COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING 38, 275–298 (1987)

A Dynamic Screen Technique for Shaded Graphics Display of Slice-Represented Objects

R. ANTHONY REYNOLDS*

General Robotics and Active Sensory Processing Group, Department of Computer and Information Science, Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, Pennsylvania 19104

DAN GORDON†

Department of Computer Science, University of Cincinnati, Cincinnati, Ohio 45221

AND

LIH-SHYANG CHEN‡

Medical Image Processing Group, Department of Radiology, University of Pennsylvania, Philadelphia, Pennsylvania 19104

Received March 21, 1985; accepted June 19, 1986

We present a very rapid method of constructing realistic images of 3-dimensional (3D) objects on a 2-dimensional (2D) display screen. Our technique is well suited to objects represented by slices, since it traverses the slices in a front-to-back sequence relative to the observer, accessing each slice just once. A dynamic data structure—the dynamic screen—is used to represent the unlit screen pixels. When each slice is accessed, only unlit pixels are processed and newly-lit pixels are efficiently removed from the data structure. Implementation of the method for large medical objects results in display times significantly faster than previous software methods. © 1987 Academic Press, Inc.

1. INTRODUCTION

There are many applications where computer techniques for creating realistic images of 3-dimensional (3D) objects are useful. One example is the generation and display of synthetic or artificial objects for entertainment or educational purposes. Another is the display of computer models of real objects (such as airports or cities) on training devices such as flight simulators. A third example is the display of objects which are unknown a priori, from the machine-readable output of automatic data-acquisition systems. In this latter case, shaded graphics techniques are used to convey the exact shape and structure of the objects to the viewer, and facilities for exploring and modifying the objects are important tools for their comprehension. We address the third application area in this paper.

*Research performed in part while with the Medical Image Processing Group, Department of Radiology, University of Pennsylvania. Present address: Dynamic Digital Displays, Inc., P.O. Box 15911, Philadelphia, Pa. 19103-0911.

†Present address: School of Mathematical Sciences, Tel-Aviv University, Tel-Aviv 69978, Israel (on leave from Haifa University). Research performed in part while visiting the Medical Image Processing Group, Department of Radiology, University of Pennsylvania, July–August 1983 and August 1984.

‡Present address: Department of Computer and Information Science, Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, Pa. 19104.

0734-189X/87 $3.00

In recent years, machines have become available which automatically digitize 3-dimensional scenes by measuring some physical property (known as the *density*) to be assigned to each of an array of volume elements in space (known as *voxels*). A typical example is in medical imaging, where computed tomography (CT) scanners estimate the radiological density (X-ray attenuation) for each voxel within a 3-dimensional region of a live patient. From these data, collections of voxels representing individual organs can be extracted and displayed on a 2-dimensional (2D) surface, with realistic shading applied and hidden surfaces removed to enhance the perception of depth.

In this paper, we use the term *$2\frac{1}{2}$-dimensional* ($2\frac{1}{2}$D) to refer to images which are displayed on a 2D surface, yet contain depth cues to indicate that a 3D scene is represented. This is to distinguish such images from 2D images of 2D scenes (e.g., CT slices), or 3D images of 3D scenes (e.g., holograms). This terminology is not universal: some authors refer to 2D images of 3D scenes as "3D images," and "$2\frac{1}{2}$D" has different meanings in other disciplines.

In medical applications, computer-generated $2\frac{1}{2}$D images can lead to improved diagnosis and are useful for radiation therapy and surgical planning [1, 2, 3]. Other areas where computer-generated $2\frac{1}{2}$D images of unknown objects are potentially useful include robotics (active sensory perception), machine vision, industrial simulation and process control, and geophysics (display of rock formations from seismological data).

Many automatic data-acquisition systems generate data as a 3D array of integers, each integer representing the density of an associated voxel. The data are usually generated in a raster-scan sequence such as slice-by-slice, row-by-row, and voxel-by-voxel within each row. If the voxels are not cubical in shape, it is often convenient to convert them to cubes by suitable interpolation: the representation of objects by cube-shaped voxels has been formalized into what is known as the *cuberille model* [4]. One approach to displaying digital scenes such as cuberilles is to obtain a surface representation for each object of interest within the scene, by tiling 1-dimensional (1D) contours [5, 6] or by surface tracking [7]. This approach incurs a time penalty for the surface formation, but rapid surface rendering algorithms can be used [8]. One disadvantage of the surface formation approach is that exploring or modifying the object interactively, or generating cut-away views, requires the formation of new surfaces which may be time consuming.

On the other hand, cuberilles are *spatially presorted* [9], that is, for each viewing direction it is possible to find (without sorting) a priority ordering such that voxels with high priority cannot be obscured by voxels with lower priority. This property makes *volume rendering* methods practical, where no surface formation is performed and every voxel is available for access at display time. Typical volume rendering techniques which have been implemented in both software and hardware include octree encoding [10, 11, 12], and the back-to-front voxel display method [13, 14]. Unlike the surface rendering approach, exploratory procedures such as cutting open the object to reveal internal structure can be performed at display time without additional time penalty. In this paper, we describe a new volume rendering method which is faster than any other *software* display method we have investigated, yet retains the capability of exploring and modifying the object at display time. The method is particularly well suited to data captured automatically by medical

imaging systems, since it exploits the slice-by-slice, row-by-row storage scheme in which the data are obtained.

The volume rendering method presented here combines three techniques to generate $2\frac{1}{2}$D images rapidly: (i) front-to-back traversal of the data; (ii) choice of coordinate transformations so that rows of voxels of the object remain parallel to scanlines of the image; and (iii) an efficient data structure for image representation (the dynamic screen). The basic idea of the dynamic screen is to represent only the *unlit* screen pixels, that is, those pixels onto which no voxel has yet projected. Combined with front-to-back traversal of the object, this enables us to "light-up" each pixel just once and eliminates the need to test pixels that have already been lit. The sequence of rotational transformations is chosen to ensure that line-segments representing the object are parallel (after projection) to line-segments representing the image, and can be combined with them in an efficient merging process.

Many issues are relevant to rapid shaded graphics display, of which we single out the following for special study: display algorithms, object representation, image representation, and shading schemes. We discuss these issues in Section 2, reviewing methods previously used to address them. The sequence of rotational transformations which makes the dynamic screen practical is discussed in Section 3. We describe the dynamic screen itself in detail in Section 4.

The dynamic screen technique can be used with many slice-by-slice object representations involving grey-scale or binary data; the optimum format depends on the application and also the interactive facilities needed. In Section 5 we discuss our implementation of the technique with binary data in the form of line segments.

In Section 6 we investigate the time and space complexity of our algorithm as a function of screen size and object characteristics. Experimental results are presented and compared with other display methods in Section 7. In Section 8 we give our conclusions: the dynamic screen technique enables us to display large objects more rapidly than any other software method for which we are aware of comparable timing measurements.

2. PREVIOUS METHODS FOR SHADED GRAPHICS DISPLAY

We summarize the methods that have previously been used for volume rendering under the following headings: display algorithms, object representation, image representation, and shading techniques. For completeness, we mention also methods that have been used for surface rendering.

2.1. Display Algorithms

Adopting the notation of our previous work [13, 15], we refer to the coordinate system in which the voxels are represented as *object space*, and the 3D space in which the pixels are represented as *image space* [Fig. 1]. The following steps are needed to generate a $2\frac{1}{2}$D image of a 3D scene: transformation from object space to image space, projection onto the screen, visible surface determination, scan-conversion (area-fill), and shading. Applicable transformations include rotations, translations, and dilations (scaling). General projections may be parallel or perspective, but in our application we use only orthographic (parallel) projections, for two reasons: perspective does not enhance the 3D perception of small, irregular objects such as human organs, and orthographic projections preserve distances and other geometrical information for direct measurement.

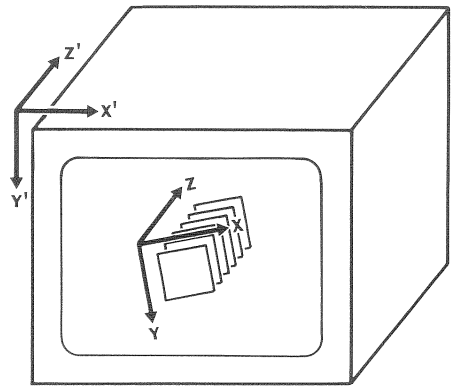


FIG. 1. Object space (X, Y, Z) and image space (X', Y', Z').

Visible surface determination (finding which parts of objects are not obscured by others) has been in the past a major bottleneck in shaded graphics display. Methods which have been used for medical objects include the well-known Z-buffer algorithm [8, 9], ray tracing [16, 17], a priori priority determination using separating planes [18] and algorithms which rely on spatial presortedness [10, 13, 16]. Since the latter are the most relevant to our present work we discuss them in further detail.

