The Visual Computer

Hypertexturing complex volume objects

Richard Satherley,
Mark W. Jones

Department of Computer Science, University of Wales Swansea, Singleton Park, Swansea SA2 8PP, United Kingdom
E-mail: {csrich, m.w.jones}@swansea.ac.uk

Published online: 15 March 2002

This paper will examine hypertexture rendering techniques and will demonstrate how volume data sets may be adapted in order for hypertexture to be applied. Details are given of a process for the conversion of complex objects, such as CT scans, into accurate distance fields. Hypertexture is applied to these objects and example renderings include the UNC CThead, a chess piece, a dodecahedron and a tank. Additional information is given about soft objects, density modulation functions, ray marching and controlling hypertexture application.

Key words: Hypertexture – Distance transform – Distance field

1 Introduction

Traditional texture mapping techniques, such as Peachey [1] and Perlin's [2] *solid textures*, are limited to the reproduction of textures that have *simple* surface definitions. Many natural textures, such as fur, have surface definitions that are at best complex. Others, such as fire and smoke, have no well-defined surface at all and are therefore unreproducible with such methods.

Hypertexture [3], developed by Perlin and Hoffert, allows the reproduction of such complex textures through the manipulation of surface densities. This method is restricted to objects which can be defined using a modified form of implicit function [$f(x, y, z) = D(x, y, z)$], such as spheres and tori. Dischler and Ghazanfarpour [4] produced hypertexture-like effects on more general objects, with the use of *geometrical based textures*, a method which involves defining a *skeleton* for an object, from which densities are interpolated. Again this method is limited as it is only applicable to closed objects.

This paper aims to introduce a new method for the application of hypertexture to complex (open or closed) volume objects, such as CT scans and voxelised polygon meshes, with the use of *distance fields*. Distance fields are well suited to hypertexture application, as each point in the field gives the distance to its closest surface point, from which density can be easily calculated.

Section 2 will briefly introduce the process of hypertexture application, summarising the concepts of *soft objects* (Sect. 2.1) and *density modulation functions* (Sect. 2.2). In Sect. 2.3 *ray marching* is introduced, and examples of hypertexture effects are given in Sect. 2.4. Section 3 will develop the new idea of using distance fields as a basis for applying hypertexture to complex objects. Section 3.1 will demonstrate how scanned data, such as CT data sets, and previously voxelised (shell) data sets [5] can be converted into distance fields using a *distance transform*. Section 3.2 will introduce a new, faster and more accurate vector distance transform, the *vector-city vector distance transform* (VCVDT). Example images will be given in Sect. 3.3. Finally, Sect. 4 introduces means of controlling the application of hypertexture.

2 Hypertexture

It has already been stated that Perlin and Hoffert developed hypertexture [3] as a method for replicat-

Digital Object Identifier (DOI) 10.1007/s003710100143

ing natural phenomena, such as fire and fur, on implicit surfaces. This section will briefly summarise the main aspects of hypertexture, giving examples of the achievable effects.

2.1 Soft objects

Before describing how to produce hypertexture effects, it is necessary to introduce *soft objects*. A soft object is an object where the thin surface boundary has been extended into a larger *soft region*, across which density varies from 1 (inside the object) to 0 (outside the object).

The soft region (and therefore the object) is modelled using an *object density function*, $D(p)$, where $p = (x, y, z)$ is a point in three-dimensional space. An example of which, describing a sphere, is given in (1) and is illustrated in Fig. 1. Note that the outer surface has been cut in order to give a better view of the two surfaces.

$$D(p) = \begin{cases} 1 & \text{if} |p|^2 \leq r_\mathrm{i}^2 \,, \\ 0 & \text{if} |p|^2 \geq r_\mathrm{o}^2 \,, \\ \frac{r_\mathrm{o}^2 - |p|^2}{r_\mathrm{o}^2 - r_\mathrm{i}^2} & \text{otherwise} \,. \end{cases} \tag{1}$$

where r_i = inner radius, r_o = outer radius and $p = (x, y, z) \in \mathbf{R}^3$.

Hypertexture effects can now be achieved by the repeated application of *density modulation functions* (DMF, Sect. 2.2) to the soft region of $D(p)$, as shown in the following:

$$H : \mathbf{R} \times \mathbf{R}^3 \rightarrow r, g, b, \alpha$$

$$H\big(D(p), p\big) = \mathrm{DMF}_n\Big(\ldots \Big(\mathrm{DMF}_0\big(D(p)\big)\Big)\Big) \,. \tag{2}$$

2.2 Density modulation functions

DMFs can be separated into the following categories:

Position dependent: Functions that are dependent on p.

