Hybrid Distance Field Computation

Richard Satherley and Mark W. Jones

University of Wales Swansea, Singleton Park, Swansea SA2 8PP, UK
{csrich, m.w.jones}@swansea.ac.uk

Abstract. Distance fields are a widely investigated area within the area of Volume Graphics. Research is divided between applications; such as – skeletonisation, hypertexture, voxelisation, acceleration of rendering techniques, correlation and collision detection; and the fundamental algorithmic calculation of the distance fields. This paper concentrates on the latter by presenting a new method for calculating distance fields and comparing it with the current *best* approximate method and the *true* Euclidean distance field. Details are given of the algorithm, and the acceleration methods that are used for calculating the true distance field. Brief descriptions of applications for these accurate distance fields are given at the end of the paper.

1 Introduction

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

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

For each point p in the domain, we evaluate $D(p)$ in order to find the distance to the surface of the object of interest. Where $D(p)$ is negative if we know that the point is inside the surface. (The choice of sign may depend upon the application). More usually D is stored as a discrete voxel array and where p is not a known voxel, its value is interpolated from its neighbouring voxels.

Calculating D can be very expensive if implemented without acceleration (either spatial data structures or approximation algorithms). In a typical voxelisation of a chess piece consisting of 1500 triangles to a grid of $60 \times 60 \times 60$ the brute force method calculates the distance 324 million times. Converting a CT dataset (such as the UNC CThead) to a distance field requires the distance to be measured from every voxel to every other voxel – approximately 50 trillion operations. The brute force algorithm was implemented for comparison purposes – a rook chess piece requires 287 seconds for calculation on a 1GHz Athlon, and the CT head required about 64 days (although this was executed in parallel over 8 machines and the distance was calculated to sub-voxel accuracy (Section 3.3)).

This sheer computational expense has led many researchers to discover approximate methods for the calculation of the distance field, which are briefly described in Section 2.2. Of these methods, the vector propagation method is the most successful, and our extension to this algorithm (the VCVDT) is presented in full in Section 3, along with a detailed comparison to existing techniques. We further improve the quality of the algorithm by proposing that the distance transform is applied to a sub-voxel accurate distance shell, rather than a binary segmentation of the object as is most widely carried out currently (Section 3.3). Our most important result in this paper is the fact that a hybrid technique employing accurate distance calculations steered by propagation distance calculations can produce a very accurate approximation in a reasonable time (Section 4). In Section 4 we compare all of the methods and demonstrate that the latter is far more accurate than any previous method, whilst still having the benefit of being reasonably quick to compute. Section 5 reviews our hypertexture and morphological operators for forensic science applications using these accurate distance fields.

2 Related Work

2.1 Distance Fields

Distance fields have been used as an intermediate step for the creation of triangular meshes from contour data by Jones [1]. Essentially the algorithm involves 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 represented by those contours can be obtained by triangulating the iso-surface for the distance of zero. It was found that the use of the distance fields helped avoid costly and difficult point correspondence problems – particularly in 1 to many and many to many branching cases.

The use of distance fields was later developed by Jones for the accurate encoding of objects as volume datasets for the area of volume graphics [2]. It was found that triangular mesh objects could be efficiently voxelised into volume data using shell distance fields (distances only calculated in the vicinity of the surface). As the first derivative of the distance field is normal to the surface, the normal for shading can be calculated using simple central differences. This method improved upon previous binary segmentation methods by removing the stepped edges of the object caused by the simple binary occupancy decision, and allowing objects to be represented with a much sparser grid. Kaufman and Šrámek followed an alternative voxelisation method using object filtering [3], and Gibson [4] smooths the *stepping* effects, while maintaining thin crevasses and protrusions, with the use of an *elastic surface net*. A smooth surface is obtained by (iteratively) relaxing the positions of a set of connected *nodes* (initially place on the binary surface), to reduce the energy of the surface net. Each node is constrained to stay within its original cube, ensuring the correct representation of the binary object. A distance field is calculated from the surface net once it has reached its final position by triangulating it and using a method similar to [2].

Frisken *et al.* [5] also demonstrated the hierarchical encoding of objects using distance fields giving a sparse set of points away from the object, and a higher resolution close to the surface. This gives the advantages of a good surface representation (as in

the shell method) and the extra information away from the surface that can be used to generate rendering effects such as glow.

Breen, Mauch and Whitaker [6] calculate sub-voxel accurate distance fields for Constructive Solid Geometry objects modelled using superellipsoid primitives. Their distance field is computed in two stages. Firstly, a narrow band of points near the evaluated surface (much like our distance shell) and a second set of points lying on the surface (the *zero set*) are calculated with a modified version of Breen's [7] Constructive Cubes algorithms. Each point in the zero set is associated to a point in the narrow band, which is next propagated throughout the distance field, using a variation of Sethian's [8] Fast Marching Method. The accuracy of their method is dependent upon the resolution of the voxelising grid used when calculating the zero set, and the method suffers from similar problems as described in section 4, caused by not recalculating the distance with the underlying primitive (superellipsoids), which the hybrid technique in this paper goes some way towards solving for triangular mesh objects.

