This straightforward 3-D display algorithm traverses voxels slice by slice to project each voxel on the screen. No surface detection or z-buffer is needed.

Back-to-Front Display of Voxel-Based Objects

Gideon Frieder
University of Michigan

Dan Gordon
University of Cincinnati

R. Anthony Reynolds
University of Pennsylvania

With the advent of medical imaging devices such as computed tomography, positron emission tomography, single photon emission computed tomography, magnetic resonance imaging, and ultrasound scanners, methods are widely available for capturing the shape and other properties of real objects (human organs) directly in machine-readable form. One way of presenting the information for review is to display the organ as a three-dimensional object, using established computer graphics techniques for creating images by projection onto a two-dimensional surface such as the screen of a raster-scan device.

Many approaches to 3-D display are available; both hardware and software methods have been reviewed recently.[1,2] Most methods require some preprocessing of the input to reduce data and to provide object representations better suited to the available display algorithms. One approach that has been applied widely to medical applications uses an automatic boundary-detection algorithm to extract the surface of a single connected object from the data.[3-5] A second approach extracts a set of 1-D primitives (contours) describing the boundaries of the object on a slice-by-slice basis. Surface representations can be obtained from the contours directly, or indirectly by tiling or by spline techniques.[6-14] A third approach retains 3-D voxels (cubes or rectangular parallelepipeds) as primitives but achieves data compression through octree encoding, which provides an efficient object representation.[15,16]

While surface representations achieve extensive data reduction, surface formation is time-consuming. In addition it is not possible to explore the interior of the object interactively, or generate cut-away views, without forming a new surface. The back-to-front (BTF) method presented in this article displays entire solid objects composed of voxels without extracting the object surface first and with a minimum of preprocessing.

Data collection and the cuberille model

One representation that corresponds closely to the format in which data are collected by medical imaging systems stores the entire object in terms of voxels. A voxel is a rectangular volume element obtained when space is divided by three sets of parallel planes, each set being orthogonal to the other two. The voxels making up an object are usually the same size; that is, all the planes in a given set are equally spaced. On the other hand, the spacing of one set of planes need not be the same as the spacing of another. For example, when a patient is examined with a computed tomography (CT) scanner, the voxels are usually rectangular parallelepipeds with a square cross section within a transverse slice and maximum dimension along the long axis of the patient (Figure 1). Associated with each voxel are three integer coordinates representing

0272-1716/85/0100-0052$01.00 © 1985 IEEE

its location in space, and an integer called its density, representing some object property at this location (x-ray attenuation, radiopharmaceutical concentration, etc.). The voxels can be converted to cubes by suitable interpolation, and we will henceforth assume that all voxels to be displayed are cubical in shape. Such a dissection of space into cubes is called a cuberille.[3,17,18]

Previous methods for direct display of voxel-based objects

Display methods that do not extract or fit surfaces to the data but rather process all the voxels in the scene at display time have been developed largely as a result of the work of Meagher on octree encoding.[15,16] The use of octrees for object representation apparently was suggested first by Hunter;[19] in some respects, octrees can be considered to be a special case of the tree structures devised for hidden-surface removal by Schumacker[20] and later extended by Fuchs et al.[21] Octrees and related approaches have been reviewed by Srihari.[22] The basic idea is that the voxels making an object are represented by a hierarchical 8-ary tree structure, which achieves data compression through spatial coherence. An advantage of octree encoding is that simple operations (union, intersection, and difference of objects; translation, rotation, and scaling; interference detection and hidden-surface removal) can be accomplished by accessing each node of the tree once at most. Furthermore these operations require only simple arithmetic such as integer additions, shifts, and comparisons. Another useful feature of the octree approach is the ability to trade off computation time against precision: A coarse image can be generated very quickly with the high-frequency details emerging later as more processing is carried out.

Hidden-surface removal can be accomplished by reading out the cubes that correspond to nodes of the octree in a recursive back-to-front sequence (Figure 2). This property, which derives from the 3-D array from which the octree is built, is referred to as "spatial pre-sortedness."[18] Changing the observer's viewpoint corresponds to simply visiting the nodes of the tree in a different sequence; no modification of the octree is required. Front-to-back readout is also possible and is more efficient in the average case: Once a region of the screen has been painted, nodes projecting on it can be ignored.[16] On the other hand, building an octree requires more steps than simply preparing an array of voxels for display, and, on a conventional computer, traversing a tree incurs more overhead than sequentially accessing the elements of an array.

A front-to-back algorithm was proposed by Strat as a means of realistically depicting mountains and other geographical features on flight simulators.[23] Strat's algorithm addresses a special case of the 3-D problem, since each input (x,y) point has a unique z (height) associated with it. A front-to-back octree display algorithm was given by Meagher.[16] Simplified front-to-back display methods have been used by Vannier et al. and Gibson for display of medical objects.[24,25] Their methods are less general than ours in that only certain viewing directions are permitted. Similarly restrictive octree display algorithms have been given by Doctor and Torborg.[26] An alternative slice-by-slice front-to-back approach has been given by Farrell et al.[27] Their method differs from others in that the data are rotated and new slices are constructed before display. Methods of ray-tracing voxels have been

