Using Distance Fields for Object Representation and Rendering

Mark W. Jones and Richard Satherley
Department of Computer Science, University of Wales Swansea
United Kingdom SA2 8PP

March 12, 2001

Abstract

Distance fields are a two- or three-dimensional array of values, where each value is the minimum distance to the encoded object. Usually distance fields are signed, such that negative values signify the point is inside the object, and positive are outside. Distance fields are becoming a popular research area as more applications are discovered, and many of these application areas are discussed in this report. More importantly, algorithms for computing distance fields quickly are also examined. Both existing algorithms, and those developed by ourselves at Swansea are discussed.

1 INTRODUCTION

The work presented in this paper is closely related to the field of Volume Graphics [1]. Volume Graphics is an emerging area of Computer Graphics which is concerned with the input, storage, construction, modelling, analysis, manipulation, display and animation of spatial objects in a true three-dimensional form. When comparing Volume Graphics to traditional surface graphics, the relationship has been likened to the relationship of that between two-dimensional images and vector graphics [2]. The employment of volume graphics techniques offers the benefits of *scalable rendering* algorithms for extremely large scenes, *visual effects* such as fire, fur and roughness (Section 4.3) and *consistency of rendering* (the ability to render objects of different source types under the same conditions).

In this paper we shall introduce distance fields in Section 2 and examine various methods for calculating distance fields in Section 3. We will discuss a naive algorithm using binary segmentation in Section 3.1, and improve this using sub-voxel accuracy (Section 3.2) and octrees (Section 3.3). Approximate methods are introduced in Section 3.4, and improved in Section 3.5. A new approximate method is presented in Section 3.6, and an analysis of all of these methods is presented in Section 3.7. Section 4 presents three of our applications of distance fields, namely contour connection (Section 4.1), the morphological closing operator (Section 4.2) and hypertexture (Section 4.3). Further work and conclusions are given in Section 5.

2 BACKGROUND

A *distance field* data set D representing a surface S is defined as: $D : \mathbb{R}^3 \rightarrow \mathbb{R}$ and for $p \in \mathbb{R}^3$,

$$\begin{aligned} &D(p) = sgn(p) \cdot min\left\{|p-q| : q \in S\right\} \\ &sgn(p) = \begin{cases} -1 & \text{if p inside} \\ +1 & \text{if p outside} \end{cases} \\ &\text{where } || \text{ is the Euclidean norm} \end{aligned} \tag{1}$$

For each voxel in the three-dimensional grid, the distance to the closest point on the surface is stored. Additionally, all voxels inside the object are set to be negative, and all voxels outside are set to positive. This distance field has effectively voxelised the object as the original surface can be displayed by rendering the iso-surface of level 0 from D – i.e. $S = \{q : D(q) = 0\}$ where $q \in \mathbb{R}^3$. Non-integer grid points can be calculated from surrounding integer grid points using trilinear interpolation or any other appropriate scheme of interpolation. Surface normals can be calculated from the grey-level data [3], and due to the choice of the sign function point away from the object centre, aswell as being perpendicular to the surface due to the distance field. Figure 1 demonstrates several different forms of surface representation.

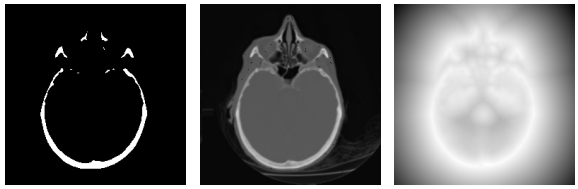

Figure 1: Binary segmented, original and distance field slices from the CThead dataset.

