Feature Preserving Distance Fields

Huamin Qu Nan Zhang Ran Shao Arie Kaufman Klaus Mueller

Center for Visual Computing (CVC) and Department of Computer Science
Stony Brook University *

ABSTRACT

We present two distance field representations which can preserve sharp features in original geometric models: the offset distance field (ODF) and the unified distance field (UDF). The ODF is sampled on a special curvilinear grid named an offset grid. The sample points of the ODF are not on a regular grid and they can float in the cells of a regular base grid. The ODF can naturally adapt to curvature variations in the original mesh and can preserve sharp features. We describe an energy minimization approach to convert geometric models to ODFs. The UDF integrates multiple distance field representations into one data structure. By adaptively using different representations for different parts of a shape, the UDF can provide high fidelity surface representation with compact storage and fast rendering speed.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations; I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types;

Keywords: Distance fields, Sampling, Irregular grids, Feature Preserving, Volume Sculpting

1 INTRODUCTION

Distance fields are an important volume representation for surfaces. Usually, the distance field is a scalar field sampled on a regular 3D grid. Each voxel specifies the minimum distance to a surface. If the surface represents a closed shape, this distance is usually signed to distinguish between the interior and exterior of the shape. Distance fields have important applications in constructive solid modeling [12], morphing [2], etc.

However, as noticed before [6, 7, 8], surfaces reconstructed from distance fields often lack sharp edges or corners indicated in the original models. Recently, there has been some notable research to tackle this problem [3, 6, 7, 8]. These methods usually achieve high quality surface reconstruction by adaptive sampling and by using extra information stored with the distances. For simplicity, all these methods sample the distance field on a regular or adaptive rectangular grid.

Converting a geometric model into a distance field is a sampling problem. The sampling pattern can be regular or irregular. Compared with regular sampling, irregular sampling provides more flexibility. It is possible to accurately align feature points with sample points in an irregular sampling grid. In this paper, we investigate sampling distance fields on a special irregular sampling pattern called an offset grid. The sample points in the offset grid are no longer on a regular grid and they can have a small offset to their corresponding regular grid points. Thus, the offset grid is actually a relatively regular curvilinear grid because each sample is only allowed to float in its correspondent regular grid cell. The offset grid strikes the middle ground between a regular pattern and a totally irregular pattern, and thus, it is a kind of semiregular pattern. We name our representation the "offset distance field" (ODF).

The ODF has the following advantages:
First, the offset grid can naturally adapt to curvature variations in the original mesh. Because the sample points of the ODF can float in the cells of a regular grid, we can align the sample points with the sharp corners and edges in the data. Thus, a high fidelity surface with sharp features can be reconstructed from the ODF.
Second, the semiregular structure of the ODF keeps most of the simplicity of a regular grid. We can still use a 3D array to store the ODF and Marching Cubes can still be applied for surface reconstruction (see Section 3.3). The CSG operations, such as volume sculpting on the ODF, are still intuitive and easy to implement (see Section 7.1).
Third, compared with other methods [6, 7, 8], the ODF shifts some expensive computations from the rendering stage to the construction stage. After carefully encoding sharp features into the ODF at preprocessing, our method does not need to explicitly identify and process these features at the rendering and manipulation stages. For many applications, such as volume sculpting, fast reconstruction and manipulation of a distance field is more desirable than fast construction.

With the ODF, we now have several distance field representations [6, 7, 8] which make different tradeoffs between quality of the surface reconstructed and storage size of the distance field. When converting any model into a distance field representation, different parts of the model may have different levels of complexity. Some part is smooth while another part may have feature points and edges. Thus, it may be desirable to develop a mixed distance field representation which uses different distance representations for different parts of the model. Motivated by this, we propose the unified distance field (UDF) which can seamlessly integrate various distance field representations into one data structure. For each sample point in the UDF, in addition to the signed minimum distance, we also optionally store one of the following three sets of information: (a) the directed distances in x, y, and z direction; (b) the position of a feature point; (c) the position of this sample point.

The UDF has the following advantages:
Adaptiveness: the UDF can select the best representations for different parts of a shape. We only store extra information when it is really necessary. For example, if accurate intersection points on edges that show sign changes can be reconstructed from the minimum distance, then we do not need to store these intersection points explicitly. If a surface can be reconstructed accurately from the intersection points in a cell, then we do not need to store an extra feature point in this cell. Similarly, if the mesh can be reconstructed accurately from the intersection points and an extra feature point, then we do not need to use the ODF. Therefore, the UDF is an accurate and compact representation.

*Department of Computer Science, Stony Brook University, Stony Brook, NY 11794-4400, USA. Email: {huamin|nanzhang|rshao|ari|mueller}@cs.sunysb.edu

IEEE Symposium on Volume Visualization and Graphics 2004
October 11-12, Austin, Texas, USA
0-7803-8781-3/04/$20.00 ©2004 IEEE

Accuracy: Unlike other methods [7, 8], we explicitly store the position of some feature points along with the distance values when we convert a geometric model to a distance field. The positions of these feature points are thus more accurate than reconstructed positions in other methods because they are derived directly from the original geometry.
Efficiency: During rendering and manipulation stage, we do not need to reconstruct feature points which are already stored in the distance field representation. Thus, rendering and manipulation speed of the UDF is faster.