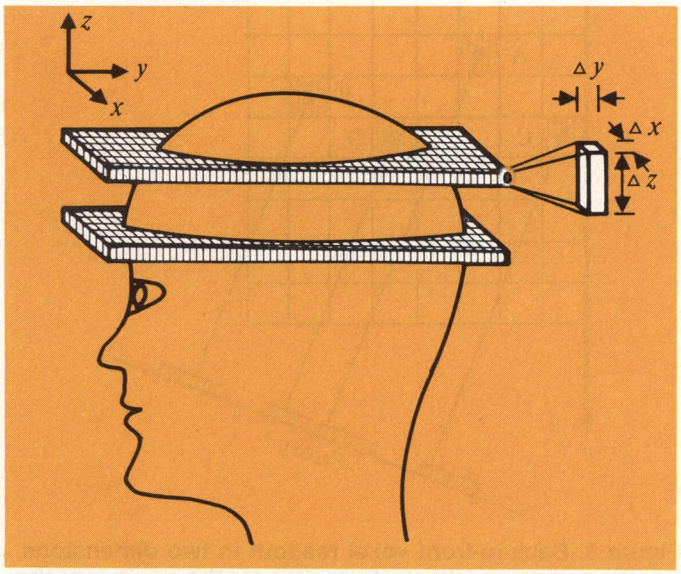


Figure 1. Typical CT slices. $\Delta x = \Delta y$ = pixel size; Δz = slice thickness.

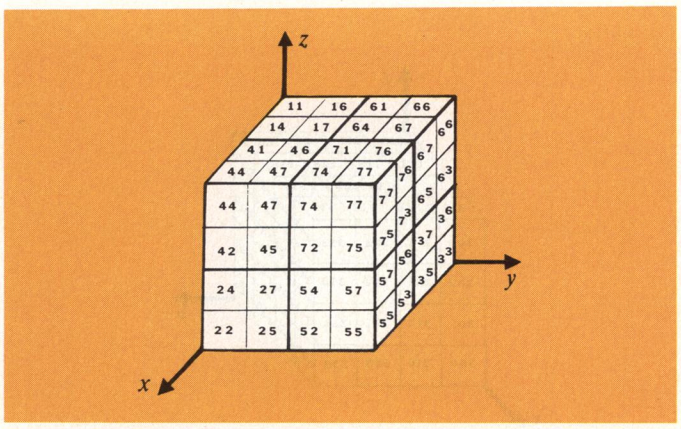


Figure 2. Recursive back-to-front voxel readout. Note that 01234567, 02134567, and 03216547 are some possible octant readout sequences for the object orientation shown. The object in this example could be represented by an octree of depth 2: Voxels are labeled with two digits, the first identifying the octant, the second identifying the voxel within the octant. The voxels can be read out in several recursive back-to-front sequences including 00, 01, 02, 03...07, 10, 11, 12...74, 75, 76, 77.

proposed by Gordon and implemented by Tuy and Tuy.[28,29]

The back-to-front display method

The BTF algorithm for three-dimensional arrays, first proposed by Gordon, is based on traversing the slices, rows, and columns of the array in order of decreasing distance to the observer. For purposes of exposition, a 2-D analog of the problem is presented, using a binary picture. Assume that in Figure 3 the voxels A,B,C,D are full and are to be projected onto the screen; all other voxels are empty. Assume the x,y axes are arranged as in the diagram, so that the origin is farthest from the observer. The voxels should be traversed in order of increasing values of x and y; either x or y may be chosen as the fastest changing index. Thus, the sequences A,B,C,D (x changes fastest) or C,A,D,B (y changes fastest) will both result in a correct rendition of the scene. The reason for this is that if (part of) a voxel with coordinates (x,y) is obscured by (part of) a voxel with coordinates (x',y'), then $x \leq x'$ and $y \leq y'$ so that by projecting (x,y) before (x',y'), the correct image is obtained.

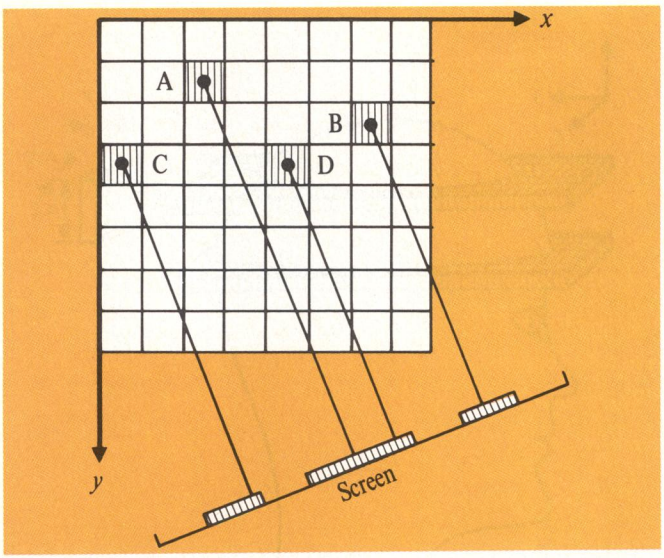


Figure 3. Back-to-front voxel readout in two dimensions. For the purpose of illustration, assume the origin is at the corner farthest from the observer. The voxels are traversed in order of increasing values of *x* and *y*; either *x* or *y* may be chosen as the faster running index.

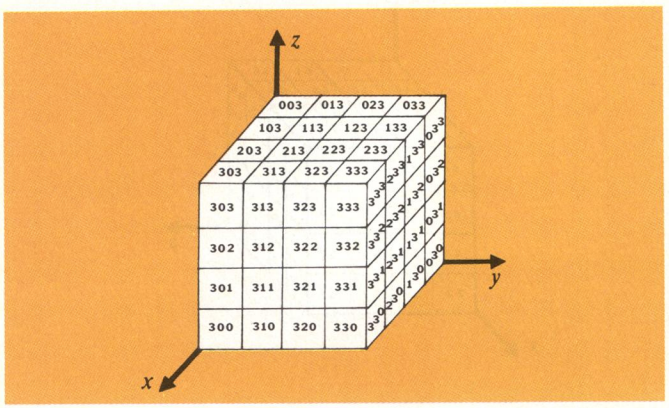


Figure 4. Slice-by-slice back-to-front voxel readout. Slices are read out starting with the slice farthest from the observer. Within each slice, voxels are read out staring with the corner farthest from the observer. The voxels are labeled with three digits, identifying the *x*, *y*, and *z* coordinates in object space. For the object orientation shown, the voxels can be read out in several slice-by-slice back-to-front sequences including 000, 100, 200...010, 110, 210...133, 233, 333.

