest cost paths found in the object.

5.2 Morphology

Distance fields have been found useful for morphological operations [104] of erosion (upon which the medial axis algorithms are usually based) and dilation.

The mathematical morphological operation of erosion [49] will remove external parts of an object (depending upon the structuring element). Dilation will add parts to the boundary of the object. Opening will enlarge cracks and cavities, and closing will remove cavities and smooth spikes.

Given a structuring element B, erosion of an object X is defined as:

$$X \ominus B = \{\mathbf{p}|B_{\mathbf{p}} \subset X\} \tag{15}$$

and dilation as:

$$X \oplus B = \{\mathbf{p}|B_{\mathbf{p}} \cap X \neq \emptyset\}. \tag{16}$$

Closing is a dilation followed by an erosion:

$$X \bullet B = (X \oplus B) \ominus B \tag{17}$$

and opening is an erosion followed by a dilation:

$$X \circ B = (X \ominus B) \oplus B. \tag{18}$$

If B is defined as a ball of radius r, then erosion can be defined as:

$$X \ominus B = \{\mathbf{p}|d(\mathbf{p}) \leq -r\} \tag{19}$$

and dilation as:

$$X \oplus B = \{\mathbf{p}|d(\mathbf{p}) \leq r\}. \tag{20}$$

This gives us a practical algorithm for closing volumetric objects [59]. First, the distance field, d is calculated (2). The surface, $S^{\oplus r}$, representing a dilation with a ball of radius r is equivalent to $S^{\oplus r} = \{\mathbf{q} : d(\mathbf{q}) = r\}$, where $\mathbf{q} \in \mathbb{R}^3$. To calculate the erosion with a ball of radius r for this surface $S^{\oplus r}$, we calculate a new distance field, d' based upon distances from surface $S^{\oplus r}$ using one of the techniques described earlier (measuring to the isosurface). The surface $S^{\bullet r}$ which has been closed with a ball of radius r using a dilation of degree r followed by an erosion of degree r is given by $S^{\bullet r} = \{\mathbf{q} : d'(\mathbf{q}) = -r\}$. Opening is defined in a similar manner.

Morphological operators have been carried out on binary segmented data by Höhne [52]. The use of three-dimensional erosions and dilations was reported as a useful application for the extraction of homogenous regions in body scans. They have also been used by Museth et al. [73] as editing operators for level sets. Distance fields allow the intuitive view of morphological operations that a dilation or erosion of 5mm should be an offset surface 5mm outside or inside the object.

Fig. 9 demonstrates accurate 3D closing operations of 20, 10, and 5 voxels on the eye socket and cheek of the UNC CThead.

5.3 Visualization

5.3.1 Accelerated Ray-Tracing of Volumetric Data

In volume graphics [64], objects are represented by 3D grids of densities, distances, or other attributes. The *Volume of Interest* (VOI) usually only occupies part of the complete grid and the proportion of time spent on processing the irrelevant areas can be significant. One possibility to minimize the time spent processing these areas is to identify the empty background voxels by segmentation and to gather them into macroregions which can be ignored during rendering. In ray-tracing, this increases the mean step length, and results in the speed up of single ray tracing in comparison with fixed length algorithms.

In distance-based acceleration, empty cuboid, octahedral, or (at least approximately) spherical macro regions are defined for each background voxel with a radius equal to its distance to the nearest surface point. This idea was introduced by Zuiderveld et al. [141], [142] in the Ray Acceleration by Distance Coding (RADC) scheme, which works in two phases. First, a chamfer approximation of the Euclidean distance (using a $5 \times 5 \times 5$ kernel according to [129], Section 3.2.2) is computed for each background voxel and stored in a volume of the same dimension as the data grid. The distance information permits skipping of an appropriate number of samples in the background, while in the foreground the ray is traversed with unit step length.

The IsoRegion Leaping technique [39] extends the RADC in that also homogeneous foreground regions are identified and filled by distances to their boundary. Once a voxel with a certain density and step length is reached, color and opacity of the whole segment is accumulated, and the appropriate length is skipped. A similar Skip Field technique [40] enables higher acceleration rates by trading quality for speed. The gradient base error metrics [33] are used to identify the homogeneous regions. If no error is tolerated, the inside distance information is ignored, otherwise the homogeneous inside regions are skipped rapidly.

The Proximity Clouds (PC) technique, similar to RADC, was proposed by Cohen and Sheffer [29]. It differs from

RADC in that it takes the shape of the free zone—the proximity cloud—around a background voxel into account: therefore, the step size depends not only on the distance value, but also on the type of distance and ray direction.

The Chessboard Distance (CD) voxel traversal algorithm [114] takes advantage of the simple cubic geometry of macroregions defined by the chessboard distance which enables the accelerated traversal of cubic, regular, and even rectilinear voxel grids. The algorithm defines a ray as a nonuniform sequence of samples, which are positioned exactly at voxel faces. This property enables rapid ray-surface intersection computation by gradient-based root finding techniques. Further, a variant of the CD algorithm was proposed in [114] for traversal of anisotropic macro-regions. Its motivation resides in that distance to already passed objects is often the limiting factor of the traversal speed. Therefore, each background voxel is assigned to eight macroregions, each registering directed distance to the nearest object in the direction of the main body axis of the volume. A similar technique was proposed in by Semwal and Kvanrstrom [112], which needs only six distance volumes but requires more complex distance computation.

In [56], [114], the CD distance field was used to speed up the template-based voxel traversal [138] with 26-connected ray templates. If such a template is implemented as an array of voxel positions, a known CD value d of a given voxel enables a jump of d voxels in this array template.

5.3.2 Distance Acceleration in Endoscopy

Using distance-based acceleration for endoscopy often leads to low values, which reflect distances to structures which are out of the viewing frustum. Therefore, in addition to the standard distance-based acceleration [134], [136], techniques have been developed which attempt to find a better acceleration than that obtained by the basic distance transform. Shargi and Ricketts [120] assigned most of the voxels inside the colon cavity to nonoverlapping spheres, which the ray can safely skip. The spheres are constructed by means of the 3-4-5 chamfer distance transform (Section 3.2.2) and identification of its local extrema. Spheres are obviously not the optimal shape to approximate tubular structures. The technique proposed by Vilanova et al. [128] therefore uses cylinders for their approximation and, consequently, they obtain a higher speed up than the aforementioned plain distance and sphere techniques. Distances play a twofold role, they are used both in the cylinder construction and rendering.

