Vector-City Vector Distance Transform

Richard Satherley and Mark W. Jones

Dept. of Computer Science, University of Wales, Swansea,
Singleton Park, Swansea, UK, SA2 8PP

E-mail: csrich@swan.ac.uk; m.w.jones@swan.ac.uk

This paper will examine the current chamfer and vector distance transforms for encoding objects as distance fields. A new vector distance transform is introduced which uses the city-block chamfer distance transform as a basis. Detailed error analysis using real CT data is presented demonstrating the improved accuracy the new approach gives over existing methods. The production of a sub-voxel accurate distance field is also demonstrated by employing an improved classification. Distance fields are shown for skull and chess piece datasets.

Key Words: distance field; distance transform; chamfer distance transform; vector distance transform; Euclidean distance; sub-voxel accuracy; voxelisation

1. INTRODUCTION

Distance fields are required by many applications such as hypertexture (Figure 1(a)), voxelisation, morphing (Figure 1(b)), facial reconstruction, measurement and flight path calculation. Our work in this area arose from the need to compute complete Euclidean distance fields from polygonal meshes and CT data for use with some of these applications.

The method for obtaining a *true Euclidean distance field* is to repetitively calculate the distance (Eq.(1)) between a voxel and every voxel on the surface, taking the minimum value as the result. This naive, brute force, method is extremely computationally expensive, taking over 62 hours when applied to the skull of UNC CThead – a $256\times256\times113$ greyscale dataset, 183,194 features. (All timings in this paper were taken on a Athlon 800.) This can be improved to two and a half hours through the use of an octree and various neighbour information, but even this significant improvement does not render the method feasible.

$$D_E = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \tag{1}$$

The computational expense of the Euclidean distance calculation is due to its global nature. *Distance transforms* (DT), also known as *distance field calculations*, devised by Rosenfeld and Pfaltz [1], approximate Euclidean distance via *local dis-*

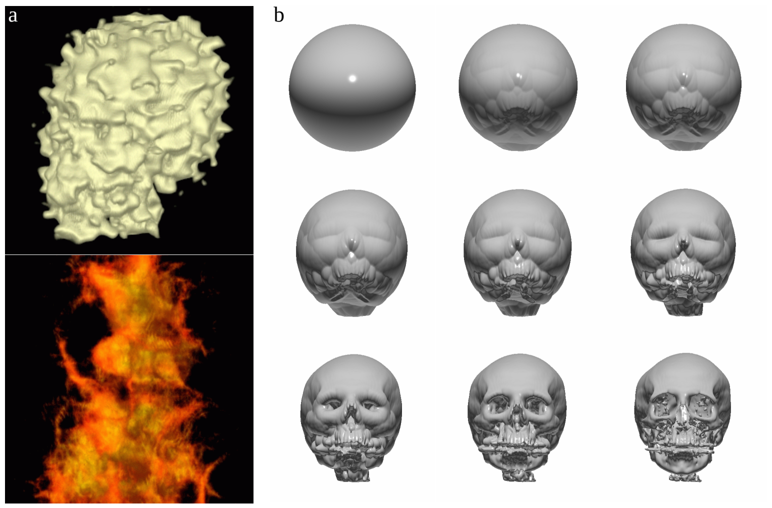

FIG. 1. (a) Hypertextured CT skull and chess piece. (b) A sphere morphing into a skull.

tance propagation, thereby reducing the global operation to simple addition. This reduces the execution time considerably, but at the cost of reduced accuracy.

Since their original proposal DTs have been the subject of numerous research papers; see Danielsson [2], Borgefors [3, 4, 5], Ragnemalm [6, 7], and Mullikin [8]. The magnitude of this interest has resulted in numerous improvements to DTs, including the introduction of *vector distance transforms (VDT)*. This paper will present a thorough review and analysis of vector distance transforms in large three-dimensional dataset situations. In addition, the two main contributions of this paper are; a further improvement to DTs, through the expansion of the basic *city-block chamfer distance transform* into the vector domain; and the computation of a distance field with true sub-voxel accuracy.

Section 2 introduces DTs with a brief description and comparison of the two categories. In Section 3 we will introduce and analyse our new *vector-city vector distance transform (VCVDT)*, giving comparisons to distance fields obtained from Mullikin's [8] EVDT, a version of the EVDT with a full vector grid, and the true Euclidean distance field. We conclude Section 4 with a further extension of DTs, by invoking a voxelisation process to classify the feature voxels, thus removing the need to perform a binary segmentation of the surface of interest prior to applying the DT. This allows the production of a distance field with sub-voxel accuracy.

2. DISTANCE TRANSFORMS

Distance transforms can be split into two categories;

- *chamfer distance transforms (CDT)* and
- *vector distance transforms (VDT)*.

Both categories work as a two stage process. Firstly, the *feature voxels* (or *surface of interest*), S, are *segmented* from the dataset, f, via a binary thresholding operation, τ (Eq.(2)).

$$S = \{(x,y,z) : f(x,y,z) \geq \tau\} \quad \text{where } x,y,z \in \mathbb{Z} \tag{2}$$

Eq.(2) returns an *object* surface of interest, in that, voxels which can be considered to be inside the surface are marked as being features. The actual surface interface can be segmented if the voxels which satisfy Eq.(2) are also checked to be transverse [9]. A voxel is transverse if at least one of its 26 bounding voxels is inside the surface and at least one is outside. Transverse voxels can be identified by comparing the bounding voxels to the threshold, τ, and setting a 26 bit flag, $b_{26}b_{25}b_{24}\ldots b_3b_2b_1$, as indicated in Eq.(3). Therefore, a voxel is transverse if its flag has a binary value between $000\ldots001$ and $111\ldots110$. Thus, non-transverse voxels are removed from S. Figure 2 gives an example of a transverse voxel, the central voxel is transverse.

