Volume 15, (*1996*), *Number 5 pp. 311–318*

The Production of Volume Data from Triangular Meshes Using Voxelisation

Mark W. Jones

Abstract
Voxelisation is the term given to the process of converting data from one source type into a three dimensional volume of data values. The techniques known collectively as volume visualisation can then be applied to the data in order to produce a graphical representation of the object. This paper gives a practical approach to the voxelisation of data in the form of triangular meshes, and demonstrates the use of the method on various datasets. Visualisation is achieved by a method also described in the paper.

Keywords: voxelisation; volume data; polygonal meshes; volume visualisation; volume morphing; rendering; ray-tracing;

1. Introduction

In this paper the production of volume data from surface meshes is examined. An algorithm to convert closed triangular meshes into volume data is given along with optimisations which make the conversion realistically possible on most computers. The visualisation of this data is explored with a few examples given. The motivation behind this work is to produce volume data from triangular meshes in order to achieve realistic metamorphosis between the mesh objects. Volume morphing, whilst retaining object coherency, is difficult to achieve, and the disc morphing method[1] has given good results. In order to apply this morphing technique the data must be available as volume data.

This work fits in with other work that has sought to voxelise object data of one description into volumetric data with the specific aim of using a unifying approach to the visualisation of data from different sources. James Kayija stated at SIGGRAPH '91[2] "... in 10 years, all rendering will be volume rendering." He views volume rendering as the only realistically scalable rendering method for complex scenes. Through the use of advanced volume visualisation techniques it would also be possible to view visualisation as an all encompassing rendering process. Such a model of rendering would allow the consistent application of visual effects to all objects, rather than applying various rendering methods to several models and combining results.

The goals of this paper are to give specific details of a voxelisation method for triangular meshes, and to demonstrate the method does not suffer from many of the problems of existing voxelisation methods. The method is primarily aimed at producing voxel data for morphing. Visualisation of various meshes is presented in this paper to demonstrate that this voxelisation process is successful and may also become a worthwhile technique simply to enable efficient visualisations of certain data sets. This process of voxelisation and subsequent visualisation is supported by evidence of a comparison of results from this method and a traditional ray-tracer - POV-Ray.

Most previous voxelisation algorithms convert solid objects into voxel data using spatial-occupancy enumeration[3−6]. The result of this process is a three dimensional regular array of cubes, sometimes known as a cuberille. The main problem associated with this method is the fact that the model is limited by the resolution of the grid. The result of coarse grids is an image which is very blocky in appearance due to the fact that only cube faces are displayed. Since there are only 6 cube faces, the resulting image only has 6 different normals with which to be shaded. Shading algorithms such as depth shading[7] and congradient shading[8] attempt to improve this situation, but only manage to reduce surface artifacts slightly. As mentioned, coarse grids increase this problem, and as such this method suffers badly when a close view of the surface is required.

Volumetric data is produced using filters[9], which can then be volume rendered to produce anti-aliased images. The technique is described for objects such as spheres and cones, but detailed descriptions for triangular meshes are not given.

© The Eurographics Association and Blackwell Publishers 2000.Published by Blackwell Publishers, 108 Cowley Road, Oxford OX4 1JF, UK and 350 Main Street, Malden, MA 02148, USA.

Where this paper differs from previous work is that it provides the following :

- a detailed description of a practical approach to producing voxel data,
- production of voxel data which results in accurate images when rendered,
- details of optimisations which make the process realistic,
- details of the rendering algorithm used to achieve a high quality visualisation,
- a comparison in time and quality of the images produced with those of a traditional renderer - POV-Ray.

Details about the method are given in Section 2. Section 3 shows how the efficiency of the algorithm can be increased, and Section 4 discusses the results for some meshes.

2. Production of voxel data

Many methods exist for the visualisation of three dimensional voxel data, but they can be broadly divided into two categories – direct volume rendering and isosurfacing. Isosurfacing[10] does not apply in this case since it aims to produce a surface mesh from volume data. Using this method a surface mesh would be produced from volume data, which had been produced from a surface mesh. The methods of volume rendering[11−15] produce images directly from the volume data and by using accurate optical models[16−19] the method can be used to effectively model various lighting effects such as scattered light, clouds and haze. Many of these methods have been implemented using hardware[20−23], and as prices decrease these systems will become widely available.

