Shape Representation Using Space Filled Sub-Voxel Distance Fields

M. W. Jones, R. A. Satherley
Department of Computer Science
University of Wales, Swansea
Singleton Park, Swansea
United Kingdom
{m.w.jones, csrich}@swan.ac.uk

Abstract

Voxelisation is the process of converting a source object of any data type into a three-dimensional grid of voxel values. This voxel grid should represent the original object as closely as possible, although some inaccuracies will occur due to the discrete nature of the voxel grid representation. In this paper we report our on-going research into methods for representing objects as voxelised distance fields, in particular we report fast methods for accurate distance field production. A review of current alternative voxelisation methods is also given.

1 Introduction

The work presented in this paper is closely related to the field of Volume Graphics [2]. 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 raster images and vector graphics [16]. The need for the process of voxelisation is a direct result of the employment of volume graphics techniques, which in turn offer the benefits of *scalable rendering* algorithms for extremely large scenes, *visual effects* such as fire, fur and roughness (Section 4.1) and *consistency of rendering* (the ability to render objects of different source types under the same conditions). Indeed, as mentioned later in this paper, many rendering effects are only available during volume rendering, and therefore employing voxelisation techniques, enables these effects to be carried out upon the voxelised objects.

In this paper we shall introduce distance fields in section 2 and examine alternative methods for voxelisation in section 3. Section 4 will present *Space Filled Distance Fields* in detail. Binary segmentation and sub-voxel segmentation will be examined, and several methods for calculating the complete distance field will be demonstrated. Full details of the process for representing objects as distance fields will be given. Results will show how effective these methods are, including images and an animation (available via WWW site). Further work and conclusions are given in sections 6 and 7.

2 Background

We have previously used distance fields as an intermediate step to create triangular meshes from contour data [13]. 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. We discovered early on that the normals calculated from these distance fields (using trilinear interpolation within voxel cells) do not become quantised as they can with other methods, and that solid objects (obtained from contours or irregular points) can be represented quite accurately using this method. We have also used distance fields to voxelise triangular mesh objects [11]. That previous work described the production of a shell distance field (also briefly reviewed in this paper), and an efficient algorithm for measuring the distance of a point from a triangle (which is somewhat more difficult to achieve efficiently than one would assume). The work presented here extends this previous work by generating space-filled distance fields efficiently, which as mentioned later in this paper, have many more uses.

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

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

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 isosurface 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.

This signed distance field has proven to be of use for many years, and has become widespread for many applications within the field of graphics in the last few years, although its use for modelling objects has not been fully explored. Figure 1 demonstrates the voxelisation of a Rook chess piece (original object is a triangular mesh). Representing the object as a distance field compares favourably with the other representation methods (Section 3).

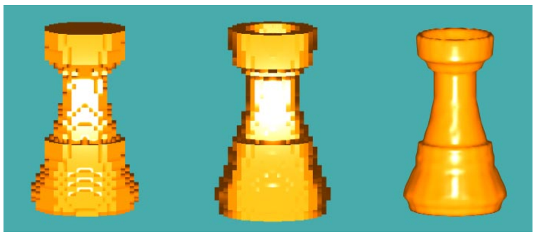

Figure 1. Rendering of rook voxelised using (a) 1 regular sample, (b) $7 \times 7 \times 7$ regular samples (Section 3.1), (c) distance shell

The main drawback of the distance field method is that it is computationally expensive when a complete space filled distance field is calculated. For the rook chess piece, the distance at each voxel (216,000) is evaluated to the closest point on a mesh of about 1500 triangles (taking about 6 minutes). This takes into account very efficient methods for calculating the distance of a point to a triangle [10], and using an octree as mentioned in Section 4.4. For another example it takes approximately 8 hours to convert the UNC CThead (7 million voxels) into a space filled distance field (also using acceleration methods such as an octree).

This method, such as it stands, is too computationally expensive to employ, and therefore we have researched methods of acceleration which we present in this paper, after examining alternative methods for voxelisation.

3 Alternative Methods For Voxelisation

3.1 Sampling

The basic method for converting objects is spatial occupancy. A regular grid of voxels is placed over the domain of the source object, and for each voxel a binary decision is made as to whether each voxel is inside or on, or outside the object [15, 6]. Each data type has been given a different treatment in the literature which takes advantages of the source data type to increase the speed of conversion. The recognised problem with this method is that the representation produces a discrete surface which is recognisably blocky. Simple solutions have been to increase the resolution of the data set, but this in turn increases the memory requirements and rendering times. It was identified early on that the primary cause of the visible artifacts is the reconstruction of the surface normal from the voxelised data. A thorough examination of the normals by Höhne et al. [26], indicated that whilst it is possible to use various sampling strategies to overcome this problem, it is preferable to have access to grey-scale data, such as that from CT scan data. In such datasets we no longer have a binary classification – each voxel takes a value depending upon its density.