In [13] we demonstrated that the voxels making up a cuberille can be read out in an iterative back-to-front sequence, starting with the corner of the scene furthest from the observer [Fig. 2]. The back-to-front voxel read-out can be accomplished with three loops: the outer loop over slices, the middle loop over rows, and the inner loop over voxels within each row. In this scheme (originally proposed in [16]), changing the observer's viewpoint simply requires determining for each loop whether the indices should be increasing or decreasing.

In principle, it is equally possible to access the voxels in a front-to-back sequence relative to the observer; this has the advantage that only the first non-zero voxel that projects onto a given pixel need be considered. However, a naive implementation of front-to-back has no advantage over back-to-front since finding the pixel onto which a given voxel projects, and determining whether that pixel has already been lit, may be as expensive as simply painting over the previous value of the pixel. Thus, for front-to-back to be effective, coherence properties of the image must be exploited. A front-to-back display scheme that uses an octree to represent the object and a quadtree to represent the image has been given by Meagher [10]. The octree and the quadtree are traversed simultaneously: when an octree node is to be projected, the relevant nodes of the quadtree are examined to determine if the corresponding region of the screen has already been painted. In this paper we give an alternative, highly efficient implementation of the front-to-back approach.

2.2. *Object Representation*

When data are obtained automatically in machine-readable format, the choice of object representation is frequently influenced by the format in which the data are captured. This format may or may not be suitable for $2\frac{1}{2}$D display. One can consider two options: display the object directly from the raw data, accepting a time

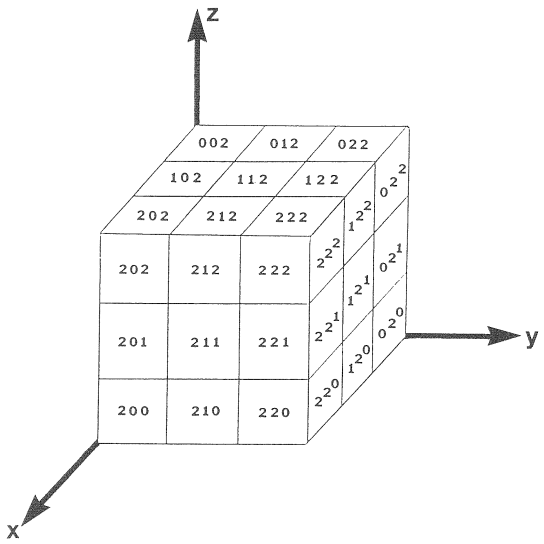


FIG. 2. Back-to-front voxel read-out. Voxels are read out slice-by-slice, starting with the slice furthest from the observer. Within each slice, voxels are read out starting with the corner furthest from the observer. The voxels are labelled with three digits, identifying the x, y, z coordinates in object space. For the object orientation shown, one possible read-out sequence is 000, 100, 200 . . . 010, 110, 210 . . . 022, 122, 222.

penalty due to inefficient representation; or preprocess the data to obtain a format that enables more rapid display. In some applications, a further requirement is a format that facilitates dissection or other interactive object manipulation. The following are some typical voxel-based methods of object representation:

1. Grey-scale voxels stored slice-by-slice and row-by-row within each slice. This format is typically used in medical imaging procedures such as computed tomography (CT) and magnetic resonance imaging (MRI). Volume rendering from grey-scale data may be accomplished by the back-to-front method, using software [13] or special hardware [14, 19]. The use of grey-scale voxels makes the entire data-set available to the user for interactive modification, but software implementations are time-consuming because of the vast amounts of data involved.

2. Compressed grey-scale data derived from (1) by techniques such as run-length encoding. Compression of CT data by 40–50% without loss of information has been reported [20].

3. Binary voxels, derived from (1) by thresholding. The voxels are stored in a 3-dimensional array, packed one bit per voxel. This format is the starting point for many surface formation schemes [7] and is probably the most useful for user interaction which does not involve manipulating grey-scale densities.

4. Compressed binary data derived from (3) using run-length encoding or equivalently, binary line segments represented by their endpoints [21]. This method achieves excellent data compression and can be adapted to support object modification. We have adopted this representation for the implementation of the dynamic screen technique described in this paper.

5. Directed contour representation [6], derived from (3) by finding the intersections of object surfaces with each slice. Directed contour representation achieves excellent data compression and facilitates interaction with and modification of the object on a slice-by-slice basis [22]. Very rapid display has been achieved using directed contours in the special case where the viewing direction is parallel to the plane of the slices [23]. However, this rapid display method does not appear to extend easily to arbitrary viewing directions.

6. Octree encoding [10, 11, 12], derived from (1). Each object in the digital scene is represented by an octree, a hierarchical 8-ary tree structure which usually requires less storage than the equivalent 3D array. Many operations (union, intersection and difference of objects; translation, rotation, and scaling; interference detection and hidden surface removal) can be accomplished by accessing each octree node (which may represent many voxels) at most once. Hidden surface removal is accomplished by displaying the cubes which correspond to octree nodes in a recursive back-to-front or front-to-back sequence. Changing the observer's viewpoint corresponds to visiting the nodes of the tree in a different sequence; modification of the octree is not required.

Efficient shaded graphics display of large objects on conventional computers is often controlled by the amount of main memory available. Ideally, the entire object should be held in main memory, but this is impractical with most conventional computers. The trend to larger memories is matched by a trend to larger objects (or data of higher resolution), and also a trend to process grey-scale rather than binary objects; thus the issue of limited storage is still important. Most commonly, the object is stored on disk and only a small portion is read into main memory at any time. A primary objective of object representations and associated display algorithms is to minimize the number of disk accesses that must be made.

2.3. Image Representation

For front-to-back display to be effective, an efficient image representation is required. The most common image representation is simply to store a 2-dimensional array of 8- or 16-bit integers, one for each pixel. The disadvantage of this scheme is that testing a pixel is typically as expensive as writing a new value into it. A more efficient approach for front-to-back display is to use a quadtree to represent the image, as previously described. In past implementations [10], the quadtree has been used to represent the entire grey-scale image; however we note that, for checking pixels, only a binary representation is needed provided it is used in conjunction with a 2-dimensional array which contains the grey-scale values. One advantage of such an arrangement is that the grey-scale image need not be retained in main memory, since pixels are written once but never read.

In this paper we describe a new representation which describes the image on a scanline-by-scanline basis, using a linked list structure for each scanline. This represents a departure from the quadtree approach in several respects. First, the data structure is simple and avoids the overhead associated with tree-traversal. Second, it is compact because only a binary representation is used: the associated grey-scale pixels are stored in a separate 2-dimensional array. Third, only the *unlit* pixels are stored: pixels which have been lit are removed from the data structure and never subsequently checked. This last feature, coupled with the dynamic nature

of the linked lists, helps to ensure that only moderate storage is required for most images commonly encountered.

2.4. Shading

One advantage of surface rendering is that vectors normal to the surface can be stored with the object representation and used at display-time to implement sophisticated shading schemes [24]. With volume rendering methods the surface is not known a priori, therefore storing surface normals is impractical. We have devised a technique known as *gradient shading* [15] which estimates the surface normals from a depth-shaded pre-image, thereby enabling oblique surfaces to be shaded realistically. A similar method was used by Horn for applications in cartography [25].

Gradient shading is an image space technique whereby a gradient operator runs over the depth-shaded pre-image to estimate the surface normals of the objects making up the 3D scene. These surface normals are then used to obtain surface orientations with respect to the incident light rays. Computational complexity of gradient shading is a function of image size rather than surface complexity or object size. Gradient shading is quite appropriate for use with the dynamic screen technique and has been applied to all the images presented in this paper.

3. CHOOSING THE ROTATIONAL TRANSFORMATIONS

In the new volume rendering method, we make use of a fundamental property of affine geometry: that parallel lines are transformed into parallel lines. This is appropriate for our application since we do not generate perspective views of medical objects, therefore the transformation from object space to image space is essentially affine. By choosing the transformations appropriately we can reduce the scan-conversion problem to simply painting in pixels one scanline at a time. The reasons for this are based on the following theorems.

THEOREM 1. *Let X_1, Y_1, Z_1 be axes through the object center, fixed in image space, and parallel to X', Y', Z' [Fig. 3]. Then if the object is rotated first about the X_1 axis and second about the Y_1 axis, line segments parallel to the X_1 axis are transformed into line segments which, when projected onto the $X'Y'$ plane, remain parallel to the X_1 axis.*

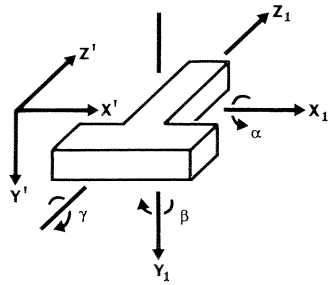


FIG. 3. Rotations α, β, and γ.