Distance field information has been used to accelerate rendering by Cohen and Shefer [9], Semwal and Kvarnstrom [10] and Šrámek and Kaufman [11]. The general principal behind each method is to use the distance information to skip over large empty spaces. Breen and Whitaker [12, 13] and Cohen-Or *et al* [14] use distance fields and warp functions to create morph sequences between volume objects. Gagvani and Silver [15] have used distance fields for creating bounding spheres for collision detection, and for creating skeletal representations for animation [16]. Danielsson [17] and Zhou and Toga [18] also use distance information to extract the skeletal representation of an object.

2.2 Distance Transforms

As stated in Section 1, a *true Euclidean distance field* is computed such that each point within the field retains the minimal distance to the object's surface. The brute force method can be improved through the use of an *octree* [19] and various neighbour information. This reduces the calculation time to around two hours for the CThead, but even this significant improvement does not render the method feasible.

The computational expense of the Euclidean distance field calculation is due to its global nature. *Distance transforms (DT)*, introduced by Rosenfeld and Pfaltz [20], reduce this global operation to simple addition, by approximating the Euclidean distance calculations via *local distance propagation*, achieved with a number of passes of a *distance matrix*. This considerably reduces the time taken to calculate the distance field, at the cost of reduced accuracy.

Since their original proposal distance transforms have seen numerous improvements. Ranging from superior distance matrices [21] to extending the local propagation to pass vectorial information [17, 22–24]. These *vector* (or *Euclidean*) *distance transforms (VDT)* are more accurate than their chamfer counterparts with Mullikin's *efficient vector distance transform (EVDT)* [24] (Figure 1) being the most accurate method reported in the literature.

Cuisenaire [25] presents an excellent review of 2D and 3D distance transforms and identifies many of the error situations which may arise. His main contribution is the correction of these errors as a post-processing operation. Section 3.2 identifies a further

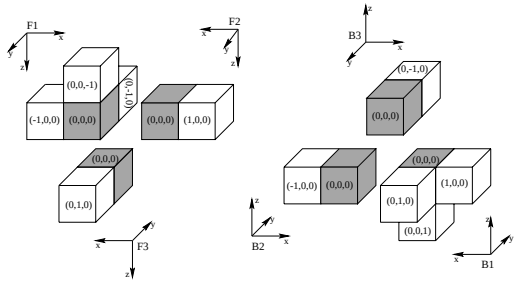


Fig. 1. Matrix portions used by the six passes of the EVDT.

error which the VCVDT avoids. His error correction techniques could be implemented in conjunction with our method to improve it further.

Mullikin developed the EVDT from Ye's 4SSED [22], to compute accurate distance fields from anisotropically sampled data. To reduce memory requirements the EVDT stores the signed vector components for only two slices of the dataset, that is, only the vector components for the current and previous slices are stored, with new slices being created, and old ones removed, as the scan passes through the dataset. To further improve efficiency, costly multiplications are avoided using three lookup tables (one for each of the vector components).

Apart from Breen, Mauch and Whitaker's CSG encoding [6], all methods start by creating a segmentation function f as:

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

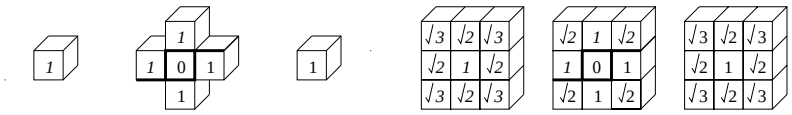
An initial distance field, D, is computed using the segmentation:

$$D(p) = \begin{cases} 0 & \text{if } f(p) = 1, \exists\ q \in p_{26}, f(q) = 0 \\ \infty & \text{otherwise} \end{cases} \tag{3}$$

where $p \in \mathbb{Z}^3$, and p_{26} is the set of voxels which are the 26 neighbours of p

It is this distance field which is then propagated using the chosen distance transform. For example, a chamfer distance transform propagates the distances using Equation 4, where d_M (the distance matrix) provides the local distances. Example chamfer distance matrices are given in Figure 2. The most elementary distance transform, the *city-block CDT* (Figure 2(a)), only considers its direct neighbours when computing the distance field. More accurate distance fields can be obtained if a larger local neighbourhood is considered and realistic distance values are used (Figure 2(b)).

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



(a) City-block (b) Complete $3 \times 3 \times 3$

Fig. 2. Example chamfer distance matrices.

The next section demonstrates the improved accuracy of our new VCVDT for three-dimensional binary segmented data and identifies some of the problem cases for EVDT. Later we demonstrate even greater accuracy by employing a sub-voxel accurate calculation for triangular meshes (and for voxel data employing a tiling algorithm).