One method by which this can be achieved is by taking more samples per voxel than just the above mentioned single central sample. With oversampling we consider each cube and sample at several points within the cube. If all points fall inside the object to be voxelised we can consider the corresponding voxel to take on the maximum value. Conversely if all points are outside, it can take on a minimum value. Values can be generated between the two values corresponding to the number of samples inside the object, and the number of bits available to represent the object. Obviously an increase in sample points results an increase in accuracy, but at a trade-off in an increase of computation. We must also consider storage. For example we could have 2^n samples and store voxel data using n bits per voxel. In practise 8 bits appears to be a good compromise and can produce images displaying consistent normals.

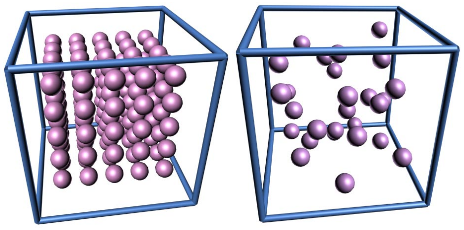

Figure 2. Regular $5{\times}5{\times}5$ sampling, and $3{\times}3{\times}3$ jittering.

Sampling theory [28] suggests that a stochastic sampling should provide better images as the high frequencies will be dispersed. Poisson sampling using a minimum distance constraint (Poisson disk), such that no two points are closer than a certain distance produces a good distribution for sampling but is expensive to calculate. In this instance the computational expense for calculating a Poisson disk for one cube and then using the same sampling pattern throughout the data set would be insignificant compared to the other operations. A simpler, but not much less effective method, is that of jittering (Figure 2). Each sample is randomly perturbed from its centre on a regular grid, but is still located within its grid cell. This is simple to calculate (and implement) as it just involves a random shift of the sample point.

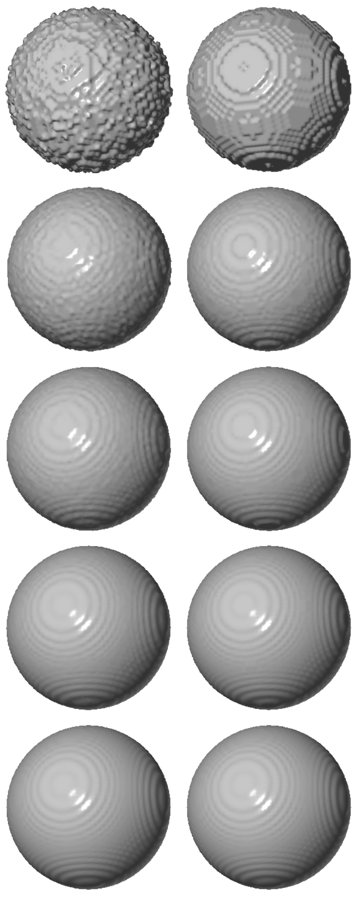

Figure 3. Rendering of spheres using jittered (left column) and regular sampling (right column). Samples per voxel (from top) 1, 8, 27, 125, 9261.

Figure 3 shows an implementation of regular and jitter sampling for the case of a sphere, and several samples per voxel on a sampling grid of 60^3. Table 1 gives the running times for each one (all timings are on an 800MHz Athlon). The artifacts apparent in the image are due to the grey-level central difference method being used for the calculation of the normal. Only a 1 voxel layer encodes the surface – all other voxels are either completely inside the surface or completely outside, and therefore there is not enough encoded information to reconstruct the normal accurately using this method. It is acknowledged that other methods exist for this [26], but as it was not the main thrust of this work, they have been omitted.

Samples per voxel	time (secs.)
$1 \times 1 \times 1$	0.02
$2 \times 2 \times 2$	0.13
$3 \times 3 \times 3$	0.45
$5 \times 5 \times 5$	2.12
$21 \times 21 \times 21$	222.31

Table 1. Computational time for various numbers of samples per voxel ($60 \times 60 \times 60$ voxels)

We can observe that sampling within one slice of a grid of voxel values is similar to rendering the mesh to a pixel image. Therefore we can accelerate the voxelisation process by using hardware renderers to render the triangular mesh to an image. The image can then be processed to determine if each pixel was inside or outside the object. Fang [6] has covered this process for point sampling, but this work could be easily extended to oversampling by simply increasing the size of the image, and decreasing the stepping size through the volume. This is fine in the case of regular sampling, but jittering presents problems using this method.