Proof. Let the object be rotated by angle α about the X_1 axis followed by angle β about the Y_1 axis. Then the rotation matrix is of the form

$$\begin{bmatrix} \cos\beta & \sin\alpha\sin\beta & \cos\alpha\sin\beta \\ 0 & \cos\alpha & -\sin\alpha \\ -\sin\beta & \sin\alpha\cos\beta & \cos\alpha\cos\beta \end{bmatrix}. \tag{1}$$

The y coordinate of a transformed segment is independent of the x coordinate of the original segment, from which the result follows immediately. □

THEOREM 2. *Given an arbitrary orientation of an object (resulting from rotations about any axes through its center, from a given initial orientation) it is always possible to find three rotations through angles α about the X_1 axis, then β about the Y_1 axis, then γ about the Z_1 axis, which will result in the same object orientation when applied to the same initial orientation.*

Proof. Let $T = [t_{ij}]$ be any rotation matrix, then we assert that it is always possible to find α, β, γ such that

$$T = \begin{bmatrix} \cos\beta\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \cos\beta\sin\gamma & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma \\ -\sin\beta & \sin\alpha\cos\beta & \cos\alpha\cos\beta \end{bmatrix}. \tag{2}$$

The right side of Eq. (2) is the transformation matrix corresponding to the rotations described in the statement of the theorem. Consider the following equations:

$$\begin{aligned} \cos\beta\cos\gamma &= t_{11} \\ \cos\beta\sin\gamma &= t_{21} \\ -\sin\beta &= t_{31} \\ \sin\alpha\cos\beta &= t_{32} \\ \cos\alpha\cos\beta &= t_{33}. \end{aligned} \tag{3}$$

From these, α, β, and γ can clearly be determined in the case $\cos\beta \neq 0$. Note that there are two solutions, since $\sin\beta = \sin(\pi - \beta)$. In the case $\beta = \pi/2$ the following equations may be used:

$$\begin{aligned} \sin\alpha\cos\gamma - \cos\alpha\sin\gamma &= t_{12} \\ \cos\alpha\cos\gamma + \sin\alpha\sin\gamma &= t_{13}. \end{aligned} \tag{4}$$

In this case there is an infinite number of solutions. Similarly for the case $\beta = -\pi/2$. □

The significance of these results is as follows. Theorem 1 states that rows of voxels making up the object will be parallel to scanlines of the image after rotation and projection. Theorem 2 states that any desired object orientation can be obtained with rotations α, β, and γ. The rotations α and β are in fact sufficient to view an object from any direction, since γ is simply a rotation of the image about an axis

perpendicular to the screen. It follows that hidden surfaces can be removed prior to rotation γ. But after rotations α and β, line segments of the object can be simply "merged" with the line segments of the image to which they are parallel, making use of the dynamic screen data structure. The result is that hidden surface removal and scan-conversion become quite trivial.

Rotations α and β together provide any desired *viewing direction*, and are in fact sufficient for many applications, since rotation γ does not uncover any new information. To obtain any desired *object orientation*, in general three angles must be specified, corresponding to rotations about three predetermined axes. The choice of these axes is somewhat arbitrary, so one can choose angles which have particular significance to the medical user [13, 26]. Given an orientation specified by any three angles which are convenient for the application at hand, as a result of Theorem 2 it is always possible to find three rotations α, β, and γ (about X_1, Y_1, and Z_1) which not only result in the object orientation the user requested, but also make possible very rapid display. This is accomplished by developing a pre-image using line segments which are parallel to scanlines of the screen, then rotating the pre-image. The pre-image may be supersampled to avoid aliasing errors. Methods of rotating 2D images are well known [27, 28].

4. PRINCIPLES OF THE DYNAMIC SCREEN

We assume that the object to be displayed resides on disk; at any time, only one slice can be in main memory. Consider the following display algorithm.

> Read in the slices one-by-one in front-to-back sequence. Within each slice, traverse the rows of voxels in front-to-back sequence. Light up those pixels onto which the voxels of the slice project, provided they are not already lit.

The problem with this approach is that we have to test every pixel that a slice projects onto. Our solution to this problem is to use a dynamic data structure to represent unlit contiguous segments of each scanline. The advantage of this technique is that only unlit pixels are tested, and as the screen gets filled, these become fewer and fewer.

We consider the object to be composed of line segments parallel to the X axis of object space, and the image to be composed of line segments parallel to the X' axis of image space [Fig. 1]. We use the terms "slice," "row," and "voxel" to refer to the object, and "scanline" and "pixel" to refer to the image. The segments for a given row of the object are stored in sorted order at the time the object representation is created; the segments for a given scanline of the image are kept sorted by means of the dynamic data structure. As a consequence of Theorem 1, rotated and projected object segments will be parallel to image segments. Thus, a line of unlit image segments and a line of rotated and projected object segments can be efficiently "merged" in a manner similar to the well-known technique used to merge two sorted sequences. Under these conditions the display algorithm becomes the following:

1. Read in the slices in front-to-back sequence. For each slice, loop over the rows in front-to-back sequence.

2. For each row, find the scanline onto which the row projects. (This may be assisted by look-up tables, as explained in [13]).

```
procedure FRONT_TO_BACK;

begin
   for k := 1 to number_of_slices do begin
       s := NEXT_SLICE; (* in front-to-back sequence *)

       for j := 1 to number_of_rows do begin
           r := NEXT_ROW; (* in front-to-back sequence *)

           if r contains object segments then begin
              L := scanline onto which r projects;
              if L contains unlit pixels then
                 MERGE(SEGMENTS(L), TRANSFORMED_SEGMENTS(r))
           end (* if *)

       end (* for *)
   end (* for *)
end;   (* FRONT_TO_BACK *)
```

FIG. 4. Simplified front-to-back display procedure.

3. Traverse the row in front-to-back sequence, transforming the object segments to image space and merging their projections with the image segments in the appropriate scanline.

4. If the merge operation results in pixels which are to be painted ("lit"), write the appropriate grey-scale values into the image array and remove the newly-lit pixels from the dynamic screen.

A formal description of this algorithm is given in Fig. 4. In Section 5 we present a modification to incorporate more effective scan-conversion.

The dynamic screen is similar to the data structures commonly used for bucket sorting and graph representation, and is not unlike the structure used in the Watkins algorithm for visible surface rendering [29]. Assume the image size is $N \times N$ pixels. An array of pointers YCHAIN[$0 \ldots N - 1$] is set up; each YCHAIN[L] points to a linked list representing contiguous unlit segments of scanline L. Each list element has three fields: MIN and MAX for the minimum and maximum x values of the unlit segment, and LINK—a pointer to the next list element [Fig. 5].

The MERGE procedure lies at the core of the display method and so we explain it in some detail. The arguments to the procedure are a row of transformed object

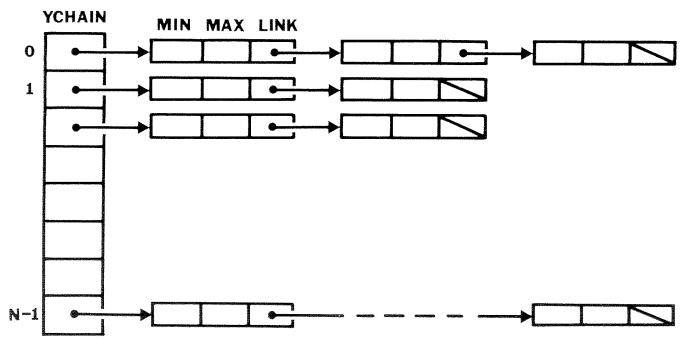


FIG. 5. The dynamic screen data structure.

segments, and a scanline of unlit image segments onto which the row will project [Fig. 6a]. Starting with the first object segment (in front-to-back order) and the first unlit image segment (left-to-right or right-to-left as appropriate), MERGE traverses the object and image segments until the end of one or the other sequence is reached. Let I and J be the pixels onto which the endpoints of an object segment project; let K be an unlit image segment. The MERGE procedure compares I and J against the endpoints of the unlit image segment, K.MIN and K.MAX. Depending on the outcome of the comparisons, there are eight distinct actions to be taken, as shown in Figs. 6b and c.

The outcomes of these comparisons (together with the proper actions) fall into four different categories. These are

* One segment completely precedes another (cases 1 and 8). In these cases, no changes are made to the image segment.