5.3.3 Distance Field Visualization

Distance fields may be visualized like any other implicit representation, but they also have one big advantage: Except for points belonging to the cut locus, we know that a point $\mathbf{p}$ can be projected onto the surface using a simple formula

$$\mathbf{p}_f = \mathbf{p} - \nabla d_S(\mathbf{p})d_S(\mathbf{p}), \qquad (21)$$

where $\mathbf{p}_f \in \partial S$ is sometimes denoted as the *foot point* of $\mathbf{p}$. The foot point can be drawn directly as a point primitive [100]. This method was used in [10], and in [88], Perry and Frisken implement a similar method in the context of adaptive distance fields.
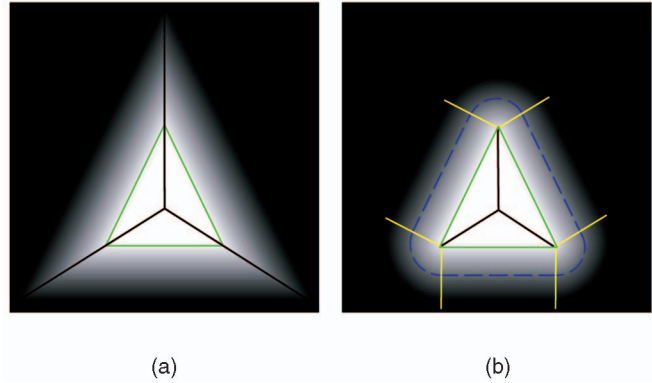

Fig. 10. A cut through a tetrahedron defined by intersection of halfspaces. (a) Intersection by the fuzzy intersection operator and (b) DF representable intersection. Green lines: inner surface, black: medial axis, and blue: surface of the DF representable Intersection.

5.4 Modeling

5.4.1 CSG Operations

In standard CSG, the basic operations union, intersection, and difference on solids are defined by the regularized Boolean set operations [38]. This concept was extended also to solids represented by scalar fields [19], [132], [30], with the Boolean operations replaced by fuzzy set operations, and later even to distance fields [16], [115]. Unfortunately, such an implementation often leads to sharp details, where the resulting values do not represent distance to the boundary of the solid δS anymore (Fig. 10a). While this situation occurs mainly in the vicinity of object edges, it is not a problem in many applications (for example, in visualization, unless it causes visible reconstruction artifacts [81]). However, in volumetric shape modeling and sculpting this becomes a major problem [11].

Assuming that the operands of a CSG operation are DFs of representable solids (Section 2.3), in order to obtain a correct result (i.e., a distance field of a representable solid), it is necessary to incorporate the DF representability criterion (Section 2.3) also in the distance field CSG operators. Such a scheme based on morphology was developed by Bærentzen and Christensen [11] for the purpose of volumetric sculpting and implements volumetric CSG operations between a discrete distance field and an unsampled continuous distance field of a sculpting tool.

Another scheme proposed by Novotny et al. [81] processes two sampled distance fields. They assume that the DFs of input objects are truncated (Section 4) at a distance equal to the radius R_s, as it is defined in the DF representability criterion. Thus, each object O partitions the space in three areas: a transition area T_O with $-R_s < d < R_s$, inner area I_O with $d = -R_s$ and outer area O_O with $d = R_s$. The boundary between the inner (outer) and transition area is called inner (outer) surface (Fig. 10). In CSG intersection (extension to other operations is straightforward), the fuzzy operators can be used for both inner and outer areas, and even for most voxels which are in the transition area of just one solid. Such areas are delineated in Fig. 10 by the yellow lines. However, in the edge vicinity, its is necessary to ensure that the corresponding medial surface at most touches the inner surface of the result, in order to keep the representability criterion fulfilled. Therefore, it is necessary to detect the intersection line of the operands'

inner surfaces and to store the distance to this intersection in the edge area voxels.

Schroeder et al. [116] use distance fields to create swept surfaces and volumes. The object to be swept is voxelized to create a distance field. For each segment along the swept path voxels within the domain are inverse transformed back to the distance field, and the distance field is trilinearly sampled. The result is that the level 0 isosurface of the produced DF represents the swept volume.

5.4.2 Level Set Method

The level set method (LSM) [86], [107], [84] is a technique for tracking deforming interfaces with a great number of applications [107], [85]. The interface is defined as the 0-level set of a function $\Phi : \mathbb{R}^3 \to \mathbb{R}$, and it is deformed indirectly by changing Φ according to the speed of the interface F.

Almost invariably, Φ is initialized to a signed distance field, and in the interest of numerical stability this property is usually maintained using some method for reinitialization (c.f. Section 3.5). It has been observed that the distance property can be maintained to some degree of accuracy if the speed function is constant along a line perpendicular to the interface [139], [27], [3]. Alternatively, Gomes and Faugeras [42] propose a reformulation of the LSM which preserves the distance field property apparently without need for reinitialization. Yet, another approach is the sparse field method proposed by [131] where a simplified reinitialization is an integral part of the method which keeps explicit lists of layers (ordered by distance) of adjacent grid points.

5.4.3 Animation

Gagvani and Silver [46], [47] use distance fields to create a skeletonized model of a volumetric data set. The distance field model is thinned (or peeled) to create a 1 voxel thick representation of the volume model. The user then defines those voxels about which motion can take place, and then this inverse kinematic (IK) skeleton is loaded into and manipulated using standard animation software to create motion. Finally, the volumetric data set is reconstructed around the manipulated IK skeletons for each frame of the animation. Reconstruction is carried out by scan filling known spheres about the skeleton where the size of the sphere corresponds to the isosurface data.

5.4.4 Collision Detection

For collision detection, distance fields have been used both in the context of rigid [41], [36] and deformable [51], [15] body simulations.