Many methods produce images by converting the original values into opacity depending upon some function. This function usually states that regions of large gradient, that is object interfaces, are the most interesting parts of the volume and are to be visualised. Any voxel model produced from surface mesh data must be able to distinguish the object interface from the interior of the object and the surrounding empty uninteresting portions of the data set. One simple method would be to assign voxels a value depending upon whether they are inside or outside the object:

$$f(x,y,z) = \begin{cases} -1 & \text{if } (x,y,z) \text{ is outside the surface} \\ 0 & \text{if } (x,y,z) \text{ is on the surface} \\ 1 & \text{if } (x,y,z) \text{ is inside the surface} \end{cases} \quad (1)$$

It is assumed throughout this paper that a voxel is a point in 3D space. The advantage of this function is that a three dimensional scan conversion algorithm can be employed to determine the state of each voxel within the domain. This results in a fast conversion of the object into voxel data, but the surface produced is highly dependent upon the density of the chosen voxel data. As mentioned before, images produced from such data often exhibit a blockiness[24] because a binary decision is being made at each voxel – it is either inside or not. In order to get a smoother description of the surface, a function which varies smoothly in the region of a surface is required. The function selected for this method is the distance function defined as:

$$f(x,y,z) = \begin{cases} -dist(x,y,z) & \text{if } (x,y,z) \text{ is outside} \\ 0 & \text{if } (x,y,z) \text{ is on} \\ dist(x,y,z) & \text{if } (x,y,z) \text{ is inside} \end{cases} \quad (2)$$

where $dist(x,y,z)$ is the Euclidean distance from (x,y,z) to the closest point on the surface. This results in a volume that represents a continuous function sampled at discrete voxel locations, that is not so dependent upon the density of the voxel data[25]. Using the distance function has the effect that the voxelisation process results in a smoother representation of the original surface. This can be seen in the comparison between the images produced using this method and the image produced using POV-Ray in the results section. Such meshes are usually a tesselated approximation of a smooth surface, and thus the smoothing introduced by the voxelisation function may be desirable. Certainly the voxelisation process and subsequent surface rendering produce far better images for these meshes.

Since voxels are considered to be points and it is assumed that the surface mesh is closed, there is a notion of whether a point is inside the surface. As a result of using the distance function of Equation 2, points inside the surface are positive, and points outside the surface are negative. This method produces coherent surfaces if we assume no surface detail is smaller than a cell bounded by 8 voxels. That is, all points outside the surface in the original representation will be outside in the voxelised representation (similarly for inside). The only occasion that this is not true is when the original surface mesh passes into an 8 voxel bounded cell, with all voxels remaining negative (outside).

This data can be visualised using the direct surface rendering method[26]. The algorithm bears a resemblance to the direct volume rendering method in principle, but is only interested in the surface contained in the volume, rather than the intensity and colour of every voxel. For each pixel in the image plane, a ray is cast into the volume and is traced through cubes on its path until it hits a transverse cube. A cube, bounded by eight voxels, is said to be transverse if at least one of its voxels is inside the surface and one outside the surface, where the surface is defined as an isosurface of value τ. Transverse cubes can be identified by comparing values of its bounding voxels against the threshold value and setting an eight bit flag, $b_8b_7b_6b_5b_4b_3b_2b_1$, as:

$$b_i = \begin{cases} 0 & \text{if the value of voxel } i \leq \tau \\ 1 & \text{if the value of voxel } i > \tau \end{cases} \quad (3)$$

A cube is transverse if its flag is of the binary value be-

© The Eurographics Association and Blackwell Publishers 2000.

tween 00000001 and 11111110. Once a transverse cube is found, a further check is made to see if the ray intersects the surface contained within the cube. Given the ray entry and exit points, namely α and β respectively, the function values at those points are calculated using an interpolation function $F(\delta)$. Given a point δ lying on one of the six square facets of a cube, $F(\delta)$ is defined as the bilinear interpolation of the values associated to the four voxels which bound the facet. The ray intersects the surface if the ray span defined by α and β is transverse, that is, $F(\alpha) \leq \tau < F(\beta)$ or $F(\beta) \leq \tau < F(\alpha)$. The actual intersection point γ on the surface is calculated as

$$\gamma = \alpha + (\beta - \alpha)\left(\frac{F(\gamma) - F(\alpha)}{F(\beta) - F(\alpha)}\right) \tag{4}$$

Once the intersection point γ has been found, the surface normal at γ, and hence the intensity, can be computed.