* The object segment covers one end of the unlit image segment (cases 2, 5, 7). The covered pixels are painted in and the image segment is shortened at one end by changing K.MIN or K.MAX. These operations are performed by procedures PAINT and CHANGE [Fig. 6d].

* The entire image segment is covered by the object segment (cases 4 and 6). The image segment is painted in and deleted from the linked list by procedure DELETE.

* The object segment falls strictly within the image segment (case 3). In this case, the covered pixels are painted in and the image segment "splits" into two segments (corresponding to the uncovered parts). The splitting operation is performed by procedure SPLIT, which adds a new list element and updates the fields of the old ones.

As part of the appropriate action, MERGE selects the next object segment, or next image segment, or both. This is done through procedures NEXT_OB_SEG and NEXT_IM_SEG, respectively. Note that the next object segment is always selected in front-to-back order, and this may result in left-to-right or right-to-left traversal along the scanline; the next image segment must be selected accordingly. MERGE advances through the object segments and image segments in a manner exactly analogous to the well-known method for merging sorted sequences; note how the linked list mechanism handles the CHANGE, DELETE, and SPLIT procedures in an efficient manner.

5. IMPLEMENTATION OF THE DYNAMIC SCREEN

When large amounts of data are involved and rapid display methods are proposed, some data-compression is in order. One simple compression technique is to obtain binary line segments by thresholding the grey-scale data, then represent each line segment by the coordinates of its endpoints. The basic idea is not new: in two dimensions it is essentially the same as run length encoding and has been described by Merrill [30]. In three dimensions, Udupa [6] used endpoints of segments to represent masks in certain surgical simulation tasks, and Trivedi [21] used line segments to represent 3D objects to facilitate their display and manipulation. Martin and Aggarwal [31] also used line segments as a 3D object representation. A major advantage of line-segment representation is that only simple operations are

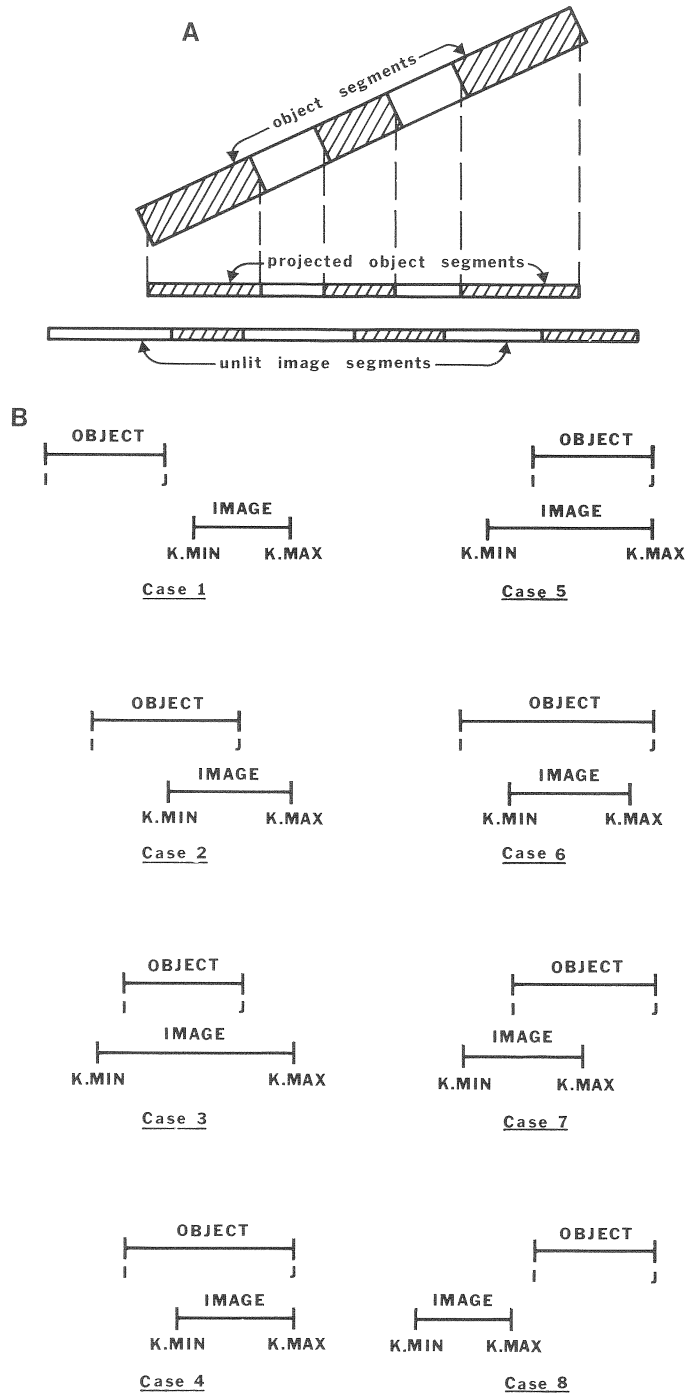


FIG. 6. The merge operation. (A) A row of object segments projected onto a scanline containing unlit image segments. (B) The eight cases arising from comparisons between a projected object segment and an unlit image segment. (C) The corresponding actions. (D) Some procedures called by MERGE.

C

CASE	CONDITION	ACTION
1	J < K.MIN	NEXT_OB_SEG
2	I ≤ K.MIN and K.MIN ≤ J < K.MAX	PAINT(K.MIN,J); CHANGE(J+1,K.MAX,K); NEXT_OB_SEG
3	I > K.MIN and J < K.MAX	PAINT(I,J); SPLIT(I-1,J+1,K); NEXT_OB_SEG
4	I ≤ K.MIN and J = K.MAX	PAINT(K.MIN,K.MAX); DELETE(K); NEXT_OB_SEG; NEXT_IM_SEG
5	I > K.MIN and J = K.MAX	PAINT(I,J); CHANGE(K.MIN,I-1,K); NEXT_OB_SEG; NEXT_IM_SEG
6	I ≤ K.MIN and J > K.MAX	PAINT(K.MIN,K.MAX); DELETE(K); NEXT_IM_SEG
7	K.MAX ≥ I > K.MIN and J > K.MAX	PAINT(I,K.MAX); CHANGE(K.MIN,I-1,K); NEXT_IM_SEG
8	I > K.MAX	NEXT_IM_SEG

D

```
procedure PAINT(I, J);
begin
   (* paint pixels I through J of current scanline *)
end;  (* PAINT *)

procedure DELETE(K);
begin
   (* delete segment K *)
end;  (* DELETE *)

procedure CHANGE(I, J, K);
begin
   (* modify segment K *)
    K.MIN := I;
    K.MAX := J
end;  (* CHANGE *)

procedure SPLIT(I, J, K);
begin
   (* split segment K into two new segments *)
    new(K');
    K'.MIN := J;
    K'.MAX := K.MAX;
    K'.LINK := K.LINK;
    K.MAX := I;
    K.LINK := K'
end;  (* SPLIT *)
```

FIG. 6–*Continued*.

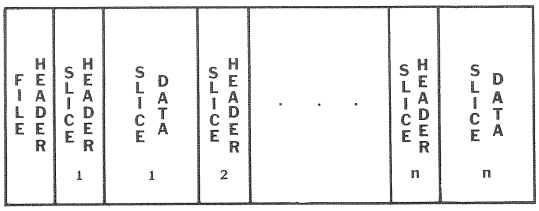


FIG. 7. Structure of input disk file. The file header contains an entry for each slice, from which can be computed the first block and the number of blocks of data for that slice. The first block of data for each slice contains a slice header, from which can be computed the first word and the number of words of data for each row. Thus, the slices and rows can be accessed in any sequence.

needed to convert from grey-scale CT data to binary segment endpoint format, because of the slice-by-slice, row-by-row storage scheme used by most CT manufacturers.

We use a stand-alone program to interpolate and threshold the CT data within a 3D volume of interest, producing cube-shaped binary voxels from which we generate line segments parallel to the X axis of object space. Cube-shaped voxels (and the interpolation step) are not strictly necessary, but in our experience, produce more pleasing images (see [32] for a discussion of this point). Each line segment is represented by two integers specifying the x coordinates of its endpoints; an efficient storage structure enables line segments belonging to any slice to be retrieved from disk [Fig. 7]. At display time, the dynamic screen data structure is initialized to the projection of the bounding box enclosing the object. The line segments representing the object are read from disk and accessed in front-to-back sequence. The x, y, and z coordinates of these line segments are integers from a finite set, therefore, look-up tables are pre-computed and used to determine the scanline of the image onto which a given line segment projects (see [13]). If there are no unlit pixels along this scanline, no further computation is performed. Otherwise the dynamic screen is updated and new visible pixels are computed and buffered for transmission to the display device.