Distance fields have been used for applications within Computer Graphics for morphing [4], virtual endoscopy or skeletal representation [5, 6] and accelerating ray tracing [7] (amongst other applications). In each

case a chamfer distance transform has been used (Section 3.4). This paper demonstrates more accurate methods than the basic CDT. A great body of work describes distance transforms in detail, in particular the interested reader is directed to Cuisenaire and Macq's review [8]. This paper improves the current best vector distance transform (in Section 3.6) and proposes a distance shell (Section 3.5) rather than the widely used binary segmentation (Section 3.1) for computation. It also describes a practical, efficient algorithm for the computation of an exact solution. Frisken et al. [9] have given a heirarchical computation for distance fields. This gives a compact storage, but trades off accuracy. In this paper we are concerned with accuracy. We are also interested in giving practical algorithms for calculation, and demonstrating new effects (for general objects) such as hypertexture (Section 4.3) and true distance morphological algorithms (Section 4.2). The next section progresses through each algorithm, from simplest to most complex, highlighting our contributions on the way.

3 Distance Field Calculation

3.1 Naive Calculation

A relatively straightforward method of calculation is to first apply a segmentation function f as:

$$f(v) = \begin{cases} 1 & \text{if } v \text{ is inside the surface} \\ 0 & \text{otherwise} \end{cases} \quad \text{where } v \in \mathbb{Z}^3 \tag{2}$$

Then for each voxel v, we set the distance field at voxel v to ∞ ($D(v) = \infty$). For each voxel w, we set $D(v) = min(D(v), |v - w|)$ where $f(w) = 1$, and $||$ is the Euclidean distance between two voxels' positions. Each voxel will be taken in turn, and its distance will be calculated to all voxels which comprise the object. The min operator ensures that the minimum distance is stored.

This algorithm produces an accurate distance field in the sense that each integer grid point in the field has an accurate distance to the surface encoded by the segmentation function. Unfortunately as a discretised segmentation function is used, this has the effect that the distance field represents a discretised surface. Another negative aspect of this algorithm is that the complexity of O(n^2) results in long computational times. If we take the CThead dataset for example ($256 \times 256 \times 113$) we must make around 50 trillion calculations. Even rejecting voxels immediately where their x, y or z position is further away than the current minimum distance does not bring the rendering time down to a useable level. Figure 2 shows the discretised surface that arises when sampling a sphere using a binary grid. Even when oversampling is used, the fact that no information is available away from the vicinity of the surface, means that normals have a restricted orientation.

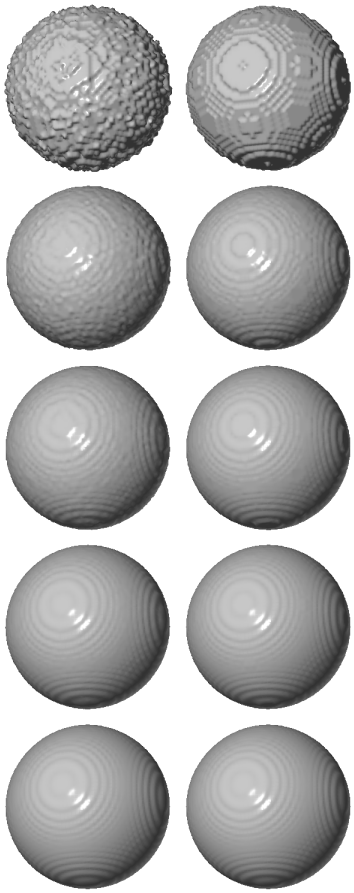

Figure 2: Rendering of spheres using jittered (left column) and regular sampling (right column). Samples per voxel (from top) 1, 8, 27, 125, 9261.