3 Vector-city Vector Distance Transform

3.1 Implementation

The fact that the city-block chamfer distance transform is the most elementary of all the distance transforms, has lead to it being the basis of the majority of the vector distance transforms.The new *vector-city vector distance transform (VCVDT)* extends this heuristic to include all four matrix passes used to compute the distance field, Figure 3 shows the matrix portion used by each pass.

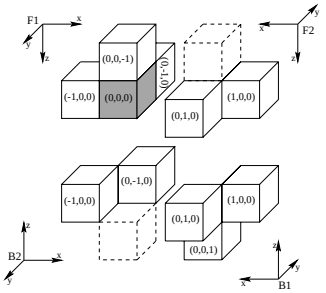


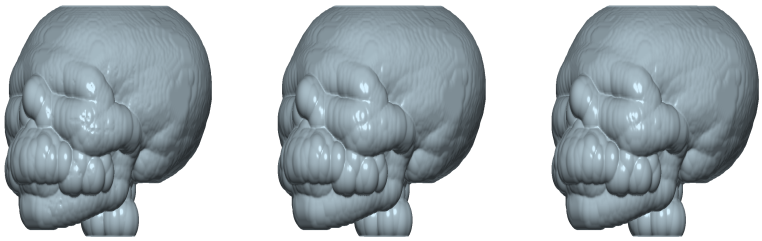
Fig. 3. Distance matrix used by the four pass vector-city vector distance transform.

The VCVDT is implemented in a manner similar to that of EVDT. During each pass ($F1$, $F2$, $B1$ and $B2$) the corresponding matrix segment is applied in the direction indicated in Figure 3. At each voxel, its neighbours' vectors are altered according to the overlying matrix element and the minimal vector stored. Unlike the EVDT, the VCVDT stores a complete vector copy of the distance volume (the reasons for which will be given in Section 3.2). Furthermore, the VCVDT employs Ye's [22] and Leymarie and Levin's [23] method of storing (after each pass) the minimal distance along with the minimum vector. This strategy allows Leymarie and Levin's [23] optimisations to be used and removes the need to recalculate the (current minimum) distance for the central voxel, saving three distance calculations per voxel. To further increase efficiency the matrix positions that only need to be checked once, i.e. the vertical positions (shown as dashed positions in Figure 3), are not included in subsequent distance calculations. Therefore, only eleven distance calculations and ten comparisons are made per voxel.

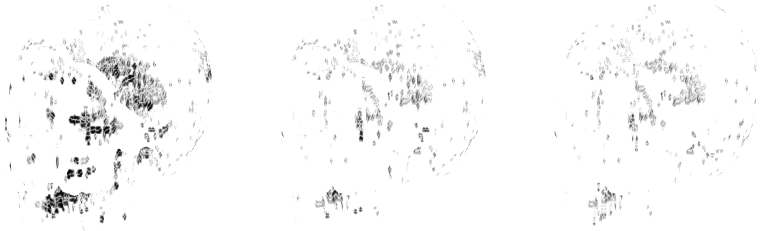
3.2 Effects of Different Vector Storage Methods

From the previous section it can be seen that the major differences between the VCVDT and the EVDT, are the distance matrices used and the manner in which the vectors are stored. The difference between the two distance matrices is obvious – the EVDT performs six passes over the dataset, whereas the VCVDT only makes four passes. This along with the other (distance calculation) savings, amounts (when using the UNC CThead) to a reduction of over 37 million distance calculations.

The remainder of this section will compare the two types of vector storage. To ensure a balanced comparison, a version of the EVDT which makes use of a complete vector representation of the distance field rather than the proposed implementation [24], has been implemented. The distance field generated by each implementation are displayed (at an offset of 10 units) in Figure 4(a) – 4(c), with the difference between the computed distance field and the (binary segmented) Euclidean distance field shown in Figure 4(d) – 4(f). ***N*ote** – *The images have been darkened to make the difference more visible.*



(a) EVDT (b) Evdt grid (c) VCVDT



(d) Euclidean - (a) (e) Euclidean - (b) (f) Euclidean - (c)

Fig. 4. Offset surfaces and the difference from the true Euclidean distance field for the implemented VDTs.

The inclusion of the full grid implementation of the EVDT has brought to light a new flaw which only occurs when a distance transform is implemented using a limited number of vector slices. The distance fields generated in this manner have a large number of errors compared to those obtained when a full vector grid is used.

Mullikin [24] detailed one of the causes of these errors (which is later corrected by Cuisenaire [25]). The reported errors occur when the feature voxels are arranged in such a way that a feature's propagation front is unable to reach a point within its own Voronoi tile. Arrangements of this kind cause local errors, in that only the voxel at the focal point of the arrangement and possibly a few of it neighbours are given incorrect distances.

Whilst analysing Mullikin's EVDT, a voxel arrangement which produces errors that are far more widespread was discovered. A diagonal line of three or more feature voxels on the *z*-plane, lying in the same direction as the matrix passes are applied, causes the wake like error propagation illustrated in Figure 5.