The primary contributions of this paper are:

- We introduce the ODF as an extension to the conventional distance field sampled on a regular grid. We demonstrate that the ODF provides more accuracy and flexibility, keeps most of the simplicity of a regular grid distance field, and can preserve sharp features in the original model.
- We describe an energy minimization approach to convert a triangle mesh to an ODF. The energy function we use consists of three terms: a distance energy which measures the fidelity of the conversion, a regularity energy and an orthogonal energy which measure the quality of the ODF grid. Several user specified parameters allow the tradeoff between fidelity of the reconstructed surface and quality of the ODF grid.
- We introduce the UDF to integrate multiple distance field representations into one data structure. We demonstrate that the UDF can provide high fidelity surface representation with compact storage size and can better preserve features and can be rendered and manipulated more efficiently.

This paper is organized as following. After a brief introduction to the related work in Section 2, we introduce the ODF in Section 3. Then, we present the energy function in Section 4 and our optimization method in Section 5. The UDF and its related data structure, construction, and surface reconstruction methods are presented in Section 6. We explore the applications of the ODF and UDF in volume sculpting in Section 7. Finally, we present our experimental results in Section 8 and conclude in Section 9.

2 Related Work

Distance fields have been studied extensively. Various methods exist to convert geometric models into distance volumes [1, 4, 17]. However, as mentioned before, the reconstructed surface from these distance fields lacks the sharp features indicated in the original models.

One solution is to use supersampling or adaptive sampling to preserve fine details to any user required precision. Frisken et al. [3] presented adaptive distance fields which can dramatically reduce volume storage size while preserve details compared with the supersampling method.

Kobbelt et al. [8] proposed an enhanced distance field which stores directed distances in x, y, and z direction with each voxel. They used an Extended Marching Cubes algorithm to detect those grid cells containing sharp features and then additional sample points lying on the feature are computed and inserted into the mesh. By this way, they can reconstruct triangle meshes at a much improved quality compared with conventional methods.

Huang et al. [6] introduced the complete distance field which stores the original triangle mesh with the distance volume. However, their method needs auxiliary data structures and the computational and storage costs of their method are substantial. Their method is good for high end graphics where accuracy is very important. Otherwise, storing both distance fields representation and original triangle meshes is an overkill for most applications.

Ju et al. [7] further stored normals along with the exact intersection points for edges which show sign changes in a distance field. They use these normals to define a quadratic error function for each cell. They then generate a vertex positioned at the minimizer of the quadratic function. By this way, they avoid the needs to explicitly identify and process "features". By using dual methods, the generated polygonal meshes also have higher quality. One disadvantage to this method is that they need normals which dramatically increase the total storage space.

3 Offset Distance Field

3.1 Definition

We introduce a structured irregular grid, called an *offset grid*. The offset grid is based on our previous work on O-buffer (or offset buffer) [13, 14]. The offset grid is generated by connecting samples stored in a uniform O-buffer. From another point of view, the offset grid is formed by placing a sample point in each cell of a regular grid. These sample points are then connected to form a structured curvilinear grid. Therefore, the offset grid is a more regular curvilinear grid because each sample cannot leave its corresponding cell in the regular grid. The *Offset Distance Field* is defined as any distance field sampled on an offset grid. Typically, the distance field is defined everywhere as an Euclidean distance field. In order to simplify the presentation, we use the following terms: *grid points* as the points on a regular grid; *sample points* as the points on an offset grid; *feature edges* as the edges whose adjacent faces enclose a dihedral angle less than a threshold; *feature points* as the joint points of more than two feature edges.

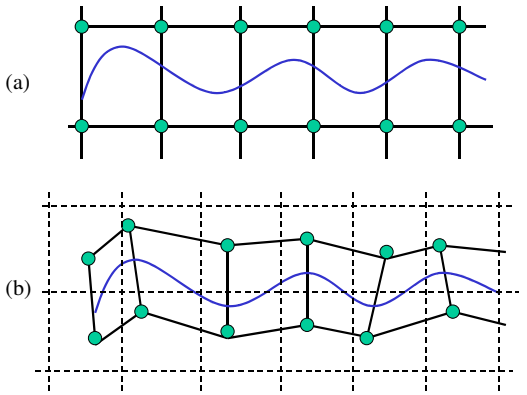
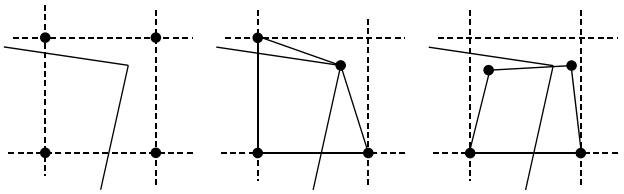


Figure 1: (a) A distance field sampled on a regular grid. (b) A distance field sampled on an offset grid. The dotted lines show the underlying regular grid.

Figure 1a shows a distance field sampled on a regular grid. Figure 1b shows an ODF where the distance field is sampled on an offset grid. From the figure we can see that the grid of an ODF can naturally adapt to curvature variations in the original mesh, and thus preserve the fine details in the data.

The ODF can also be used to capture the sharp features in the original data. Figure 2 shows two scenarios of using the ODF to preserve sharp features. For any feature point, we can choose either aligning a sample point in the ODF with this feature point (see Figure 2b) or align an edge of the ODF with it (see Figure 2c) so the feature point can be reconstructed easily at the rendering stage by the Marching Cubes algorithm.