A simple approach in the case of rigid bodies is to use a dual representation. A set of sample points on the object surface (e.g., the vertices) is one representation and a distance field is the other. When two objects collide, the points of one object can be tested against the distance field of the other. If a point is detected to be inside the other object (according to the distance field sign) it counts as a contact point, and the gradient of the distance field can be used as contact normal [36]. Alternatively, one might trace a ray in the direction of the gradient to find the exact contact point and the true surface normal at that point [41].

5.4.5 Dynamic Simulation

Fujishiro and Aoki [37] approximated thawing ice by using erosion and dilation operations (Section 5.2) on binary
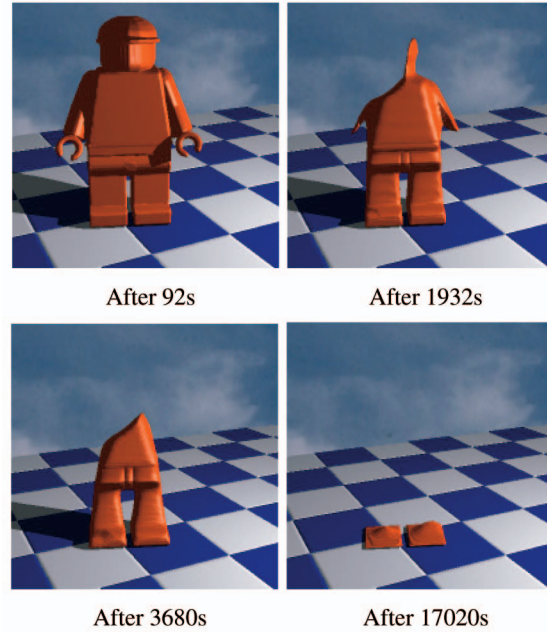

Fig. 11. 24cm high wax legoman. $k = 0.18$ W/(m°C), $\rho = 1,000$ kgm^{-3}, $c = 2,000$ J/kg°C, $h = 0.001$m, $L = 105$kJ, with radiation from a 100°C heat source at the top left.

voxelized data (each of their voxels is two-state—inside or outside). Their method provides a certain amount of realism (irregular melting) by using different sized structuring elements (in their case different radii spheres) in areas where the object surface can see more of the heat source. Their model is not physically based as there is no calculation of heat energy and by using a distance field the process is just visually improved.

Jones [60] introduces a model that accurately models both thermal flow and latent heat in rigid solids with complex surfaces. By using the phase mixture formulation based upon the distance field, the method allows a fast (volume preserving) physically-based animation that allows effective control of the melting process by using well-defined parameters derived from the physical properties of the material undergoing the phase transition. One major advantage of employing a distance field on a voxel grid is that it provides a structure that allows the fast computation of heating due to radiative transfer.

By replacing the partial derivatives with finite differences, Jones [60] shows that it is relatively straightforward to solve the Enthalpy formulation of the Heat Equation using a discrete voxel grid giving results like that of Fig. 11.

5.4.6 Morphing

Cohen-Or et al. [28] implement morphing with a user-defined geometric warping between the source and target distance fields. Interpolation between the warped distance fields is then used to create the in-between object. Breen and Whitaker [22] use level sets and distance fields to create a morph between solid models. First, the source and target objects are converted into signed distance fields. The two objects are registered in order to create an affine transformation between them. A level set is fitted to the zero level of the source object, and using the transformation a corresponding point in the distance field of the target object is

Fig. 12. (a) Bozo and melting texture on tank. (b) Clipped hair on skull.

found. The level set evolves in the direction of the local surface normal with a speed proportional to the signed distance to the target at the corresponding transformed point. They demonstrate that this method creates convincing morphs between objects with little or no user input.

5.4.7 Hypertexture

Perlin and Hoffert [89] proposed hypertexture as a way of adding texture detail or noise (such as fur, smoke, and fire) to objects. The method relies upon being able to specify a soft region $D(p)$ around the object, such that each point p can be classified as inside the object, in the soft region (where the hypertexture effect is applied), or completely outside the influence of the object and its texture. This is relatively straightforward for implicitly defined objects $f(x, y, z) = r$, such as spheres and torii, where the soft region can be established for all points p where $r_i < f(p) < r_o$ for some inner radius r_i and outer radius r_o.

Objects defined using triangular mesh data do not readily have such a property, but Satherley and Jones [111] demonstrated that a space-filled distance field d could be used, and a soft region $D(p)$ could be defined about the object:

$$D(p) = \begin{cases} 1 & \text{if } \mathrm{d}(p)^2 \le r_i^2 \\ 0 & \text{if } \mathrm{d}(p)^2 \ge r_o^2 \\ \frac{r_o^2 - \mathrm{d}(p)^2}{r_o^2 - r_i^2} & \text{otherwise,} \end{cases}$$

where   (22)

$r_i =$ inner radius,

$r_o =$ outer radius,

and $p \in \mathbb{R}^3$.

$D(p)$ is used in the hypertexture gain, bias, noise and turbulence functions of Perlin and Hoffert [89]. Using voxelization, this approach allows hypertexture to be applied to general objects. Fig. 12 shows hypertexture on a voxelized tank and the UNC CThead data set, and also demonstrates combining hypertexture and clipping hypertexture. Miller and Jones [76] demonstrate the implementation of hypertexture on a GPU to enable real-time rendering of animated fire texture amongst other effects on complex objects.

5.4.8 Mesh Processing

Distance fields are potentially very useful in mesh manipulation algorithms. In [9], Botsch et al. argue that when multiple manipulations are applied to a mesh, it is difficult to precisely compute the cumulative error.

An effective solution to this problem is to construct a distance field around the original surface. The manipulated surface can then be checked against this distance field. In practice, Botsch et al. propose to render the manipulated mesh using the distance field of the original as a 3D texture. Thus, we can render the manipulated mesh with the distance to the original mesh mapped on as a color using 3D texture mapping. Pixels are then culled if the mapped distance values are below some tolerance. Consequently, pixels are only drawn if they are above the tolerance. Thus, a simple occlusion query [100] can be used to check whether the manipulated mesh is within a given distance of the original.