Another method by which a grey-level dataset can be computed is by using a different sampling technique. Wang and Kaufman [27] volume sample primitives using a spherical volume set at 3 units. Calculating this for arbitrary objects (e.g. triangular meshes) is difficult, so they have specific functions to volume sample each kind of primitive (sphere, cone etc.). The method produces values in the vicinity of the object, and therefore the volume is not space filled. This has advantages in terms of storage, but disadvantages for applications mentioned in Section 4.1. Šrámek and Kaufman [25] identify that a linear density profile in the vicinity of the surface give the best results. They produce such a profile using convolution with a box filter. Again this produces values in the vicinity of the object (whereas this paper is primarily concerned with space filled voxelisations). Their method may benefit from the improvements made in this paper (as will become clear later). They suggest a method for voxelising surface represented objects in which the surface is voxelised using a field, such that a thin solid is produced during visualisation – e.g. a plane would become a box. Although the methods of Section 3.3 onwards are discussed in the context of solid modelling, they could be applied to surface representations in a similar manner (i.e. by creating slightly thick objects in place of infinitely thin surfaces).

3.2 Implicit Functions

Sampling is a valid method for producing voxelised objects, but it suffers from resolution problems (blocky images), and restricted orientation of normals. One class of object for which this volume sampling is not a problem is that of implicit functions [14, 8]. For example rendering a sphere on a grid of 20^3 can give the top left image in Figure 4. The grey level normals are calculated using trilinear interpolation from the 8 normals creating a cube containing the intersection point with the ray. As can be seen these normals give a high accuracy for shading. The voxel grid is calculated by evaluating the function of the implicit surface – in this case $f(x, y, z) = x^2 + y^2 + z^2 - r^2$. Various other implicit functions have been voxelised and rendered (also Figure 4), including some which have been genetically bred [12]. In all cases the normals from such data produce naturally shaded objects. All images in this paper have been rendered using direct surface rendering [14]. Most images in Figures 4–6 take less than a second to render at 300×300 pixels (800Mhz Athlon).

The implicit functions encoded as above can be considered as *space filled* voxelisation – there is a value at every point in the domain of the object. The sampled voxelisations only produce values in the vicinity of the object and can be considered to be *shell* voxelisations. In the next sections we shall examine distance fields, both in their space filled and shell incarnations.

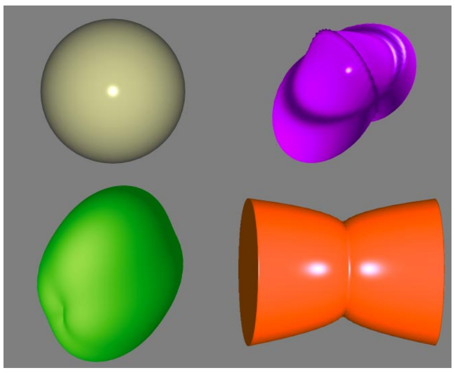

Figure 4. Rendering of sphere and other implicit functions.

Samples per voxel	time (secs.)
$1 \times 1 \times 1$	0.04
$7 \times 7 \times 7$	6.30
Distance shell	7.10
Full distance	230.00

Table 2. Computational time for voxelising rook ($60 \times 60 \times 60$ voxels)

3.3 Distance Shells [11]

If we are just interested in encoding the surface and do not need a space filled distance field, we can voxelise the object just in the vicinity of the surface. We have called such a voxelisation a *distance shell* – the computational expense is significantly less, and the surface representation is far superior when compared to the oversampling method which takes an equivalent time (Table 2 and Figure 1).

The shell for which the distance must be calculated is given by the set of voxels S_n. First we define a segmentation function f as:

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

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 $p \in v_{26}$.

Calculating the distance for these voxels is enough to encode the surface – the uncalculated voxels are either inside the surface, and all their neighbours are inside the surface, or outside the surface with all their neighbours outside the surface. Using this shell to render the encoded surface results in voxels outside this shell S_v being used during normal calculation when central differences are calculated. To include all of these additional voxels 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).