We turn now to the issue of scan-conversion. The algorithm presented in Fig. 4 assumes each object segment projects onto a single scanline; this approach requires using scale factors no greater than $1/\sqrt{2}$, to avoid the occurrence of aliasing errors or "holes" at certain orientations [13]. This problem may be solved by scan-converting a long, thin rectangular parallelopiped for each object segment [Fig. 8]. Making use of coherence properties, we have implemented a modified front-to-back procedure which performs this scan-conversion quite efficiently.

The following discussion is valid for $-45° \leq \alpha < 45°$ and $135° \leq \alpha < 225°$ only. For other object orientations, the roles played by slices and rows should be interchanged. Suppose that an object line-segment in row r_1 of slice s_1 has endpoints e_1 and f_1; suppose further that e_1 is closer to the screen than f_1. Let e_1 and $f_1 + 1$ transform and project onto pixels i_1, j_1 respectively of scanline l_1. Then we paint in all pixels from i_1 to $j_1 - 1$ inclusive, using an appropriate grey-value for each: thus no holes can occur along the X' direction. Applying the same logic to Y', we determine the scanline l_2 onto which row r_2 will project (where r_2 is the next row in front-to-back sequence), and paint in pixels i_1 through $j_1 - 1$ in all scanlines

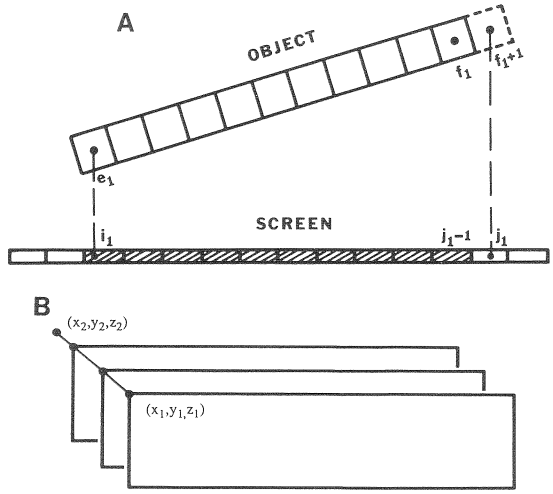


FIG. 8. Each line-segment is scan-converted as a long, thin rectangular parallelopiped: (A) view from above; (B) front view.

from l_1 through $l_2 - 1$, thus filling in a rectangular region of the screen. The final step is to determine where row r_1 of slice s_2 will project (where s_2 is the next slice in front-to-back sequence), and use this information to convert the rectangle into the projection of a parallelopiped. In general, this involves stepping along a diagonal path which may be obtained from a simple digital differential analysis (DDA) computation [33]. The modified algorithm is given in Fig. 9. The method is efficient because the dynamic screen automatically avoids scan-converting parts of the projected parallelopiped which are invisible.

In our implementation we have found it convenient to retain the grey-scale image in main memory, although (as pointed out in Sect. 2.3) this is not strictly necessary.

6. ANALYSIS OF THE ALGORITHM

Any display algorithm takes time at least proportional to the image area (number of lit pixels). We assume the screen size is $N \times N$ pixels and make a time and space analysis of our method which shows how these factors change as functions of N and other parameters. The actual run times are functions of many things, including the object, its orientation, and the computational facilities; thus, our analysis can only be performed to within multiplicative constants.

We assume that, for some constant μ, the object is bounded by a cube of size μN^3. This assumption can be justified on the following grounds: If the number of voxels is very much less than N^3, the object resolution is too coarse and individual voxels will be discernible (the object will appear to be made out of cubes). If the number of voxels is much larger than N^3, the screen resolution is too coarse to convey all the information inherent in the object. In practical terms, our assumption means that screen and object resolution are compatible. Since our analysis is to within multiplicative constants only, we can make the stronger assumption that $\mu = 1$ and the object is bounded by N^3.

```
procedure FRONT_TO_BACK_2;
begin
   s2 := NEXT_SLICE;  (* first slice *)

   for k := 1 to number_of_slices do begin
       s1 := s2;
       s2 := NEXT_SLICE;  (* in front-to-back sequence *)
       x1 := XOFFSET(s1);    x2 := XOFFSET(s2);
       y1 := YOFFSET(s1);    y2 := YOFFSET(s2);
       z1 := ZOFFSET(s1);    z2 := ZOFFSET(s2);

       repeat
          r2 := NEXT_ROW  (* first row in slice *)

          for j := 1 to number_of_rows do begin
              r1 := r2;
              r2 := NEXT_ROW;  (* in front-to-back sequence *)
              if r1 contains object segments then begin
                 L1 := scanline onto which r1 projects;
                 L2 := scanline onto which r2 projects;
                 if L1 < L2 then Ln := L2 - 1 else Ln := L2 + 1;
                 for L := L1 to Ln do begin
                     if L contains unlit pixels then
                        MERGE(SEGMENTS(L),
                                 TRANSFORMED_SEGMENTS(r1))
                 end (* for *)
              end (* if *)
          end; (* for *)

          x1 := x1 + XINC;  y1 := y1 + YINC;  z1 := z1 + ZINC;
       until DDADONE(x1, y1, z1, x2, y2, z2)

   end  (* for *)
end;  ( FRONT_TO_BACK_2 *)
```

FIG. 9. Front-to-back display procedure modified for scan-conversion. Only the case $-45° \leq \alpha < 45°$ or $135° \leq \alpha \leq 225°$ is given; the other case is similar. NEXT_SLICE and NEXT_ROW select the next slice and the next row in front-to-back sequence, respectively. XOFFSET(s), YOFFSET(s), and ZOFFSET(s) give (x_1, y_1, z_1), the projection of the first voxel in slice s, see Fig. 8; these offsets are used in computing the projections of subsequent voxels. XINC, YINC, ZINC, and DDADONE form a DDA procedure which moves (x_1, y_1, z_1) towards (x_2, y_2, z_2) and terminates when the latter point is reached.

Since each pixel is accessed at most once, $O(N^2)$ time is spent on accessing and painting in pixels. To analyze the time spent by the other operations, we first consider the operations that are not involved in the input of data. There are N scanlines to process, and each scanline is matched up against λN object lines, for some constant λ. Again, we can simply assume $\lambda = 1$. Testing an object segment against an image segment (and modifying the image segment) is done in constant time. Thus, the "merging" of k screen segments against l object segments takes $O(k + l)$ time, the same as merging two sorted sequences [34].

Let C be the minimal number of convex objects into which our object can be partitioned. Any object line then contains at most C segments and a scanline at any given time contains at most $C + 1$ segments, so the merging effort per scanline per object line is $O(C)$. C itself may be very large—as large as $N^3/2$—so $O(C)$ by itself may not be a very good bound. This bound can be sharpened by observing that image lines and object lines can have at most $N/2$ segments, so the merging effort is also bounded by $O(N)$.

Setting $M = \min\{C, N\}$, we find that the time to merge an object line with an image line is $O(M)$. Multiplying this result by N scanlines and by N object lines per

scanline, we get an overall time of $O(MN^2)$ for the merging part of the algorithm. Adding to this the $O(N^2)$ for painting pixels, we have a total time of $O(MN^2)$.

We now consider the effort involved in reading the data. There are N slices to be read in, each slice consisting of N lines. Each line has at most $\min\{C, N/2\}$ segments so the space taken up by a slice is at most $O(MN)$. Assuming that the input time per slice is proportional to its space requirement, we have a total input time of $O(MN^2)$. Note that the amount of work done by the gradient shading algorithm, and by rotations about the Z_1 axis, is also $O(N^2)$, so the total time, including shading and arbitrary orientation, is still $O(MN^2)$.

The space requirement of our algorithm is $O(N^2)$ for the screen data and $O(MN)$ for each slice, so the total space is $O(N^2)$. The multiplicative factors μ and λ which were taken to be 1 in the analysis can only change the constants in the $O(MN^2)$ runtime and $O(N^2)$ space, so these results remain unchanged. We have thus proved

THEOREM 3. *The dynamic screen technique can generate and display a shaded image of an object at any orientation in time $O(MN^2)$, where N^2 is the screen size, the object is bounded by a cube of size $O(N^3)$, $M = \min\{C, N\}$, and C = minimum number of convex parts into which the object can be partitioned. The space requirement is $O(N^2)$.*

The run time of our algorithm thus depends on C—which may be viewed as a measure of the "convexity" of the object. For a sphere or a cube, $C = 1$ and our algorithm runs in time $O(N^2)$. For the worst possible object, the run time would be $O(N^3)$.