5.4.9 Mesh Generation

For many applications, such as soft body simulations, simplicial meshes (i.e., triangle meshes in 2D or tetrahedral meshes in 3D) are required. These meshes are typically generated from some known geometry. Distance fields can be a useful representation of this geometry, since it is very easy to detect whether a mesh vertex is inside the shape, and the vertex is easily projected onto the boundary. See Bridson et al. [95] and the PhD thesis of Per-Olof Persson [87].

6 DISCUSSION

Building upon the properties of continuous distance fields (Section 2), this paper has presented a survey of the main techniques for the production of three-dimensional discrete distance fields (Section 3). In particular, we gave detailed directions for creating correctly signed distance fields of triangular mesh data (Section 3.1). To reduce voxelization time, we have shown that it is possible to clamp the distance field to a shell around the object, and make use of distance transform methods (such as chamfer, vector and FMM) to enumerate the remainder of the discrete space (Section 3.2). As these methods are approximating, it becomes natural to ask the questions we gave in the abstract—*How accurate are these methods compared to each other?*, *How simple are they to implement?*, and *What is the complexity and runtime of such methods?* Section 3.6 gave an analysis to provide answers to those questions. In summary, it appears that for best accuracy, we should ensure the the initial surface shell (Section 3.2.1) contains vectors to the surface rather than just distances, and then use a vector transform. Where just distances are available, then it appears that using the new SFMMHA method will give the most accurate results, in the least time, and is easier to implement than the previously published FMMs. After giving a review on the production of distance fields, we documented alternative representations (Section 4), and their application to various situations (Section 5).

REFERENCES

[1]   H. Aanæs and J.A. Bærentzen, "Pseudo-Normals for Signed Distance Computation," *Proc. Conf. Vision, Modeling, and Visualization,* pp. 407-413, 2003.
[2]   M. Akmal Butt and P. Maragos, "Optimum Design of Chamfer Distance Transforms," *IEEE Trans. Image Processing,* vol. 7, no. 10, pp. 1477-1484, 1998.
[3]   D. Adalsteinsson and J.A. Sethian, "The Fast Construction of Extension Velocities in Level-Set Methods," *J. Computational Physics,* vol. 148, no. 1, pp. 2-22, 1999.
[4]   J.A. Bærentzen and H. Aanæs, "Signed Distance Computation Using the Angle Weighted Pseudo-Normal," *IEEE Trans. Visualization and Computer Graphics,* vol. 11, no. 3, pp. 243-253, May/ June 2005.
[5]   J.A. Bærentzen, "Volumetric Manipulations with Applications to Sculpting," PhD thesis, IMM, Technical Univ. of Denmark, 2001.

[6] J.A. Bærentzen, "On the Implementation of Fast Marching Methods for 3D Lattices," Technical Report IMM-REP-2001-13, DTU.IMM, 2001.

[7] J.A. Bærentzen, "Robust Generation of Signed Distance Fields from Triangle Meshes," *Proc. Int'l Workshop Volume Graphics,* pp. 167-175, June 2005.

[8] A.H. Barr, "Superquadrics and Angle-Preserving Transformations," *IEEE Computer Graphics and Applications,* vol. 1, no. 1, pp. 11-23, Jan. 1981.

[9] M. Botsch, D. Bommes, C. Vogel, and L. Kobbelt, "GPU-Based Tolerance Volumes for Mesh Processing," *Proc. Conf. Pacific Graphics '04,* pp. 237-243, 2004.

[10] J.A. Bærentzen and N.J. Christensen, "Interactive Modelling of Shapes Using the Level-Set Method," *Int'l J. Shape Modeling,* vol. 8, no. 2, pp. 79-97, Dec. 2002.

[11] J.A. Bærentzen and N.J. Christensen, "A Technique for Volumentric CSG Based on Morphology," *Proc. Conf. Volume Graphics '01,* pp. 71-79, June 2002.

[12] I. Bitter, A.E. Kaufman, and M. Sato, "Penalized-Distance Volumetric Skeleton Algorithm," *IEEE Trans. Visualization and Computer Graphics,* vol. 7, no. 3, pp. 195-206, 2001.

[13] *Introduction to Implicit Surfaces,* J. Bloomenthal, ed., Computer Graphics and Geometric Modeling, Morgan Kaufman, 1997.

[14] H. Blum, "A Transformation for Extracting New Descriptors of Shape," *Proc. Symp. Models for the Perception of Speech and Visual Form,* Nov. 1964.

[15] R. Bridson, S. Marino, and R. Fedkiw, "Simulation of Clothing with Folds and Wrinkles," *Proc. ACM SIGGRAPH/Eurographics Symp. Computer Animation (SCA),* pp. 28-36, 2003.

[16] D.E. Breen, S. Mauch, and R.T. Whitaker, "3D Scan Conversion of CSG Models into Distance Volumes," *Proc. IEEE Symp. Volume Visualization,* pp. 7-14, Oct. 1998.

[17] D. Breen, S. Mauch, and R. Whitaker, "3D Scan Conversion of CSG Models into Distance, Closest-Point and Colour Volumes," *Volume Graphics,* M. Chen, A.E. Kaufman, and R. Yagel, eds., pp. 135-158, London: Springer, 2000.

[18] G. Borgefors, "On Digital Distance Transforms in Three Dimensions," *Computer Vision and Image Understanding,* vol. 64, no. 3, pp. 368-376, 1996.

[19] D. Breen, "Constructive Cubes: CSG Evaluation for Display Using Discrete 3D Scalar Data Sets," *Proc. EUROGRAPHICS '91,* pp. 127-142, 1991.

[20] S. Bouix and K. Siddiqi, "Divergence-Based Medial Surfaces," *ECCV '00: Proc. Sixth European Conf. Computer Vision-Part I,* pp. 603-618, June 2000.

[21] I. Bitter, M. Sato, M. Bender, K.T. McDonnell, A. Kaufmann, and M. Wan, "CEASAR: A Smooth, Accurate and Robust Centerline Extraction Algorithm," *Proc. Conf. Visualization '00,* pp. 45-52, 2000.