The correctness of the three-dimensional method is now obvious (Figure 4). Assuming the origin is farthest from the observer, then it is necessary to simply traverse the voxels in the 3-D array in order of increasing x, y, and z. If the origin is not the farthest corner from the observer, some of x,y,z should be increasing and some should be decreasing. However, the choice of which index changes fastest can be arbitrary. This fact can be used to advantage in the case of data that are stored slice by slice, making it possible to read one slice (or portion of a slice) at a time from disk storage into main memory for processing. Once a slice is read in, the voxels in it are projected in the appropriate increasing or decreasing sequence of x and y values. Note that the order in which voxels are projected on the screen is not in strictly decreasing distance, so our algorithm is not an example of the well-known "painter's algorithm."

The following conventions are useful in the description of the BTF algorithm. We refer to the coordinate system in which the voxels of the object are defined as object space and the coordinate system in which they are displayed as image space (Figure 5). Object space coordinates are specified by integers, and object space and image space are related by affine transformations, the products of rotations, translations, and dilations (scaling). We use orthographic projections, since these preserve distances and other important geometrical information for direct measurement. For implementational efficiency we usually threshold the data on the basis of density; that is, we ob-

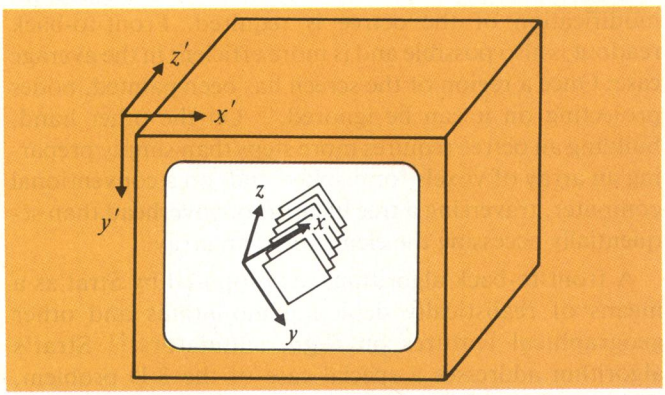


Figure 5. Object space *(x, y, z)* and image space *(x ', y ', z ')*.

tain binary data by assigning 1 to those voxels whose density lies within a given range, and 0 otherwise. We assume that a single (imaginary) light source is used to illuminate the object, and that it is coincident with the observer's position; thus we do not describe methods to generate shadows. Lastly, we assume that the surfaces of the objects are diffuse reflectors, and our present shading implementation does not produce highlights.

Implementation of the back-to-front method

We have implemented a back-to-front program on the Data General Eclipse S/140 minicomputer, using Fortran programming. This machine is representative of the minicomputers supplied as standard equipment on most CT scanners (e.g., General Electric 8800 and 9800 CT/T). One disadvantage of such minicomputers is their small address space (32K 16-bit words). For efficient main memory utilization, our program displays binary objects (one bit per voxel), although on a machine with more main memory, gray-scale data could be used (8-16 bits per voxel). The object to be displayed is assumed to be divided into slices perpendicular to the z axis. The x and y axes are attached to the object as shown in Figure 5.

To specify any desired orientation of an object for display, the user must specify three angles, *A*, *B*, and *C*, which result in rotations about three predetermined axes. We have chosen our angles to correspond to manipulations of the object that are natural to the medical user. Imagine a patient lying supine (face up) in a CT scanner, with feet toward the observer; axes through the center of the patient are labeled x_1, y_1, and z_1. The angle A indicates swivel about the longitudinal z_1 axis (Figure 6), the angle B indicates tilt about the x_1 axis, and the angle C indicates rotation about the y_1 axis. The transformations are applied in the order given (first swivel, then tilt, then vertical rotation).

The physical organization of the data prescribes the order in which the voxels are traversed for display generation. For the particular case of CT data, the voxels are traversed as follows:

(1) Outer loop on the slices (z-direction).
(2) Within each slice, an outer loop on the rows (y-direction).
(3) Innermost loop in the x-direction.

This traversal is chosen because CT data are usually organized in a slice-by-slice fashion, with the data within each slice stored row by row.

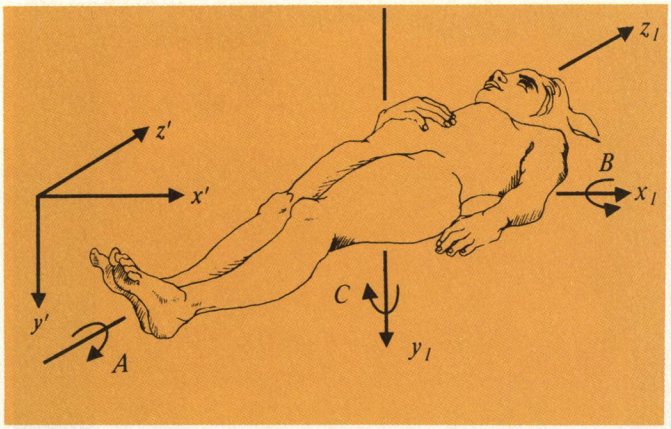


Figure 6. Angles *A* (swivel), *B* (tilt), and *C* (vertical rotation) are entered by the user to specify the object orientation.