We make a few comments on the motivation behind the ODF. The irregular volumetric grids can be divided into structured grids (i.e., curvilinear grids) and unstructured grids (i.e., tetrahedral



(a) (b) (c)

Figure 2: (a) A sharp feature point of a geometric model; (b) Aligning a sample point of an ODF with the feature point; (c) Aligning an edge of an ODF with the feature point.

meshes). The tetrahedral meshes provide better flexibility and adaptiveness. However, sampling a distance field on a tetrahedral mesh has some disadvantages: First, for unstructured grids, the connectivity information has to be stored explicitly. This will greatly increase the storage requirement [11]. Second, for CSG operations, quickly locating the corresponding cell for a given sample point is critically important. However, this task is not trivial for tetrahedral meshes. The offset grid is a structured grid. Thus, we do not need to store the connectivity information. As a special constrained curvilinear grid, locating a cell in the ODF for a sample point is relatively straightforward and easy. Thus, the offset grid achieves a fine balance between a regular grid and a totally irregular grid.

3.2 Data Structure

We can use the same data structure as a conventional distance volume to store an ODF, which is just a 3D array of samples. For each sample, we store the position of this sample point and the minimum distance to a shape from this sample point. This data structure can be improved in two ways: First, for most voxels which are not close to the shape, their offset is simply zero. Thus, we can use a flag to identify if this is a regular voxel or an offset voxel for quick processing. Second, if storage space is a big concern, the position of a sample point can be recorded as an offset to its nearest regular grid point and this offset can be quantized for compact representation.

3.3 Rendering of ODFs

Distance fields can be either directly rendered by ray casting or can be first converted into another representation such as triangle meshes [9] or points [6], which can then be rendered by projection or splatting, respectively. Ray casting can generate high quality images but high rendering speed is often unattainable in practice. A triangle mesh is the most popular primitive and considerable research has been conducted to extract triangle meshes from volume data [4, 9, 15, 19]. There are two major approaches: cube-based methods [4, 9] and deformable methods [15, 19].

Marching Cubes is the most famous cube-based method [9]. The original Marching Cubes algorithm has been further extended to solve the ambiguity cases [10] and to extract surfaces from multiresolution volume representations [7, 16, 18]. The SurfaceNets method [4] can be considered as a dual method of Marching Cubes. Instead of generating vertices on cell edges, the SurfaceNets algorithm generates one representative vertex inside each cell. Then, these representative vertices are connected to form a surface according to the sign changes on cell edges.

In this paper, we use the cube-based algorithms as the surface reconstruction method for ODFs. Even though the cell of the ODF is no longer a cube (it is actually a hexahedral cell), the Marching Cubes (or its dual method, the SurfaceNets) algorithm can still be directly used to render ODFs as long as these cells are not in bad shape (see Section 5.2). The algorithm structure is identical to the original Marching Cubes method. Every cell of the ODF is processed separately and a surface patch is generated for cells with sign changes. For each edge which shows a sign change, we use linear interpolation to compute an intersection point. After that, the intersection points in a cell are connected into triangles using the lookup table of Marching Cubes.

4 Definition of the Energy Function

We need to consider the following requirements when we develop algorithms to convert geometric models into ODFs:
First, the fidelity of the conversion. Given a triangular mesh and its corresponding ODF, we can measure the error between the original mesh and the reconstructed mesh from its ODF. The smaller the error, the better the conversion. Also, the sharp features in the data should be faithfully reconstructed from the ODF.
Second, the quality of the offset grid. The quality of the offset grid will affect the aspect ratios of the triangles generated. We want to avoid slivery/thin triangles. The more regular the grid is, the better aspect ratios the final triangles will have. As a volume representation, a bad quality grid will affect the precision of some parts of the volume. This may degrade the performance of CSG operations.

In order to satisfy the competing desires of these two criteria, we define an energy function and cast the conversion problem into an optimization problem of minimizing this energy function. Our approach is inspired by the method used by Hoppe et al. [5] for mesh optimization . Let the original triangle mesh be $M_o(V,T)$, where $V = v_1, v_2, ..., v_m, v_i \in R^3$ is a set of vertex positions and T is the triangle list. Its corresponding ODF is *O(G, D)*, where $G = g_1, g_2, ..., g_n, g_i \in R^3$ is a set of sample points and D is the corresponding set of minimum distances. Let the mesh reconstructed from the ODF be M_r. The energy function is:

$$E(M_o, O) = \lambda_d E_{dist}(M_o, M_r) + \lambda_s E_{spring}(O) + \lambda_o E_{ortho}(O) \quad (1)$$

The first term corresponds to the fidelity of the conversion, and the last two terms correspond to the quality of the offset grid. The distance energy E_{dist} equals the sum of squared distances between the original mesh M_o and the mesh reconstructed from the ODF M_r. The distance energy can be computed as:

$$E_{dist}(M_o, M_r) = min(\sum d^2(v_i, M_r), \sum d^2(g_i, M_o)) \quad (2)$$

The grid quality for structured grids consists of regularity and orthogonality. To guarantee the regularity of the grid, a spring with the rest length of l is placed on each edge of the mesh. The potential energy is:

$$E_{spring}(O) = \sum (|g_i - g_k| - l)^2, \quad (3)$$

where g_i and g_k are any two adjacent sample points. The spring energy measures the equi-distribution of sample points and penalizes samples which get too far or too close. The rest length of the spring equals the unit length of the ODF's underlying regular grid.

However, we notice that just preventing any two sample points from getting too close and too far cannot guarantee good results because the cell still can get deformed very much without even changing the spring energy. Thus, we add the third term, the orthogonality energy. We can think that samples are linked by a rigid structure and rods which are free to extend and those adjacent ones are connected by torsion springs. The orthogonality energy can be computed as:

$$E_{ortho}(O) = \sum (|e_i . e_j|)^2 \quad (4)$$

where vectors e_i and e_j are any two adjacent edges which should be perpendicular to each other if the grid is regular. The orthogonality energy penalizes cells which get deformed.