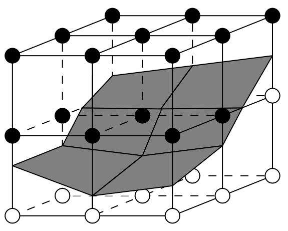
FIG. 2: A transverse voxel.

$$b_i = \begin{cases} 0 & \text{if voxel } i \leq \tau \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

2.1. Chamfer Distance Transforms

The second stage of a distance transform is the propagation of local distances throughout the field, which (for chamfer distance transforms) is initialised as shown in Eq.(4). If a signed field (voxels inside the object have negative distances) is required, the datasets must also be classified to indicate whether a voxel is internal, external or on the surface – Eq.(5). The signed distance field is thus the unsigned field multiplied by the corresponding classification value – Eq.(6). Local distance propagation is achieved with a number of passes of a *distance matrix*, d_{mat} – Eq.(7).

$$D(p) = \begin{cases} 0 & \text{if } p \in S, \text{ where } p \in \mathbb{Z}^3 \\ \infty & \text{otherwise} \end{cases} \tag{4}$$

$$C(p) = \begin{cases} -1 & \text{if } b(p) = 111\ldots111 \\ 0 & \text{if } p \in S \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

$$D(p)_{signed} = D(p) \times C(p) \tag{6}$$

$$D(x,y,z) = min\Big(\big(D(x+i,y+j,z+k) + d_{mat}(i,j,k)\big)\ \forall\, i,j,k \in d_{mat}\Big) \quad \text{where } x,y,z,i,j,k \in \mathbb{Z} \tag{7}$$

Chamfer distance transforms propagate local distance by addition of known neighbourhood values obtained from the distance matrix, d_{mat}, the basis of which is shown in Figure 3, where *a, b* and *c* represent the local distances.

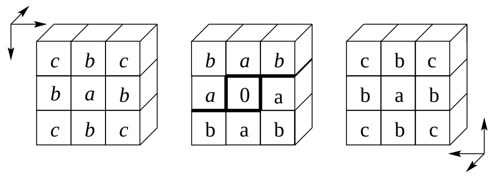

FIG. 3. Basis of the 3×3×3 distance matrix.

All CDTs employ two passes of the 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. In essence the forward past of the matrix is applied once for every voxel according to Eq.(7). In the second pass, the backward part of the matrix is applied similarly. Overall each voxel is considered twice, its calculation depending purely upon the addition of the elements of the matrix to its neighbours, and taking the minimum value. It is therefore computationally inexpensive.

There are numerous chamfer distance matrices, with varying error minimisation criteria, available in the literature. For example, Borgefors [4, 5] attempts to minimise the maximum difference, whereas Vossepoel [10] also minimises the root-mean-square difference. Marchand-Maillet and Sharaiha [11] measure the number of topological inconsistencies [12, 13] in the (discrete) distance fields. As this report introduces a new vector distance transform, only a summary of the more common matrices will be given here.

The simplest and least accurate of the CDTs is the *city-block CDT* [1] (Figure 4(a)), which uses only the orthogonal neighbours (a), which are unit distance away. Accuracy can be improved by including more neighbours – the *chess board CDT* (Figure 4(b)) and the *quasi-Euclidean 3×3×3 CDT* (Figure 4(c)) also use neighbours that are diagonal along two axes (a and b). Whereas, the *complete 3×3×3 CDT* (Figure 4(d)) uses all 26 local neighbours (a, b and c) and propagates realistic distances.

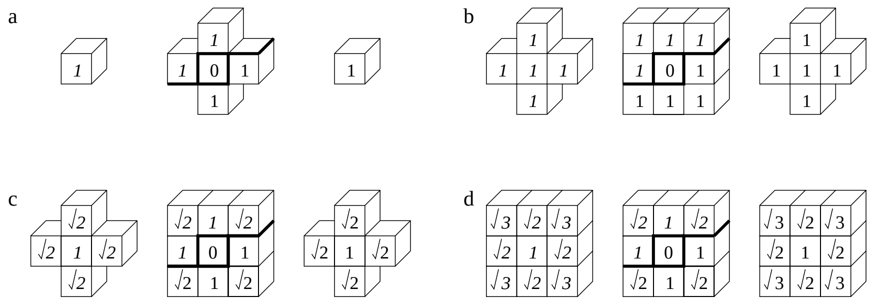

FIG. 4. (a) City-block, (b) chess board, (c) quasi-Euclidean and (d) complete 3×3×3.

The accuracy can be further improved by increasing the matrix size, Figure 5 gives the distance matrix for the *quasi-Euclidean 5×5×5 CDT*. Note that some matrix positions are unused, this is due to these positions being equal to a double step of a position closer to the centre of the matrix. This removes any unnecessary calculations.

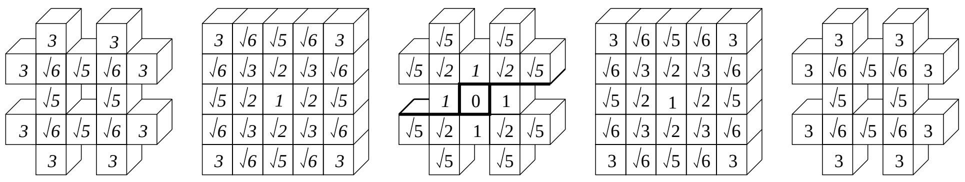

FIG. 5. Quasi-Euclidean 5×5×5 chamfer distance matrix.