3.2 Sub-Voxel Accuracy

The quality of the distance field representation can be improved by measuring the distances to a sub-voxel accuracy. Instead of employing a binary segmentation function, the distances to the actual surface are calculated. In the case of the chess piece we can calculate the distance to the closest point on each triangle. This effectively *voxelises* the chess piece. Voxelisation is the process of converting an object of one source type into voxel (volume) data. (In fact the binary segmentation function mentioned in the previous section is often used for voxelisation). We have studied the methods for voxelising triangular mesh objects in a previous paper [10], and discovered that it is often quicker to voxelise and employ a volume renderer than to ray trace the original triangular mesh. This is mainly because volume rendering scales so well. The time to render is based upon the size of the volume data rather than the complexity of the object. Another benefit of voxelisation is the fact that once all scene objects are encoded by a common data type, it is possible to apply rendering effects uniformly across all objects. Also rendering effects such as texture mapping or bump mapping which are traditionally carried out in the rendering phase can be carried out

during the voxelisation phase, and therefore view independent images can be rendered consistently from the voxelised objects, thus removing the need to perform the texture and bump mapping for each new view.

The distance to a triangle is a non-trivial problem, and is treated in detail in a separate report [11]. Depending upon the position of the voxel, the closest distance to the triangle may be a vertex, an edge between vertices, or the plane of the triangle. We therefore presented an efficient algorithm for projecting a point onto the triangles plane, performing tests to determine which area the point is in, and then calculating the distance appropriately.

In the case of triangular mesh objects the complexity becomes O(nm), where n is the number of voxels and m is the number of triangular facets. For example Figure 3 shows volume rendered chess pieces, each voxelised to a grid of $60 \times 60 \times 60$ and originally comprising of about 1500 triangles. In that case 324 million calculations are required for each piece taking 287 seconds on a 1GHz Athlon (for the pawn).

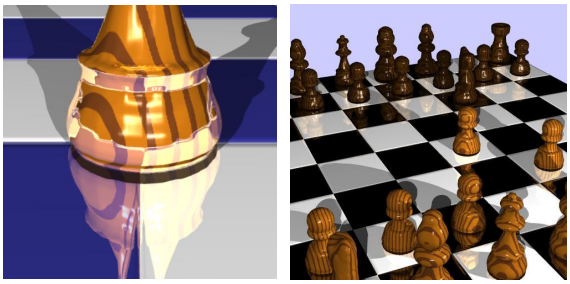

Figure 3: Distance field encoded chess pieces rendered with VLib [12].

One of our other contributions to this area is to convert CT data (such as the UNC CThead data set) into a sub-voxel accurate distance field. A similar algorithm to the naive algorithm can be used, but instead of f encoding a binary segmentation, f encodes those cells which are transverse. A cell is a cuboid made up of 8 neighbouring voxels. It is transverse when at least one voxel is inside the surface, and at least one is outside the surface. This cell will have a piece of the surface passing through it. We triangulate that part of the surface using the tiling tetrahedra algorithm [13] as this removes the ambiguities of marching cubes. The distance is then calculated to the triangles using the same method as for voxelisation. There are alternative methods for constructing a surface from the field, although these are more complex to compute and as we shall see this is already a computationally intense process.

Using this method an accurate Euclidean distance field can be created for the UNC CThead. Using the naive algorithm this would take about 60 days on a 1GHz Athlon. We actually implemented it using coarse grain parallelisation, and executed it on 8 processors (1 1GHz Athlon, 5 800MHz Athlons and 2 866MHz Pentium IIIs) in just over 8 days. We needed this dataset to compare the accuracy with the faster methods reported in the next sections.

3.3 Employing an Octree

We can improve the calculation of distance to O(nlogn) using an octree (in our case a Branch on Need Octree [14] (BONO) as the data size is not a power of 2). We can reject those parts of the octree which are known not to contain any transverse voxels. Primarily it is used to reject portions of the octree (and thus large numbers of voxels) which are further away than our current minimum distance (D(x,y,z)). This can be improved further by using a neighbouring voxel's closet point as a good starting point for the current voxel, thus removing large parts of the octree immediately. These two methods improve performance considerably, particularly in the regions close to the surface. An identical Euclidean distance field to that of the previous section can be computed in 126 minutes on a single 1GHz Athlon.