The user-specified parameters λ_d, λ_s, and λ_o provide a controllable trade-off between fidelity of conversion and quality of the offset grid. For example, a larger λ_d indicates that a high fidelity conversion is preferred over a high quality grid. Large λ_s and λ_o can avoid concave and other badly shaped cells in the ODF.

5 Minimization of the Energy Function

We use a regular grid distance field as a starting point for an optimization process. During the optimization, we vary the position of sample points and try to reduce the total energy. In principal, we can iterate until some formal convergence criterion is met. In practice, we often perform a fixed number of iterations. Like many optimization problems in computer graphics, there is no guarantee of finding a global minimum. Our goal is to find heuristic methods which can work in practice with a wide variety of data sets.

5.1 General Solution

We describe some general methods to minimize the energy function. We first compute the energy for each cell of the initial regular grid. For those cells whose energies are beyond a threshold, the vertices of these cells are tagged as candidates to move.

Random Move and Simulated Annealing: For any candidate sample point, we randomly move it to a new point and compute the energy function. If the energy is reduced, accept it. If not, we try again. If a large number of trials fails to reduce the energy function, we terminate the iteration. This is a brute-force method. However, it is found that even this simple strategy of random descent can generate good results [5]. Simulated annealing refines the random descent approach by accepting some moves which increase the energy. Simulated annealing can overcome local minimum and achieve global optimization. However, it is very slow and we have not seen much of its application in computer graphics.

Gradient Search: For each point, we pick some discretized directions and compute the gradients of the energy function along these directions. Then, we move this point along the steepest descent gradient direction. This is a kind of greedy algorithm and is widely used in practice. In our case, we consider 6 discretized directions (positive and negative x, y, and z directions) for any sample point. For each direction, we can move the point along this direction at a small step. Then, we compute the energy function again for the new position. The difference gives the gradient. We then move this sample along the direction that can reduce the energy function most. Good results can often be achieved by decreasing the step size of the move after each round of iterations. In this paper, we use the gradient search method to minimize the energy function.

5.2 Legal Move

Some moves of sample points may cause badly shaped cells. Two kinds of shapes will cause problems and make Marching Cubes inappropriate for surface reconstruction: self-intersection cell (see Figure 3a) and concave cell (see Figure 3b). We define the legal move of a sample point as a move which does not cause any self-intersection and concave cells in the ODF grid.

The structure of the offset grid can avoid any self-intersection cell if a sample point is only allowed to float in its corresponding cell of the underlying regular grid. Thus, we will focus on how to avoid concave cells. Figure 3c illustrates our method. For a sample point, we will not allow any moves which cause the penetration of any sample a of the ODF through the plane passing through three adjacent samples b, c, and d in the same cell.

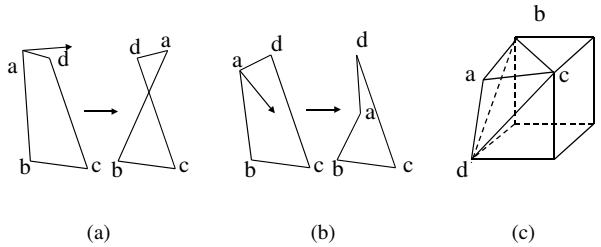


Figure 3: (a) A self-intersection cell; (b) A concave cell; (c) A legal cell.

5.3 Acceleration

In this section, we describe some heuristic acceleration methods which can dramatically reduce the running time. First, we introduce an approximate energy function computation method. After that, we present Laplacian smoothing for inside voxels.

5.3.1 Energy Function Computation

During the iteration, we need to know in advance how the energy function will be affected by moving the position of a sample point. There are two different ways to compute the energy function: exact evaluation and approximate evaluation. For exact evaluation, after each move, we update the energy function for each cell. For approximate evaluation, we use some faster heuristic method to estimate the energy and use the estimated energy to update the total energy function after an accepted move. By this way, we can avoid computing the exact energy function after each move.

Among the three terms of the energy function, the computation of the spring and orthogonality energies are easy and straightforward. To evaluate the distance energy, it is necessary to compute the sum of squared distances between the original mesh and the reconstructed mesh. This is an expensive operation. We develop a fast approximation method for the distance energy.

It has been demonstrated that if a surface is smooth, a distance field can accurately represent this surface [6]. Thus, the error between the reconstructed surface and the original surface is mainly caused by feature points and edges which have high curvature. For each cell with feature points falling in or feature edges passing through, we compute the minimum distances from each feature point and feature edge in this cell to the vertices of the triangle mesh reconstructed in this cell from the distance field using Marching Cubes. These vertices are on the cell edges which show sign changes. The smaller the distance, the more likely the surface is separated into smooth patches in each cell, and the smaller the real distance energy. Thus, we approximate the distance energy by the sum of squared minimum distances from all feature points or feature edges to the vertices of the reconstructed mesh.

5.3.2 Laplacian Smoothing for Inside Voxels

For the inside voxels whose adjacent cells contain no segment of the shape, the energy function only contains regularity and orthogonality energy. Instead of using the gradient search, we can simply use Laplacian smoothing to improve the grid quality. For each sample point in an ODF, there are up to eight adjacent cells. Thus, the new position of the sample point can be simply computed as the average of the centers of these adjacent cells. This can greatly improve the grid quality.

6 UNIFIED DISTANCE FIELD

Besides the ODF, there are four other different distance field representations: the conventional distance field which only stores minimum distances; the enhanced distance field [8] which stores directed distances in x, y, and z direction; the Hermite distance field [7] which stores exact intersection points and their normals for edges with sign changes; the complete distance field [6] which stores minimum distances and the original mesh. All these methods have their advantages and disadvantages. They make different tradeoff between quality of the reconstructed mesh and space requirement of the distance field representations. Some method may be suitable for some situation. No one method can win in all situations.