Table 1 compares the (signed) distance fields, computed by the above CDTs for the skull of the UNC CThead, to the true Euclidean distance field. Emphasis is given to the average (absolute) error per voxel, although the error range of each distance field is also given. The comparison entails 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 subtraction and dividing the result by the number of voxels in the dataset (7,405,568 for the UNC CThead).

TABLE 1
Comparison of the chamfer distance transforms to the true Euclidean distance field.

Distance matrix	Execution time (s)	Distance range min	Distance range max	Error range min	Error range max	Average error per voxel
Euclidean	2.5+ hours	-4.243	109.595	0.000	0.000	0.000000
City-block	1.320	-5.000	184.000	-2.000	76.060	12.269071
Chess board	3.176	-4.000	92.000	-26.924	1.243	5.356728
Quasi-Euclidean 3^3	3.236	-4.414	130.108	-0.828	22.428	3.365570
Complete 3^3	4.842	-4.414	118.116	-0.415	11.769	2.196785
Quasi-Euclidean 5^3	20.438	-4.243	111.786	-0.363	5.149	0.612223

Note: The chess board CDT produces mainly negative errors as, unlike the other implemented CDTs, its distance range is smaller than the distance range of the true Euclidean distance field.

2.2. Vector Distance Transforms

Vector distance transforms (also known as Euclidean distance transforms (EDT)) differ from chamfer distance transforms in that vector components are propagated. Distances are calculated by evaluating the vector components once they have been propagated. Vector propagation is possible if the basic stages of a distance transform (Eq.(4) and Eq.(7)) are altered as shown in Eq.(8) and Eq.(9).

$$\vec{V}(p) = \begin{cases} (0,0,0) & \text{if } p \in S \\ (\infty,\infty,\infty) & \text{otherwise} \end{cases} \tag{8}$$

$$\vec{V}(x,y,z) = min\left(\left|(\vec{V}(x+i,y+j,z+k) + d_{mat}(i,j,k))\right| \forall i,j,k \in d_{mat}\right)$$
$$\text{where } x,y,z,i,j,k \in \mathbb{Z} \text{ and } d_{mat}(i,j,k) = (\vec{mat_x}, \vec{mat_y}, \vec{mat_z}) \tag{9}$$

Vector distance transforms generally require more passes of the distance matrix. During each pass the vector components are added to the necessary vector position, a decision is made as to whether any of the new vectors are minimal and, if so, the minimal vector is stored. The minimal vector is found by calculating the magnitude of the vectors (Eq(10)) and comparing the results. In the situation where two or more distances are minimal *(a tie)* no preference is made as to which vector is stored.

$$|\vec{V}(x,y,z)| = \sqrt{\vec{V_x}^2 + \vec{V_y}^2 + \vec{V_z}^2} \tag{10}$$

Vector distance transforms were first introduced (in two-dimensions) by Danielsson [2], through a description of the *sequential Euclidean distance mapping (SED)* algorithms – the 4SED and 8SED, where the numeral denotes the number of neighbours used in a 3×3 matrix . Implemented in a manner similar to chamfer distance transforms, the SED algorithms provide a 180° propagation angle, computing a distance field in four passes (Figure 6) – later refined to three passes by Ragnemalm [7].

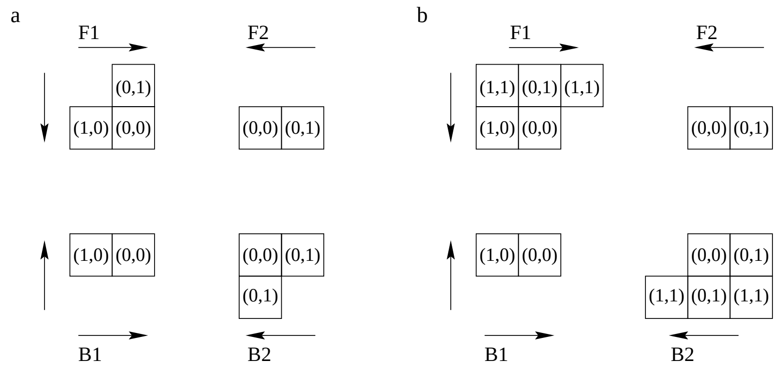

FIG. 6. (a) 4SED and (b) 8SED distance matrices.

The superiority of VDTs, over CDTs, was immediately apparent – Danielsson shows the 4SED (8SED) to have an *absolute error* of no more than 0.29 (0.09) pixel units. Improvements were made to the SED algorithms by Ye [14] and Leymarie and Levine [15], who extended the algorithms to include signed vectors (*signed sequential Euclidean distance mapping (SSED)*) and removed the need to perform multiplication and square root operations, respectively.

Mullikin [8] extended the 4SSED into three-dimensions, developing the *efficient vector distance transform (EVDT)* – the most accurate 3D VDT available in the literature, computing a distance field in six passes (Figure 7). Table 2 shows how using the EVDT compares to computing the true Euclidean distance field for the datasets listed in Table 3. (Execution times in Table 3 are for the brute force computation)

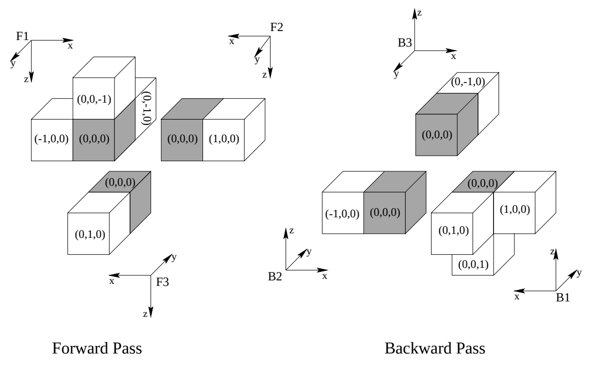

FIG. 7. The six passes of Mullikin's EVDT.