3.4 Approximation Methods – Chamfer Distance Transforms

The sheer computational expense of the naive implementation has led to the application of *distance tranforms*. These have been implemented as multiple dilations from a binary segmented surface, and more efficiently as a 2 pass process. First we begin with an initial distance field, D (equation 3). Local distance propagation (equation 4) is achieved with a number of passes of a *distance matrix*, d_M. Each pass loops through each voxel in a certain order and direction according to the needs of the matrix.

$$D(p) = \begin{cases} 0 & \text{if } f(p) = 1, \exists\ q \in p_{26}, f(q) = 0 \\ \infty & \text{otherwise} \end{cases}$$
$$\text{where } p \in \mathbb{Z}^3, \text{ and } p_{26} \text{ is the set of voxels which are the 26 neighbours of } p \tag{3}$$

$$D(i,j,k) = min\big(D(x+i, y+j, z+k) + d_M(i,j,k)\big)$$
$$\forall\ i,j,k \in d_M \text{ where } i,j,k \in \mathbb{Z} \tag{4}$$

Chamfer distance transforms propagate local distance by addition of known neighbourhood values obtained from the distance matrix, d_M, (an example of which is in Figure 4). Each value in the matrix represents the local distance value. This matrix is applied in two passes.

```
/* Forward Pass */
FOR(z = 0; z < f_z; z++)
  FOR(y = 0; y < f_y; y++)
    FOR(x = 0; x < f_x; x++)
      D(x,y,z) = Eq.4
```

```
/* 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--)
      D(x,y,z) = Eq.4
```

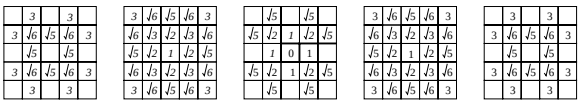

Figure 4: Quasi-Euclidean $5 \times 5 \times 5$ chamfer distance matrix.

The forward pass (using the matrix above and to the left of the bold line, shown in *italic* font) calculates the distances moving away from the surface towards the bottom of the dataset, with the backward pass (using the matrix below and to the right of the bold line) calculating the remaining distances. This method is computationally inexpensive since each voxel is only considered twice, and its calculation depends upon the addition of elements of the matrix to its neighbours. It is also very inaccurate, and hence all of the applications that rely on the method are using inaccurate (although in many cases acceptable) data. We desired distance fields to produce hypertextured objects, and we found that the best distance transforms did not produce accurate enough data for the hypertexture to operate convincingly. This led us to investigate vector propagation methods (Section 3.6).

3.5 Distance Shells

Usually these chamfer (Section 3.4) and vector (Section 3.6) distance transforms are carried out upon binary segmented data. Alternatively a *distance shell* can be calculated, and use those values as a basis upon which we can propogate the surface throughout the volume. This indeed increases the accuracy of these type of distance transforms (both those of the previous and next sections). To completely encode the surface of an object, we need to calculate a shell of voxels which are either side of the surface interface. All those voxels which are inside the object and have a neighbour outside the object should be part of the computed shell, as should all those voxels which are outside the object and have a neighbour inside the object. In fact a quick voxelisation algorithm is to *segment* those voxels, and calculate the distance for them. The surface can be reconstructed accurately from this shell, but the normals calculated using the central difference operation sample voxel values outside of this shell, and therefore as they are of a uniform background value, it can lead to quantization effects (the normals have a restricited number of unique directions). We can solve this by also including all those voxels that are used within the grey-level normal calculation function.