Position independent: Scalar argumented functions.

Geometry dependent: Functions that are dependent on local geometry, for example, surface normals.

Complex DMFs are constructed, as shown in (2), using a combination of the following basis functions,

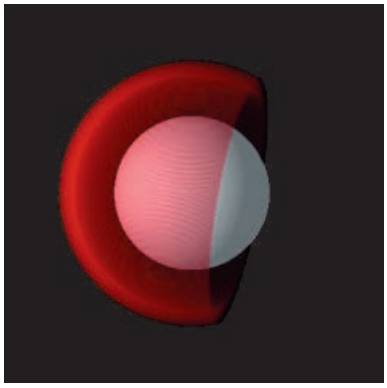


Fig. 1. Illustration of (1)

along with control functions such as *sine* and *cosine* (cf. Perlin's solid textures [2]):

Noise: Approximates band-limited white noise; returns a pseudo-random value in the range $(-1, 1)$. Several implementations are given by Lewis [6].

Turbulence: Simulates the appearance of Brownian motion (turbulent flow) by the summation of noise at increasing frequencies (3), introducing a self-similar f^{-1} pattern, where f is the initial frequency.

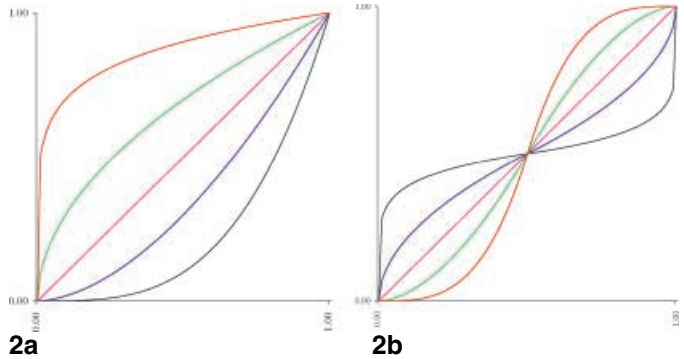
$$\mathrm{turb}(p) = \sum_i \mathrm{abs}\left(\frac{\mathrm{noise}(2^i p)}{2^i}\right) \,. \tag{3}$$

Bias: Defined by the power curve of (4) (illustrated in Fig. 2a), bias is used to shape the density across the soft region.

$$\mathrm{bias}_b\big(D(p)\big) = D(p)^{\frac{\ln(b)}{\ln(\frac{1}{2})}} \,. \tag{4}$$

Gain: Defined as a combination of two bias curves [(5), Fig. 2b], gain is used to alter the rate at which the density changes in the midrange of the soft region.

$$\mathrm{gain}_g\big(D(p)\big) = \begin{cases} \frac{\mathrm{bias}_{1-g}\left(2D(p)\right)}{2} & \text{if} D(p) < \frac{1}{2}, \\ 1 - \frac{\mathrm{bias}_{1-g}\left(2-2D(p)\right)}{2} & \text{otherwise} \,. \end{cases} \tag{5}$$



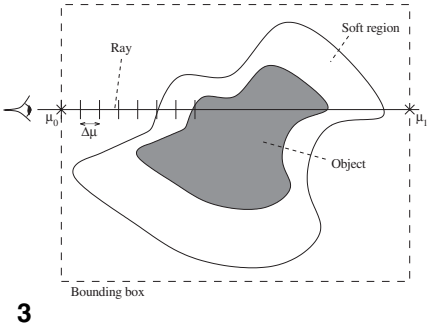


Fig. 2. The *bias* **(a)** and *gain* **(b)** power curves
Fig. 3. The ray marching process

Fast alternatives for the bias and gain functions have been developed by Schlick [7].

2.3 Implementation

The absence of a well-defined surface for the natural phenomena modelled by hypertexture means that hypertextured objects will also have no well-defined surface. This necessitates the use of a *ray marcher*, a simplified form of volume rendering [8], when creating hypertexture effects.
A simplified model of a ray marcher is given in Fig. 3. A ray, r, is fired from every pixel into the object space. If a ray intersects the object's bounding box, the entry (μ_0) and exit (μ_1) points for this ray are calculated. $D(p)$ is now evaluated at fixed points along the ray according to

$$r(k) = \mu_0 + k\Delta\mu\ , \quad \text{where} \quad k \in \mathbf{N}\ . \tag{6}$$

If p lies within the soft region, the necessary DMF combination is applied, and a colour, C_λ ($\lambda = $ r, g, b), and opacity, α, for this point of the ray are obtained. This process is repeated along the ray, accumulating colour and opacity, until one of following three criteria is met:

Ray termination: $r(k) = \mu_0 + k\Delta\mu \geq \mu_1$.