TABLE 2
Comparison of distance fields computed using the EVDT to the true Euclidean distance field.

Dataset	Execution time (s)	Distance range min	Distance range max	Error range min	Error range max	Average error per voxel
CThead skull	7.540	-4.242	109.595	-0.518	2.533	0.004761
Sphere	0.284	-27.677	39.268	-1.659	1.984	0.010045
Pawn	0.116	-7.000	37.670	-0.394	1.013	0.009697
Queen	0.116	-6.633	39.674	-0.381	0.928	0.008117
Rook	0.116	-9.000	34.496	-0.740	0.945	0.033425
Pawn and rook	0.236	-9.000	37.670	-0.740	1.013	0.044117

TABLE 3
Datasets used to test the distance transforms.

Dataset	Resolution	Number of feature voxels	Distance range min	Distance range max	Execution time (s)
CThead skull	$256 \times 256 \times 113$	183194	-4.242	109.595	2.5+ hours
Sphere	$80 \times 80 \times 80$	15425	-27.441	39.268	1212.410
Pawn	$60 \times 60 \times 60$	3340	-7.000	37.670	98.650
Queen	$60 \times 60 \times 60$	2357	-6.633	39.674	66.360
Rook	$60 \times 60 \times 60$	4050	-9.000	34.438	127.480
Pawn and rook	$120 \times 60 \times 60$	7390	-9.000	37.670	446.440

3. VECTOR-CITY VECTOR DISTANCE TRANSFORM

It has already been stated in Section 2.1, that the city-block CDT is the most elementary distance transform. This fact has led to the majority of VDTs being based, in some way, on the city-block CDT. In most cases the similarities stop at the neighbourhood used by the distance matrix (for example Danielsson's 4SED [2]), whereas others can include the two city-block passes in their own (for example passes *F1* and *B1* of Mullikin's EVDT (Figure 7)). Our new *vector-city vector distance transform (VCVDT)* goes one step further than this, in that the extra passes are those that would be used by the city-block CDT if it were extended to employ the same number of passes. (Increasing the number of passes made by a CDT has no effect on the final distance field.)

To reduce memory requirements the EVDT only maintains vectors for 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 progresses. For reasons which will be explained in Section 3.1, the VCVDT stores a complete vector copy of the distance field. Also the minimal distance is stored along with the minimum vector after each pass. This strategy allows Leymarie and Levine's [15] 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, that is, the vertical positions, are not included in subsequent distance calculations. Figure 8 illustrates the four passes employed by the VCVDT.

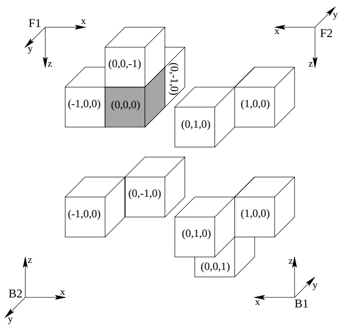

FIG. 8. Four pass vector-city vector distance transform.

3.1. The Effects of Using Different Vector Storage Methods

The previous section briefly compared the implementation of the vector-city VDT to the most accurate vector distance transform available in the literature (Mullikin's efficient VDT [8]), showing the major differences to be:

- *Different distance matrices*, and
- *Different methods of vector storage.*

The difference between the two distance matrices is obvious – the EVDT calculates the minimum distance six times per voxel, whereas the VCVDT only calculates the minimum distance four times per voxel.

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 has also been implemented. The distance field generated by each implementation is displayed (at an offset of 10 units) in Figure 9(b)–(d), with the difference between the computed distance field and the true Euclidean distance field (at this offset) shown in Figure 9(e)–(g). The difference images have been darkened for improved visibility.

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

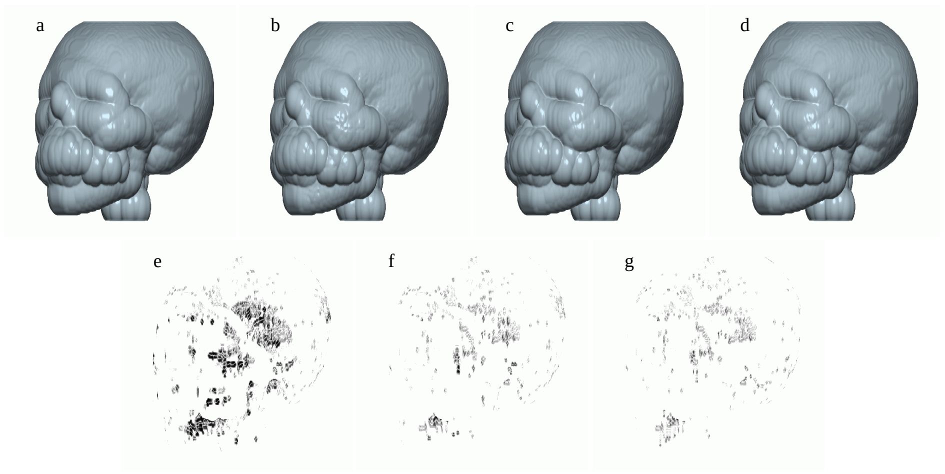

FIG. 9. Offset surfaces for (a) true Euclidean, (b) EVDT, (c) EVDT with complete vector grid and (d) VCVDT distance fields. Also difference between (e) *a* and *b* (f) *a* and *c* (g) *a* and *d*

(during the backward passes) as a result of the wrong feature voxel being selected as closest. This is due to important vector information (from the forward passes) being lost. Indeed, if the forward and backward passes are computed separately, the two implementations produce identical distance fields.

The average time taken, by each approach, to compute a distance field for the skull of the UNC CThead, is given (along with comparative information) in Table 4. Examination of this table shows that storing only two slices of vector information increases the computation time. This increase in computational time is caused by the extra management needed to ensure that the vector slices are organised correctly.