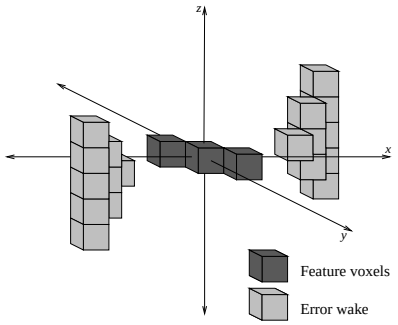


Fig. 5. Error wake caused by diagonal feature voxels.

The error wake only occurs when a limited number of vector slices are used. This can be easily proven by performing the following simple experiment – apply each distance transform to a (small) binary dataset, containing only three feature voxels arranged as described above. Next calculate the true Euclidean distance field for the dataset and compare the results. The comparison shows that only the distance fields generated by the standard EVDT is erroneous.

Figure 6 illustrates, for the true Euclidean distance field (on the slice containing the feature voxels), which feature voxel (shaded voxels) is closest to each of the background voxels, where a voxel with two patterns is equidistant from both of the corresponding features. Figures 7 and 8 show how the relationship between the feature and background voxels develops with each pass of the standard EVDT and VCVDT respectively, where an empty voxel indicates that it has not been reached by the propagation front. Notice that pass $B1$ of the standard EVDT (Figure 7(d)) does not alter the relationship between the feature and background voxels, whereas pass $B1$ of the VCVDT (Figure 8(c)) does. In Figures 1 and 3 it can be seen that these two passes are identical. The explanation for the existence of the error wake is the loss of vector information when only using a limited number of vector slices.

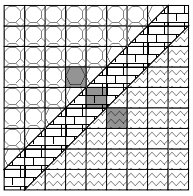
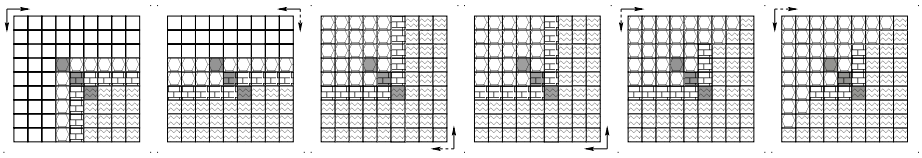


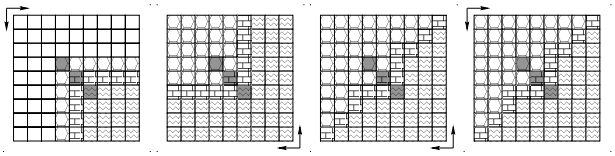
Fig. 6. Feature voxel to background relationship.

Table 1 gives the computational results for each distance transform when computing the distance field for the (binary segmented) skull of the UNC CThead. The error range is obtained by subtracting the voxel values of the true Euclidean distance field from those of the computed fields. The minimum error value is equal to the largest negative difference, with the maximum error value being equal to the largest positive difference. The average error per voxel is thus calculated by summing the absolute value of each



(a) $F1$ (b) $F2$ (c) $F3$ (d) $B1$ (e) $B2$ (f) $B3$

Fig. 7. Feature to background voxel relationship after each pass of the standard EVDT.



(a) $F1$ (b) $F2$ (c) $B1$ (d) $B2$

Fig. 8. Feature to background voxel relationship after each pass of the VCVDT.

subtraction and dividing the result by the number of voxels in the dataset. It can be seen from the table that the extra memory management needed when using a limited number of vector slices, increases the time taken to compute the distance field. Furthermore, Table 1 confirms that the use of a complete vector copy of the distance field greatly increases the accuracy of the EVDT distance field.

Table 1. Average execution times and a comparison of the three vector distance transforms to the true Euclidean distance of the CThead. Timings where made on a 800 MHz Athlon

Distance matrix	Execution time (s)	Error range min	max	Mean error per voxel
EVDT	7.540	-0.518	2.533	0.004761
EVDT Grid	6.348	-0.504	1.053	0.000786
VCVDT	3.420	-0.334	0.446	0.000644

A final observation from Table 1 is that the VCVDT produced, in less time, a distance field which has a smaller average error per voxel compared to the distance field produced by the EVDT with full vector grid. Table 2 presents the results of the application of these distance transforms on various datasets. (For means of comparison the average error per voxel is also given for the *quasi-Euclidean chamfer distance transform (Q-E CDT)*). The datasets are listed in Table 3 along with the true Euclidean computation times. From Table 2 it is clear that the VCVDT out performs the EVDT with full vector grid in all cases. The table also shows that the performance of VDTs is only dependent on the size of the dataset.

Table 2. Results of applying the EVDT with full vector grid and the VCVDT to the datasets of Table 3. (800 MHz Athlon).