Inner surface reached: $D(p) = 1$.

Opaque surface: $\alpha \geq 1$.

Evaluating implicit surfaces by ray marching is inefficient and does not guarantee the retrieval of all the roots. A root in the implicit function $f(p) = 0$ can be easily missed if a zero value occurs between sample points. Worley and Hart [9] remove this inefficiency by replacing the ray marcher with *sphere tracing* [10] which guarantees a *safe* stepping distance by augmenting function evaluations with the Lipschitz condition.

2.4 Examples of hypertexture

Figure 4 illustrates some of the effects which may be achieved with the application of hypertexture. The DMFs used in each case are as follows:

Unconditional noise: Noise is added to each component of p (Fig. 4a).

Directed noise: Noise is added to only one of the components of p, creating a melting effect (Fig. 4b).

Fractal noise: Noise of increasing frequency and decreasing amplitude is summed and added to p (Fig. 4c).

Fire: Turbulence is added to each component of p (Fig. 4d).

Fur: Fur filaments are *grown* from random positions (Fig. 4e).

Melting fire: A fire hypertexture is applied to a previously melted sphere (Fig. 4f).

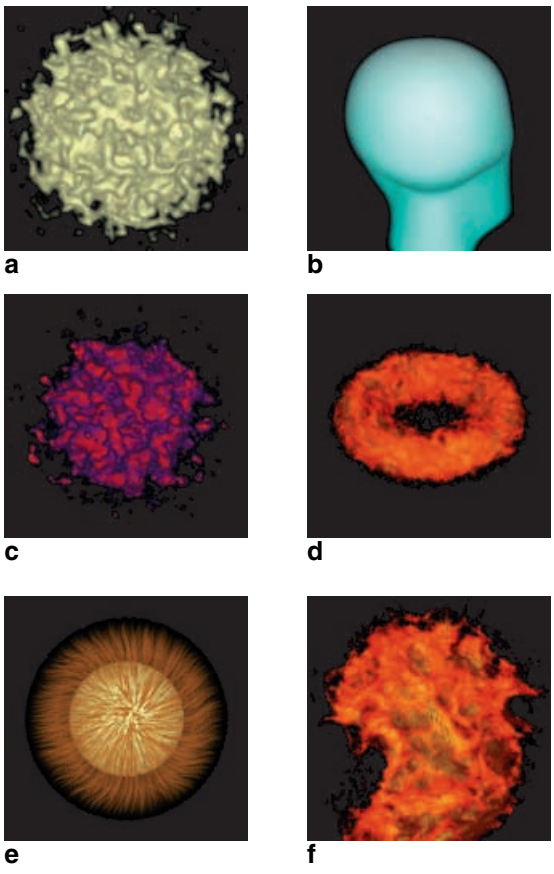


Fig. 4a–f. Examples of hypertexture effects: **a** noise; **b** melting; **c** fractal; **d** fire; **e** curled fur; **f** melting fire

3 Hypertextured volume data

The fact hypertexture is implemented with a ray marcher (theoretically) allows it to operate on volume data. Problems arise when it is not possible to derive an implicit function for an object which is traditionally rendered using volume-based methods, for example, CT scan data sets.

Examination of the data sets used for the images of Fig. 4 and other mathematically generated data sets, shows that the voxels give values related to the *distance* to some *feature point/surface*. Therefore, complex objects could be rendered with hypertexture if the data set could be converted into a distance volume, D, where $D(p)$ is the distance from p to the closest point on the object's surface. For volume data, this can be achieved by measuring the distance between every background voxel and every surface voxel and then taking the minimum. This is extremely computationally expensive – taking over 62 h for the UNC CThead. This can be improved to 2.5 h by employing an octree and using a neighbour's closest voxel as a candidate to eliminate parts of the tree.

The run-time of the distance conversion process can be considerably improved, at the cost of reduced accuracy, with the use of a *distance transform* (DT) [11–20] – a process used to approximate Euclidean distance calculations via *local distance propagation*. In Sect. 3.1 the DT process will be introduced, followed by a description of the *vector-city vector distance transform* (VCVDT) – a new more accurate DT, in Sect. 3.2. Example images will also be given.