TABLE 4
Average execution times and a comparison of the EVDT, EVDT with full vector grid and the VCVDT to the true Euclidean distance.

Distance matrix	Execution time (s)	Distance range min	Distance range max	Error range min	Error range max	Average error per voxel
EVDT	7.540	-4.242	109.595	-0.518	2.533	0.004761
EVDT full grid	6.348	-4.242	109.595	-0.504	1.053	0.000786
VCVDT	3.420	-4.242	109.595	-0.334	0.446	0.000644

Mullikin originally proposed the two vector slice approach to save on memory requirements. Advances in technology have removed the need to limit resources in this way, thus allowing all vector distance transforms to be implemented with a complete vector grid. This and the fact that a full vector grid generates distance volumes, that are more accurate, in less time, has led to it being used by the VCVDT. Table 5 presents the results of the application of the VCVDT and EVDT, with full vector grid, distance transforms on the datasets listed in Table 3, clearly the VCVDT out performs the EVDT with full vector grid in all cases. Further-more, Tables 2 and 5 also show that the performance of distance transforms is only dependent on the size of the dataset.

TABLE 5
Results of applying the EVDT with full vector grid and the VCVDT to the datasets of Table 3.

Dataset	Execution time (s)		Average error per voxel	
	EVDT grid	VCVDT	EVDT grid	VCVDT
CThead skull	6.348	3.420	0.000786	0.000644
Sphere	0.424	0.204	0.005282	0.002734
Pawn	0.184	0.094	0.001683	0.001546
Queen	0.184	0.094	0.001415	0.001380
Rook	0.184	0.094	0.002552	0.002307
Pawn and rook	0.360	0.174	0.002118	0.001930

The accuracy of the VCVDT can be improved by increasing the number of passes made by the distance matrix from four to eight, as shown in Figure 10, producing a VDT resembling Ragnemalm's [7] *corner EDT*. Ragnemalm's corner EDT is briefly mentioned in [7] and very little information regarding computation, accuracy and performance is given. The results of applying the eight pass VCVDT to the datasets listed in Table 3 are given in Table 6, which again highlights the improvement the VCVDT technique provides in terms of computation time and accuracy.

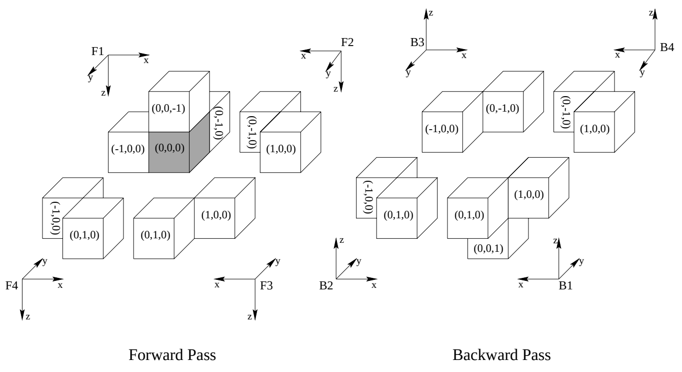

FIG. 10. Eight pass vector-city vector distance transform.

3.2. Error Analysis

In the reports on their respective algorithms Danielsson [2] and Mullikin [8] detailed one of the causes of errors in a distance field. The reported errors occur when the feature voxels are in a specific arrangement, preventing a propagation front from reaching a point within its own Voronoi tile, as demonstrated (in 2D) in Figure 11(a), where the dashed line represents the desired propagation and the solid lines represent the achievable propagation. That is, the feature voxels are arranged such that the vectors satisfy the inequalities of Equation 11 [8], see Figure 11(b) for an example.

TABLE 6

Results of applying the 8 pass VCVDT to the datasets of Table 3.

Dataset	Execution time (s)	Distance range min	Distance range max	Error range min	Error range max	Average error per voxel
CThead skull	5.470	-4.243	109.595	-0.334	0.334	0.000223
Sphere	0.302	-27.677	39.268	-0.268	0.268	0.000559
Pawn	0.134	-7.000	37.670	-0.310	0.278	0.000233
Queen	0.134	-6.633	39.674	-0.268	0.268	0.000179
Rook	0.134	-9.000	34.438	-0.334	0.268	0.000302
Pawn and rook	0.260	-9.000	37.670	-0.334	0.278	0.000268

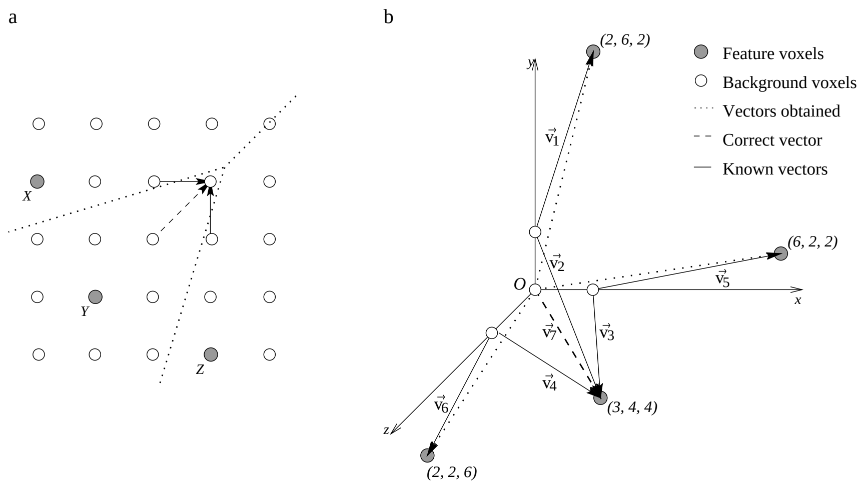

FIG. 11. Examples of (error causing) feature voxel arrangements, (a) 2D and (b) 3D.