Dataset	Execution time (s)		Error range				Mean error per voxel		Q-E CDT
	EVDT grid	VCVDT	EVDT grid		VCVDT		EVDT grid	VCVDT	mean error per voxel
			min	max	min	max			
CThead skull	6.348	3.420	-0.504	1.053	-0.334	0.446	0.000786	0.000644	0.612223
Sphere	0.424	0.204	-1.053	1.713	-0.385	0.425	0.005282	0.002734	0.163602
Pawn	0.184	0.094	-0.394	0.504	-0.334	0.268	0.001683	0.001546	0.406946
Queen	0.184	0.094	-0.268	0.386	-0.268	0.334	0.001415	0.001380	0.466726
Rook	0.184	0.094	-0.504	0.597	-0.334	0.299	0.002552	0.002307	0.368889
Pawn and rook	0.360	0.174	-0.504	0.597	-0.334	0.299	0.002118	0.001930	0.291697

Table 3. Computational time for the full Euclidean distance field for the various datasets used in the comparisons of Tables 2 and 4 (800 MHz Athlon).

Dataset	Resolution	# feature voxels	Execution time (s)
CThead skull	$256 \times 256 \times 113$	183194	7560.000
Sphere	$80 \times 80 \times 80$	15425	1212.410
Pawn	$60 \times 60 \times 60$	3340	98.650
Queen	$60 \times 60 \times 60$	2357	66.360
Rook	$60 \times 60 \times 60$	4050	127.480
Pawn and rook	$120 \times 60 \times 60$	7390	446.440

The accuracy of the VCVDT can be further improved by increasing the number of passes made by the distance matrix from four to eight, as shown in Figure 9, adding an extra eight distance calculations and comparisons per voxel. Thus, the eight pass VCVDT makes a total of nineteen distance calculations and eighteen comparisons per voxel.

The results of applying the eight pass version of the VCVDT to selected datasets from Tables 3 are given in Table 4, which highlights that the 8VCVDT is over five times more accurate than the improved EVDT for a comparable time, and over a thousand times more accurate than the quasi-Euclidean CDT (which is used in the volume graphics literature).

3.3 Sub–voxel Accuracy

The use of binary segmentation to extract a surface, prior to applying a distance transform, results in the surface being misrepresented – the discretisation of the surface is visually obvious. We can see this in Figure 10, where Figure 10(a) demonstrates the binary segmented data, 10(b) shows the original data before binary segmentation and 10(c) shows the sub-voxel distance shell.

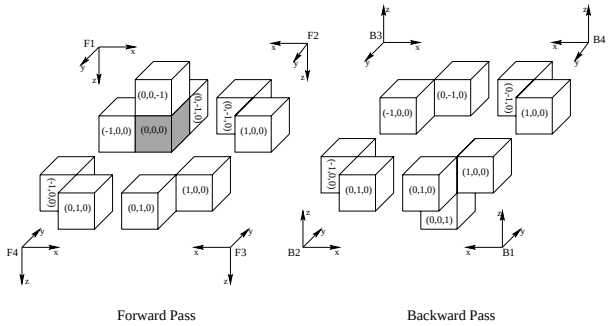
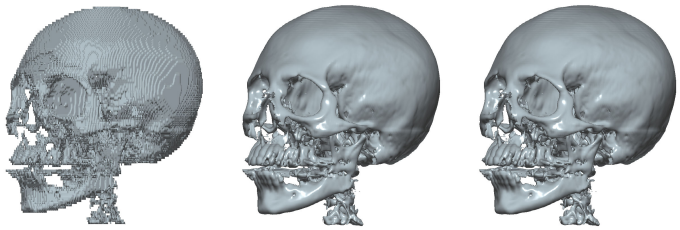


Fig. 9. Matrix used by the eight pass VCVDT.

Table 4. Results of applying the 8VCVDT to selected datasets from Table 3. (800 MHz Athlon).

Dataset	Execution time (s)	Error range min	Error range max	Mean error per voxel
CThead skull	5.470	-0.334	0.334	0.000223
Sphere	0.302	-0.268	0.268	0.000559
Pawn	0.134	-0.310	0.278	0.000233
Queen	0.134	-0.268	0.268	0.000179
Rook	0.134	-0.334	0.268	0.000302
Pawn and rook	0.260	-0.334	0.278	0.000268



(a) Binary (b) Original (c) Sub-voxel

Fig. 10. The UNC CThead (a) binary segmented, (b) original and (c) sub-voxel shell distance field. (Note the similarity of the distance field normal shading to the original grey-level normal shading.)

As previously mentioned, the solution could be to smooth the data [4] or to propagate a distance shell calculated using sub-voxel accurate distance measurements. Breen, Mauch and Whitaker [6] use a method similar to this for CSG models.

Our contribution is to calculate sub-voxel accurate distance fields for objects represented by triangular meshes and sampled volume datasets. In the case of triangular mesh objects we can calculate the distance to the closest point on each triangle. This

method may also be used for sampled volume data, if the surface is first triangulated. We use Payne and Toga's [26] tetrahedral decomposition and tiling algorithm as it removes the ambiguous cases of Marching Cubes and creates a topologically correct closed surface tiling. There are alternative methods for obtaining the surface contained within a dataset directly from a reconstructed trilinear interpolation function [27], although these are more computationally expensive to compute. Further work should determine the error between using such a reconstruction function and the method of triangulating the volume cell on the fly using Payne and Toga's [26] method.