3.1 Distance transforms

DTs can be separated into two categories, *chamfer distance transforms* (CDTs) and *vector distance transforms* (VDTs). The difference being CDTs propagate distance by addition of known local neighbourhood distances, whereas vectorial information, from which distances are calculated, is propagated by VDTs. Both categories are implemented as a two-stage process – surface extraction followed by distance propagation.

The first stage of a DT is to *segment* the data set, f, on a regular grid, via a thresholding operation, τ, to extract the surface of interest, S.

$$S = \{(x, y, z) : f(x, y, z) \geq \tau\}\,, \tag{7}$$

where $x, y, z \in \mathbf{Z}$.

3.1.1 Chamfer distance transforms

The preliminary distance field for a CDT is constructed using

$$D(p) = \begin{cases} 0 & \text{if } p \in S \text{ and } \exists, q \in p_{26}, q \notin S\,, \\ \infty & \text{otherwise}\,, \end{cases} \tag{8}$$

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

If a signed field (voxels inside the object have negative distances) is required, the data sets must also be classified to indicate whether a voxel is internal, external or on the surface (9). The signed distance field is thus the unsigned field multiplied by the corresponding classification value (10).

$$C(p) = \begin{cases} -1 & \text{if } p > \tau \text{ and } p \notin S\,, \\ 0 & \text{if } p \in S\,, \\ 1 & \text{otherwise}\,, \end{cases} \tag{9}$$

$$D(p)_{\text{signed}} = D(p) \times C(p)\,. \tag{10}$$

CDTs propagate distances with the use of two passes of the *distance matrix*, d_M, applying (11) to each voxel in turn, as demonstrated in the pseudo-code below.

$$D(p) = \min[D(x+i,\, y+j,\, z+k) + d_M(i,\, j,\, k)\big] \quad \forall\, i,\, j,\, k \in d_M, \tag{11}$$

where $p \in \mathbf{Z}^3$ and $i,\ j, k \in \mathbf{Z}$.

```
/* Forward Pass */
FOR(z = 0; z < fz; z++)
FOR(y = 0; y < fy; y++)
FOR(x = 0; x < fx; x++)
D(x,y,z) = (11)
/* Backward Pass */
FOR(z = fz-1; z ≥ 0; z--)
FOR(y = fy-1; y ≥ 0; y--)
FOR(x = fx-1; x ≥ 0; x--)
D(x,y,z) = (11)
```

Two examples of a chamfer distance matrix are given in Fig. 5 (see [14] for more examples). Note that each matrix element gives the distance to the central element.

Overall, each voxel is considered twice, with its distance calculation depending purely upon the addition of the matrix elements to its neighbours and taking the minimum.

3.1.2 Vector distance transforms

VDTs [12, 16–18] generally require more passes of the distance matrix. During each pass the vector components are added to the necessary vector position, the distance is calculated, a decision is made as to whether any of the new distances are minimal, and finally the minimal vector is stored.

To allow vector propagation, (8) and (11) are modified as follows:

$$\boldsymbol{V}(p) = \begin{cases} (0,0,0) & \text{if } p \in S \text{ and } \exists\, q \in p_{26},\, q \notin S, \\ (\infty,\infty,\infty) & \text{otherwise}\,. \end{cases} \tag{12}$$

$$D(p) = \min \left| \boldsymbol{V}(x+i,\, y+j,\, z+k) + d_M(i,\, j,\, k) \right| \quad \forall\, i,\, j,\, k \in d_M\,, \tag{13}$$

where $d_M = (\boldsymbol{M}_x, \boldsymbol{M}_y, \boldsymbol{M}_z)$.

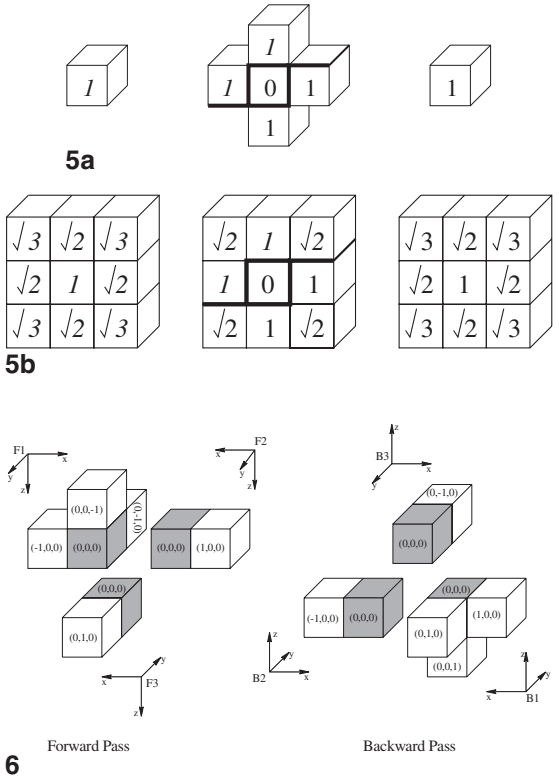


Fig. 5a,b. Chamfer distance matrices: **a** city-block; **b** complete CDT

Fig. 6. Mullikin's EVDT matrix passes

Figure 6 shows the matrix passes employed by Mullikin's *efficient VDT* (EVDT) [18] – the best VDT given in the literature.