$$\begin{aligned}
|\vec{v_1}| &< |\vec{v_2}| \\
|\vec{v_5}| &< |\vec{v_3}| \\
|\vec{v_6}| &< |\vec{v_4}| \\
|\vec{v_7}| &< |\vec{v_5} + (1,0,0)| \\
|\vec{v_7}| &< |\vec{v_1} + (0,1,0)| \\
|\vec{v_7}| &< |\vec{v_6} + (0,0,1)| \\
\text{where } \vec{v_7} &= \vec{v_2} + (0,1,0) = \vec{v_3} + (1,0,0) = \vec{v_4} + (0,0,1)
\end{aligned} \tag{11}$$

Arrangements of this kind cause local errors – only the voxel at the focal point of the arrangement (O in Figure 11(b)) and possibly a few of its neighbours are given incorrect distances. Cuisenaire and Macq [16] observe that these errors only occur at the corners of Voronoi tiles, and show how they may be detected and corrected if each voxel also indicates which feature point is its *nearest object pixel (NOP)*. Whilst analysing Mullikin's EVDT, a voxel arrangement was discovered which produces errors that are far more widespread. 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 12.

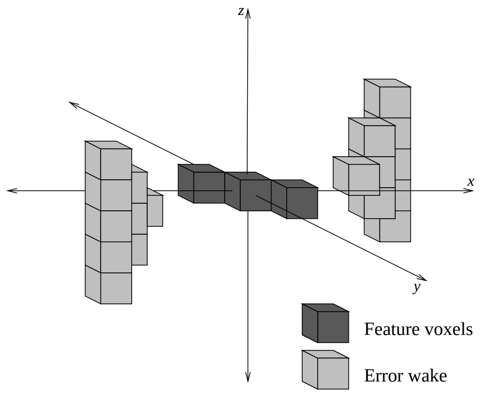

FIG. 12. Error wake caused by diagonal feature voxels.

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 field generated by the EVDT is erroneous.

Figure 13 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 feature voxels. Figures 14 and 15 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 does not alter the relationship between the feature and background voxels, whereas pass $B1$ of the VCVDT does. In Figures 7 and 8 it can be seen that these two passes are identical. The only explanation for the existence of the error wake is the loss of vector information when only using a limited number of vector slices.

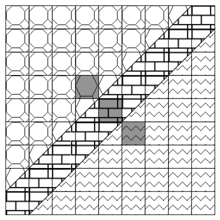

FIG. 13: Background to feature voxel relationship.

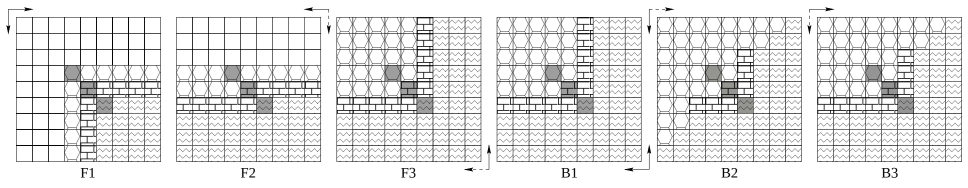

FIG. 14. Feature to background voxel relationship after each pass of the standard EVDT.

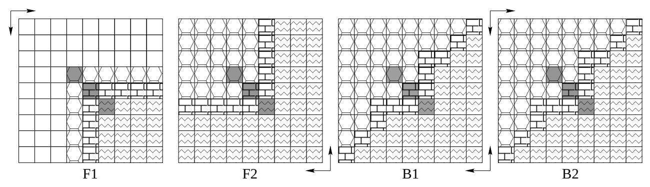

FIG. 15. Feature to background voxel relationship after each pass of the VCVDT.

Various methods for error reduction and correction have been detailed in the literature. For example, Ragnemalm [7] and Borgefors [5] increase the number of local neighbours, Cuisenaire and Macq [17] use larger matrices in the vicinity of possible errors and Mullikin [8] keeps track of tie and near tie vectors. All of these methods may be employed by the new VCVDT to even further improve accuracy.

4. SUB-VOXEL ACCURACY

Applications using distance transforms employ a segmentation algorithm in order to determine the zero distance surface from which the distance field is computed. This approach results in the segmented surface being an approximation to the correct surface, and therefore the resulting computed distance field being an approximation to the correct distance field. Figure 16 shows a rendering of the original UNC CThead before segmentation, and the resulting rendering after segmentation. This demonstration should highlight the fact that the binary segmentation process is unsatisfactory, and leads to large inaccuracies during subsequent computation (e.g. distance transforms).

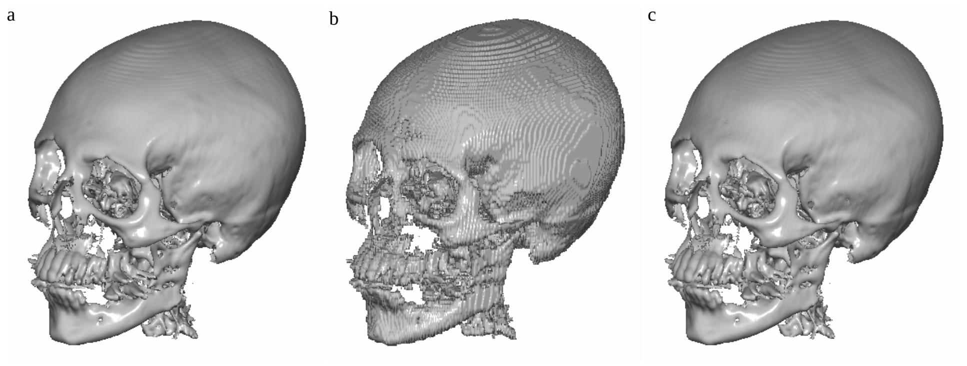

FIG. 16. UNC CThead rendered from: (a) original data set (b) distance field based upon binary segmented data set (c) distance field based upon new sub-voxel approach.