The whole procedure is to use the segmentation function f (equation 2). Then for each voxel, v, we add v and v_{26} (the 26 neighbours of v) to S_v, when $f(v) = 1$ and $\exists p$ such that $f(p) = 0$ where $p \in v_{26}$. To include all of the additional voxels required for central distance calculation of the normals, we create the shell S_n where for each $v \in S_v$ we add v and v_{26} to S_n. S_n now contains all voxels which are used to display the surface (*including values* used just in normal calculation). We have called the voxels S_n the *distance shell* of the encoded object. The distance shell adequately represents the object, and is a valid method for voxelising objects where only the surface needs to be encoded. As a shell voxelisation it benefits from the advantage of requiring less memory to store (if run length encoding is employed). The chess piece takes 11.4 seconds to encode using this method, and the UNC CThead takes 124 seconds to convert into this form. Such datasets can then be used as the basis in the distance transform process (and therefore give a much more accurate result).

3.6 Vector Distance Transforms

Vector methods [15] store a vector to the closest surface point at each voxel. These vectors are propagated to neighbouring voxels in a prescribed way similar to the propagation of distances via the distance matrix. VDTs generally require more passes of the distance matrix. During each pass the vector components are added to the necessary vector position, the distance calculated, a decision made as to whether any of the new distances are minimal, and finally the minimal vector is stored.

To allow vector propagation Equations 3 and 4 are modified as shown in Equations 5 and 6 respectively.

$$\vec{V}(p) = \begin{cases} (0,0,0) & \text{if } p \in S \text{ and} \\ & \exists\ q \in p_{26}, q \notin S \\ (\infty,\infty,\infty) & \text{otherwise} \end{cases} \quad (5)$$

$$D(p) = min\left|\vec{V}(x+i,y+j,z+k) + d_M(i,j,k)\right| \quad \forall i,j,k \in d_M, \text{ where } d_M = (\vec{M}_x, \vec{M}_y, \vec{M}_z) \quad (6)$$

The previous best algorithm is Mullikin's EVDT [16] operating on binary segmented data.

Figure 5 shows the matrix passes employed by our Vector City VDT.

3.7 Comparison

We have compared each of the methods in the previous sections to the correct distance field in terms of the execution time, the average voxel error, and the most negative and positive errors. Table 1 presents these results in detail, and Figure 6 present images for each of these methods. If perfect Euclidean distance fields are required, we have shown that it is possible to compute them in a reasonable time, by using the octree based algorithm running in parallel. It would be possible to

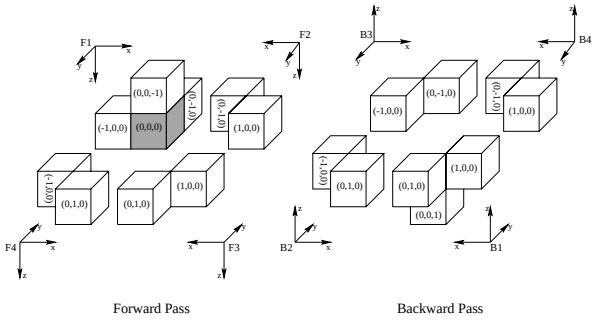

Figure 5: Eight pass vector-city vector distance transform.

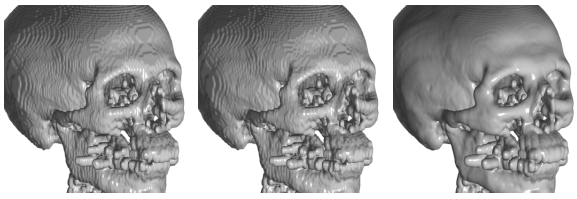

Figure 6: 5x5x5 DT on binary, efficient vector DT on binary, vector-city VDT on sub-voxel accurate data.

calculate each voxel individually from all other voxels, although in reality we calculate each slice individually. Therefore, the best run-time would arise when we have as many processors as slices. For our experiments we usually employed 8 processors giving us a time of about 15 minutes for the UNC CThead. We have also shown that where possible, it is better to reject the binary segmentation in favour of a sub-voxel accurate segmentation. We have also demonstrated a procedure for generating a distance shell, which can then be fed to a distance transform (either vector or chamfer). The use of this shell rather than a binary segmentation increases accuracy, and further we have shown that our new VCVDT produces results which are 50% more accurate than the previous best algorithm (in a comparable time). It is hoped that researchers are able to digest these results and decide to what accuracy they need distance fields, and use the appropriate method.