The normal at γ is calculated by trilinearly interpolating the normals of the eight voxels bounding the unit cube. The normal, G at a voxel located at (x, y, z) is calculated using difference approximation, that is

$$\begin{aligned} G &= (g_x, g_y, g_z), \\ g_x &= F(x+1, y, z) - F(x-1, y, z), \\ g_y &= F(x, y+1, z) - F(x, y-1, z), \\ g_z &= F(x, y, z+1) - F(x, y, z-1). \end{aligned} \tag{5}$$

With the normal, a shading technique, such as Phong shading, can be used to calculate the intensity at γ.

3. Increasing the efficiency

The main problem with a naive approach to the production of the voxel data using the distance function is the fact that the complexity of the algorithm is so high. If the number of triangles in the mesh is N and the size of the voxel data set in each dimension is X, Y, and Z respectively, the distance function is calculated $NXYZ$ times.

Taking a mesh of 100 triangles and a volume of 100^3 voxels, for example, would result in 100 million calculations of the distance function. Since each function computation involves calculating the distance to a three dimensional triangle, and involves at least a square root, this is costly in terms of computer processing. Three areas for acceleration can be identified:

- reducing the number of voxels for which the distance has to be computed,
- making the distance calculation as efficient as possible,
- reducing the number of triangles which the distance has to be computed with.

These three areas are all addressed in the next sections.

3.1. Reduction of distance function calculations

It can be observed that it is not necessary for the value of every voxel to be accurately computed. In the direct surface rendering phase, the algorithm examines each cube, bounded by eight voxels, along a ray path. When it encounters a cube which contains the object interface it determines the values at the ray entry and exit points. If those values indicate there is a surface interface along the ray, the position of the surface is determined, and a surface normal is calculated. Voxel values at the eight vertices of the cube need to be known in order to determine the surface. The 6-neighbours (two in each of the x, y and z directions) of each voxel also need to be known in order to calculate the surface normal. In other words, only the values of voxels near the actual surface are required in order to display the surface. At all other voxels an indication of whether the point is inside or outside the surface is sufficient. This leads to a practical method for reducing the computational cost by restricting the distance computation to those voxels near the surface interface.

Each voxel is associated with two fields, namely state and distance. The state field indicates whether a voxel is interior (1), exterior (-1) or on the surface mesh (0), and is first computed by scan-converting the object. The state function (Equation 6) can be calculated for each voxel (x, y, z).

$$f(x,y,z) = \sum_{k=z-1}^{z+1} \sum_{j=y-1}^{y+1} \sum_{i=x-1}^{x+1} \text{state of voxel } (i, j, k) \tag{6}$$

For a voxel (x, y, z) there is no need to apply the distance function (Equation 2), if:

- its state field is zero, or
- $f(x,y,z) = 27$ or -27 and $f(x', y', z') = 27$ or -27 for each 6-neighbour (x', y', z').

It is obvious that there is no need to apply the distance function when the state field is zero which indicates the voxel is on the surface. The second condition shows that if a voxel and all its 26-neighbours are all interior (or all exterior) to the surface and all the 26-neighbours of its 6-neighbours have the same state, its distance value will not influence the surface display in any way. The first part states that a voxel need not be known if it is not used during linear interpolation of position, and the second part states that a voxel need not be known if it is not used during determination of the surface normal. This process eliminates all unnecessary voxels from the expensive distance computation by identifying those voxels, and only those voxels, for which the distance function must be calculated.

If the number of voxels on each slice for which the distance must be calculated is R_i, the number of distance calculations is now $N\sum_{k=1}^{z} R_k$. Usually R_i is less than both X and Y, therefore this method results in a reduction of an order of magnitude of the number of distance calculations.

© The Eurographics Association and Blackwell Publishers 2000.

3.2. Point to triangle distance calculation

During the computation of the voxel data, the distance from each voxel to the surface must be found. This requires the calculation of the distance between a 3D point, P and a triangle in 3D, $P_0P_1P_2$. Since this is likely to be done many times, and is the most expensive operation in the algorithm, it is worth examining in depth to create efficient code for the function.

It can be observed that there are several possible cases:

- the point could be closest to a vertex of the triangle (P_0, P_1 or P_2)
- the point could be closest to an edge of the triangle (P_0P_1, P_1P_2 or P_2P_0)
- the point could be closest to an interior point of the triangle. ($P_0P_1P_2$)

It would be costly to calculate the distance to each possible case and then use that to determine the minimum distance. It is far better to determine which case is applicable, and then calculate the distance only once for each triangle.