Given a voxel with coordinates (x,y,z) in object space, its coordinates in image space are given by

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \mathbf{T} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + \begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix} \qquad (1)$$

where:

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix};$$

$\mathbf{T} = [t_{ij}]$ is the rotation matrix (see box this page); $[2x_c, 2y_c, 2z_c]$ are the dimensions of a bounding box enclosing the object; $[2x_s, 2y_s, 2z_s]$ are the display device bounding dimensions (screen size is $2x_s \times 2y_s$, and $2z_s$ is the number of gray levels); and S is a scale factor.

The matrix $\mathbf{T}$ is computed after the angles A, B, C, and the scale-factor S have been read in. Table lookups are used for the transformations as follows:

(1) Each slice determines a value of z, so the three products $t_{13}z_1$, $t_{23}z_1$, $t_{33}z_1$ are computed just once for each slice.

(2) All possible values for $t_{12}y_1$, $t_{22}y_1$, $t_{32}y_1$ are stored in three arrays. For any given y (whose value is always an integer) these products are retrieved from the arrays using y as an index.

(3) All possible values for $t_{11}x_1$, $t_{21}x_1$, $t_{31}x_1$ are stored in three arrays and retrieved as needed.

Rotation matrix

$$\mathbf{T} = [t_{ij}] = S \begin{bmatrix} \cos A \cos C + \sin A \sin B \sin C & -\sin A \cos C + \cos A \sin B \sin C & \cos B \sin C \\ \sin A \cos B & \cos A \cos B & -\sin B \\ -\cos A \sin C + \sin A \sin B \cos C & \sin A \sin C + \cos A \sin B \cos C & \cos B \cos C \end{bmatrix};$$

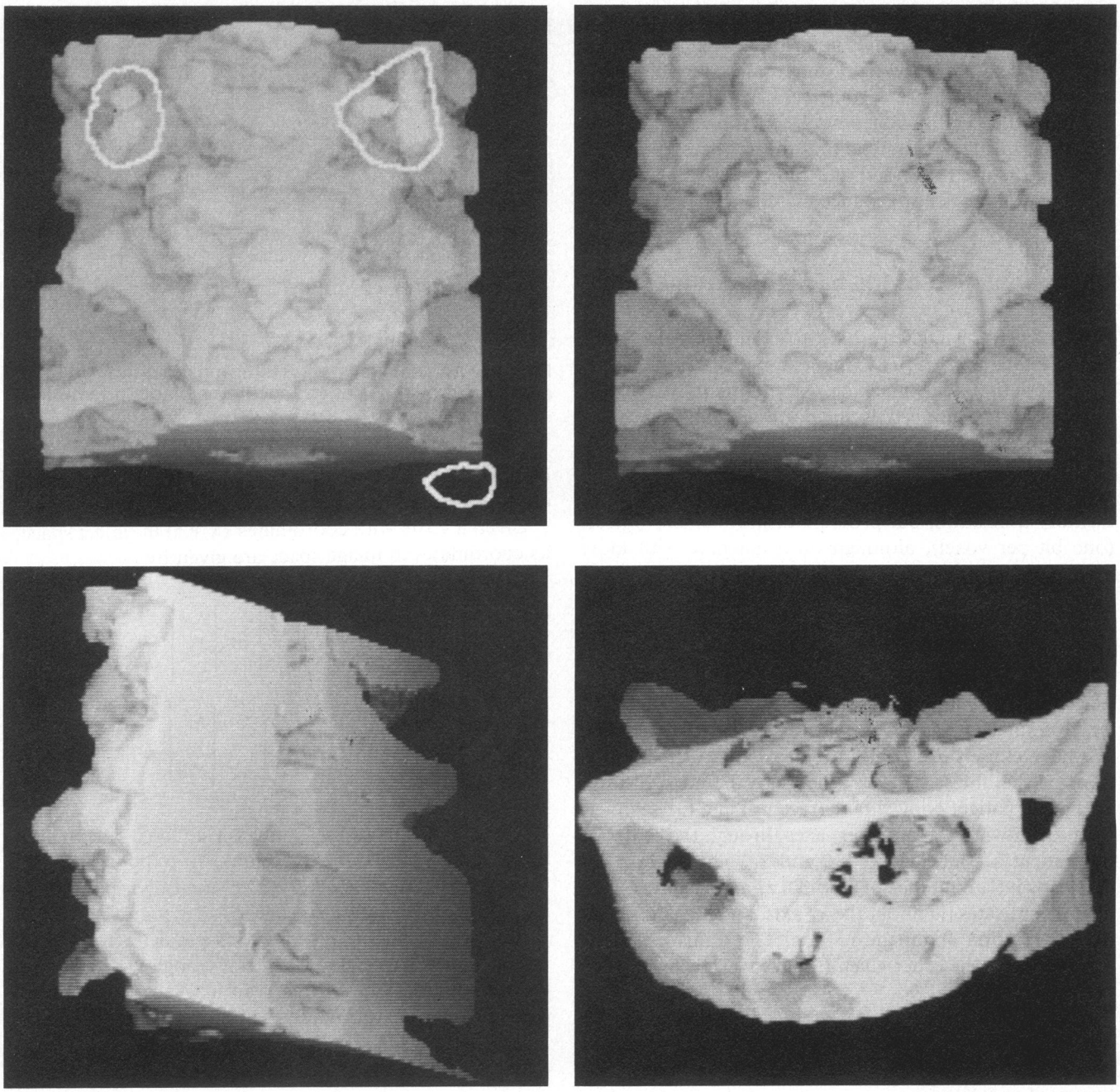


Figure 7. Three views of a human spine and one of a human skull: User outlines material to be removed with a trackball (a); unobscured view with unwanted material removed (b); part of the same spine cut open with object space clipping plane to reveal internal structure (c); and a skull of another patient (d).