4 Applications

4.1 Contour Connection

We originally proposed distance fields as an intermediate step to create triangular meshes from contour data [17]. Essentially the algorithm involved calculating the distance from each voxel within the domain, to the closest point on the set of contours representing the object of interest. One possible surface that is described by those contours can be obtained by triangulating the isosurface for the distance of zero. It was found that the use of the distance field helped avoid costly and difficult point correspondence problems – particularly in 1 to many and many to many branching cases. Figure 7 demonstrate some particularly difficult situations (for other methods), and the surfaces produced for those cases.

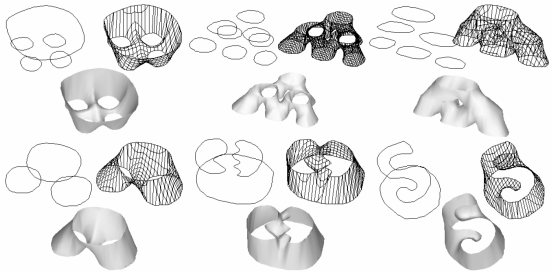

Figure 7: Some cases of contour connection.

4.2 True 3D Closing Operator

One particularly interesting area for the use of distance fields is that of morphological operations. A closing operation can be carried out by using n dilation operations followed by n erosion operations. This has the effect of closing holes in the dataset, smoothing out sharp spikes and filling cavities. It is particularly useful for shape and object recognition (although we have a further use described in Section 5). The operation of n dilations indicates those voxels which are n voxels away from the surface, and if displayed will show a shape that has appeared to have grown. When followed by n dilations, it will return the original surface, although some of the artifacts mentioned above will have been removed. The method is carried out on binary segmented data. Höhne [18] has reported the use of three-dimensional erosions and dilations as a useful application for the extraction of homogenous regions in body scans. An alternative to erosions and dilations upon binary segmented data, is to increase or decrease the grey-level voxel values themselves. For example, if the template is a $3 \times 3 \times 3$ matrix where each value in the matrix is 1, for a dilation from a binary segmentation f, each voxel v where $f(v) = 1$ will be included in the new surface along with all of its 26 neighbours. Using the original voxel values, a grey-level dilation will set a voxel to the value of its maximum neighbour (plus 1).

Intuitively a dilation of degree n should give us the exact surface where each point on the surface is n voxels away from the original surface. An erosion of degree n should give us the exact surface where each point on the surface is n voxels inside the original surface. We therefore propose that a true 3D closing operation should be based on the accurate sub-voxel distance field and used in the following manner.

First we calculate the distance field D for our surface as in equation 1. The surface, S, representing a dilation of degree n is equivalent to $S = \{q : D(q) = n\}$ where $q \in \mathbb{R}^3$. To calculate the erosion of degree n for this surface S, we then calculate a new distance

Table 1: Comparison of Distance Algorithms

Method	Time	Error Range	Average Error
True (octree)	126 minuts	0	0
City (binary)	1.280s	-2.000000 → 76.229546	12.519548
City (shell)	1.290s + 124s	-2.376824 → 73.186974	10.775360
EVDT (binary)	6.890s	-1.732051 → 3.081223	0.261987
EVDT (shell)	7.100s + 124s	-1.873284 → 1.392951	0.015674
VCVDT (shell)	7.640s + 124s	-1.873284 → 1.392951	0.010767

field, D' based upon distances from surface S using the techniques described earlier (measuring to the triangulation of the isosurface). The surface S^{C_n} which has been closed by n units through a dilation of degree n followed by an erosion of degree n is given by $S^{C_n} = \{q : D'(q) = -n\}$. Figure 8 demonstrates closing operations of 20, 10 and 5.