[22] D.E. Breen and R.T. Whitaker, "A Level-Set Approach for the Metamorphosis of Solid Models," *IEEE Trans. Visualization and Computer Graphics,* vol. 7, no. 2, pp. 173-192, 2001.

[23] D.L. Chopp, "Some Improvements of the Fast Marching Method," *SIAM J. Scientific Computing,* vol. 23, no. 1, pp. 230-244, 2001.

[24] *Volume Graphics,* M. Chen, A.E. Kaufman, and R. Yagel, eds., London: Springer Verlag, 2000.

[25] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms.* MIT Press, 1990.

[26] O. Cuisenaire and B. Macq, "Applications of the Region Growing Euclidean Distance Transform: Anisotropy and Skeletons," *Proc. ICIP '97 VOLI,* pp. 200-203, 2000.

[27] S. Chen, B. Merriman, S. Osher, and P. Smereka, "A Simple Level Set Method for Solving Stefan Problems," *J. Computational Physics,* vol. 135, no. 1, pp. 8-29, July 1997.

[28] D. Cohen-Or, A. Solomovic, and D. Levin, "Three-Dimensional Distance Field Metamorphosis," *ACM Trans. Graphics,* vol. 17, no. 2, pp. 116-141, 1998.

[29] D. Cohen and Z. Sheffer, "Proximity Clouds—An Acceleration Technique for 3D Grid Traversal," *The Visual Computer,* vol. 10, no. 11, pp. 27-38, Nov. 1994.

[30] M. Chen and J.V. Tucker, "Constructive Volume Geometry," *Computer Graphics Forum,* vol. 19, no. 4, pp. 281-293, 2000.

[31] O. Cuisenaire, "Distance Transformation: Fast Algorithms and Applications to Medical Image Processing," PhD thesis, Univ. Catholique de Louvain, Oct. 1999.

[32] P.-E. Danielsson, "Euclidean Distance Mapping," *Computer Graphics and Image Processing,* vol. 14, pp. 227-248, 1980.

[33] J. Danskin and P. Hanrahan, "Fast Algorithms for Volume Ray Tracing," *Proc. Workshop Volume Visualization,* pp. 91-98, 1992.

[34] F. Dachille and A. Kaufman, "Incremental Triangle Voxelization," *Proc. Graphics Interface '00,* pp. 205-212, 2000.

[35] K. Erleben and H. Dohlman, personal communication, 2005.

[36] K. Erleben, "Stable, Robust, and Versatile Multibody Dynamics Animation," PhD thesis, Dept. of Computer Science, Univ. of Copenhagen, 2004.

[37] I. Fujishiro and E. Aoki, "Volume Graphics Modeling of Ice Thawing," *Proc. Conf. Volume Graphics '01,* pp. 69-80, 2001.

[38] J.D. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics.* Addison-Wesley, 1982.

[39] P.-F. Fung and P.-A. Heng, "Efficient Volume Rendering by Isoregion Leaping Acceleration," *Proc. Sixth Int'l Conf. Central Europe Computer Graphics and Visualization '98,* pp. 495-501, 1998.

[40] J. Freund and K. Sloan, "Accelerated Volume Rendering Using Homogenous Region Encoding," *Proc. IEEE Conf. Visualization '97,* pp. 191-196, Nov. 1997.

[41] E. Guendelman, R. Bridson, and R.P. Fedkiw, "Nonconvex Rigid Bodies with Stacking," *ACM Trans. Graphics,* vol. 22, no. 3, pp. 871-878, 2003.

[42] J. Gomes and O.D. Faugeras, "Reconciling Distance Functions and Level Sets," *Scale-Space Theories in Computer Vision,* pp. 70-81, 1999.

[43] S.F.F. Gibson, "Using Distance Maps for Accurate Surface Representation in Sampled Volumes," *Proc. IEEE Symp. Volume Visualization,* pp. 23-30, Oct. 1998.

[44] J. Gielis, "A Generic Geometric Transformation that Unifies a Wide Range of Natural and Abstract Shapes," *Am. J. Botany,* vol. 90, pp. 333-338, 2003.

[45] S.F. Frisken Gibson, R.N. Perry, A.P. Rockwood, and T.R. Jones, "Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics," *Proc. SIGGRAPH '00,* pp. 249-254, 2000.

[46] N. Gagvani and D. Silver, "Parameter-Controlled Volume Thinning," *Proc. CVGIP: Graph. Models Image Process.,* vol. 61, no. 3, pp. 149-164, 1999.

[47] N. Gagvani and D. Silver, "Animating Volumetric Models," *Graph. Models,* vol. 63, no. 6, pp. 443-458, 2001.

[48] A. Guéziec, "Meshsweeper: Dynamic Point-to-Polygonal Mesh Distance and Applications," *IEEE Trans. Visualization and Computer Graphics,* vol. 7, no. 1, pp. 47-60, Jan.-Mar. 2001.

[49] R. Gonzalez and R. Woods, *Digital Image Processing.* Addison-Wesley, 1993.

[50] E. Hartmann, "On the Curvature of Curves and Surfaces Defined by Normalforms," *Computer Aided Geometric Design,* vol. 16, no. 5, pp. 355-376, 1999.

[51] G. Hirota, S. Fisher, and A. State, "An Improved Finite Element Contact Model for Anatomical Simulations," *The Visual Computer,* vol. 19, no. 5, pp. 291-309, 2003.

[52] K.H. Höhne and W.A. Hanson, "Interactive 3D Segmentation of MRI and CT Volumes Using Morphological Operations," *J. Computer Assisted Tomography,* vol. 16, no. 2, pp. 285-294, 1992.

[53] J. Huang, Y. Li, R. Crawfis, S.-C. Lu, and S.-Y. Liou, "A Complete Distance Field Representation," *Visualization, 2001, VIS '01 Proc.,* pp. 247-254, 2001.

[54] C.M. Hoffmann, *Geometric and Solid Modeling.* Morgan Kaufmann, 1989.

[55] J. Helmsen, E. Puckett, P. Colella, and M. Dorr, "Two New Methods for Simulating Photolithography Development in 3D," *SPIE 2726,* pp. 253-261, 1996.