This section explores the method by which we produce sub-voxel accurate distance fields for both voxel (CT, MRI etc.) and polygonal mesh data.

Our new approach is to compute a shell of vectors about the surface of interest, where each vector indicates the distance of the closest point on the surface from the current position. For positions not in the shell we again assign a large value. Vectors are propagated to these positions using the 8VCVDT or some other DTs.

4.1. Volume Data Closest Point Calculation

For volume data such as CT or MRI data we must calculate the vector from each required voxel, to the closest point on the surface, S, of the object encoded within the data (Eq.(2)).

At each voxel $v = (x, y, z)$ we calculate the vector V as the vector which points to the closest point $s \in S$ from v.

To compute V we must calculate the minimum distance from v to the sub-voxel surface contained in every cell bounded by 8 voxels. This can be done by calculating a function to represent the local surface passing through a cell, but in practice this produces a highly complex solution (see Webber [18]). The approach employed here, is to calculate the triangulation of the cell and to measure the distance of the voxel to each triangle, and subtract the point of minimum distance from v to give the vector, V. The triangulation is created by dividing the cell up into tetrahedra, and producing either 1 or 2 triangles separating voxels inside the surface from voxels outside the surface. This method is used by Payne and Toga [19], and has the advantage that it does not suffer from the ambiguities that Marching Cubes [20] does. Each triangle has vertices determined by interpolation from the known voxel values, and therefore approximates local field changes with whichever interpolation method is decided upon.

This computation is quite complex, and can be accelerated by creating an octree [21] of the voxel values. Each node in the octree contains the maximum and minimum of the voxel values contained within that node. We also keep track of the current minimum distance discovered so far, and then each node, having a transverse voxel, which is within that distance is considered. The children of that node which are within the current minimum distance are considered recursively. Once we reach a leaf node, the triangulation mentioned above is carried out, and the minimum distance is updated before continuing the traversal of the octree. By using a neighbouring voxel's closet point as a candidate for a new voxel, we can start with a good approximation to the minimum distance, and therefore reject large portions of the octree. We can also reject portions of the octree that do not contain the surface threshold τ.

The above method of finding the closet point $s \in S$ from voxel v was used for all voxels when computing the brute force complete Euclidean distance field, and as mentioned took $2\frac{1}{2}$ hours.

4.2. Computing the required shell

To fully define the surface, the vectors need to be calculated just inside, and just outside the surface (otherwise the vector propagation will not produce an accurate representation of the surface). Therefore, the required shell, R is the set of voxels that are either just inside, on or just outside the surface. These are stored along with a flag indicating whether the point is inside or outside, which is used to determine the sign in the distance field.

For each voxel v, where $f(v) \geq \tau$, which has a neighbour n where $f(n) < \tau$ we add v and its 26-neighbours to R. This has the desired effect of creating a shell of voxels around the surface as stated in the previous paragraph. For each $v \in R$ we calculate the vector V as in Section 4.1, and store this in our vector grid. For all other voxels we store a large vector. The 8VCVDT is then used to propagate the vectors, from which the distance field is then computed. This vastly reduces the number of voxels to be computed using the method of Section 4.1, the remaining being computed in the 8VCVDT passes. This reduces the overall run-time to produce a distance field from $2\frac{1}{2}$ hours to less than 5 minutes. Figure 16(c) and 17 show renderings from this distance field.

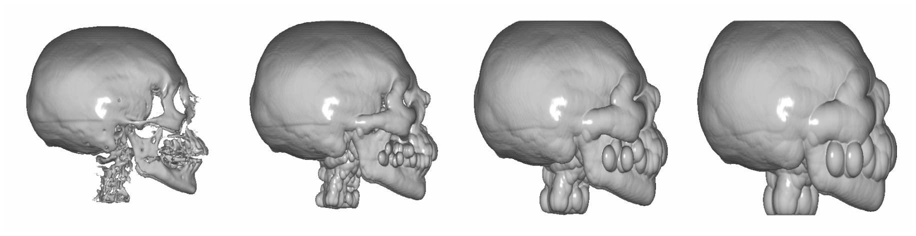

FIG. 17. Iso-surfaces from the CThead distance field

Table 7 shows the computational times for calculating the complete distance field from the shell distance field using various chamfer and vector methods.

TABLE 7
Comparison of the binary segmented and sub-voxel classified distance fields to the sub-voxel accurate Euclidean distance field.

Distance matrix	Execution time (s)	Error range min	Error range max	Average error per voxel
True Euclidean	2.5+ hours	0.000	0.000	0.000000
City-block (binary)	1.320	-2.000	76.230	12.519548
City-block (shell)	1.290 + 124	-2.377	73.187	10.775360
EVDT (binary)	7.540	-1.732	3.081	0.261987
EVDT (shell)	7.200 + 124	-1.873	1.393	0.015674
4VCVDT (binary)	3.420	-1.732	1.739	0.257894
4VCVDT (shell)	4.534 + 124	-1.873	1.393	0.013266
8VCVDT (binary)	5.470	-1.732	1.729	0.257474
8VCVDT (shell)	8.648 + 124	-1.986	1.393	0.012215

4.3. General Polygonal Meshes

The sub-voxel accurate approach of the previous sections can also be applied to polygonal mesh objects as a process known as voxelisation [22]. Once the resolution of the resulting data set has been decided, the grid is sampled at regular points to determine if the voxel is inside the surface. The shell of voxels for which vectors must be computed explicitly is calculated using the method of section 4.2. Then for each voxel $v \in R$ we calculate the closest point on the triangular mesh from v. We use the vector to that point to create our shell vector field (which are then propagated by applying 8VCVDT). Once again an octree could be used to accelerate the process of finding the closest triangle, although we use a slab based approach which matches the number of slices in our voxel grid. A voxelised chess piece is shown in Figure 18, along with several distance field iso-surfaces. The full distance field takes 1050 seconds to compute, whereas calculating the shell of vectors and propagating them to create the fields takes a combined time of 12 seconds.