Overall each voxel attempts to update its distance once during each pass, with the number of calculations and comparison made each time dependent on the matrix used. For example, the EVDT passes over the data set six times, making a total of 16 distance calculations and 11 comparisons per voxel.

Table 1 summarises the findings of a comparison between the true Euclidean distance field, for the skull of the UNC CThead, and those generated using the city-block, complete CDT and EVDT distance matrices. From the table it can be seen that the EVDT is by far the superior of the implemented DTs, having an almost negligible average error per voxel.

Table 1. Results of comparing the generated distance fields to the ture Euclidean distance field

Distance matrix	Execution time (s)	Error range min ⇒ max	Incorrect voxels (%)	Average error per voxel
True Euclidean	7560.000	0.000 ⇒ 0.000	0.000	0.000000
City-block	1.320	−2.000 ⇒ 76.060	91.579	12.269071
Complete 3×3×3	4.842	−0.415 ⇒ 11.769	88.774	2.196785
EVDT	7.540	−0.518 ⇒ 2.533	3.449	0.004761

3.2 The vector-city VDT

The previous section concluded that VDTs are far more accurate than CDTs. In this section a new, more accurate VDT, based on the city-block CDT, will be introduced.

The city-block CDT is the most elementary DT. A fact which has led to it being the basis for the majority of VDTs. For example, Danielsson's 4SED [12] uses the same local neighbourhood and Mullikin's EVDT [18] uses the vector equivalent of the two city-block passes in its own. The new VCVDT extends this heuristic to include all four matrix passes. That is, the extra passes could also be implemented as part of the city-block CDT. Figure 7 illustrates the matrix used during each pass. The dashed positions in Fig. 7 are omitted during the matrix passes as they represent redundant calculations, that is, they perform the same calculation as a previous position.

The VCVDT is implemented in a manner similar to that of the EVDT. During each pass (*F1*, *F2*, *B1* and *B2*) the corresponding matrix segment is applied in the direction indicated by Fig. 7. At each voxel, its neighbours' vectors are altered according to the overlying matrix element and the minimal vector stored. If the situation arises where two or more of the distances are minimal, *a tie*, no preference is made as to which of the corresponding vectors is stored. This is also the case in the EVDT.

Unlike the EVDT, which only stores two vector slices, the VCVDT stores a complete vector copy of the distance volume. Storing only two vector slices was found to be the inferior of the two approaches, as vector information is lost as a pass progresses through the data set.

Furthermore, by storing (after each pass) the minimal distance along with the minimum vector, the VCVDT is able to employ Leymarie and Levin's [17] optimisations. Thus removing the need to recalculate the distance for the central voxel, saving three distance calculations per voxel. Therefore, the VCVDT makes only 11 distance calculations and 10 comparisons per voxel.

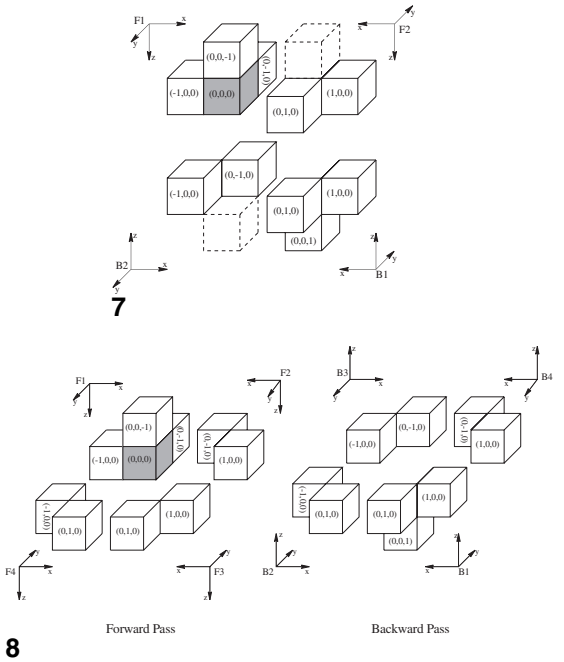


Fig. 7. VCVDT matrix passes
Fig. 8. The 8VCVDT matrix passes

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 Fig. 8, producing a VDT resembling Ragnemalm's [21] *corner EDT*. As before, the extra passes could be implemented as part of the city-block CDT, adding an extra eight distance calculations and comparisons per voxel. Thus, the eight pass VCVDT makes a total of 19 distance calculations and 18 comparisons per voxel.