[56] I. Holländer and M. Sramek, "An Interactive Tool for Manipulation and Presentation of 3D Tomographic Data," *Proc. CAR '93 Computer Assisted Radiology,* pp. 278-383, 1993.

[57] T. Ju, F. Losasso, S. Schaefer, and J. Warren, "Dual Contouring of Hermite Data," *Proc. 29th Ann. Conf. Computer Graphics and Interactive Techniques,* pp. 339-346, 2002.

[58] M.W. Jones, "The Production of Volume Data from Triangular Meshes Using Voxelisation," *Computer Graphics Forum,* vol. 15, no. 5, pp. 311-318, 1996.

[59] M.W. Jones, "Facial Reconstruction Using Volumetric Data," *Proc. Vision Modeling and Visualization Conf. '01 (VMV-01),* pp. 135-150, Nov. 2001.

[60] M.W. Jones, "Melting Objects," *The J. WSCG,* vol. 11, no. 2, pp. 247-254, 2003.

[61] M.W. Jones, "Distance Field Compression," *The J. WSCG,* vol. 12, no. 2, pp. 199-204, 2004.

[62] A. Kaufman, "Efficient Algorithms for 3D Scan-Conversion of Parametric Curves, Surfaces, and Volumes," *Computer Graphics (SIGGRAPH '87 Proc.),* vol. 21, no. 4, pp. 171-179, July 1987.

[63] L.P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel, "Feature Sensitive Surface Extraction from Volume Data," *SIGGRAPH '01: Proc. 28th Ann. Conf. Computer Graphics and Interactive Techniques,* pp. 57-66, 2001.

[64] A. Kaufman, D. Cohen, and R. Yagel, "Volume Graphics," *Computer,* vol. 26, no. 7, pp. 51-64, July 1993.

[65] S. Kim, "An O(N) Level Set Method for Eikonal Equations," *SIAM J. Scientific Computing,* vol. 22, no. 6, pp. 2178-2193, 2000.

[66] C.-Y. Kao, S. Osher, and J. Qian, "Lax-Friedrichs Sweeping Scheme for Static Hamilton-Jacobi Equations," *J. Computational Physics,* vol. 196, no. 1, pp. 367-391, May 2004.

[67] S.G. Krantz and H.R. Parks, "Distance to C^k Hypersurfaces," *J. Differential Equations,* vol. 40, no. 1, pp. 116-20, 1981.

[68] R. Kimmel and J.A. Sethian, "Computing Geodesic Paths on Manifolds," *Proc. Nat'l Academy of Sciences,* vol. 95, no. 15, pp. 8431-8435, 1998.

[69] H. Löffelmann and E. Gröller, "Parameterizing Superquadrics," *Proc. Winter School of Computer Graphics '95,* pp. 162-172, Feb. 1995.

[70] S. Mauch, "A Fast Algorithm for Computing the Closest Point and Distance Transform," Technical Report caltechASCI/2000.077, Applied and Computational Math., Calif. Inst. of Technology, 2000.

[71] S. Mauch, "Efficient Algorithms for Solving Static Hamilton-Jacobi Equations," PhD thesis, Calif. Inst. of Technology, Pasadena, Calif., 2003.

[72] O. Monga, S. Benayoun, and O.D. Faugeras, "From Partial Derivatives of 3D Density Images to Ridge Lines," *Proc. SPIE-The Int'l Soc. for Optical Eng.,* vol. 1808, pp. 118-26, 1992.

[73] K. Museth, D.E. Breen, R.T. Whitaker, and A.H. Barr, "Level Set Surface Editing Operators," *SIGGRAPH '02: Proc. 29th Ann. Conf. Computer Graphics and Interactive Techniques,* pp. 330-338, 2002.

[74] D. Meagher, "Geometric Modeling Using Octree Encoding," *Computer Graphics and Image Processing,* vol. 19, no. 2, pp. 26-147, June 1982.

[75] G. Malandain and S. Fernández-Vidal, "Euclidean Skeletons," *Image and Vision Computing,* vol. 16, no. 5, pp. 317-327, 1998.

[76] C.M. Miller and M.W. Jones, "Texturing and Hypertexturing of Volumetric Objects," *Proc. Int'l Workshop Volume Graphics '05,* pp. 117-125, 2005.

[77] U. Montanari, "A Method for Obtaining Skeletons Using a Quasi-Euclidean Distance," *J. ACM,* vol. 15, no. 4, pp. 600-624, 1968.

[78] J.C. Mullikin, "The Vector Distance Transform in Two and Three Dimensions," *CVGIP: Graphical Models and Image Processing,* vol. 54, no. 6, pp. 526-535, 1992.

[79] M.B. Nielsen and K. Museth, "Dynamic Tubular Grid: An Efficient Data Structure and Algorithms for High Resolution Level Sets," *J. Scientific Computing,* 2006.

[80] P. Novotny, "CSG Operations with Voxelized Solids," master's thesis, Comenius Univ., Faculty of Math., Physics and Informatics, Bratislava, Slovakia, May 2003.

[81] P. Novotny, M. Sramek, and L.I. Dimitrov, "CSG Operations with Voxelizad Solids," *Proc. CGI Conf. '04,* pp. 370-373, 2004.

[82] F.S. Nooruddin and G. Turk, "Simplification and Repair of Polygonal Models Using Volumetric Techniques," *IEEE Trans. Visualization and Computer Graphics,* vol. 9, no. 2, pp. 191-205, Apr.-June 2003.

[83] P. Novotny and M. Sramek, "Representation of Objects with Sharp Details in Truncated Distance Fields," *Proc. Conf. Volume Graphics '05,* pp. 109-116, 2005.

[84] S.J. Osher and R.P. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces,* first ed., Springer Verlag, Nov. 2002.

[85] *Geometric Level Set Methods in Imaging, Vision, and Graphics,* S. Osher and N. Paragios, eds., Springer, 2003.

[86] S. Osher and J.A. Sethian, "Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations," *J. Computational Physics,* vol. 79, no. 1, pp. 12-49, Nov. 1988.