Two approaches for the calculation of point to triangle distances were implemented and compared[27]. The first calculated the distance in three dimensions, whereas the second rotated the triangle to make the problem 2D. It was found that the second method, which is described here, is the more efficient by a factor of 4. The simplest way to calculate the distance is by determining the translation and rotation matrix to rotate the triangle $P_0P_1P_2$ so that P_0 lies on the origin, P_1 lies on the z axis, and P_2 lies in the yz plane.

This transformation matrix can be calculated once for each triangle in a preprocessing step, and can then be used to transform P to P'. P' can be trivially projected onto the triangles plane giving P'' by ignoring its x coordinate since the triangle is in the yz plane. If P'' is inside the triangle $P_0P_1P_2$, the distance from the point to the triangle is simply the x coordinate of P.

Using P'', determination of the closest part of triangle $P_0P_1P_2$ to P can be found by using the edge equation[28], and once determined the distance can be found in a standard way. The edge equation is simply:

$$E(x,y) = (x-X)\,dY - (y-Y)\,dX \qquad (7)$$

for a line passing through (X,Y) with gradient $\frac{dY}{dX}$ with respect to a point (x,y). If $E<0$ the point is to the left of the line, if $E>0$ to the right, and if $E=0$, it is on the line. In Figure 1 we see the different possibilities.

If P'' is left of P_2P_0 it is closest to P_2P_0 if it is to the right of P_2P_{21} and to the left of P_0P_{01}. The proximity to the other edges of the triangle can be similarly determined. For P'' to be closest to vertex P_0 it must be right of P_0P_{01} and left of P_0P_{02}, where P_0P_{02} is defined at right angles to P_0P_1. Using just these edge equations, the closest vertex or edge of $P_0P_1P_2$ to P'' can be determined. The lines P_0P_{01}, P_0P_{02}, P_2P_{21}, P_2P_{22}, etc. and their directions can be precomputed, thus enabling simple applications of the edge equation to determine which part of the triangle the point P'' is closest to. Once determined, the distance can be calculated to that part in the normal way.

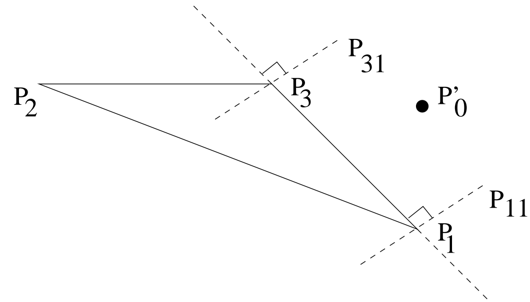

Figure 1: *Calculating point position relative to triangle.*

3.3. Reduction of the number of triangles

As mentioned in the previous section the distance must be calculated to each triangle in order to determine the minimum distance to the surface. It is obvious that some triangles are far away from the voxel in question and should not have their distance calculated from the voxel. The complexity of the algorithm can be reduced further by determining these triangles quickly and eliminating them from the process of calculating the minimum distance from a voxel to a surface.

Using the methods of the previous sections results in a shell of known voxel values surrounding the surface which represents the original triangular mesh. From Section 3.1 we know that the furthest any calculated voxel can be from the surface is the distance covered by three diagonal voxels – $2\sqrt{3}$ units.

A simple method of removing triangles from the process is that of rejecting immediately any triangle whose plane is greater than $2\sqrt{3}$ units away from the current voxel. Using the rotation method of the previous section the distance to the plane is known as soon as the appropriate transformation has been applied to find the transformed x coordinate of the voxel. Since this can be calculated immediately for each triangle, a minimum of calculation is carried out upon the triangle. In addition to this we know that if the transformed y coordinate of the voxel is greater than $2\sqrt{3}$ units away from the minimum and maximum y coordinate of the triangle, then the triangle can be rejected immediately. Similarly for the z coordinate. This introduces just four new comparisons.

This does not remove triangles completely from the process, but does remove them from the *closest part* and distance calculations. It is shown by the results (Section 4) that

© The Eurographics Association and Blackwell Publishers 2000.