Frisken et al. [7] have presented an intermediate representation where objects are adaptively sampled – i.e. more samples are taken in the vicinity of the surface, and less further away from the surface. Samples are stored in an octree structure to reduce the amount of storage required (over using a uniform grid). This representation will reduce calculation time (as less samples are made than a full distance field), but will result in less accuracy away from the surface. Essentially their method is the same as a distance shell, with the added benefit of less accurate distances available away from the surface, rather than none at all as in the case of the distance shell.

4 Space Filled Distance Fields

4.1 Overview

We have already demonstrated a simplistic approach to calculating space-filled distance fields. Such distance fields offer a powerful method by which objects can be represented, manipulated and rendered. Animators are interested in them for collision detection and morphing purposes and vision and image understanding researchers use them for skeletonisation, thickening, thinning, shape interpolation and general distance calculations. An example of a space-filled distance field for the rook is shown in Figure 5. At a distance of zero we have the original surface, and for various offsets we obtain the other surfaces as depicted.

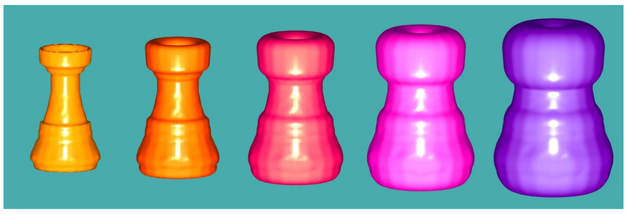

Figure 5. Rendering of rook space filled distance field at several different offsets.

It is now being appreciated that these distance fields have many uses, such as:

- Skeletonisation – Danielsson [5], Paglieroni [19], Zhou *et al.* [31] and Zhou and Toga [32] use distance information to extract the skeletal representation of an object.
- Machine vision applications (Paglieroni [19]) – Including, thickening, thinning, correlation, and convergence.
- General distance calculations – e.g. the calculation of distance to objects or contours and the production of Voronoi tessellations.
- Shape-based interpolation (Levin [17] and Herman *et al.* [9]) – Intermediate slices in scanned data can be interpolated from distance information. Interpolation from the original data causes abrupt changes at boundary locations.
- Distance field manipulation [20] – Distance information is added to surface models to allow their manipulation. Such manipulations include: interpolation between two surface models, offset surfaces, blending of two surface, and surface blurring.
- Volume morphing – A morph between two distance volumes can be easily created with the use of simple linear interpolation [1], Figure 6 gives some images from such a morph. Cohen-Or et al. [4] use distance fields along with warp functions to create a morph between two general topological objects.

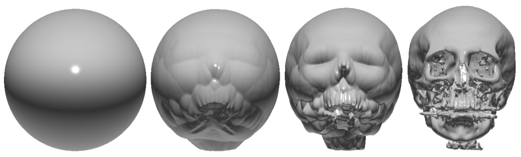

Figure 6. Simple morph between a sphere and a CT skull.

Fig. 7 demonstrates the application of hypertexture to distance fields of triangular mesh objects and the UNC CThead.

- Accelerating ray tracing – Yagel and Shi [30], Cohen and Sheffer [3], and Semwal and Kvarnstrom [24] all use CDT (see later) distance information to accelerate ray tracing. The general principal behind each method is to use the distance information to skip over large empty spaces.
- Hypertextures (Satherley and Jones [23]) – Non-geometrically definable volume datasets, such as CT scans, can be converted to distance fields, allowing the application of Perlin and Hoffert's hypertexture [21] effects, Figure 7.

All of the applications listed above require and use a space-filled distance field. As it is so costly to compute accurately, a faster (very inaccurate) propagation method known as a *distance transform* (DT) is used upon the binary segmentation of the underlying object.

We will describe this basic method, and place it into context with other methods in our following classification. As many computer graphics researchers are using the simple model, the classification serves not only to demonstrate our own methods, but also to bring other methods to a wider audience.