We now compare the complexity of the present algorithm with some previous approaches. The back-to-front display method [13] takes time $O(N^3)$, proportional to the volume of the object. In fact, experiments show that about 90% of processing time is spent on computations involving the projections of voxels on the screen.

Analysis of octree algorithms is a difficult problem. As an example of the complicated issues involved, consider the following case: Assume the object is a $2^k \times 2^k \times 2^k$ solid occupying $\frac{1}{8}$ of a $(2^{k+1})^3$ universe. If the object happens to occupy exactly one octant of the universe, then the octree would consist of just the root and its 8 sons. However, shifting the object by just one voxel in a principal direction produces branches of length k in the octree representation. Octree algorithms thus depend not only on the nature of the object but on its location in object space. This property makes analysis of octree algorithms very difficult—a complete analysis should consider some form of average-case behavior, where the average is taken over the different possible locations of the object in object space.

An alternative display approach is the well-known ray tracing method: from the center of each unlit pixel, trace a light-ray through the object until it hits a full voxel, and light up the pixel accordingly. Analysis of a straightforward implementation of ray tracing shows that the time requirement depends not on the convexity of the object, but on the volume of the complement of the object in the $N \times N \times N$ cube. For example, if the object is the entire $N \times N \times N$ cube, then ray tracing takes $O(N^2)$ time. If the object is a sphere of radius $N/2$, the volume of its complement is proportional to N^3 and so the time to trace all rays to its surface is $O(N^3)$. Note however that ray tracing requires $O(N^3)$ space compared to our $O(N^2)$. One can do ray tracing with only $O(N^2)$ space by reading in a slice every

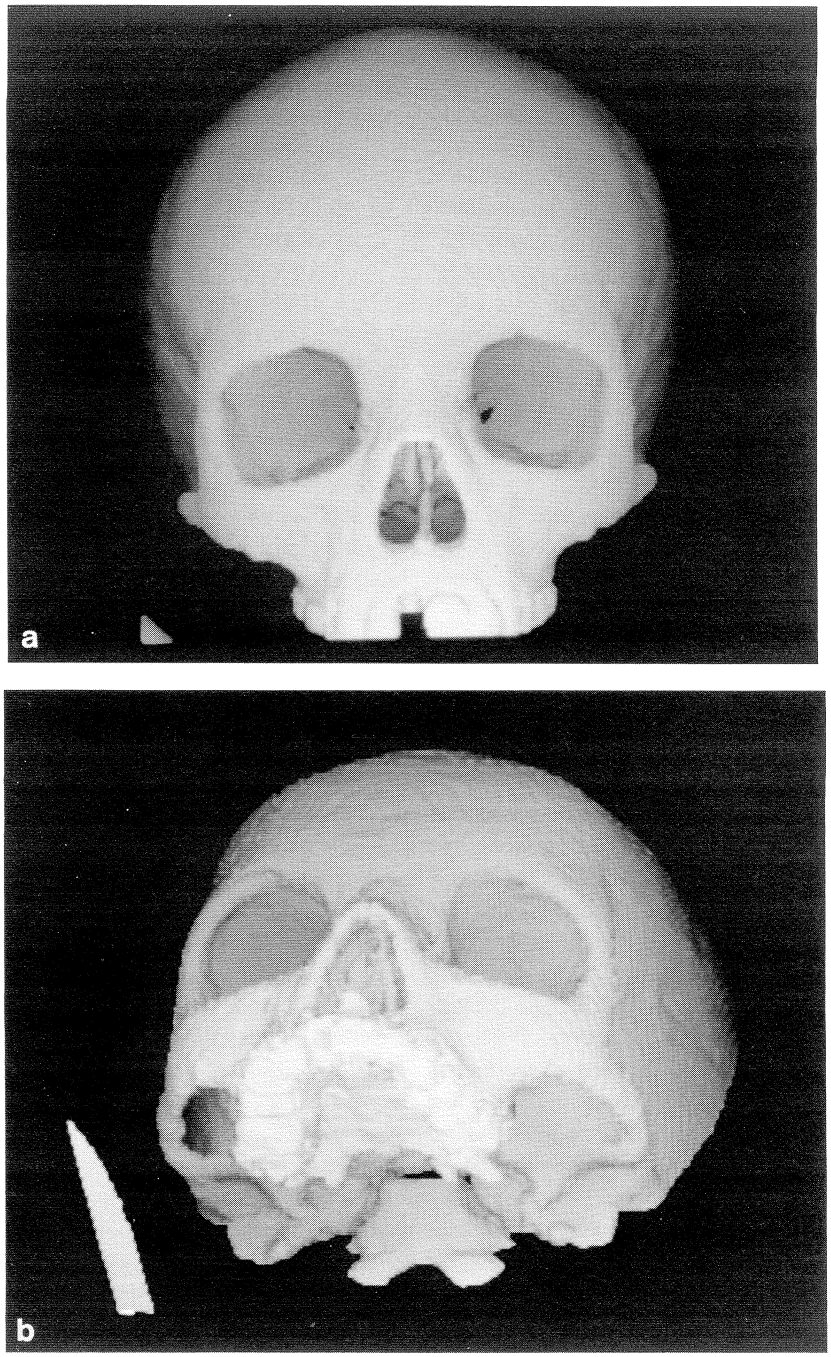
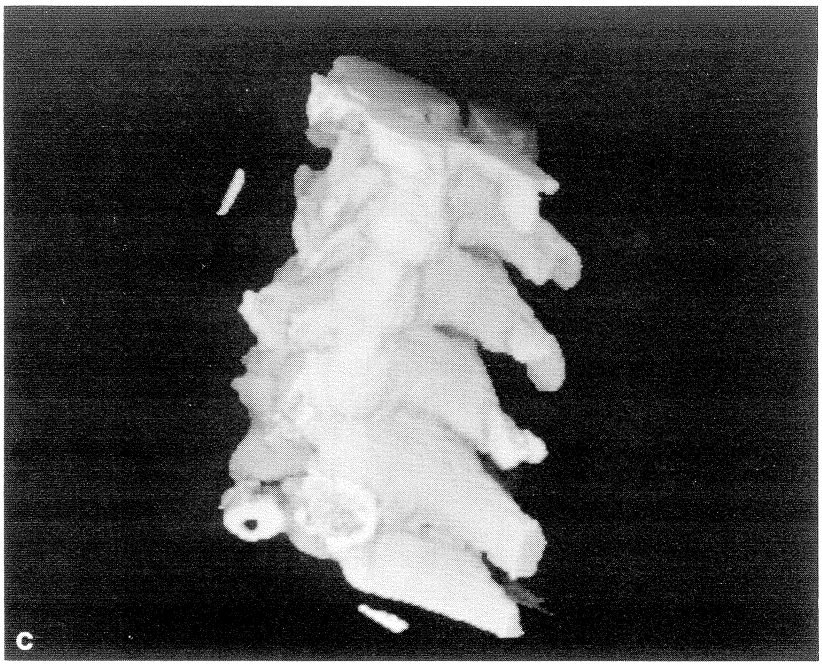


FIG. 10. Some images generated with the dynamic screen front-to-back program. (a) Dry skull, scanned in air, displayed with no rotation. The small object in the lower left corner is part of the apparatus used to hold the skull while scanning. (b) Same data-set, oriented so that $\alpha = -45°$, $\beta = 15°$, and $\gamma = 0°$. (c) Cervical spine of live patient, oriented so that $\alpha = 60°$, $\beta = 120°$, and $\gamma = 0°$. (d) Same data-set, oriented so that $\alpha = 30°$, $\beta = 10°$, and $\gamma = 0°$.



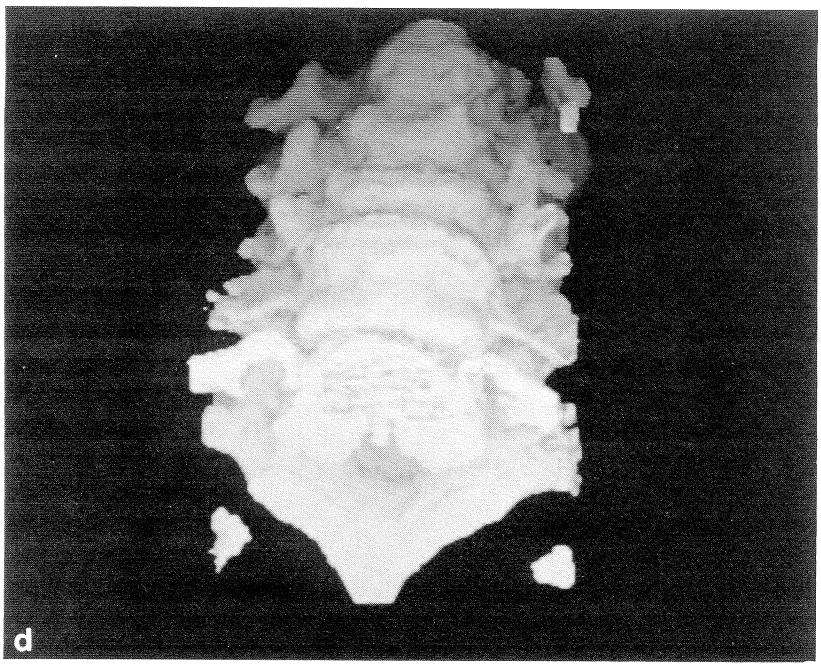


FIG. 10–*Continued*.

time it is required, but that would increase the run time considerably. This space requirement puts ray tracing in an entirely different category of algorithms.