Data set	No. of triangles	Distance computations	% of voxels calculated	Time taken
Octahedron	8	97128	5.12%	0.350
Dodecahedron	36	1170754	14.65%	1.467
Soccerball	116	3850470	15.24%	4.216
Teapot	252	5717484	10.98%	7.233
King	3080	17740398	2.67%	24.349
Queen	2600	16378497	2.92%	22.216
Bishop	2360	14092809	2.76%	19.583
Pawn	1600	14791639	4.28%	19.766
Knight	1524	10191575	3.09%	14.133
Rook	1600	18501156	5.35%	24.499

Table 1: *Table showing voxelization timings*

the time to produce volume data is reduced by about 50% using this method.

4. Results

The voxelisation process was implemented in C on a 275Mhz Alpha. The three acceleration steps of Section 3 were included in the implementation. Several meshes were voxelised – octahedron, dodecahedron (Figure 2(a)), soccerball (Figure 2(b)), teapot, and chess pieces (Queen – Figure 3(a)). In each case the triangular mesh was converted into a voxel data set of size 60^3 (216,000 voxels), for which the timing is given in Table 1. The dodecahedron and soccerball (Figures 2(a) and 2(b)) show that the voxelisation process is effective for data which contains flat faces and sharp edges which we wish to be retained. The queen chess piece (Figure 3(a)) demonstrates that fine surface details are not lost by the voxelisation process. The smooth curvature of the piece is retained, and small details such as the ridge halfway up, and the knob on the top are still visible. Although voxelised at 60^3 the piece only occupies an area of 15 voxels squared at the base, and about 3 voxels squared at the apex. The image in Figure 3(b) shows pawn data combined with CT head data and demonstrates that one single visualisation technique can be used to display data which is originally from different sources. In fact the facets visible at the base of the pawn show that in this case the voxelisation is limited by the original triangular mesh.

The mesh data set and the number of triangles in the data set is given in the first two columns of Table 1. The third column gives the number of times the distance was computed between a voxel and a triangle. The percentage of the number of voxels that enter the distance calculation to the number of total voxels is given in column 4, with the time taken to produce the voxel data, in seconds, given in the final column.

In order to show the results of the acceleration techniques over the naive method an additional table (Table 2) shows the results for the queen data set which can be regarded as typical. The first line gives the timing for the brute force naive method. The second line gives the timing for a method using the first two optimisations, with the final giving the timing for all three methods.

Algorithm	Time taken
Naive method	30 mins.
With 3.1 and 3.2	52.331 secs.
With all methods	22.216 secs.

Table 2: *Table showing voxelization timings for different acceleration techniques.*

The final comparison was that of the voxelisation method and subsequent visualisation compared with the rendering of the original mesh using POV-Ray. The queen data set was used for this comparison, and the results are presented in Table 3. As can be seen, the combined voxelisation and direct surface rendering of the data set is quicker than the POV-Ray ray-tracing of the original mesh. In this case the results of the direct surface rendering visualisation are more pleasing than the POV-Ray rendering (Figure 4). This is due to the fact that the queen mesh has been smoothed as a result of the voxelisation process. For the record, the voxel data set occupies 860Kbytes and the POV-Ray text input file occupies 650KBytes.

Future extensions to this work could involve extending visual effects by incorporating shadows, fog and global illumination complementing the already existing reflectance model.

In conclusion, a useful method for the voxelisation of triangular meshes has been presented, along with the visualisation process. This method works well and robustly for

© The Eurographics Association and Blackwell Publishers 2000.

Computation	Time taken
Voxelisation	22.2 secs.
Direct surface rendering	42.6 secs.
POV-Ray rendering	120.0 secs.

Table 3: *Table showing voxelization and POV-Ray comparison.*

arbitrary closed meshes, and computes the data in a realistic time. In order to achieve this three acceleration methods were described and implemented. The first was to reduce the number of voxels the distance is computed for, the second was to achieve an efficient implementation of the point to triangle distance function, and the third was to reduce the number of triangles the distance calculation is performed upon. The results of the method have been visualised using a general volume visualisation technique, and inspection shows the images to be accurate with respect to the original model. Some smoothing of the voxelised data takes place due to the distance function used, which can be desirable. Comparisons have been made between this rendering process and a conventional ray-tracing package - POV-Ray, which show that equivalent images are produced, smoother surfaces are rendered and less time is taken to produce the images. It is envisaged that as scenes become more complex the voxelisation and subsequent direct surface rendering process will scale better than standard ray-tracing.