We call this sub-voxel accurate distance field the *true distance field*, and will compare the results of all methods to this true field in the remainder of this paper (previous comparisons were to the binary segmented distance field).

The large number of calculations results in high computation times – computing a true sub-voxel accurate distance field, for the skull of the UNC CThead, using only the shortest distance to a triangle calculation and an octree requires a few hundred million of distance calculations and takes about 2 hours.

The execution time can be greatly reduced by employing a distance transform, but instead of f encoding a binary segmentation, f encodes the triangulated surface. For the purposes of propagation we only need to have information about the surface itself, all other voxels can be computed via the transform process from this initial information.

We call those voxels that need to be computed the *distance shell* of the object. The distance shell consists exactly of those voxels that are required to encode the surface, and if displayed at an iso-value of zero they will give an accurate representation of the surface. To create the distance shell we 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 $p \in v_{26}$.

The rook 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). The impact on accuracy is demonstrated in Table 5 where we can see the large difference between chamfer distance transforms on binary segmented data and the 8VCVDT on the shell data (compared to the true sub-voxel accurate distance field of the CThead).

Table 5. Comparison of Distance Algorithms. (1 GHz Athlon).

Method	Execution time (s)	Error range min	Error range max	Mean error per voxel
True (octree)	7560.000	0.000	0.000	0.000000
City (binary)	1.280	-2.000	76.230	12.519548
City (shell)	1.290	-2.377	73.187	10.775360
EVDT (binary)	6.890	-1.732	3.081	0.261987
EVDT (shell)	7.100	-1.873	1.393	0.015674
8VCVDT (shell)	7.640	-0.451	0.316	0.009668

4 Hybrid Method

Employing a sub-voxel accurate distance measurement for each voxel in a distance field produces a more accurate representation of the object and its offset surfaces, and improves the visual appearance of any rendering of the surface (e.g. using hypertexture). When comparing true distance field dataset with the vector propagation of a binary segmented dataset and with the vector propagation of the sub-voxel accurate distance shell there was a great increase in accuracy in the latter case. The remaining accuracy problems are due to the fact that the measured surface is now continuous. For each known voxel, a vector is stored which points to the closest point upon the surface of the object within a cell. When this is propagated to any neighbour, the neighbouring voxel will also have a vector which points to that point (if the cell is minimal), even though it may not be the closest point (within that cell). Under binary segmentation, both voxels would have correctly pointed to the same voxel, whereas in the sub-voxel accurate situation they should now point to different points within the same cell. This can be seen in Figure 11.

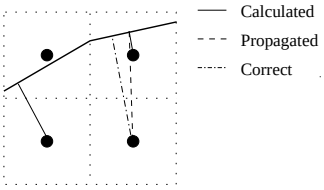
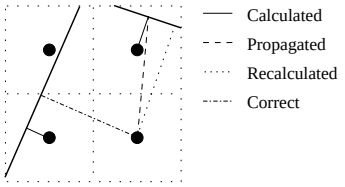
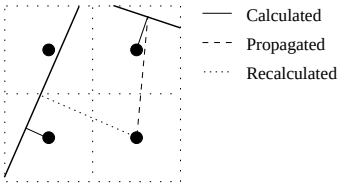


Fig. 11. The bottom left and top right points have the correct calculated vectors to the surface. The bottom right point has a propagated vector which is incorrect.

Our hybrid method follows a three stage process. Firstly it calculates the voxel shell of the object. This is then propagated using the most accurate method – the 8VCVDT, although the vectors are stored rather than the final distances. Finally these vectors are used to direct a second pass through the dataset where each voxel has its sub-voxel accurate distance calculated to the cell indicated by the propagated voxel. This will produce the exact result for all voxels where the cell indicated by the propagated vectors is the correct one (the correct vector in Figure 11). Where the indicated cell is incorrect the problem in Figure 12(a) will be created. To solve this problem it is a good idea to calculate the distance to the 8 cells that a voxel belongs to. This would result in the correct computation of the vector as in Figure 12(b).



(a) Recalculating the surface within indicated cell can lead to incorrect surface.



(b) Recalculating within neighbouring cells gives the correct surface.

Fig. 12. Propagated vectors are recalculated in the final step of the hybrid method.

For a dataset consisting of n voxels, the full distance computation algorithm has complexity $O(nlogn)$ by employing an octree. If we have m voxels in the voxel shell of the dataset, the approximate sub-voxel accurate propagation method reduces to $O(n + mlogn)$ (where the first term represents the distance transform passes through the dataset). The hybrid method also has complexity $O(n + mlogn)$ as it just makes an extra pass through the dataset at the end. Therefore if it was possible to ensure that the hybrid method calculated the distance to the correct cell in every case, we could state that the hybrid technique fundamentally improves the Euclidean distance calculation algorithm. Figure 13 shows the case where the algorithm fails to find the correct closest point. The vector propagation method suggests that the left cell is the closest (which it is), however the surface within the left cell is further away than the surface within the right cell, and therefore the cell used to compute the sub-voxel accurate distance is incorrect.