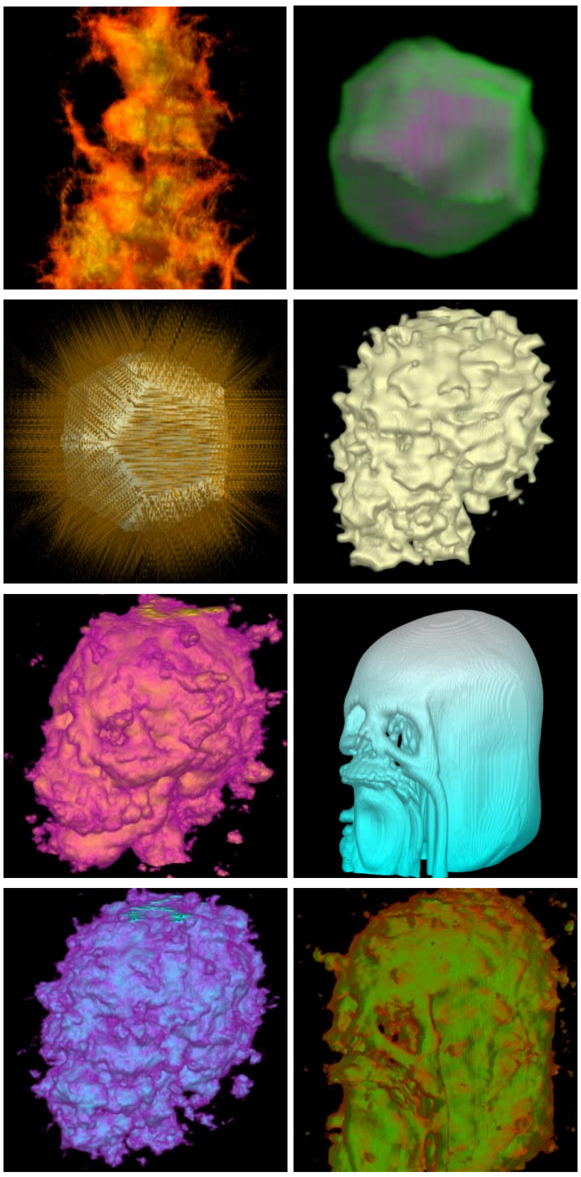

Figure 7. Hypertexture applied to the distance fields for a chess piece, a dodecahedron and the UNC CThead dataset.

4.2 Classification

In this section we will classify the different available types of distance fields and indicate computational time and accuracy. As a rough guide Figure 8 demonstrates the main classification with the faster less accurate computation towards the left. For the purposes of this paper we will report our research on the conversion of the UNC CThead into a distance field. The methods reported have also been applied to triangular mesh data (Figure 5).

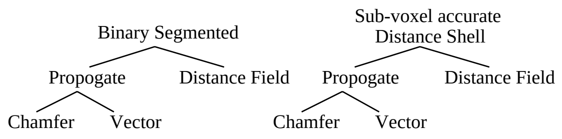

Figure 8. Classification (less accurate, but faster towards left).

4.3 Binary Segmentation

Almost all previous research has employed a binary segmentation (equation 2) of the surface to be encoded as this is easier to compute (than the method given in Section 4.4). After binary segmentation using equation 2, the next stage is to construct an initial distance field, D using equation 3:

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

4.3.1 Distance Field

At this stage we could create a space filled distance field by carrying out a full distance field calculation:

```
FOR each voxel v
   min=infinity
   FOR each voxel c
      IF D(c)=0 THEN
         dist=|c-v|
         min=min(dist,min)
```

On an 800MHz Athlon this algorithm takes approximately 2 weeks with no acceleration (such as the use of octrees). Obviously such a method is too time consuming and therefore propagation methods (or distance transforms) have been employed.

4.3.2 Chamfer Distance Transform

The simplest method is the chamfer method: 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.

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

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 9). Each value in the matrix represents the local distance value. This matrix is applied in two passes (and not recursively propagating one distance at a time as some authors have reported in previous work).

```
/* 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 $\geq$ 0; z--)
  FOR(y = $f_y$-1; y $\geq$ 0; y--)
    FOR(x = $f_x$-1; x $\geq$ 0; x--)
      $D$(x,y,z) = Eq.4
```

	3		*3*	
3	$\sqrt{6}$	$\sqrt{5}$	$\sqrt{6}$	*3*
	$\sqrt{5}$		$\sqrt{5}$	
3	$\sqrt{6}$	$\sqrt{5}$	$\sqrt{6}$	*3*
	3		*3*	

3	$\sqrt{6}$	$\sqrt{5}$	$\sqrt{6}$	*3*
$\sqrt{6}$	$\sqrt{3}$	$\sqrt{2}$	$\sqrt{3}$	$\sqrt{6}$
$\sqrt{5}$	$\sqrt{2}$	*1*	$\sqrt{2}$	$\sqrt{5}$
$\sqrt{6}$	$\sqrt{3}$	$\sqrt{2}$	$\sqrt{3}$	$\sqrt{6}$
3	$\sqrt{6}$	$\sqrt{5}$	$\sqrt{6}$	*3*