In a typical setting, the bounding box could be $256 \times 256 \times 256$, and each application of Equation 1 requires nine multiplications. This would have resulted in $9 \times 2^{24} = 1.51 \times 10^8$ multiplications instead of the $9 \times 256 = 2304$ required by our use of look-up tables. A further saving is achieved by computing the sums $t_{12}\, y_1 + t_{13} z_1$, $t_{22}\, y_1 + t_{23}\, z_1$ and $t_{32}\, y_1 + t_{33}\, z_1$ just once for each row of a slice.

The BTF program produces a depth-shaded image, which is somewhat lacking in fine detail. This image is used as input to our gradient-shading method, which uses both the distance from the light source and the object surface orientation to compute the intensity to be assigned to each pixel.[30] We used gradient shading to produce the images shown in Figure 7.

Before our BTF program can be applied, the slice data

Table 1.
Statistics for the objects displayed in Figure 7. These examples have been chosen to illustrate the wide range of object size and complexity encountered in clinical applications. Timings are shown for gradient-shaded images; depth-shaded images take 25-30 seconds less.

Figure	Preprocessing without AP* (min./sec.)	Preprocessing with AP* (min./sec.)	Bounding Box	Number of Full Voxels	Display with BTF (min./sec.)
7a	14/35	6/35	205×251×178	3,015,430	7/12
7b			205×251×178	3,004,880	7/09
7c			100×251×178	1,363,060	3/37
7d	3/44	2/26	20×147×77	222,228	1/09

*Array processor

from the CT scanner must be converted into the cuberille format. This preprocessing is not strictly part of the BTF algorithm (since similar conversion is required by many other 3-D display techniques); its details are described in Frieder et al.[31] Table 1 lists the preprocessing times, with and without an array processor. The array processor (Floating Point Systems AP120B) is standard equipment on GE 8800 CT scanners.

Interaction and further developments

Because the BTF method accesses all voxels of the scene at display time (not just those voxels making up the object surface), several simple techniques are available for interacting with and modifying the object. These interactive techniques can be used to plan and simulate surgical procedures or simply to obtain an unobstructed view of an organ of interest.[32] We describe three interactive options here.

The simplest way to remove parts of the object is to use object space clipping planes orthogonal to the axes; these can be implemented by changing the loop limits for the x,y,z indices. An alternative is to use image space clipping planes, fixed with respect to the observer; these are also easily implemented, since voxels that lie on the observer's side of a clipping plane are simply ignored. Clipping planes result in cut-away views of the object's interior (Figure 7c). A third option is to directly specify parts of the object to be altered or deleted. In our implementation, the observer uses a trackball to outline a cylinder perpendicular to the screen surface and specifies its depth; voxels whose projections fall within the cylinder during the BTF process are deleted from the input file. We have found this type of interaction most useful in removing unwanted parts of the scene to obtain an uncluttered display (Figure 7a,b).

Our BTF program is capable of generating arbitrary views of complex objects from a binary voxel representation in just a few minutes (Table 1); the preprocessing required to obtain the binary file takes from three to 15 minutes (two to seven minutes with an array processor). We have experimental evidence that the BTF computation is dominated by the number of 1-voxels to be transformed and displayed. For typical objects, approximately six percent of the running time is spent reading the input file, and another six percent is spent scanning empty words (i.e., 16 0-voxels packed into a single machine word). On the other hand, approximately 50 percent of the full voxels encountered will be obscured later by a full word (i.e., 16 1-voxels packed into a single machine word). We have exploited these observations in a special-purpose program, which displays objects from the six major viewing directions only (along the positive and negative coordinate axes). This choice of special viewing directions avoids the need for coordinate transformations and allows a simple "look-ahead" procedure to avoid displaying any full voxel that will be obscured later by a full word. The special program runs approximately five times faster than the general BTF program.

As mentioned earlier, one advantage of octree encoding is that a rough (low-resolution) image can be generated quickly. The following modification of the BTF method will achieve the same result. The data are packed so that each byte codes a $2\times2\times2$ cube of voxels. For a low-resolution image, if a byte has more than four bits set, it is considered full, otherwise empty; the number of bits set is obtained from the byte value (zero to 255) by a look-up table. Processing the data byte by byte would produce an image of one half the linear resolution in approximately one eighth of the time; for a normal-resolution image the bytes are unpacked, and all the voxels are traversed back to front in the normal fashion.

In our implementation of the BTF display method, we have restricted the scale factors so that they will not be greater than unity. This restriction is imposed because, unless the pixels of the image are larger than the voxels, not all pixels will be painted in, and artifact holes could