Figure 4 shows some scenarios. Figure 4a demonstrates that if there are no feature points or edges in a cell, the minimum distances and the standard Marching Cubes method sometimes provide better fidelity than a more sophisticated method such as the Dual Contouring method. In this case, it is hard to find a good position for a sample point inside the cell which is used by the Dual Contouring method. Figure 4b shows that if there are sharp corners or edges in a cell, the Extended Marching Cubes and the Dual Contouring method provide better reconstruction. However, the reconstructed feature point is only an estimation of the real feature point of the original mesh. Figure 4c shows that there are two feature points in a cell of a regular grid. The ODF can align these feature points with different edges of cells by adjusting the positions of the sample points of the ODF.

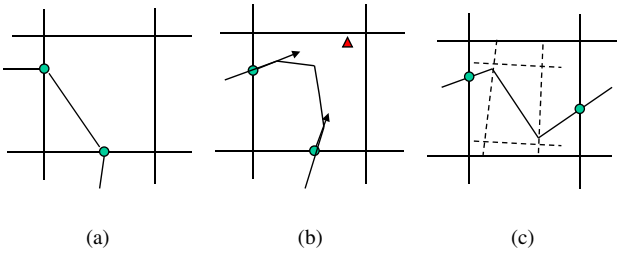


Figure 4: (a) No feature point; (b) One feature point; (c) Two feature points. The dotted lines show the ODF grid.

All these scenarios may appear simultaneously in one model. Thus, it may be desirable to develop a mixed representation which uses different distance representations for different parts of the model. We propose the unified distance field (UDF) representation which integrates multiple distance field representations into one data structure.

6.1 Data Structure

For each sample point, we store one of the following four sets of information. Figure 5 illustrates these four sets in 2D.

- D_m: the minimum distance to the surface.
- $(D_m,(D_x,D_y,D_z))$: the minimum distance and the directed distance in positive x, y, and z directions. The exact intersection points in the edges can be computed from these distances.
- $(D_m,(V_x,V_y,V_z))$: the minimum distance and the position of one feature point inside this cell.
- $(D_m,(O_x,O_y,O_z))$, the minimum distance and the position of this sample point. This sample point is only allowed to float in its correspondent cell of a regular grid. In other words, it is a sample point in the ODF.

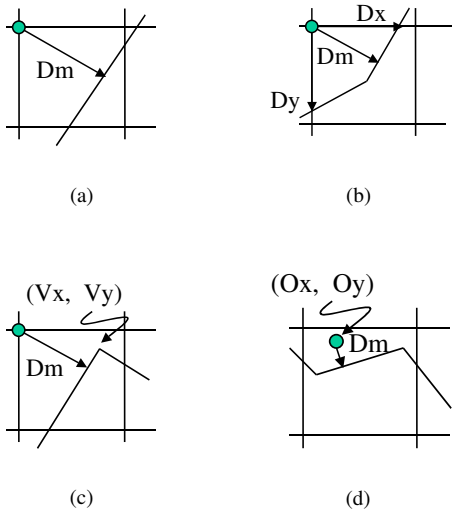


Figure 5: Four sets of information: (a) D_m, which is the minimum distance; (b) D_m and (D_x,D_y), which are the distances in positive x and y direction; (c) D_m and (V_x,V_y), which is the position of a feature point; (d) D_m and (O_x,O_y), which is the position of this sample point.

There are two features of this representation: First, we may explicitly store a feature point in a cell. This is inspired by the complete distance field (CDF) [6]. Unlike the CDF which stores whole triangle meshes, we only store one feature point. This makes our representation more compact and we believe that this is good enough for most applications. Compared with the Extended Marching Cubes method [8] and the Dual Contouring method [8], our method provides more accuracy for the position of the feature points by getting them directly from the original data and provides more efficiency for rendering and manipulation of the distance field by avoiding the expensive reconstruction of these feature points. Second, we always store the minimum distance. The minimum distance plays a key role to integrate different representations. We will demonstrate this in Section 6.3.

The implementation of this representation is straightforward. For each sample point, we store a minimum distance and a pointer to additional information. The additional information includes three floating point numbers and a 2-bit flag. Based on this flag, these three numbers can be interpreted as either the directed distances, the position of a feature point, or the position of this sample point. If only the minimum distance is stored, the pointer to additional information is simply null.

6.2 Construction of UDFs

When we convert geometry models into UDFs, we need to decide which one of the four sets of information to store in each sample point. The principle is that we only store extra information when necessary. We start from the minimum distance and then gradually store directed distances, feature points, and the positions of samples with the increase of the complexity of the shape. Even though directed distances, feature points, and the positions of samples require equal amount of memory, their underlying grid quality, construction and reconstruction algorithms, and CSG operations are different. The criteria to choose which information to store are

the fidelity of the representation, the regularity of the final grid, and the simplicity of the construction and reconstruction algorithms.

The steps to convert a triangle mesh model to a UDF are:
First, we generate a regular grid distance field representation with the minimum distance to the surface computed and stored in each grid point. For each edge which shows a sign change, we first try to reconstruct the intersection point by interpolating the minimum distances stored on the two vertices of this edge. Then, we compute the exact intersection point and compare the reconstructed point and exact intersection point. If their difference is less than a threshold, the information of the exact intersection point is abandoned. Otherwise, we store the directed distances computed from the exact intersection point.
Second, we detect the global feature points and the feature edges in the original data and locate their corresponding cells in the distance field. For cells with one feature point, we store the position of this feature point. For a cell with a feature edge passing through, we also store the position of one feature point along the feature edge in this cell. This position is computed using Ju et al.'s method [7] which finds the minimizer of a quadric function.
Third, we check the error of each cell between the original mesh and the reconstructed mesh. For cells with errors beyond a threshold, we can move the position of the samples to reduce the error by using the method described in Section 5. If a legal move of a sample can reduce the total energy, then we store the position of this sample along with the minimum distance.