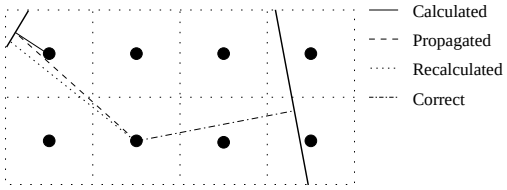


Fig. 13. Example of a problematic case for the hybrid method.

We are now examining the use of a tie-list (which has been previously proposed for a related problem [24]) to solve this problem. Table 6 shows the hybrid method, the EVDT applied to a binary segmentation and the 8VCVDT applied to a distance shell compared to the true sub-voxel accurate distance field. We can see that it is an order of magnitude more accurate, but without too much of a performance penalty. The clearest demonstration of the importance of this method is the fact that 90% of the voxels have the correct distance computed for them compared to less than 1% for the previous best. (Incorrect voxels are those where the distance stored at the voxel differs from the true Euclidean distance.)

Table 6. Accuracy of the various methods when compared to the true sub-voxel accurate distance field. (1 GHz Athlon).

Method	Execution time (s)	Mean error per voxel	Incorrect voxels (%)
True (octree)	7560.000	0.000000	0.000
Hybrid	270.000	0.001299	9.7356
8VCVDT (shell)	131.640	0.009668	72.146
EVDT (binary)	6.890	0.261987	99.682

5 Applications

5.1 Hypertexture

Figures 14 and 15 (colour plate) demonstrate various hypertexture effect on a torus, sphere, dodecahedron, chess piece, skull and tank dataset. A more detailed account of the application of hypertexture to volumetric datasets can be found in [28].

We have also used distance fields for true 3D morphological erosion, dilation, opening and closing operations (Figure 16). We will use the closing operation to create closed datasets of the skull, which will then be used to map skin over for the application of reconstucting facial features of discovered remains as a means to aid identification.

6 Conclusion

In this paper we have shown that vector propagation methods are more accurate than simple chamfer distance transforms. We have introduced the new VCVDT with 4 and 8 passes, and demonstrated in Table 2 its improvement over the existing best algorithm – the EVDT. Section 3.3 presents a further improvement to the distance measurement process by using sub-voxel accurate distances calculated to the actual surface (or surface tiling), rather than the standard calculation to the binary segmentation of the surface. Table 5 demonstrates that this results in a surface that is over 1000 times more accurate than using the binary segmentation and EVDT. Our final contribution to this paper is to use a hybrid technique to calculate distance fields. The sub-voxel accurate distance shell is propagated using the most accurate 8 pass VCVDT, and then those vectors are used to direct a further pass through the data in order to calculate the distance to the correct sub-voxel surface. Table 6 demonstrates that this method produces the correct distance for 90% of voxels, whereas the previous best (EVDT on binary segmented data) produces the correct distance to less than 1% of voxels. Section 5 shows how the new accurate distance field benefits our research in two main application areas.

References

1. M. W. Jones and M. Chen. A new approach to the construction of surfaces from contour data. *Computer Graphics Forum (Proceedings of Eurographics '94)*, 13(3):C75–C84, September 1994.
2. M. W. Jones. The production of volume data from triangular meshes using voxelisation. *Computer Graphics Forum*, 15(5):311–318, December 1996.
3. M. Šrámek and A. E. Kaufman. Alias-free voxelization of geometric objects. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):251–267, July–September 1999.
4. S. F. F. Gibson. Constrained elastic surfacenets: generating smooth surfaces from binary sampled data. In *Proceedings Medical Image Cmputation and Computer Assisted Interventions, MICCAI'98*, pages 888–898, October 1998.
5. 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. *Computer Graphics (Proceedings of SIGGRAPH 2000)*, pages 249–254, July 2000.