	$\sqrt{5}$		$\sqrt{5}$	
$\sqrt{5}$	$\sqrt{2}$	*1*	$\sqrt{2}$	$\sqrt{5}$
	1	0	1	
$\sqrt{5}$	$\sqrt{2}$	1	$\sqrt{2}$	$\sqrt{5}$
	$\sqrt{5}$		$\sqrt{5}$	

3	$\sqrt{6}$	$\sqrt{5}$	$\sqrt{6}$	3
$\sqrt{6}$	$\sqrt{3}$	$\sqrt{2}$	$\sqrt{3}$	$\sqrt{6}$
$\sqrt{5}$	$\sqrt{2}$	1	$\sqrt{2}$	$\sqrt{5}$
$\sqrt{6}$	$\sqrt{3}$	$\sqrt{2}$	$\sqrt{3}$	$\sqrt{6}$
3	$\sqrt{6}$	$\sqrt{5}$	$\sqrt{6}$	3

	3		3	
3	$\sqrt{6}$	$\sqrt{5}$	$\sqrt{6}$	3
	$\sqrt{5}$		$\sqrt{5}$	
3	$\sqrt{6}$	$\sqrt{5}$	$\sqrt{6}$	3
	3		3	

Figure 9. 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.

This method of distance generation is very inaccurate, and hence all of the applications that rely on the method are using inaccurate data. We required 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.

4.3.3 Vector Distance Transforms (VDTs)

Vector methods [18] 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. Previous vector propagation methods operated on a binary segmented data set (as in Equation 3), and so are attempting to produce a distance field from an object encoded as Figure 1(a).

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 Eqs. 3 and 4 are modified as shown in Eqs. 5 and 6 respectively.

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

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

Figure 10 shows the matrix passes employed by our Vector City VDT.

4.4 Sub-Voxel Accurate Distance Field

The previous sections dealt with the left hand side of our classification (Figure 8) and produce distance fields which poorly approximate the underlying object. Our main contribution to this area is two-fold – firstly we use our distance shell S_v of Section 3.3 (Figure 1(c)) as the starting point (although vectors to the closest surface point, rather than the distance to that point are stored), and secondly, we propagate distances using a new transform – the Vector City Vector Distance Transform (VCVDT). Complete details and analysis of the method and its comparison to previous methods can be found in our report [22] (available on http://www-compsci.swan.ac.uk/∼csmark/voxelisation/). This paper differs from the report in the fact that it presents the VCVDT method as being applied to sub-voxel accurate distances, contains information about using the method for modelling objects and is intended to present the results to a different audience.

Our method for modelling objects is to first calculate vectors for each $v \in S_v$ (as defined in Section 3.3) to the closest point on the surface. For *triangular mesh objects*, the closest point on the triangle is used. Computational time can be improved by using an octree to organise the object – parts of the octree outside the current closest distance can be ignored (thus reducing the number of triangles to be considered for each voxel). A new voxel can use its neighbours closest point as an initial starting point (so that large amounts of the octree are ignored).

For CT data (or any other non-Euclidean grey-level data), 8 neighbouring voxels are considered to make a brick cell. A cell is transverse if at least one voxel is inside the surface and at least one voxel is outside. For each $v \in S_v$

the closest transverse cells are stored in a list (again an octree and neighbour information is used to speed computation). Next, each transverse cube in the list is divided into tetrahedra, and then the closest point is calculated as the closest point on the triangular tiling of the tetrahedra [20]. This avoids the ambiguous cases present in marching cubes. The list of cells contains all cells that could contain the closest point – i.e. the furthest point in the closest cell is further away than the closest point in the furthest cell.

The next stage uses this shell of voxels S_v about the surface for which vectors to the closest point on the surface are known, and all other voxels in the domain are initialised to a large value. We consider this voxel grid to be $\vec{V(p)} \in \mathbb{R}^3$ where $p \in \mathbb{Z}^3$. The vectors are now propagated (using the VDT method) throughout the volume so that equation 6 is true (D is the final distance field).

The matrix passes 8 times through the data set as shown in Figure 10 (our VCVDT), and d_M is defined as the vector values in Figure 10 (for further details see our report [22]). As an example though, the first forward pass F1 is applied to each voxel in the direction of increasing x, y and z. The new vector at the voxel is the minimum of itself, its negative y neighbour with -1 added to the y component of its vector, and similar for its negative x and z neighbours. This small example may seem to indicate no account is taken of whether we are moving away or towards the surface. In fact later passes (and the minimum operator) ensure that account is taken. It may also make obvious the fact that the method may not give the correct closest voxel. This is true – the method is an approximation, but as we shall see, it is about 20 times more accurate than previous approximate methods. Figure 11 shows how the inaccuracies are introduced by the method. The final distance is set to negative if $f(v) = 1$, or positive if $f(v) = 0$.