Table 2. Results of the comparing the VCVDT distance fields to the true Euclidean distance field

Distance matrix	Execution time (s)	Error range min ⇒ max	Incorrect voxels (%)	Average error per voxel
VCVDT	3.420	−0.334 ⇒ 0.446	1.300	0.000644
8VCVDT	5.470	−0.334 ⇒ 0.334	0.343	0.000223

Table 2 shows the results of the comparison between the distance fields obtained from the four- and eight-pass VCVDT and the true Euclidean distance field. It can be seen from Tables 1 and 2 that the VCVDTs outperform the EVDT in both speed of execution and accuracy. A more detailed account of the implementation and performance of the VCVDT can be found in [22].

3.3 Hypertextured distance fields

Hypertexture can be easily added to a distance field by making a slight modification to the object density function (1):

$$D(p) = \begin{cases} 1 & \text{if } \|p\| \le r_\mathrm{i}\,, \\ 0 & \text{if } \|p\| \ge r_\mathrm{o}\,, \\ \frac{r_\mathrm{o} - \|p\|}{r_\mathrm{o} - r_\mathrm{i}} & \text{otherwise}\,, \end{cases} \tag{14}$$

where r_i = inner radius, r_o = outer radius and $\|p\|$ = data set value at p.
That is, the value held at voxel p is used instead of the magnitude of p. Furthermore, as the data set holds distance values, it is no longer necessary to square the inner and outer radii.
Figure 9 demonstrates the application of hypertexture to distance fields of triangular mesh objects and the UNC CThead.

4 Controlling hypertexture

There is little material available in the literature on ways to control the application of hypertexture to an object. This section will demonstrate three application control methods which are applicable to hypertexture:

Clipping: Hypertexture is only applied to selected parts of the object.

Blending: Two or more textures can be blended together.

Animation: Animations can be created by altering control parameters.

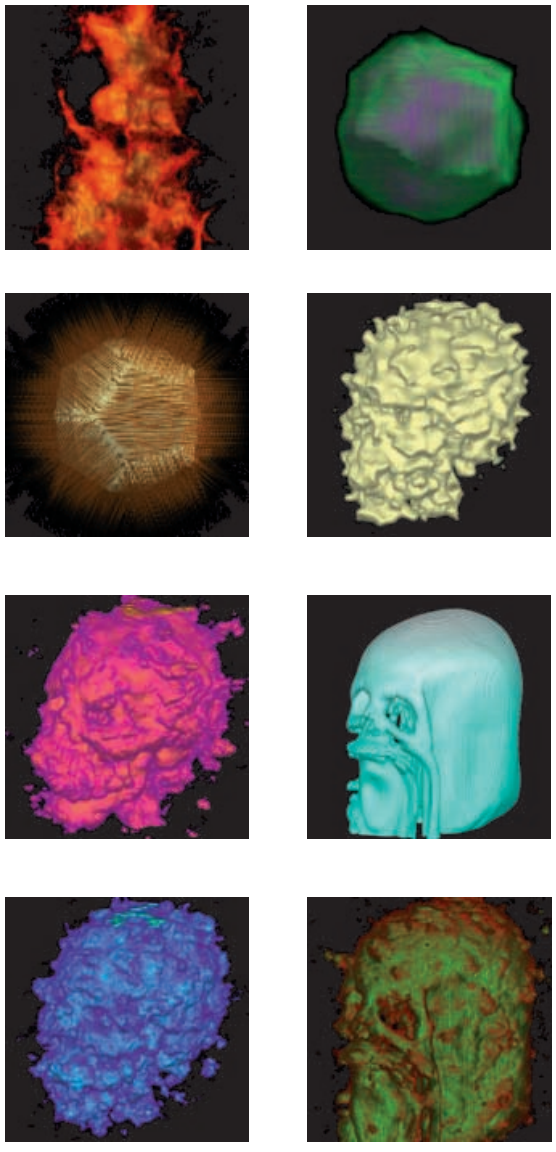
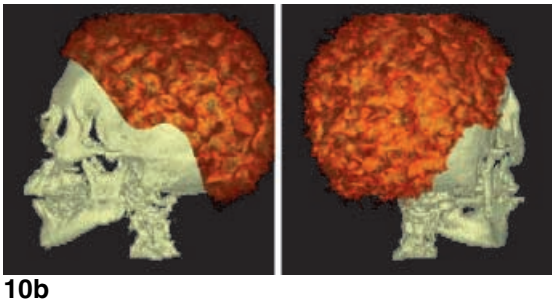
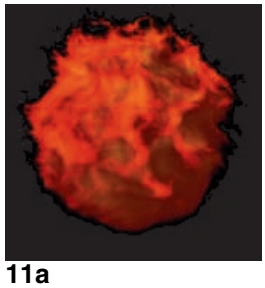