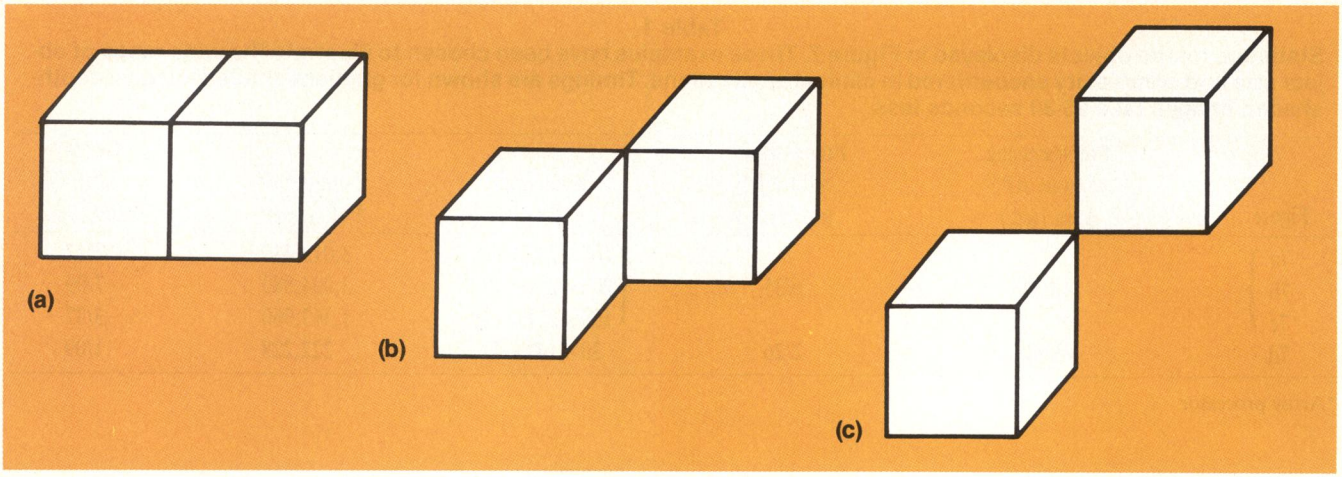


Figure 8. The maximum scale factor depends on the definition of object connectivity chosen: face-connected voxels (a); edge-connected voxels (b); and corner-connected voxels (c).

result at certain orientations. Specifically, the scale factor can be at most 1 for a head-on view, $1/\sqrt{2}$ if the object is rotated about a single axis, and $1/\sqrt{3}$ for arbitrary orientations. In fact, $1/\sqrt{2}$ can be used for arbitrary orientations by accepting the convention that corner-connected voxels may appear disconnected, but edge-connected voxels may not (Figure 8).

One way of allowing arbitrary scale factors is to replace each voxel by a sphere whose projection is a disk of diameter $a\sqrt{3}$, where a is the size of the voxel after scaling. Fast methods of rendering disks have been devised by Badler.[33] Another approach magnifies the 2-D image after display. However, the use of large voxels or large pixels (magnified displays) is undesirable because of the loss of real spatial information and the occurrence of aliasing errors.

One approach that does not result in loss of resolution considers each voxel to be a cube with three visible faces to be painted with a standard polygon-fill algorithm; this is the solution adopted in the 3D83 package.[4] We have not implemented this option in the BTF program because of the associated time penalty. However, we are currently examining faster scan-conversion methods that achieve the same result.

We have presented a simple technique for displaying voxel-based objects. The technique can operate either directly on gray-scale data or on binary data derived from them. The latter form is particularly amenable to interactive dissection and manipulation. Interaction can be accomplished efficiently because of straightforward transformations back and forth between screen coordinates, object coordinates, and physical location of voxels.

The main advantage of our technique is that it can be used immediately after the binary array is formed, and no further preprocessing is required even after user interaction. Surface formation methods require the further step of boundary detection, which must be repeated after cutting away part of the object. Octree methods require additional preprocessing to create the tree structure, plus pointer dereferencing operations to access each node. Our method gives the user quick views of the object soon after the gray-scale data are available and permits reasonably fast interaction using modest computer equipment.

We have illustrated the method with views of medical objects; however, the method can be adapted to any application where objects are represented by three-dimensional arrays of voxels. Figures 7a, b, and c show parts of the cervical spine of a live patient. In Figure 7a, some unwanted objects, which are disconnected from the organ of interest, are clearly discernible; we show how these can be identified and removed interactively to give an unobscured view (Figure 7b). Figure 7c shows part of the same spine cut open with a clipping plane to reveal the spinal canal for further study. Figure 7d shows the skull of another live patient.

The shading effects in these illustrations were produced by running a gradient operator over depth-shaded preimages.[30] Simple antialiasing techniques were used to remove the jagged appearance of diagonal silhouette edges (vertical and horizontal edges were left unchanged).

Table 1 shows relevant statistics for the objects shown in Figure 7. The complete spine is one of the largest objects one would expect to encounter in routine clinical practice; timings 7a and 7b can be interpreted as upper limits for our method. Many clinical objects fall within the ranges of 7c and 7d. These timings indicate that complex objects can be displayed in a few minutes on a minicomputer of modest capability using the BTF technique.

We turn now to the complex issue of comparing the speed of the BTF approach with that of other direct display methods. Unfortunately, comparative timing measurements for the various methods on similar computing equipment have not been published. We have per-

formed experiments that indicate that slice-by-slice BTF is faster than recursive BTF (Figure 2) or ray tracing when implemented on a conventional minicomputer. Comparing slice-by-slice BTF with a front-to-back method—using, for example, octrees—is an interesting subject for further research. The strength of the BTF method lies in its simplicity. The advantage of the front-to-back method is that once a region of the screen has been painted, subsequent voxels in the object (or nodes of the octree) that project on it need not be processed; the disadvantage is that a complex data structure is required to efficiently determine which regions of the screen have already been painted.