6.3 Surface Reconstruction from UDFs

The surface reconstruction method for the UDF has the identical framework as the Marching Cubes method or its dual, the SurfaceNets method. We process each cell one by one. For each cell which shows a sign change, we reconstruct surfaces in this cell and then connect them with surfaces reconstructed from adjacent cells.

The surface reconstruction method is:
First, for each edge which shows a sign change, check if the exact intersection point is already stored as the directed distances. If yes, we retrieve the intersection point. If not, we can always reconstruct the intersection point by linear interpolation of the minimum distances stored in two vertices of the edge. In order to stitch multiple representations together, the minimum distance is required for each sample point. If one or two of the vertices of this edges are not on the regular grid, compute the intersection point based on the minimum distance and the position of these two samples.
Second, for each cell where no feature point is stored, triangulate this cell using the Marching Cubes method.
Third, for cells with a feature point stored, we can use the Extended Marching Cubes method [8]. We first generate a triangle fan using this feature point as the center. Then, we check the neighboring cells, if there are other feature points, connect them to form a feature edge by an edge flip process.

7 Volume Sculpting with ODF and UDF

Boolean operations on the distance field provide a natural, straightforward method for sculpting[12]. In this section, we use volume sculpting to demonstrate the flexibility and accuracy of the ODF and the UDF.

7.1 Volume Sculpting with ODF

Volume sculpting starts from reconstructing and resampling the tool's distance field on each sample point of the object's distance field. Then, Boolean operations are applied for the object's distance value and tool's distance value on each sample point. Finally, surfaces are reconstructed from the resulting distance field.

Suppose the object to be sculpted is represented by an ODF. Usually, tools are less complicated than the objects. Thus, tools can be represented as parametric functions, triangle meshes, or distance fields sampled on regular grids. For each voxel in an ODF, computing its distance to a parametric surface or triangle mesh, or resampling its distance in a regular grid distance field by trilinear interpolation is straightforward (see Figure 6). Actually, this computation and the following Boolean operations are the same for an ODF and a regular grid distance field. The resulting ODF can then be rendered by Marching Cubes. Thus, volume sculpting using ODFs is still intuitive and easy to implement just like using a regular grid distance field.

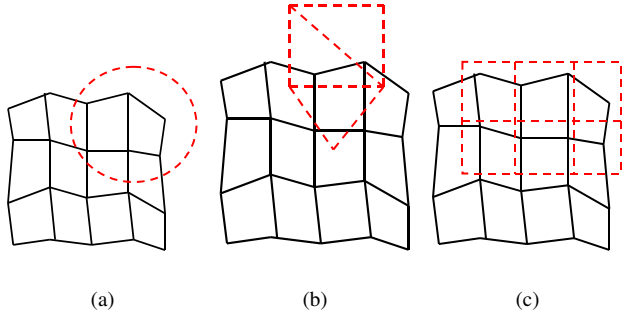


Figure 6: An ODF is sculpted by: (a) A parametric surface; (b) A mesh; (c) A regular grid distance field. The dotted lines represent the sculpting tools (e.g., a sphere, a triangle mesh, and a regular grid distance field).

Some features may be created after the CSG operations even when there is no such feature in both objects and tools. Then, we need to use either Kobbelt et al.'s method [8] to compute the intersection of the two or three planes or use Ju et al.'s method [7] to compute the point which minimizes a quadric error function . After the feature point is computed, we can locally deform the cell so this feature point can be either aligned with the sample point or the edge of the cell.

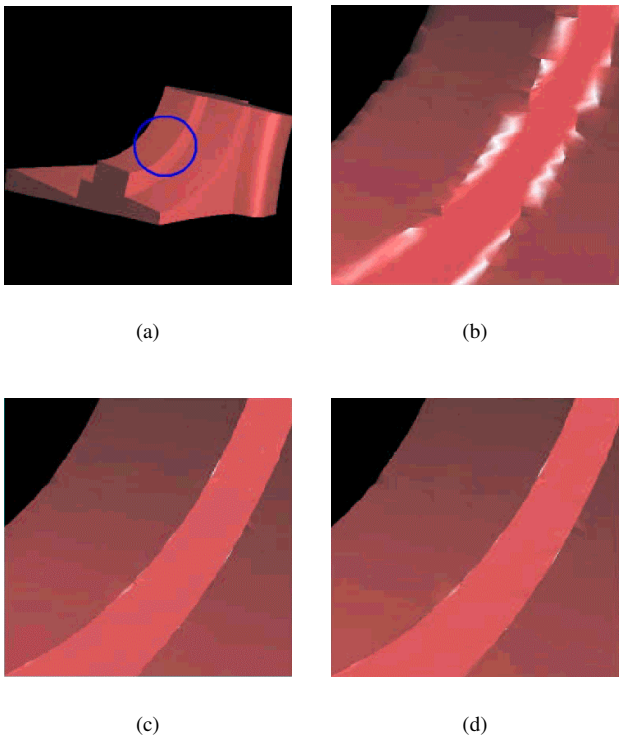
7.2 Volume Sculpting with UDF

Volume sculpting with UDFs has the same framework as volume sculpting with conventional distance fields or the ODFs. After the Boolean operation, one of the two distance values (i.e., distances to the object and the tool) is used as the new distance value for any sample point in the UDF. For each cell, if the final distance values of its eight vertices are from different sources, then a new surface is generated in this cell. We need to detect if there is any sharp feature point or edge created in the new surface. This can be done by using Kobbelt et al.'s method [8]. If yes, we store its position in this cell.