Fig. 9. Hypertexture applied to the distance fields for a chess piece, a dodecahedron and the UNC CThead data set

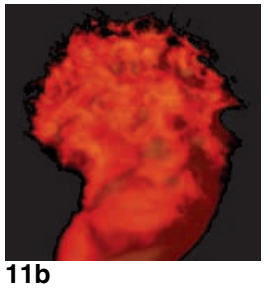
10a



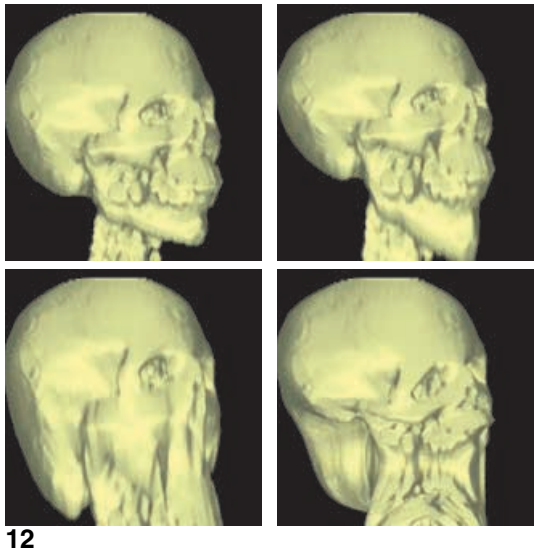
10b



11a



11b



12

Fig. 10a,b. Clipped hypertexture: **a** melting tank barrel; **b** *hairline* clipping surface
Fig. 11a,b. Examples of blended hypertexture: **a** fading; **b** merged
Fig. 12. An animated hypertexture

4.1 Clipped hypertexture

More often than not it is only necessary to texture a certain part of an object. This selective texture addition can be achieved if it is possible to identify such sections.
One approach is to define a clipping surface to separate the object into the desired sections. With the clipping surface defined, a simple calculation on the current ray location indicates whether or not the texture should be applied. In Fig. 10a a clipping surface has been constructed, allowing the barrel of the tank to be melted. Figure 10b shows a more complex clipping surface which has been designed to follow a *hairline*.
Figure 10a also shows that an object does not have to be restricted to one type of texture; the tank's camouflage was produced using Perlin's *bozo* solid texture [2].

4.2 Blending hypertexture

A problem with clipped hypertexture application is that artifacts can occur at the clipping surface if the textures used on either side are disjointed. These artifacts can be removed if the textures are blended together in the region of the clipping surface.
Examples of blended hypertexture are given in Fig. 11. Figure 11a shows a fire hypertexture *fading* away as it moves down the sphere and Fig. 11b demonstrates a blend between a fire and melting hypertexture.

4.3 Animation

As hypertexture is created by manipulating the surface densities of an object with a three-dimensional DMF, the object may be rendered from different viewpoints without any inconsistencies in the tex-

ture, thus introducing frame coherency for animation. Furthermore, making a slight alteration in the control parameters of the hypertexture can produce some interesting animations. Figure 12 shows four frames from a melting animation; here the frequency of the distorting noise is increased in each frame.

5 Conclusion

This paper has introduced a new method for the easy application of hypertexture to complex, open or closed, volume objects. In addition to showing that data sets obtained by evaluating an implicit function can be immediately hypertextured, it has also been shown that, with the use of a distance transform, complex volume objects may be made to mimic such data sets, and therefore have a hypertexture applied.

Details have been given about the transform process, in particular two new VDTs – the four- and eight-pass VCVDTs – have been introduced. The performance of the new VCVDTs has been compared with that of the best distance transform given in the literature – Mullikin's EVDT. The new methods were shown to be both faster and more accurate.