On the other hand, when special computer architectures are devised, many problematic issues can be avoided both with octree encoding and with BTF. A high-performance octree display system is available from Phoenix Data Systems, Inc., Albany, New York. A small hardware prototype of a recursive BTF system, the Voxel Processor, is now operational in the GRASP Laboratory, University of Pennsylvania, and achieves display rates of 16 frames per second.[34,35] ■

Acknowledgments

We wish to acknowledge many helpful discussions with L. S. Chen, G. T. Herman, S. S. Trivedi, and J. K. Udupa of the Medical Image Processing Group. We wish to thank D. J. Meagher, M. W. Vannier, and the anonymous referees for their helpful comments. Thanks are also due to G. T. Herman for the patient data used in Figure 7 and to D. W. Ro and S. Strommer for photography. Part of the work was carried out using the Medical Image Processing Group computing facility. This work was supported by NIH grant HL28438. The GRASP Laboratory is supported by grants ARO DAA6-29-84-K-0061, AFOSR 82-NM-299, NSF MCS-8219196-CER, NSF MCS 82-07294, AVRO DAABO7-84-K-FO77, and NIH 1-RO1-HL-29985-01.

References

1. R. A. Reynolds, "Some Architectures for Real-Time Display of Three-Dimensional Objects: A Comparative Survey," tech. report MIPG84, Dept. of Radiology, Univ. of Pennsylvania, Oct. 1983.
2. J. K. Udupa, "Display of 3D Information in Discrete 3D Scenes Produced by Computerized Tomography," *Proc. IEEE,* Vol. 71, No. 3, Mar. 1983, pp. 420-431.
3. G. T. Herman and J. K. Udupa, "Display of 3-D Digital Images: Computational Foundations and Medical Applications," *IEEE Computer Graphics and Applications,* Vol. 3, No. 5, Aug. 1983, pp. 39-46.
4. L. S. Chen, G. T. Herman, C. M. Meyer, R. A. Reynolds, and J. K. Udupa, "3D83—An Easy-to-Use Software Package for Three-Dimensional Display from Computed Tomograms," *IEEE Computer Soc. Int'l Symp. Medical Images and Icons,* Arlington, Va., July 1984, pp. 309-316.
5. E. Artzy, G. Frieder, and G. T. Herman, "The Theory, Design, Implementation and Evaluation of a Three-Dimensional Surface Detection Algorithm," *Computer Graphics and Image Processing,* Vol. 15, Jan. 1981, pp. 1-24.
6. J. K. Udupa, "Interactive Segmentation and Boundary Surface Formation for 3D Digital Images," *Computer Graphics and Image Processing,* Vol. 18, 1982, pp. 213-235.
7. E. Keppel, "Approximating Complex Surfaces by Triangulation of Contour Lines," *IBM J. Research and Development,* Vol. 19, Jan. 1975, pp. 2-11.
8. J. C. Mazziotta and K. H. Huang, "THREAD (Three-Dimensional Reconstruction and Display) with Biomedical Applications in Neuron Ultrastructure and Computerized Tomography," *AFIPS Conf. Proc.,* Vol. 45, 1976 NCC, pp. 241-250.
9. H. N. Christiansen and T. W. Sederberg, "Conversion of Complex Contour Line Definitions into Polygonal Element Mosaics," *Computer Graphics* (Proc. Siggraph 78), Vol. 12, No. 3, Aug. 1978, pp. 187-192.
10. D. L. McShan, A. Silverman, D. M. Lanza, L. E. Reinstein, and A. S. Glicksman, "A Computerized Three-Dimensional Treatment Planning System Utilizing Interactive Color Graphics," *British J. Radiology,* Vol. 52, 1979, pp. 478-481.
11. L. T. Cook, S. J. Dwyer, S. Batnitsky, and K. R. Lee, "A Three-Dimensional Display System for Diagnostic Imaging Applications," *IEEE Computer Graphics and Applications,* Vol. 3, No. 5, Aug. 1983, pp. 13-19.
12. D. S. Schlusselberg, W. K. Smith, M. H. Lewis, B. G. Culter, and D. J. Woodward, "A General System for Computer Based Acquisition, Analysis and Display of Medical Image Data," *Proc. ACM Ann. Meeting,* Oct. 1982, pp. 18-25.
13. H. Fuchs, G. D. Abram, and E. D. Grant, "Near Real-Time Shaded Display of Rigid Objects," *Computer Graphics* (Proc. Siggraph 83), Vol. 17, No. 3, July 1983, pp. 65-72.

14. A. Sunguroff and D. Greenberg, "Computer Generated Images for Medical Applications," *Computer Graphics* (Proc. Siggraph 78), Vol. 12, No. 3, Aug. 1978, pp. 196-202.

15. D. Meagher, "Geometric Modelling Using Octree Encoding," *Computer Graphics and Image Processing,* Vol. 19, 1982, pp. 129-147.

16. D. Meagher, "The Octree Encoding Method for Efficient Solid Modelling," PhD dissertation, Rensselaer Polytechnic Institute, Aug. 1982.

17. G. T. Herman and H. K. Liu, "Three-Dimensional Display of Human Organs from Computed Tomograms," *Computer Graphics and Image Processing,* Vol. 9, Jan. 1979, pp. 1-21.

18. G. T. Herman, R. A. Reynolds, and J. K. Udupa, "Computer Techniques for the Representation of Three-Dimensional Data on a Two-Dimensional Display," *Proc. SPIE,* Vol. 367, 1982, pp. 3-14.

19. G. M. Hunter, "Efficient Computation and Data Structures for Graphics," PhD dissertation, Dept. of Electrical Eng. and Comp. Sci., Princeton Univ., June 1978.

20. I. E. Sutherland, R. F. Sproull, and R. A. Schumacker, "A Characterization of Ten Hidden Surface Algorithms," *Computing Surveys,* Vol. 6, No. 1, Mar. 1974.

21. H. Fuchs, Z. M. Kedem, and B. F. Naylor, "On Visible Surface Generation by A Priori Tree Structures," *Computer Graphics* (Proc. Siggraph 80), Vol. 14, No. 3, 1980, pp. 124-133.