[87] P.-O. Persson, "Mesh Generation for Implicit Geometries," PhD thesis, Mass. Inst. of Technology, Feb. 2005.

[88] R.N. Perry and S.F. Frisken, "Kizamu: A System for Sculpting Digital Characters," *Proc. SIGGRAPH,* pp. 47-56, 2001.

[89] K. Perlin and E. Hoffert, "Hypertexture," *Computer Graphics (Proc. SIGGRAPH '89),* vol. 23, no. 3, pp. 253-262, July/Aug. 1989.

[90] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang, "A PDE-Based Fast Local Level Set Method," *J. Computational Physics,* vol. 155, no. 2, Nov. 1999.

[91] R. Peikert and C. Sigg, "Optimized Bounding Polyhedra for GPU-Based Distance Transform," *Dagstuhl Seminar 023231 on Scientific Visualization: Extracting Information and Knowledge from Scientific Data Sets,* 2003.

[92] B.A. Payne and A.W. Toga, "Distance Field Manipulation of Surface Models," *Computer Graphics and Applications,* vol. 12, no. 1, 1992.

[93] H. Qu, N. Zhang, R. Shao, A. Kaufman, and K. Mueller, "Feature Preserving Distance Fields," *VV '04: Proc. 2004 IEEE Symp. Volume Visualization and Graphics (VV '04),* pp. 39-46, 2004.

[94] I. Ragnemalm, "The Euclidean Distance Transform in Arbitrary Dimensions," *Pattern Recogn. Lett.,* vol. 14, no. 11, pp. 883-888, 1993.

[95] N. Molino, R. Bridson, J. Teran, and R. Fedkiw, "Adaptive Physics Based Tetrahedral Mesh Generation Using Level Sets," *Eng. with Computers,* vol. 21, pp. 2-18, 2005.

[96] F. Rhodes, "Discrete Euclidean Metrics," *Pattern Recognition Letters,* vol. 13, no. 9, pp. 623-628, 1992.

[97] A. Rosenfeld and J.L. Pfaltz, "Sequential Operations in Digital Picture Processing," *J. ACM,* vol. 13, no. 4, pp. 471-494, 1966.

[98] G. Russo and P. Smereka, "A Remark on Computing Distance Functions," *J. Computational Physics,* vol. 163, no. 1, pp. 51-67, 2000.

[99] E. Rouy and A. Tourin, "A Viscosity Solutions Approach to Shape-From-Shading," *SIAM J. Numer. Anal.,* vol. 29, no. 3, pp. 867-884, 1992.

[100] M. Segal and K. Akeley, *The 100 Graphics System: A Specification (Version 2.0).* SGI, 2004.

[101] H. Samet, *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1990.

[102] S. Svensson and G. Borgefors, "Digital Distance Transforms in 3D Images Using Information from Neighbourhoods up to $5 \times 5 \times 5$," *Computer Vision and Image Understanding,* vol. 88, pp. 24-53, 2002.

[103] K. Siddiqi, S. Bouix, A. Tannenbaum, and S.W. Zucker, "The Hamilton-Jacobi Skeleton," *ICCV '99: Proc. Int'l Conf. Computer Vision,* vol. 2, p. 828, 1999.

[104] J. Serra, *Image Analysis and Mathematical Morphology.* Academic Press, 1982.

[105] J.A. Sethian, "A Fast Marching Level Set Method for Monotonically Advancing Fronts," *Proc. Nat'l Academy of Sciences US-Paper Ed.,* vol. 93, no. 4, pp. 1591-1595, 1996.

[106] J.A. Sethian, "Fast Marching Methods," *SIAM Rev.,* vol. 41, no. 2, pp. 199-235, 1999.

[107] J.A. Sethian, *Level Set Methods and Fast Marching Methods,* Cambridge Monographs on Applied and Computational Math., second ed., Cambridge Univ. Press, 1999.

[108] M. Sussman, E. Fatemi, P. Smereka, and S. Osher, "An Improved Level Set Method for Incompressible Two-Phase Flows," *Computers and Fluids,* vol. 27, nos. 5-6, pp. 663-680, 1998.

[109] R. Satherley and M.W. Jones, "Hybrid Distance Field Computation," *Proc. Conf. Volume Graphics,* pp. 195-209, 2001.

[110] R. Satherley and M.W. Jones, "Vector-City Vector Distance Transform," *Computer Vision and Image Understanding,* vol. 82, no. 3, pp. 238-254, 2001.

[111] R. Satherley and M.W. Jones, "Hypertexturing Complex Volume Objects," *The Visual Computer,* vol. 18, no. 4, pp. 226-235, 2002.

[112] S.K. Semwal and H. Kvarnstrom, "Directed Safe Zones and the Dual Extent Algorithms for Efficient Grid Traversal during Ray Tracing," *Graphics Interface,* W. Davis, M. Mantei, and V. Klassen, eds., pp. 76-87, May 1997.

[113] M. Sramek and A. Kaufman, "Alias-Free Voxelization of Geometric Objects," *IEEE Trans. Visualization and Computer Graphics,* vol. 5, no. 3, pp. 251-266, 1999.

[114] M. Sramek and A. Kaufman, "Fast Ray-Tracing of Rectilinear Volume Data Using Distance Transforms," *IEEE Trans. Visualization and Computer Graphics,* vol. 6, no. 3, pp. 236-252, 2000.

[115] M. Sramek and A. Kaufman, "vxt: A C++ Class Library for Object Voxelization," *Volume Graphics,* M. Chen, A.E. Kaufman, and R. Yagel, eds., pp. 119-134, London: Springer Verlag, 2000.

[116] W.J. Schroeder, W.E. Lorensen, and S. Linthicum, "Implicit Modeling of Swept Surfaces and Volumes," *Proc. IEEE Conf. Visualization,* pp. 40-45, Oct. 1994.

[117] J.A. Sethian and A. Mihai Popovici, "3D Traveltime Computation Using the Fast Marching Method," *Geophysics,* vol. 64, no. 2, pp. 516-523, 1999.

[118] A. Sud, M.A. Otaduy, and D. Manocha, "Difi: Fast 3D Distance Field Computation Using Graphics Hardware," *Computer Graphics Forum,* vol. 23, no. 3, 2004.