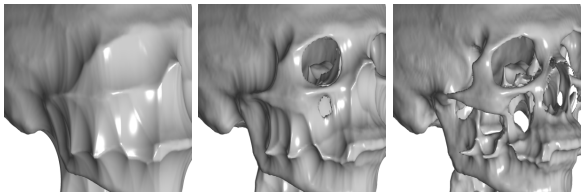

Figure 8: True 3D distance closure of 20, 10 and 5 voxels.

4.3 Hypertexture

A distance field defines the object's surface, and also the distance from and direction to the surface. This is similar to the requirements for Perlin and Hoffert's hypertexture [19] effects. We are either inside or on the surface ($D(p) \leq 0$), in the *soft* region ($D(p) \geq 0$ and $D(p) \leq \delta$) or outside of the object completely ($D(p) > \delta$) ($p \in \mathbb{R}^3$). In the soft region, noise functions are used to distort, colour or project inwards to obtain the various effects. Distance fields, as computed in this paper, conform to the data requirements for hypertexture, and can be rendered using an extension to the volume rendering algorithm. Figure 9 demonstrates the application of hypertexture to distance fields of triangular mesh objects and the UNC CThead.

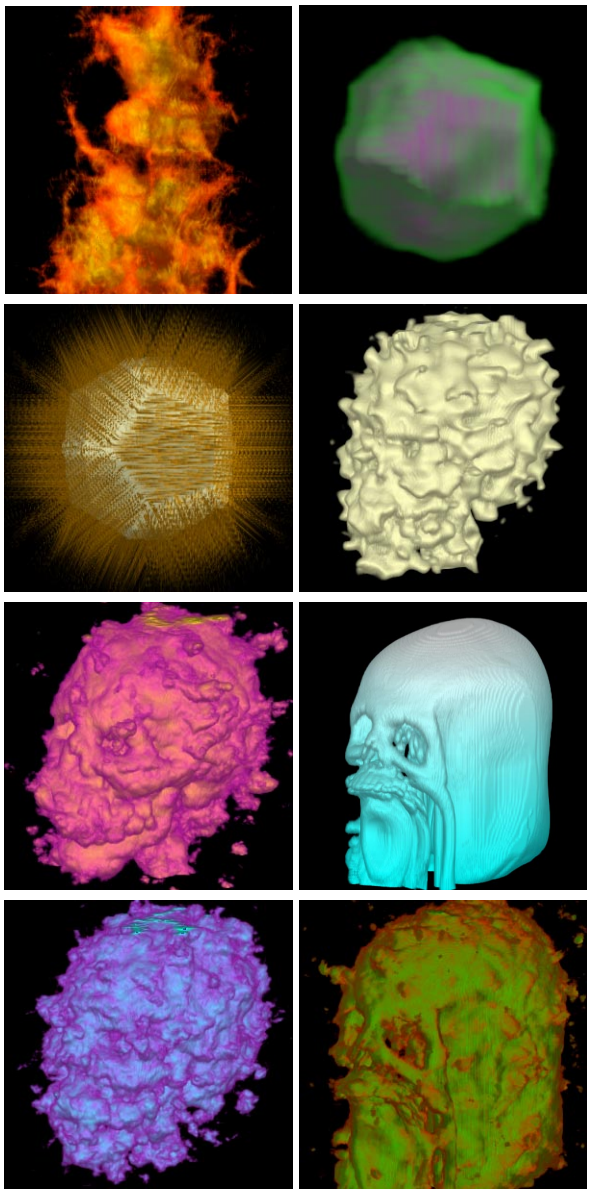

Figure 9: Hypertexture applied to the distance fields for a chess piece, a dodecahedron and the UNC CThead dataset.

4.4 Other Applications

Other applications include *skeletonization* (based upon other morphological operations), morphing by using a distance field representation, general distance operations (such as Voronoi calculation – rather than calculating the distance to a point we can calculate the actual closest point (as demonstrated by our vector shell code required by the VCVDT algorithm)), shape-based interpolation schemes, and for accelerating ray-tracing by using the distance fields as proximity clouds.