References

1. M. Chen, M. W. Jones, and P. Townsend. Methods for volume metamorphosis. In *European Workshop on Combined Real and Synthetic Image Processing for Broadcast and Video Production (Hamburg, Germany)*, November 1994.
2. T. T. Elvins. A survey of algorithms for volume visualization. *Computer Graphics*, 26(3):194–201, August 1992.
3. A. Kaufman. An algorithm for 3D scan-conversion of polygons. *Proc. of Eurographics '87, Amsterdam, The Netherlands*, pages 197–208, August 1987.
4. G. Frieder, D. Gordon, and R. A. Reynolds. Back-to-front display of voxel-based objects. *IEEE Computer Graphics and Applications*, 5(1):52–60, January 1985.
5. D. Gordon and J. K. Udupa. Fast surface tracking in three dimensional binary images. *Computer Vision, Graphics, and Image Processing*, 45:196–214, 1989.
6. J. D. Foley, A. Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics, Principles and Practice, 2nd ed.* Addison Wesley, 1990.
7. D. Gordan and J. K. Udupa. Image space shading of three-dimensional objects. *Computer Vision, Graphics, and Image Processing*, 29:361–376, 1985.
8. D. Cohen, A. Kaufman, R. Bakalash, and S. Bergman. Real time discrete shading. *The Visual Computer*, 6:16–27, 1990.
9. 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.
10. W. E. Lorensen and H. E. Cline. Marching Cubes : A high resolution 3D surface construction algorithm. In *Proc. SIGGRAPH '87 (Anaheim, Calif., July 27-31, 1987)*, volume 21(4), pages 163–169. ACM SIGGRAPH, New York, July 1987.
11. M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.
12. P. Sabella. A rendering algorithm for visualizing 3D scalar fields. In *Proc. SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988)*, volume 22(4), pages 51–57. ACM SIGGRAPH, New York, August 1988.
13. L. Carpenter R. A. Drebin and P. Hanrahan. Volume rendering. In *Proc. SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988)*, volume 22(4), pages 65–74. ACM SIGGRAPH, New York, August 1988.
14. L. Westover. Footprint evaluation for volume rendering. In *Proc. SIGGRAPH '90 (Dallas, Texas, August 6-August 10, 1990)*, volume 24(4), pages 367–376. ACM SIGGRAPH, New York, August 1990.
15. C. Upson and M. Keeler. V-Buffer : Visible volume rendering. In *Proc. SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988)*, volume 22(4), pages 59–64. ACM SIGGRAPH, New York, August 1988.
16. R. A. Crawfis and N. Max. Texture splats for 3D scalar and vector field visualization. In *Proc. Visualization 93*, pages 261–266. IEEE CS Press, Los Alamitos, Calif., 1993.
17. N. Max. Optical models for volume rendering. In *Visualization in Scientific Computing*, pages 35–40. Springer-Verlag, Wein New York, January 1995.
18. H. Meinzer, K. Meetz, D. Scheppelmann, U. Engelmann, and H. J. Baur. The heidelberg ray tracing model. *IEEE Computer Graphics and Applications*, 11(6):34–43, November 1991.
19. W. Krueger. The application of transport theory to visualization of 3D scalar data fields. In *Proc. Visualization 90*, pages 273–280. IEEE CS Press, Los Alamitos, Calif., 1990.
20. A. Kaufman and R. Bakalash. Memory and processing

© The Eurographics Association and Blackwell Publishers 2000.

architecture for 3D voxel-based imagery. *IEEE Computer Graphics and Applications*, 8(6):10–23, November 1988.

21. H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel. Pixel-Planes5: A heterogenous multiprocessor graphics system using processor-enhanced memories. In *Proc. SIGGRAPH '89 (Boston, Mass., July 31-August 4, 1989)*, volume 23(3), pages 79–88. ACM SIGGRAPH, New York, July 1989.
22. T. S. Yoo, U. Neumann, H. Fuchs, S. M. Pizer, J. Rhoades T. Cullip, and R. Whitaker. Direct visualization of volume data. *IEEE Computer Graphics and Applications*, 12(4):63–71, July 1992.
23. D. Cohen and C. Gotsman. Photorealistic terrain imaging and flight simulation. *IEEE Computer Graphics and Applications*, 14(2):10–12, March 1994.
24. 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.
25. 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.
26. 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.
27. 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.
28. J. Pineda. A parallel algorithm for polygon rasterization. In *Proc. SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988)*, volume 22(4), pages 17–20. ACM SIGGRAPH, New York, August 1988.