[119] C. Sigg, R. Peikert, and M. Gross, "Signed Distance Transform Using Graphics Hardware," *Proc. IEEE Conf. Visualization '03,* pp. 83-90, 2003.

[120] M. Sharghi and I.W. Ricketts, "A Novel Method for Accelerating the Visualisation Process Used in Virtual Colonoscopy," *Proc. Fifth Int'l Conf. Information Visualization,* pp. 167-172, Aug. 2001.

[121] S. Turek and S.-R. Hysing, "The Eikonal Equation: Numerical Efficiency vs. Algorithmic Complexity on Quadrilateral Grids," *Proc. ALGORITMY 2005,* pp. 22-31, Mar. 2005.

[122] M. Sussman, P. Smereka, and S. Osher, "A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow," *J. Computational Physics,* vol. 114, no. 1, pp. 146-159, Sept. 1994.

[123] J. Strain, "Fast Tree-Based Redistancing for Level Set Computations," *J. Computational Physics,* vol. 152, no. 2, pp. 664-686, 1999.

[124] Y.R. Tsai, "Rapid and Accurate Computation of the Distance Function Using Grids," *J. Computational Physics,* vol. 178, pp. 175-195, 2002.

[125] J.N. Tsitsiklis, "Efficient Algorithms for Globally Optimal Trajectories," *IEEE Trans. Automatic Control,* vol. 40, no. 9, pp. 1528-1538, Sept. 1995.

[126] G. Thürmer and C.A. Wüthrich, "Computing Vertex Normals from Polygonal Facets," *J. Graphics Tools,* vol. 3, no. 1, pp. 43-46, 1998.

[127] L. Velho, J. Gomes, and L.H. Figueiredo, *Implicit Objects in Computer Graphics.* New York: Springer Verlag, 2002.

[128] A. Vilanova i Bartrolì, A. König, and E. Gröller, "Cylindrical Approximation of Tubular Organs for Virtual Endoscopy," *Proc. Computer Graphics and Imaging Conf. '00,* pp. 283-289, 2000.

[129] B.J.H. Verwer, P.W. Verbeek, and S.T. Dekker, "An Efficient Uniform Cost Algorithm Applied to Distance Transforms," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 11, no. 4, pp. 425-429, 1989.

[130] M. Wan, F. Dachille, and A. Kaufman, "Distance-Field Based Skeletons for Virtual Navigation," *VIS '01: Proc. Conf. Visualization '01,* pp. 239-246, 2001.

[131] R.T. Whitaker, "A Level-Set Approach to 3D Reconstruction from Range Data," *Int'l J. Computer Vision,* vol. 29, no. 3, pp. 203-231, 1998.

[132] S.W. Wang and A. Kaufman, "Volume-Sampled 3D Modelling," *Computer Graphics and Applications,* vol. 14, no. 5, pp. 26-32, Sept. 1994.

[133] J. Wu and L. Kobbelt, "Piecewise Linear Approximation of Signed Distance Fields," *Proc. Vision, Modeling, and Visualization Conf.,* pp. 513-520, 2003.

[134] M. Wan, A. Kaufman, and S. Bryson, "High Performance Presence-Accelerated Ray Casting," *Proc. IEEE Conf. Visualization '99,* pp. 379-386, Oct. 1999.

[135] F.E. Wolter, "Cut Locus and Medial Axis in Global Shape Interrogation and Representation," technical report, MIT, 1993.

[136] M. Wan, A. Sadiq, and A. Kaufmann, "Fast and Reliable Space Leaping for Interactive Volume Rendering," *Proc. IEEE Conf. Visualization '02,* pp. 195-202, 2002.

[137] L. Yatziv, A. Bartesaghi, and G. Sapiro, "O(n) Implementation of the Fast Marching Algorithm," *IMA Preprint Series No. 2021,* Feb. 2005.

[138] R. Yagel and A. Kaufman, "Template-Based Volume Viewing," *Proc. Eurographics '92,* pp. 153-167, June 1992.

[139] H.-K. Zhao, T. Chan, B. Merriman, and S. Osher, "A Variational Level Set Approach to Multiphase Motion," *J. Computational Physics,* vol. 127, no. 1, pp. 179-195, Aug. 1996.

[140] H.K. Zhao, "Fast Sweeping Method for Eikonal Equations," *Math. of Computation,* vol. 74, pp. 603-627, 2004.

[141] K.J. Zuiderveld, A.H.J. Koning, and M.A. Viergever, "Acceleration of Ray-Casting Using 3D Distance Transforms," *Proc. Conf. Visualization in Biomedical Computing '92,* pp. 324-335, 1992.

[142] K.J. Zuiderveld, A.H.J. Koning, and M.A. Viergever, "Techniques for Speeding up High-Quality Perspective Maximum Intensity Projection," *Pattern Recognition Letters,* vol. 15, pp. 507-517, 1994.

[143] Y. Zhou and A.W. Toga, "Efficient Skeletonization of Volumetric Objects," *IEEE Trans. Visualization and Computer Graphics,* vol. 5, no. 3, pp. 196-209, July/Sept. 1999.

Mark W. Jones received the BSc and PhD degrees from the University of Wales Swansea. He is now a lecturer in the Department of Computer Science at Swansea, where his research interests include 3D object modeling, voxelization, and visualization.

J. Andreas Bærentzen received the MSc and PhD degrees from the Technical University of Denmark (DTU). He is now an assistant professor of informatics and mathematical modeling at DTU. His research is focused on the development of techniques for geometry representation and effective techniques for interactive shape manipulation and visualization.

Milos Sramek received the Ing (MSc) degree in 1982 from the Slovak Technical University, Bratislava, Slovakia, and the PhD degree in 1996 from the Vienna University of Technology, Austria. From 1997 to 1999, he worked as a postdoctorate student at the State University of New York at Stony Brook. Since 1999, he has been with the Commission for Scientific Visualization of Austrian Academy of Sciences, Vienna, Austria. His research interests include image processing, 2D and 3D data segmentation, voxelization, and volume rendering algorithms.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.