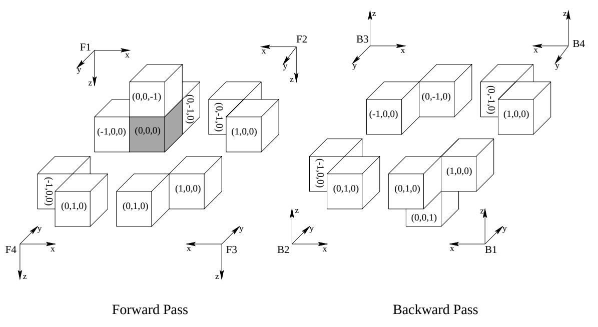

Figure 10. Eight pass vector-city vector distance transform.

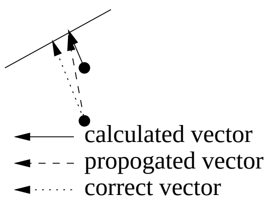

Figure 11. Incorrect vector propagation.

5 Results

Figure 12 shows several offset surfaces from the CThead rendered from the distance field produced from the distance shell S_v using the above VCVDT vector transform. A full animation can be found at http://www-compsci.swan.ac.uk/∼csmark/voxelisation/. The CThead distance shell (i.e. accurately measured sub-voxel distances to the skull for all $v \in S_v$) takes 240 secs. Table 3 shows the additional time required to propagate these vectors (and calculating the final distances), using our new VCVDT method, the current best method EVDT [18], and the CDT method, and compares this to the true calculation. It can be seen that for just over 4 minutes it is possible to compute the full distance field to a good accuracy, rather than resorting to the 8 hour computation. Propagating the shell vectors for the chess piece to create a complete distance field takes less than 2 seconds, and in fact Figure 5 was rendered from the propagated rook distance field. We found that the field was accurate enough for our purposes (hypertexture – Figure 7), and the improved accuracy over the CDT method (currently used by most researchers requiring distances), would have advantages for all the applications mentioned earlier. Figure 13 shows a volume graphics scene composed of distance field encoded chess pieces and rendered using the volume rendering library – VLib [29].

6 Further Work

We are currently examining a hybrid technique using a vector transform to direct the correct sub-voxel accurate calculation. We anticipate that this will fall between the correct true calculation and transformed distance shell interms of time. The distance field may be 100% accurate, although this depends upon keeping a large list of candidates (Section 4.4, paragraph 3). We are also investigating other application areas not already identified in the literature.

7 Conclusions

We have introduced distance fields as a modelling paradigm and in particular introduced the notions of shell distance fields and space-filled distance fields. Figure 1

Distance method	Time	Error range min-max	Avg error per voxel
True Euclidean	8 hrs	0.000–0.000	0.000000
VCVDT	5.470s	-0.334–0.334	0.000223
EVDT (previous best)	7.540s	-0.518–2.533	0.004761
CDT ($3 \times 3 \times 3$)	4.842s	-0.415–11.769	2.196785
CDT (City Block–most common)	1.320	-2.00–76.060	12.269071

Table 3. Comparison of VCVDT with other methods

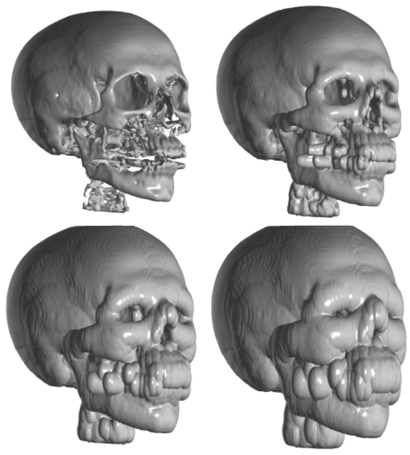

Figure 12. Rendering of CThead space filled distance field at different isovalues (offsets).