22. S. N. Srihari, "Representation of Three-Dimensional Digital Images," *Computing Surveys,* Vol. 13, No. 4, 1981, pp. 399-424.

23. T. M. Strat, "Application of Data Flow Computation to the Shaded Image Problem," working paper 163, A. I. Laboratory, MIT, Cambridge, Mass., May 1978.

24. M. W. Vannier, J. L. Marsh, and J. O. Warren, "Three Dimensional Computer Graphics for Craniofacial Surgical Planning and Evaluation," *Computer Graphics* (Proc. Siggraph 83), Vol. 17, No. 3, July 1983, pp. 263-273.

25. C. J. Gibson, "A New Method for the Three-Dimensional Display of Tomographic Images," *Physics in Medicine and Biology,* Vol. 28, No. 10, 1983, pp. 1153-1157.

26. L. J. Doctor and J. G. Torborg, "Display Techniques for Octree-Encoded Objects," *IEEE Computer Graphics and Applications,* Vol. 1, No. 3, July 1981, pp. 29-38.

27. E. J. Farrell, R. Zappulla, and W. C. Yang, "Color 3-D Imaging of Normal and Pathologic Intracranial Structures," *IEEE Computer Graphics and Applications,* Vol. 4, No. 9, Sept. 1984, pp. 5-19.

28. D. Gordon, "Boundary Detection and Display: Some Informal Research Notes," typed notes, Dept. of Comp. Studies, Univ. of Haifa, Israel, July 1982.

29. H. K. Tuy and L. T. Tuy, "Direct 2-D Display of 3-D Objects," *IEEE Computer Graphics and Applications,* Vol. 4, No. 10, Nov. 1984, pp. 29-33.

30. D. Gordon and R. A. Reynolds, "Image Space Shading of Three-Dimensional Objects," tech. report MIPG85, Dept. of Radiology, Univ. of Pennsylvania, Nov. 1983. (To be published in *Computer Vision, Graphics and Image Processing.*)

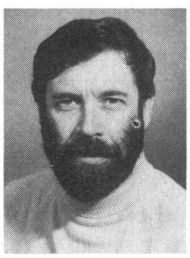
31. G. Frieder, D. Gordon, and R. A. Reynolds, "Back-to-Front Display of Voxel-Based Objects," tech. report MIPG89, Dept. of Radiology, Univ. of Pennsylvania, Aug. 1984.

32. L. J. Brewster, S. S. Trivedi, H. K. Tuy, and J. K. Udupa, "Interactive Surgical Planning," *IEEE Computer Graphics and Applications,* Vol. 4, No. 3, Mar. 1984, pp. 31-40.

33. N. I. Badler, "Disk Generators for a Raster Display Device," *Computer Graphics and Image Processing,* Vol. 6, No. 6, Dec. 1977, pp. 589-593.

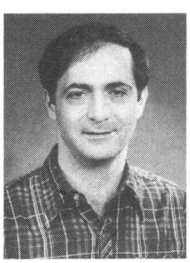
34. S. M. Goldwasser and R. A. Reynolds, "An Architecture for the Real-Time Display and Manipulation of Three-Dimensional Objects," *Proc. Int'l Conf. Parallel Processing,* Bellaire, Mich., Aug. 1983, pp. 269-274.

35. S. M. Goldwasser, "A Generalized Object Display Processor Architecture," *IEEE Computer Graphics and Applications,* Vol. 4, No. 10, Oct. 1984, pp. 43-55.



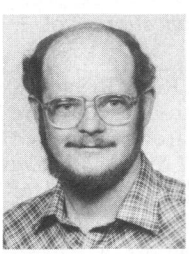
Gideon Frieder is currently professor of electrical engineering and computer science at the University of Michigan, where he also serves as chairman of the Division of Computer Science and Engineering. Prior to that he was professor of computer science at the State University of New York in Buffalo, a staff member of IBM Scientific Center at Haifa, Israel, and head of Computer Science in the Israeli Department of Defense.

Frieder is a member of ACM, where he served as national lecturer and a board member of Sigmicro and is a member of the IEEE Computer Society and TC-Micro. His interest in publication spans the areas of computer architecture, micro programming, operating system design, medically motivated algorithms for storage management in graphics, and robotic vision.



Dan Gordon is a visiting associate professor of computer science at the University of Cincinnati, where he taught from 1977 through 1979. He is on leave from the Department of Computer Studies at the University of Haifa, Israel. His current research interests include design and analysis of algorithms, computational geometry, computer graphics, VLSI, and computerized tomography.

Gordon received his BSc and MSc degrees in mathematics from the Hebrew University in Jerusalem, and his DSc from the Technion-Israel Institute of Technology. He is a member of the European Association for Theoretical Computer Science.



R. Anthony Reynolds is with the General Robotics and Active Sensory Processing group at the University of Pennsylvania. He hopes to graduate with a PhD in computer and information science in May 1985. From 1976 to 1980 he was a physicist with the Cancer Control Agency of British Columbia, Canada, working with computers in medical imaging and radiation therapy planning. He is currently working on 3-D display of medical objects, using hardware and software techniques.

Reynolds obtained a BA in physics from Trinity College, Dublin, in 1971. In 1973 he obtained an MSc in physics from the University of Alberta, Edmonton. He is a member of the Canadian Medical and Biological Engineering Society, the American Association of Physicists in Medicine, the ACM, and the IEEE.

Questions about this article may be addressed to R. Anthony Reynolds, Computer & Information Science, Moore School of Elec. Engineering, Univ. of Pa., Philadelphia, PA 19104.