8 Experimental Results

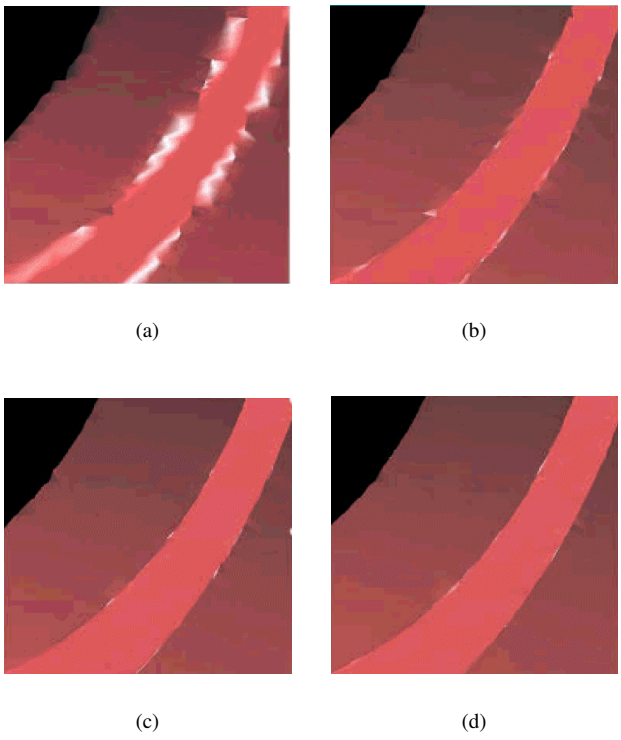
All our experimental results have been generated on a Dell Dimension 8200 desktop with a 2.53GHz Pentium 4 CPU, 1.0GB of RAM, and an Nvidia GeForce4 graphics board with 64MB memory. Our implementation is not highly optimized for speed.

First, we convert the FanDisk model into an ODF and a UDF and demonstrate that the ODF and UDF can reserve sharp features in the original model. The grid resolution for the distance fields in this example is $65 \times 65 \times 65$. All distances and errors are in grid unit. All rendering times are in second. Figure 7a shows the original



(a) (b) (c) (d)

Figure 7: Fandisk surfaces: (a) Original surfaces; (b) Surfaces reconstructed from a regular grid distance field; (c) Surfaces reconstructed from an ODF; (d) Surfaces reconstructed from a UDF.



(a) (b) (c) (d)

Figure 8: FanDisk surfaces reconstructed from: (a) A regular grid distance field; (b) The ODF after 5 iterations; (c) The ODF after 10 iterations; (d) The ODF after 15 iterations.

surface. Figure 7b shows the surface reconstructed from a regular grid distance field. The aliasing around feature edges and corners is obvious. Figure 7c shows the surface reconstructed from its ODF. The ODF is generated from the FanDisk mesh using the method presented in Section 5. Figure 8 shows surfaces reconstructed from the ODF at different stages of iterations. We can see that the surface fidelity is improved substantially and the sharp features can be faithfully reconstructed from the ODF. Figure 7d shows the surface reconstructed from the UDF. Directed distances are stored in each cell if the distance between reconstructed intersection points and the exact intersection points are greater than 0.1 grid units. We compute the Hausdorff distance and the average distance between these reconstructed surfaces and the original mesh. Table 1 shows the construction time, reconstruction time, and the errors. From the table, we can see that the UDF has the best overall fidelity performance.

Table 1: Rendering times and the errors between reconstructed surfaces and the original surface.

Surfaces:	Regular DF	ODF	UDF
Construction Time	11.8s	57.8s	12.5s
Reconstruction Time	1.319s	1.438s	1.533s
Maximum Distance Error	0.97	0.14	0.12
Average Distance Error	0.006	0.0022	0.0018

Table 2 shows the errors of the UDF using different thresholds to store the directed distances. Among all 262, 144 cells, there are 15, 406 boundary cells which surfaces pass through, and 1,354 feature cells which contain a feature edge or a feature corner. From the table we can see that if the error threshold is 0.1 grid units, there are 8,933 voxels which need to store directed distances. However, if the error threshold is 0.2 grid units, only 151 voxels need to store directed distances. Compared with Kobbelt et al.'s enhanced distance field [8] which always stores directed distances, our representation can save substantial memory and storage space with negligible approximation error.

Table 2: The errors of the UDF under different error thresholds.

Error Threshold:	0.1	0.2	0.3
Voxels with Features	1354	1354	1354
Voxels with Directed Distances	8933	151	80
Average Distance Error	0.0018	0.0022	0.00219
Maximum Distance Error	0.1224	0.1228	0.1228

Figure 9 shows volume sculpting with an ODF model. Figure 10 shows volume sculpting with a UDF model. The resolutions of these two models are $128 \times 128 \times 128$. These figures demonstrate that the newly created features can be reconstructed from our representations.