7. RESULTS AND DISCUSSION

We have implemented the dynamic screen algorithm on two machines, a VAX 11/750 and a Data General Eclipse S/140, using FORTRAN programming. The program requires 192K bytes if the 256 × 256 × 16-bit grey-scale image is retained in main memory. All timing measurements quoted here have been made on the S/140, for two reasons. First, the S/140 is standard equipment on many CT scanners, including General Electric 8800 and 9800 CT/T; our program will therefore run on these scanners, achieving the performance quoted here. Second, using the S/140 allows us to make direct comparisons with other algorithms that have previously been implemented on this machine.

We chose two previous algorithms for direct comparison with the dynamic screen technique. Method 1 is the boundary detection and surface rendering approach, as implemented in the 3D83 package (later upgraded to 3D98) that is widely used for medical applications [26]. Method 2 is the back-to-front volume rendering method we have reported on previously [13]. For convenience, we refer to method 1 as "3D83," method 2 as "BTF," and the dynamic screen front-to-back technique as "FTB" in what follows. Additionally, we have chosen two objects to display using the three methods. Object 1 is a dry skull, scanned in air for research purposes. Object 2 is a section of cervical spine, scanned *in vivo* for clinical evaluation. Both data-sets were obtained with a GE 8800 CT scanner. Relevant statistics for these objects appear in Table 1. Two views of each object were generated: these are reproduced in Fig. 10. These objects were chosen because the size of the data-sets makes them representative of the largest objects one would normally encounter in routine clinical work.

In Fig. 11 we give timings for the preprocessing required to prepare the data, and for the display programs themselves. Common to all three display methods are the preprocessing steps of *subregioning*, to spatially isolate the volume of interest; *interpolation*, to obtain cube-shaped voxels; and *thresholding*, to obtain binary data. The programs which accomplish these steps for the three methods are in principle very similar; the preprocessing programs used for BTF and FTB are somewhat faster than 3D83 because they make use of an array processor (Floating Point Systems AP 120B) which is supplied as standard equipment on GE 8800 CT scanners. Using this array processor and a more careful implementation, all three methods could probably produce binary data for each object in under 5 min. Of more significance is the 20–30 min required for boundary detection in 3D83: there

TABLE 1

Some Statistics for the Objects Shown in Fig. 10

Object	Pixel size (mm)	Slice thickness (mm)	Slice increment (mm)	No. slices before interpolation	No. slices after interpolation	Bounding box enclosing object	No. voxels enclosed by surface	No. voxel faces on surface
Skull	0.64	1.5	1.0	93	144	205 × 233 × 144	859,569	353,940
Spine	0.293	1.5	1.5	50	251	215 × 252 × 251	3,068,333	553,640

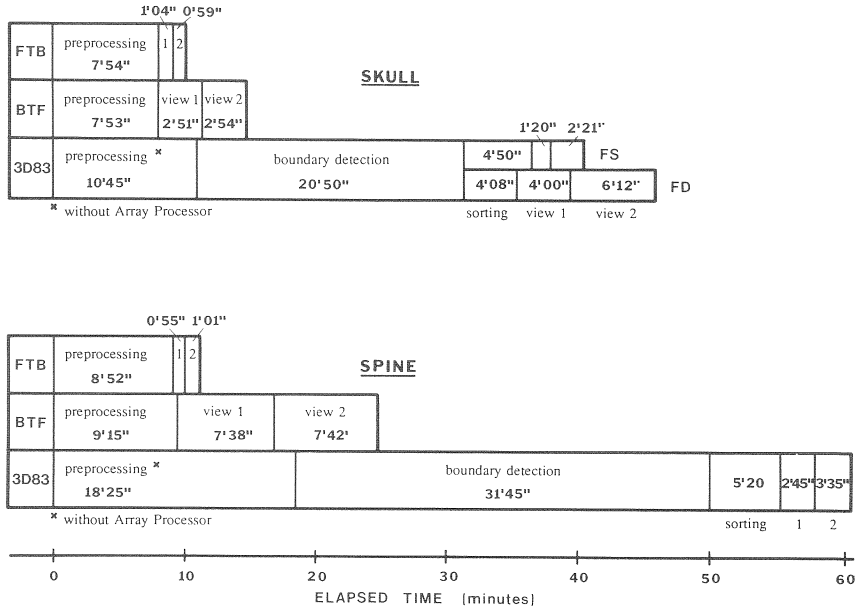


FIG. 11. Timing measurements for the objects shown in Fig. 10.

is no equivalent requirement in BTF or FTB. A sorting phase is required as a precursor to the surface rendering techniques used in 3D83 (see [8]). This sorting phase could possibly be eliminated by the use of improved data structures. Also, recent work [8] indicates that the surface rendering phase itself can be speeded by a factor of 2. However, even allowing for more efficient implementations we find that the surface formation approach makes the time to obtain the first image fundamentally longer in 3D83 than in BTF or FTB.

We turn now to a discussion of the display programs themselves. There are two surface rendering programs available in 3D83. The first ("FS") is capable of generating only certain orientations (rotations about the X_1, Y_1, or Z_1 axes), but can generate an image in just a few minutes. The second ("FD") is capable of arbitrary orientations, but is somewhat slower. We used both FS and FD for the skull, but FS only for the spine. The BTF and FTB programs can generate either depth-shaded or gradient-shaded images. The timings presented in Fig. 11 for BTF and FTB are for gradient-shaded images; timings for depth-shaded images are 25–30 s less.

A comparison of 3D83, BTF, and FTB is not complete without a discussion of image quality. The image quality obtained with the boundary detection approach used in 3D83 is uniformly excellent, due in part to an advanced shading scheme [24]. However, the use of the gradient shading technique makes the image quality obtained with FTB quite acceptable. The image quality obtained with BTF is slightly inferior to FTB; the reason for this is illustrated in Fig. 12. Let the four voxels v_1, v_2, v_3, and v_4 project onto pixels p_2, p_3, p_3, and p_4 respectively; denote by z_i the final z value associated with pixel p_i. Then in the example shown,

$$z_2 - z_3 = 2(z_3 - z_4).$$

The result is that flat surfaces can produce corrugated "ridges" rather than z values

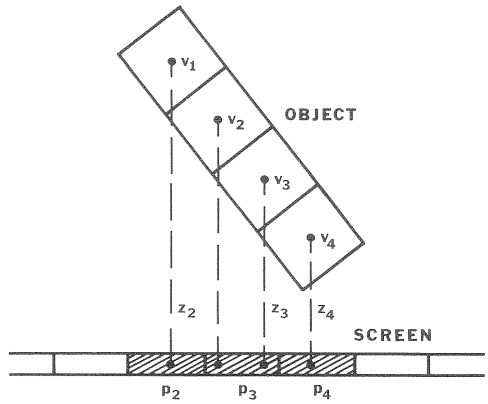


FIG. 12. In the BTF method, a flat surface can produce corrugated "ridges" (anomalous changes in z value). These ridges are later exaggerated by the gradient shading algorithm.

which change with constant slope. These ridges are later exaggerated by the gradient shading algorithm.

An important feature of volume rendering techniques in general is that they provide simple facilities for interaction with the object. Detailed methods of interaction are discussed elsewhere [13]; we mention briefly here some techniques which are relevant to the dynamic screen technique. Implementation of a clipping plane which is fixed with respect to image space is quite straightforward: voxels which lie on the observer's side are simply ignored, resulting in "cutaway" views of the object's interior. Clipping planes perpendicular to the X, Y, and Z axes of object space are also easily implemented, since they correspond to simply adjusting the loop limits over voxels, rows, and slices. Methods of deleting parts of the object can in principle be implemented as follows, although we have not added this facility to the present program. The observer uses a trackball to outline a cylinder perpendicular to the screen surface, and optionally specifies its depth: voxels whose projections fall within the cylinder during the display process are simply deleted from the input file, using procedures similar to CHANGE, DELETE, and SPLIT on the object line segments. Interaction resulting in object modification is appropriate to certain simple operations in computerized surgical planning [22]. However, it is also most useful in removing unwanted parts of the object to obtain an uncluttered display.