5 Conclusions and Future Work

In this paper we have shown that various methods are readily available for the calculation of distance fields. We have demonstrated that a choice can be made between accuracy and speed of computation. Perhaps our more significant results are the fact that true Euclidean distance fields can be computed using an efficient O(nlogn) algorithm, and to a far greater degree of accuracy by using a sub-voxel distance algorithm, rather than a binary segmentation. We have also introduced a new vector distance transform function (VCVDT), which has been shown to perform 50% more accurately, than the previous best function (EVDT). The EVDT algorithm was originally proposed for use with binary segmented data, and to be fair we have compared it when applied to sub-voxel accurate data aswell. We have demonstrated how to calculate the initial shell which is used for propogation. We have also demonstrated some applications, in particular the new suggestion of using the accurate Euclidean distance field for true 3D Closing operations, as demonstrated in Figure 4.2.

Our future work will concentrate on improving accuracy – we believe that a hybrid method directed by the distance transform field will be able to give an exact distance field. We also intend using the closed skull datasets as a basis upon which to create a facial reconstruction of a discovered skull, which will have application to the field of forensic science.

Acknowledgements

This work has been undertaken with funding from EPSRC, UK, under grants GR/L88238 and GR/R11186.

References

[1] M. Chen, A. Kaufman, and R. Yagel, editors. *Volume Graphics*. Springer-Verlag, 2000.

[2] A. E. Kaufman. State-of-the-art in volume graphics. In *Volume Graphics*, pages 3–28. Springer, 2000.

[3] U. Tiede, K. H. Höhne, M. Bomans, A. Pommert, M. Riemer, and G. Wiebecke. Investigation of medical 3D-rendering algorithms. *IEEE Computer Graphics and Applications*, 10(2):41–53, March 1990.

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

[5] Y. Zhou, A. Kaufman, and A. W. Toga. Three-dimensional skeleton and ceterline generation based on an approximate minimum distance field. *The Visual Computer*, 14:303–314, 1998.

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

[7] D. Cohen and Z. Sheffer. Proximity clouds - an acceleration technique for 3D grid traversal. *The Visual Computer*, 11:27–38, 1994.

[8] O. Cuisenaire and B. Macq. Fast euclidean distance transformation by propagation using multiple neighborhoods. *Computer Vision and Image Understanding*, 76(2):163–172, 1999.

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

[10] M. W. Jones. Voxelisation of polygonal meshes. In *Proceedings of 13th Annual Conference of Eurographics (UK Chapter) (Loughborough, March 28-30, 1995)*, pages 160–171, March 1995.

[11] M. W. Jones. 3D distance from a point to a triangle. Technical Report CSR-5-95, Department of Computer Science, University of Wales, Swansea, February 1995.

[12] Andrew S. Winter and Min Chen. vlib: A volume graphics library. 2001. Submitted to Volume Graphics 2001.

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

[14] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.

[15] P-E. Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14:227–248, 1980.

[16] J. C. Mullikin. The vector distance transform in two and three dimensions. *CVGIP: Graphical Models and Image Processing*, 54(6):526–535, 1992.

[17] M. W. Jones and M. Chen. A new approach to the construction of surfaces from contour data. *Computer Graphics Forum*, 13(3):C–75–C–84, September 1994.

[18] K. H. Höhne and W. A. Hanson. Interactive 3D segmentation of MRI and CT volumes using morphological operations. *Journal of Computer Assisted Tomography*, 16(2):285–294, 1992.

[19] K. Perlin and E. M. Hoffert. Hypetexture. In *Proc. SIGGRAPH '89 (Boston, Mass., July 31-August 4, 1989)*, volume 23(3), pages 253–262. ACM SIGGRAPH, New York, July 1989.