9 Conclusions and Future Work

We presented two novel distance field representations which can provide high fidelity surface representations. The ODF is an attempt to sample distance fields on a irregular grid. We presented a conversion method based on reducing an energy function which balances several competing goals of the conversion, such as the

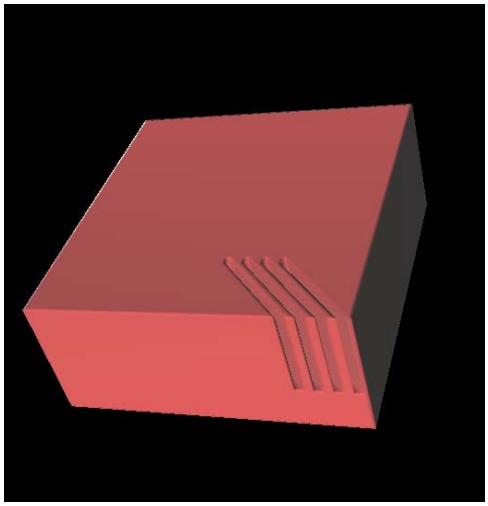


Figure 9: Volume sculpting on an ODF.

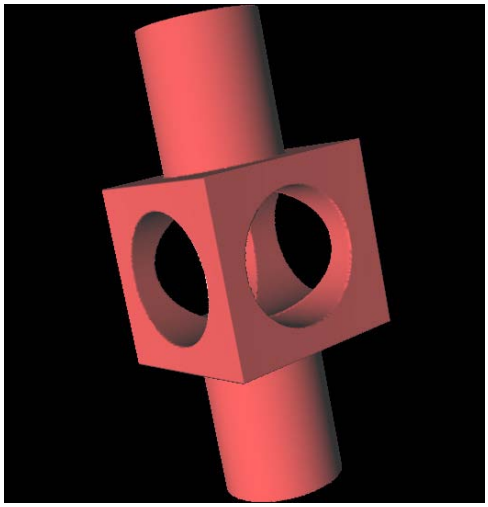


Figure 10: Volume sculpting on a UDF.

overall fidelity, feature preservation, and the quality of the mesh. We described several strategies to solve the optimization problem. We proposed the UDF as a data structure to integrate multiple distance field representations. The UDF provides accurate surface representation with compact storage size and fast rendering speed by adaptively using different representations for different parts of the model and by explicitly storing feature points. The UDF is easy to implement and the ODF can be used to fine tune the grid for more flexible and accurate representation. We demonstrated that the ODF and UDF are useful for volume sculpting.

In the future, we plan to further extend both representations to octree-based adaptive distance fields. We also want to investigate how to perform generalized Boolean operations between two ODFs, two UDFs, or one ODF and one UDF. There are a number of applications that need to employ a generalized Boolean operator (e.g., solid modeling over distance fields, management of range data in 3D scanning). How to integrate and merge ODFs or UDFs and still maintain an accurate representation of feature elements needs further research.

Acknowledgments

This work is partially supported by NSF grant CCR0306438 and ONR grant N000149710402. We would like to thank Susan Frank for proofreading a draft of this paper.

REFERENCES

[1] David E. Breen, Sean Mauch, and Ross T. Whitaker. 3D scan conversion of CSG models into distance volumes. *Proceedings of IEEE/SIGGRAPH Symposium on Volume Visualization and Graphics*, pages 7–14, 1998.

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

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

[4] Sarah Gibson. Using distance maps for accurate surface representation in sampled volumes. *Proceedings of IEEE/SIGGRAPH Symposium on Volume Visualization and Graphics*, pages 23–30, 1998.

[5] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. *Proceedings of SIGGRAPH*, pages 19–26, 1993.

[6] Jian Huang, Yan Li, Roger Crawfis, Shao Chiun Lu, and Shu Liou. A complete distance field representation. *Proceedings of IEEE Visualization*, pages 247–254, 2001.

[7] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. *Proceedings of SIGGRAPH*, pages 339–346, 2002.

[8] Leif P. Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel. Feature-sensitive surface extraction from volume data. *Proceedings of SIGGRAPH*, pages 57–66, 2001.

[9] W. E. Lorensen and H. E. Cline. Marching cubes: a high resolution 3D surface construction algorithm. *Proceedings of SIGGRAPH*, pages 163–170, 1987.

[10] Gregory M. Nielson and Bernd Hamann. The asymptotic decider: Removing the ambiguity in marching cubes. *Proceedings of IEEE Visualization*, pages 83–91, 1991.

[11] Renato B. Pajarola, Jarek Rossignac, and Andrzej Szymczak. Implant sprays: Compression of progressive tetrahedral mesh connectivity. *Proceedings of IEEE Visualization*, pages 299–306, 1999.

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

[13] Huamin Qu and Arie Kaufman. O-buffer: A framework for sample-based graphics. *IEEE Transactions on Visualization and Computer Graphics*, 10(4):410–421, 2004.

[14] Huamin Qu, Arie Kaufman, Ran Shao, and Ankush Kumar. A framework for sample-based rendering with O-buffers. *Proceedings of IEEE Visualization*, pages 441–448, 2003.

[15] I. A. Sadarjoen and F. H. Post. Deformable surface techniques for field visualization. *Computer Graphics Forum*, 16(3):109–116, 1997.

[16] R. Shekhar, E. Fayad, R. Yagel, and F. Cornhill. Octree-based decimation of marching cubes surfaces. *Proceedings of IEEE Visualization*, pages 335–342, 1996.

[17] Milos Sramek and Arie E. Kaufman. Alias-free voxelization of geometric objects. *IEEE Transactions on Visualization and Computer Graphics*, 5 (3):251–267, 1999.

[18] R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer*, 15(2):100–111, 1999.

[19] Zoë J. Wood, Mathieu Desbrun, Peter Schröder, and David Breen. Semi-regular mesh extraction from volumes. *Proceedings of IEEE Visualization*, pages 275–282, 2000.