In comparing volume rendering with surface rendering, each approach has its advantages and disadvantages. Boundary detection results in isolation of a single connected surface of an object, which is automatically separated from surrounding noise and other unwanted objects. However, for some applications (e.g., fractures, interstitial implants) it is advantageous to view several disconnected objects simultaneously, to determine their relative positions; this is facilitated by volume rendering methods. In any event, the dynamic screen technique can be used to obtain rapid images and isolate and modify objects interactively; if desired, automatic boundary detection can then be applied to the modified objects.

We conclude with some remarks about possible hardware implementations of the dynamic screen technique. Clearly, a scanline can be merged with an object row in a

manner completely independent of the other scanlines; therefore in principle, scanlines (or groups of scanlines) can be assigned to separate processors operating in parallel. Each processor will access the object rows it requires in front-to-back sequence. The main advantage of this scheme over previous work [10, 14] would be the data compression afforded by run-length encoding, and the simplified scan-conversion.

8. CONCLUSIONS

We find the dynamic screen technique to be capable of rendering and shading large objects in about 1 min on a mini-computer of modest capability; this makes it considerably faster than any other software method we have investigated. The image quality obtained using gradient shading with the dynamic screen technique is quite acceptable. The dynamic screen approach has two other advantages in addition to its speed: less preprocessing than for surface rendering methods, and the facility to explore or modify the object at display time without time penalty. We conclude that the dynamic screen technique is a very viable method of shaded graphics image generation.

ACKNOWLEDGMENTS

We gratefully acknowledge the help of Dr. Gabor T. Herman, Hospital of the University of Pennsylvania, for use of Medical Imaging Section and Medical Image Processing Group facilities, and for the CT data. MIPG is supported by NIH Grant HL28438. The dynamic screen program in its present form was developed on a VAX 11/750 and ported to the Eclipse S/140 for timing measurements. We wish to thank Dr. Ruzena Bajcsy, Dr. Sam Goldwasser, and the members of the GRASP group for use of the 11/750 and its advanced software tools. 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. Special thanks are also due to Dr. J. K. Udupa, who suggested many improvements to the original manuscript; to Edward S. Walsh, who implemented the solution for α, β, γ given in Eqs. 3 and 4; and to David A. Talton, who implemented 2D rotation about the Z_1 axis.

REFERENCES

1. 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, *Brit. J. Radiol.* **52**, 1979, 478–481.
2. D. C. Hemmy, D. J. David and G. T. Herman, Three-dimensional reconstruction of craniofacial deformity using computed tomography, *Neurosurgery* **13** No. 5, 1983, 534–551.
3. M. W. Vannier, J. L. Marsh, and J. O. Warren, Three dimensional computer graphics for craniofacial surgical planning and evaluation, *Comput. Graphics* **17**, No. 3, 1983, 263–273.
4. G. T. Herman and H. K. Liu, Three-dimensional display of human organs from computed tomograms, *Comput. Graphics Image Process.* **9**, 1979, 1–21.
5. H. Fuchs, Z. M. Kedem, and S. P. Uselton, Optimal surface reconstruction from planar contours, *Commun. ACM* **20**, No. 10, 1977, 693–702.
6. J. K. Udupa, Interactive segmentation and boundary surface formation for 3D digital images, *Comput. Graphics Image Process.* **18**, 1982, 213–235.
7. E. Artzy, G. Frieder, and G. T. Herman, The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm, *Comput. Graphics Image Process.* **15**, 1981, 1–24.
8. R. A. Reynolds, *Fast Methods for 3D Display of Medical Objects*, Chap. 4, Ph.D. dissertation, Dept. of Computer and Information Science, University of Pennsylvania, May 1985.

9. 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. Soc. Photo-Opt. Instrum. Eng.* **367**, 1982, 3–14.
10. D. Meagher, Efficient synthetic image generation of arbitrary 3-D objects, *Proc. IEEE Comput. Soc. Conf. Pattern Recognit. Image Process.*, June 1982, pp. 473–478.
11. D. Meagher, Geometric modelling using octree encoding, *Comput. Graphics Image Process.* **19**, 1982, 129–147.
12. D. Meagher, *The Octree Encoding Method for Efficient Solid Modelling*, Ph.D. dissertation, Electrical, Computer and Systems Engineering Dept., Rensselaer Polytechnic Institute, August 1982.
13. G. Frieder, D. Gordon, and R. A. Reynolds, Back-to-front display of voxel-based objects, *IEEE Comput. Graphics Appl.* **5**, No. 1, 1985, 52–60; see also *IEEE Comput. Graphics Appl.* **5**, No. 3, 1985, 83.
14. S. M. Goldwasser and R. A. Reynolds, An architecture for the real-time display and manipulation of three-dimensional objects, *Proc. Int. Conf. Parallel Processing*, Bellaire, Mich., August 1983, pp. 269–274.
15. D. Gordon and R. A. Reynolds, Image space shading of 3-dimensional objects, *Comput. Vision Graphics Image Process.* **29**, 1985, 361–376.
16. D. Gordon, Boundary detection and display: Some informal research notes, typed notes, Department of Computer Studies, University of Haifa, August 1982.
17. H. K. Tuy and L. Tuy, Direct 2-D display of 3-D objects, *IEEE Comput. Graphics Appl.* **4**, No. 10, 1984, 29–33.
18. H. Fuchs, G. D. Abram, and E. D. Grant, Near real-time shaded display of rigid objects, *Comput. Graphics* **17**, No. 3, 1983, 65–72.
19. S. M. Goldwasser, R. A. Reynolds, T. Bapty, D. Baraff, J. Summers, D. Talton, and E. Walsh, Physician's workstation with real-time performance, *IEEE Comput. Graphics Appl.*, **5**, No. 12, 1985, 44–57.
20. M. L. Rhodes, J. F. Quinn, and B. Rosner, Data compression techniques for CT image archiving, *J. Comput. Assisted Tomogr.* **7**, No. 6, 1983, 1124–1126.
21. S. S. Trivedi, Representation of three-dimensional binary scenes, in *NCGA'85 Technical Sessions Proceedings*, Dallas, Texas, Vol. III, April 1985, pp. 132–144.
22. L. J. Brewster, S. S. Trivedi, H. K. Tuy, and J. K. Udupa, Interactive surgical planning, *IEEE Comput. Graphics Appl.* **4**, No. 3, 1984, 31–40.
23. L. S. Chen. H. M. Hung, and J. K. Udupa, Towards real-time interactive display of 3D medical objects, *Proc. 18th Hawaii Int. Conf. Syst. Sci.*, January 1985.
24. L. S. Chen, G. T. Herman, R. A. Reynolds, and J. K. Udupa, Surface shading in the cuberille environment, *IEEE Comput. Graphics Appl.* **5**, No. 12, 1985, 33–43.
25. B. K. P. Horn, Hill shading and the reflectance map, *Geo-Process.* **2**, 1982, 65–146.
26. L. S. Chen, G. T. Herman, C. R. Meyer, R. A. Reynolds, and J. K. Udupa, 3D83–An easy-to-use software package for three-dimensional display from computed tomograms, *Proc. Soc. Photo-Opt. Instrum. Eng.* **515**, 1984, 309–316.
27. E. Catmull and A. R. Smith, 3-D transformations of images in scanline order, *Comput. Graphics* **14**, No. 3, 1980, 279–285.
28. K. M. Fant, A nonaliasing real-time spatial transform technique, *IEEE Comput. Graphics Appl.* **6**, No. 1, 1986, 71–80.
29. G. S. Watkins, *A Real-Time Visible Surface Algorithm*, Technical Report UTECH-CSC-70-101, Dept. of Computer Science, University of Utah, June 1970.
30. R. D. Merrill, Representations of contours and regions for efficient computer search, *Commun. ACM* **16**, No. 2, 1973, 69–82.
31. W. N. Martin and J. K. Aggarwal, *Volumetric Descriptions of Objects from Multiple Views*, Technical Report TR-82-5, Laboratory for Image and Signal Analysis, University of Texas at Austin, October 1982.
32. G. T. Herman, Three-dimensional computer graphic displays in medicine, *NCGA'84 Tutorial Sessions Proceedings*, Anaheim, Ca., Vol. I, April 1984, pp. 131–147.
33. J. E. Bresenham, Algorithm for computer control of a digital plotter, *IBM Syst. J.* **4**, No. 1, 1965, 25–30.
34. D. E. Knuth, *The Art of Computer Programming*, Vol. 3: *Sorting and Searching*, Addison–Wesley, Reading, Mass., 1973.