6. D. E. Breen amd S. Mauch and R.T. Whitaker. 3D scan conversion of CSG models into distance volumes. In *Proceedings of the 1998 Symposium on Volume Visualisation*, pages 7–14, October 1998.
7. D. E. Breen. Constructive Cubes: CSG evaluation for display using discrete 3D scalar data sets. *Computer Graphics Forum (Proceedings of Eurographics '91)*, pages 127–142, September 1991.
8. J. A. Sethian. *Level Set Methods*. Cambridge University Press, 1996.
9. D. Cohen and Z. Sheffer. Proximity clouds - an acceleration technique for 3D grid traversal. *The Visual Computer*, 11:27–38, 1994.
10. S. K. Semwal and H. Kvarnstrom. Directed safe zones and the dual extent algorithms for efficient grid traversal during ray tracing. In *Graphics Interface '97*, pages 76–87, Kelowna, British Columbia, May 1997.
11. M. Šrámek and A. Kaufman. Fast ray-tracing of rectilinear volume data using distance transforms. *IEEE Transactions on Visualization and Computer Graphics*, 6(3):236–252, July - September 2000.
12. R. T. Whitaker and D. E. Breen. Level-set models for the deformation of solid objects. In *Proceedings of the 3rd International Workshop on Implicit Surfaces*, pages 19–35, 1998.
13. D. E. Breen and R. T. Whitaker. A level-set approach for the metamorphosis of solid models. *To appear in IEEE Transactions on Visualization and Computer Graphics*, 2001.
14. D. Cohen-Or, D. Levin, and A. Solomovici. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 17(2):116–141, April 1998.
15. N. Gagvani and D. Silver. Shape-based volumetric collision detection. In *Volume Visualization and Graphics Symposium*, pages 57–61, 2000.
16. N. Gagvani and D. Silver. Realistic volume animation with alias. In *Volume Graphics*, chapter 16, pages 253–263. Springer, 2000.
17. P-E. Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14:227–248, 1980.
18. Y. Zhou and A. W. Toga. Efficient skeletonization of volume objects. *IEEE Visualization and Computer Graphics*, 5(3):196–209, July 1999.
19. J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.
20. A. Rosenfeld and J. L. Pfaltz. Sequential operations in digital picture processing. *Journal of the ACM*, 13(4):471–494, 1966.
21. G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing*, 34(3):344–371, 1986.
22. Q. Z. Ye. The signed Euclidean distance transform and its applications. In *Proceedings, 9th International Conference on Pattern Recognition*, pages 495–499, 1988.
23. F. Leymarie and M. D. Levine. Fast rater scan distance propagation on the discrete rectangular lattice. *CVGIP: Image Understanding*, 55(1):84–94, January 1992.
24. J. C. Mullikin. The vector distance transform in two and three dimensions. *CVGIP: Graphical Models and Image Processing*, 54(6):526–535, 1992.
25. O. Cuisenaire. *Distance Transformations: Fast Algorithms and Applications to Medical Images Processing*. PhD thesis, Laboratoire de Telecommunications et Teledetection, Université Catholique de Louvain, 1999.
26. B. A. Payne and A. W. Toga. Surface mapping brain function on 3D models. *IEEE Computer Graphics and Applications*, 10(5):33–41, September 1990.
27. R. E. Webber. Ray tracing voxel data via biquadratic local surface interpolation. *The Visual Computer*, 6(1):8–15, February 1990.
28. R. A. Satherley and M. W. Jones. Hypertexturing complex volume objects. In *Proceeding of the 9-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 146–153, February 2001.

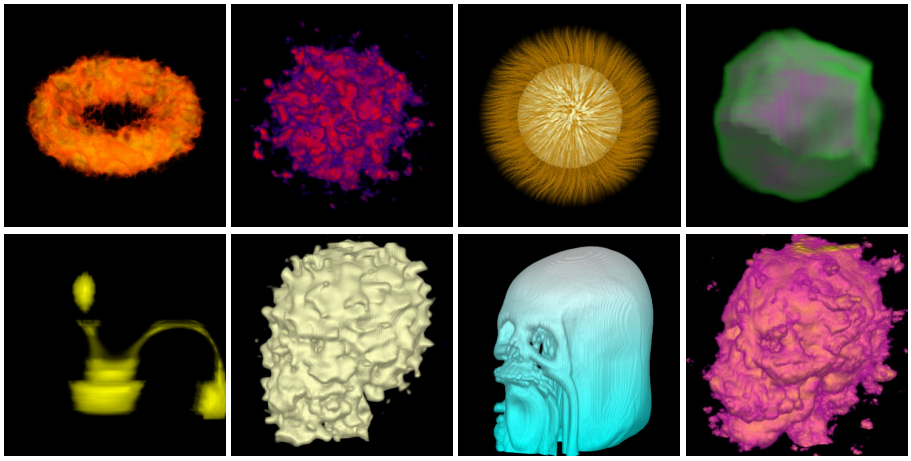


Fig. 14. Examples of hypertexture effects.

(a) Clipped hypertexture application (b) Merged hypertextures

Fig. 15. Clipped and merged hypertextures.

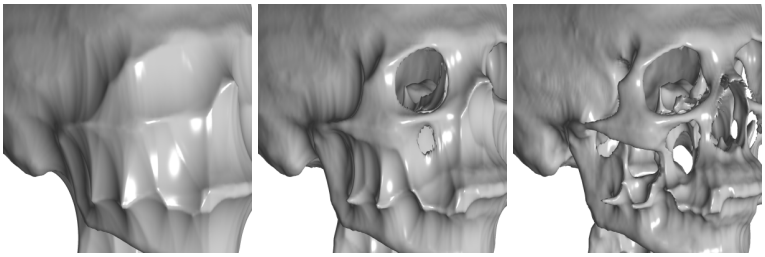


Fig. 16. True 3D distance closure of 20, 10 and 5 voxels. Skin thicknesses will be mapped onto the closed skull in order to indicate how a person may have looked.