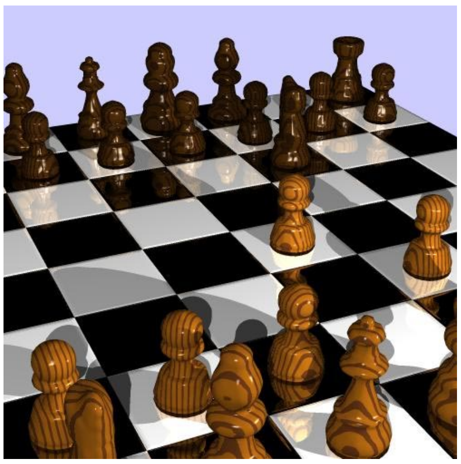

Figure 13. A Volume Graphics scene using sub-voxel accurate distance field encoded chess pieces.

demonstrated their superiority over sampling methods and table 2 their comparable execution time for distance shells. Details for reproducing distance shells are given in 3.3 and 4.4. A case for needing space-filled distance fields was made in Section 4.1, but computational expense was cited as a major problem. Section 4.4 builds upon the knowledge presented in section 4.3 to demonstrate that vector methods produce fairly accurate distance fields in less time. Our contribution to that area is a sub-voxel accurate segmentation and a better vector transform, which we present in this paper in relation to modelling objects.

We have demonstrated that distance fields can be used to represent (voxelise) objects, but our overall aim was to demonstrate to the graphics community that fast, **accurate** space filled distance fields can be calculated from distance shells and thus become a realistic modelling paradigm with many (already identified) application areas.

Acknowledgements

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

References

[1] D. E. Breen, S. Mauch, and R. Whitaker. 3D scan conversion of CSG models into distance volumes. In *Proceedings of the 1998 Symposium on Volume Visualization, ACM SIGGRAPH*, pages 7–14, October 1998.

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

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

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

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

[6] S. Fang and H. Chen. Hardware accelerated voxelisation. In *Volume Graphics*, pages 301–315. Springer, 2000.

[7] 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.

[8] S. F. F. Gibson. Using distance maps for accurate surface representation in sample volumes. In *Proceedings of the 1998 IEEE Symposium on Volume Visualization*, pages 23–30, Oct. 1998.

[9] G. T. Herman, J. Zheng, and C. A. Bucholtz. Shape-based interpolation. *IEEE Computer Graphics and Applications*, 12(3):69–79, May 1992.

[10] 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.

[11] M. W. Jones. The production of volume data from triangular meshes using voxelisation. *Computer Graphics Forum*, 15(5):311–318, December 1996.

[12] M. W. Jones. Direct surface rendering of general and genetically bred implicit surfaces. In *Proceedings of 17th Annual Conference of Eurographics (UK Chapter) Cambridge*, pages 37–46, April 1999.

[13] 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.

[14] M. W. Jones and M. Chen. Fast cutting operations on three dimensional volume datasets. In *Visualization in Scientific Computing*, pages 1–8. Springer-Verlag, Wien New York, January 1995.

[15] A. Kaufman. An algorithm for 3D scan-conversion of polygons. *Proc. of Eurographics '87, Amsterdam, The Netherlands*, pages 197–208, August 1987.

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

[17] D. Levin. Multidimensional reconstruction by set-valued approximation. *IMA J. Numerical Analysis*, 6:173–184, 1986.

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

[19] D. W. Paglieroni. Distance transforms: Properties and machine vision applications. *CVGIP: Graphical Models and Image Processing*, 54(1):56–74, Jan. 1992.

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

[21] 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.

[22] R. Satherley and M. W. Jones. Vector-city vector distance transform. Technical report, University of Wales Swansea, Singleton Park, Swansea, Feb. 2000. Submitted to Computer Vision and Image Understanding.

[23] R. Satherley and M. W. Jones. Hypertexturing complex volume objects. Technical report, University of Wales Swansea, Singleton Park, Swansea, Oct. February 2001. to appear in WSCG 2001.

[24] 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.

[25] M. Srámek and A. E. Kaufman. vxt: A class library for object voxelisation. In *Volume Graphics*, pages 119–134. Springer, 2000.

[26] 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.

[27] S. W. Wang and A. E. Kaufman. Volume sampled voxelization of geometric primitives. In *Proc. Visualization 93*, pages 78–84. IEEE CS Press, Los Alamitos, Calif., 1993.

[28] A. Watt and M. Watt. *Advanced Animation and Rendering Techniques*. Addison-Wesley, 1996.

[29] A. S. Winter and M. Chen. vlib: A volume graphics library. 2001. Submitted to Volume Graphics 2001.

[30] R. Yagel and Z. Shi. Accelerating volume animation by space-leaping. In *Proceedings of IEEE Visualization '93*, pages 62–69, Oct. 1993.

[31] 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.

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