A brief explanation of the main aspects of hypertexture – soft objects, density modulation functions and the ray marching process – has been given. Finally, examples of hypertextured distance field encoded objects have been given, and also methods for controlling the application of hypertexture have been demonstrated.

Acknowledgements. We would like to thank Šrámek and Kaufman [23, 24] for allowing us to use their voxelised tank. This work has been undertaken with funding from EPSRC, UK, under grant GR/L88238.

References

1. Peachey DR (1985) Solid texturing of complex surfaces. Comput Graph (Proc. SIGGRAPH '85) 19(3):279–286
2. Perlin K (1985) An image synthesizer. Comput Graph (Proc SIGGRAPH '85) 19(3):287–296
3. Perlin K, Hoffert E (1989) Hypertexture. Comput Graph (Proc SIGGRAPH '89) 23(3):253–262
4. J-Dischler M, Ghazanfarpour D (1995) A geometrical based method for highly complex structured textures generation. Comput Graph Forum 14(4):203–215
5. Jones MW (1996) The production of volume data from triangular meshes using voxelisation. Comput Graph Forum 15(5):311–318
6. Lewis JP (1989) Algorithms for solid noise synthesis. Comput Graph (Proc SIGGRAPH '89) 23(3):263–270
7. Schlick C (1994) Fast Alternatives to Perlin's Bias and Gain Functions, Chap VI.3, Academic Press, New York, pp 401–403
8. Levoy M (1988) Display of surface from volume data. IEEE Comput Graph Appl 8(3):29–37
9. Worley SP, Hart JC (1996) Hyper-rendering of hypertextured surfaces. In Proceedings of Implicit Surfaces '96, Eurographics, pp 99–104
10. Hart JC (1996) Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. The Visual Comput 12(10):527–545; an earlier version appeared in Comput Graph (Proc. SIGGRAPH '93)
11. Rosenfeld A, Pfaltz JL (1966) Sequential operations in digital picture processing. J Association Comput Mach 13(4):471–494
12. Danielsson P-E (1980) Euclidean distance mapping. Comput Graph Image Process 14:227–248
13. Borgefors G (1984) Distance transformations in arbitrary dimensions. Comput Vision Graph Image Process 27(3):321–345
14. Borgefors G (1986) Distance transformations in digital images. Comput Vision Graph Image Process 34(3):344–371
15. Vossepoel AM (1988) A note on distance transfromations in digital images. Comput Vision Graph Image Process 43(1):88–97
16. Ye QZ (1988) The signed Euclidean distance transform and its applications. In Proceedings, 9th International Conference on Pattern Recognition, IEEE Computer Society Press, Los Alamitos, California, pp 495–499
17. Leymarie F, Levine MD (1992) Fast rater scan distance propagation on the discrete rectangular lattice. Comput Vision Graph Image Process: Image Understand 55(1):84–94
18. Mullikin JC (1992) The vector distance transform in two and three dimensions. Comput Vision Graph Image Process: Graph Models Image Process 54(6):526–535
19. Borgefors G (1996) On digital distance transforms in three dimensions. Comput Vision Image Understand 64(3):368–376
20. Marchand-Maillet S, Sharaiha YM (1999) Euclidean ordering via chamfer distance calculations. Comput Vision Image Process 73(3):404–413
21. Ragnemalm I (1993) The Euclidean distance transform in arbitrary dimensions. Pattern Recogn Lett 14(11):883–888
22. Satherley R, Jones MW (2001) Vector-city vector distance transform. Comput Vision Image Understand 82(3):238–254
23. Šrámek M, Kaufman A (1998) Object voxelization by filtering. In IEEE Symposium on Volume Visualization, IEEE Computer Society Press, Los Alamitos, California, pp 111–118
24. Šrámek M, Kaufman A (2000) `vxt`: a c++ class library for object voxelization. In Chen, Kaufmann, Yagel (eds) Volume Graphics. Springer-Verlag, London, pp 119–134

Photographs of the authors and their biographies are given on the next page.

RICHARD SATHERLEY is a post-graduate student at the University of Wales, Swansea, UK. He received a first class BSc. degree in Computer Science in 1997. He is currently undertaking a Ph.D. in Computer Graphics and Visualisation, researching the computation and uses of distance fields in volume graphics. His research interests include texture mapping, texture analysis and synthesis, and hypertexture.

MARK W. JONES is a lecturer in the Department of Computer Sicence at the University of Wales, Swansea. His primary research interests are in the field of Volume Graphics, in particular, distance fields, voxelisation, modeling and three-dimensional reconstructions. Mark received a Ph.D. from the University of Wales in 1995. He is a member of Eurographics.