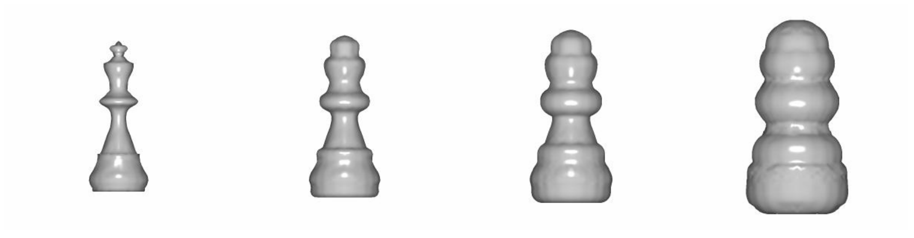

FIG. 18. Iso-surfaces from the chess piece distance field

5. CONCLUSION

In this paper we have demonstrated several improvements to DTs. Firstly it has been shown that storing a complete vector representation of a distance field rather than a limited number of vector slices, both reduces the average execution time and increases the accuracy of vector distance transforms. This can be seen in the comparison between Mullikin's EVDT [8] and the previously unpublished EVDT with full vector grid – two VDTs that only differ in the number of vector slices stored (two slices and a complete vector grid).

Next a new, faster and more accurate VDT has been introduced, the *vector-city vector distance transform (VCVDT)* – a four pass VDT modelled on the city-block CDT. After a detailed error analysis, four extra passes were added to the VCVDT, again improving accuracy.

Error analysis was carried out on real datasets, allowing a thorough test for each DT on a large number of possible error situations. Testing accuracy in this way revealed a voxel arrangement which causes errors far worse than any previously published. A diagonal line of three or more voxels in the z-plane causes a wake like error propagation. The use of a complete vector grid removes this error wake.

Finally we have shown how voxelisation may be combined with distance transforms in order to produce distance fields, from field data and triangular mesh objects, with sub-voxel accuracy, which achieves a far more accurate distance field than previous methods that have employed binary segmentation.

ACKNOWLEDGMENT

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

REFERENCES

1. A. Rosenfeld and J. L. Pfaltz. Sequential operations in digital picture processing. *Journal of the ACM*, 13(4):471–494, 1966.
2. P-E. Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14:227–248, 1980.
3. G. Borgefors. Distance transformations in arbitrary dimensions. *Computer Vision, Graphics and Image Processing*, 27(3):321–345, 1984.
4. G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing*, 34(3):344–371, 1986.

5. G. Borgefors. On digital distance transforms in three dimensions. *Computer Vision and Image Understanding*, 64(3):368–376, 1996.
6. I. Ragnemalm. Neighbourhoods for distance transformations using ordered propagation. *CVGIP: Image Understanding*, 56(3):933–409, November 1992.
7. I. Ragnemalm. The Euclidean distance transform in arbitrary dimensions. *Pattern Recognition Letters*, 14(11):883–888, 1993.
8. J. C. Mullikin. The vector distance transform in two and three dimensions. *CVGIP: Graphical Models and Image Processing*, 54(6):526–535, 1992.
9. M. W. Jones. *The Visualisation of Regular Three Dimensionl Data*. PhD thesis, University of Wales, Swansea, 1995.
10. A. M. Vossepoel. A note on distance transfromations in digital images. *Computer Vision, Graphics and Image Processing*, 43(1):88–97, 1988.
11. S. Marchand-Maillet and Y. M. Sharaiha. Euclidean ordering via chamfer distance calculations. *Computer Vision and Image Processing*, 73(3):404–413, 1999.
12. S. Forchhammer. *Representation and Data Compression of Two-Dimensional Graphical Data*. PhD thesis, Technical University of Denmark, 1988.
13. S. Forchhammer. Eculidean distances from chamfer distances for limited distances. In *Sixth Scandinavian Conference on Image Analysis*, 1989.
14. Q. Z. Ye. The signed Euclidean distance transform and its applications. In *Proceedings, 9th International Conference on Pattern Recognition*, pages 495–499, 1988.
15. 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.
16. O. Cuisenaire and B. Macq. Fast and exact signed Euclidean distance transformation with linear complexity. In *IEEE International Conference on Acoustics and Signal Processing*, 1999.
17. O. Cuisenaire and B. Macq. Fast Euclidean distance transformation by propagation using multiple neighborhoods. *Computer Vision and Image Understanding*, 76(2):163–172, 1999.
18. R. E. Webber. Ray tracing voxel data via biquadratic local surface interpolation. *The Visual Computer*, 6(1):8–15, February 1990.
19. B. A. Payne and A. W. Toga. Distance field manipulation of surface models. *IEEE Computer Graphics and Applications*, 12(1):65–71, 1992.
20. W. E. Lorensen and H. E. Cline. Marching Cubes: A high resolution 3D surface construction algorithm. *Computer Graphics (Proceedings of SIGGRAPH 87)*, 21(4):163–169, July 1987.
21. J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.
22. M. W. Jones. The production of volume data from triangular meshes using voxelisation. *Computer Graphics Forum*, 15(5):311–318, December 1996.