© The Eurographics Association and Blackwell Publishers 2000.

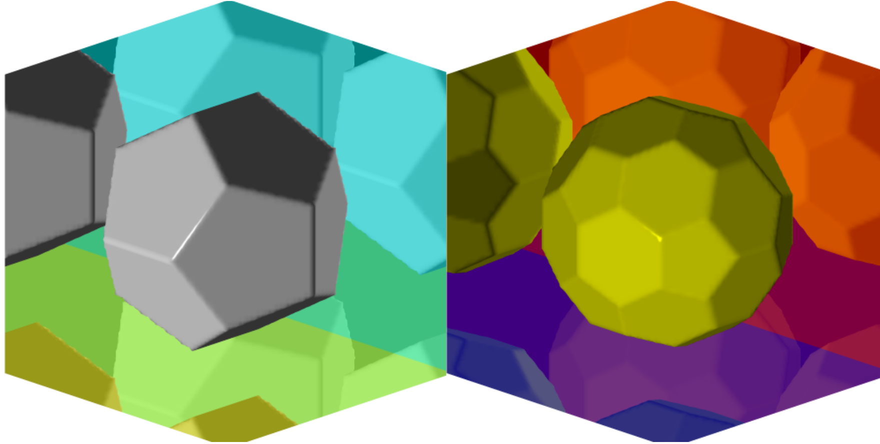

Figure 2: *(a) Voxelised dodecahedron. (b) Voxelised soccerball.*

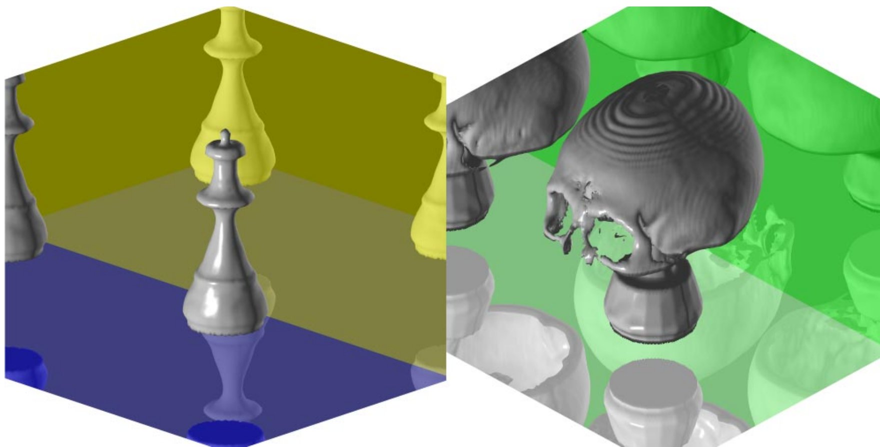

Figure 3: *(a) Voxelised chess piece (b) Voxelised CThead and pawn.*

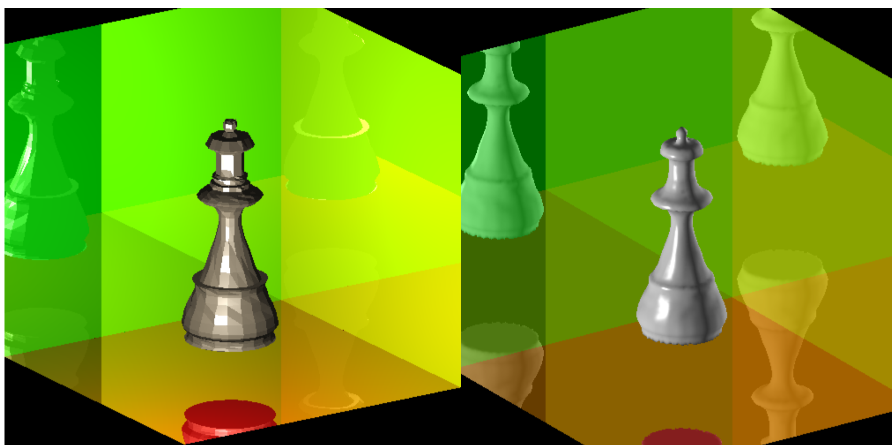

Figure 4: *(a) Mesh rendered using POV-Ray. (b) Voxelised mesh direct surface rendered.*

© The Eurographics Association and Blackwell Publishers 2000.