Representation of Three-Dimensional Digital Images

SARGUR N. SRIHARI

Department of Computer Science, State University of New York at Buffalo, Amherst, New York 14226

Three-dimensional digital images are encountered in a variety of problems, including computed tomography, biological modeling, space planning, and computer vision. A wide spectrum of data structures are available for the computer representation of such images. This paper is a tutorial survey of three-dimensional spatial-data representation methods emphasizing techniques that apply to cellular (or voxel-based) images. We attempt to unify data structures for representing interior, surface, and structural information of objects in such images by comparing their relative efficiency. The derivation of high-level representations from serial section images is also discussed The representations include topological representations (Euler characteristic and adjacency trees), geometrical representations (borders, medial axes, and features), and spatial organization representations (generalized cylinders and skeletons).

Key Words and Phrases three-dimensional analysis, image-data structures, pattern representation, computed tomography, computer vision, image reconstruction, computer graphics, space planning, three-dimensional features

CR Categories: 3.6, 5.3, 8.2

INTRODUCTION

New challenges have been created in the field of image analysis and pattern recognition by the introduction of modern image-data collection techniques such as computed tomography, scanning electron microscopy, and digital stereoscopy. These methods make possible the computation of the three-dimensional (3D) structure of scenes, ranging from organs interior to the human body to rock microstructures, in the form of a 3D array of numbers. Developing systems for processing and displaying these images has revealed the need for developing new data structures, and more generally, for developing spatial-knowledge representation schemata.

While numeric-computational methods of picture-processing and symbolic-computational methods of image analysis have evolved over the past two decades, representation methods have primarily concerned two-dimensional (2D) images (e.g., optical character recognition and chromosome analysis) and monocular 2D projections of 3D scenes (e.g., remote sensing). Methods for handling true 3D digital images are now being developed for tasks such as the display of cross-sectional and shaded-surface reconstruction images on a graphics terminal, quantitative measurement of 3D shape, and computer understanding of spatial organization. In this paper we review some of the more promising approaches for the representation of 3D discrete images. Since the topic covers a wide spectrum of techniques, from data structures for image intensity arrays to knotty problems of knowledge representation, and since much of the literature is very recent (and somewhat inaccessible),

CONTENTS

this review is not intended to be a comprehensive survey. The methods discussed are primarily concerned with discrete images, with emphasis on image analysis applications. Although some of the relevant results of computational geometry and topology are touched upon, we do not consider conventional 3D techniques such as those used in flight simulators, animation, and computer-aided design.

1. APPLICATION AREAS

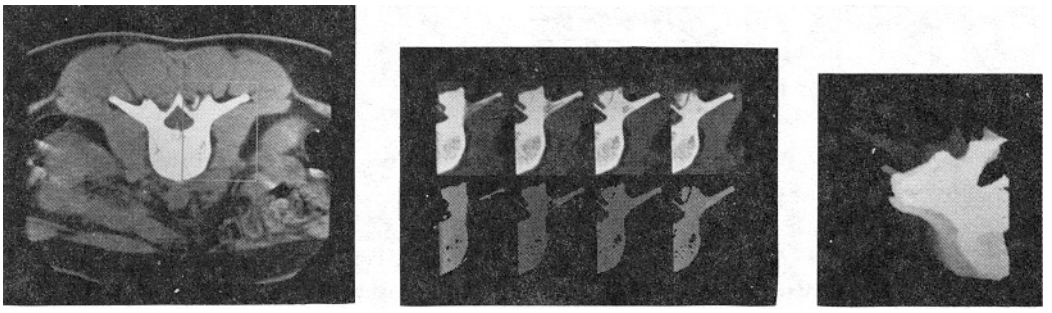
The method of computer representation of a 3D digital image is necessarily influenced by the processing for which the representation is used. It is thus useful to begin with the problem domains in which the methods discussed here are applicable. Each of the following domains have the common characteristic of the image being in the form of a 3D array, or equivalently, a folio of 2D images of serial sections.

Perhaps the most significant of the new 3D scanning techniques is the process of *computed tomography* (CT). Essentially, the attenuation caused by each object point (referred to as a *CT number*) is reconstructed from a set of translucent projections (such as by x-ray, emission, ultrasound, or nuclear magnetic resonance (NMR)) of the object [GORD74, ALTS81]. CT is useful for reconstructing internal parts of the human body such as the brain, the skull, and the heart. The potential of CT to reconstruct industrial objects for nondestructive testing is now being explored [KRUG78]. Figure 1 shows an example of a 3D image of the human body produced by CT.

Methods of 3D image representation are needed in *quantitative microscopy* [DEHO75]. In the analysis of biological microstructures, such as neurons and blood vessels, the object is physically sliced in parallel planes and each slice is photographed and the photograph digitized [REDD78]. Computer serial section methods are also useful in the analysis of grain microstructures in rocks and metals [BARR70], where sections are obtained by repeated polishing of the surface. Pattern recognition techniques for identifying 3D geometrical structures are also required in pharmacology and x-ray crystallography [LESK79].

The analysis of *time-varying* 2D images is a problem where certain 3D representations are useful. Here a stack of successive frames in a time sequence of 2D images constitutes the 3D arrays. As opposed to the intraframe approach of image analysis, for example, in detecting the boundary of an object in each frame, the 3D approach not only tends to be more efficient but also is necessary to preserve time continuity [UDUP79].

To a lesser extent, the methods discussed here are applicable to the domain of *computer vision* in which the 3D description of an object's shape is derived from information encoded in images of its surface. Methods of mapping the 3D coordinates of a surface using stereopsis [GENN77, POTE79], laser ranging [NEVA77], shadows [WALT75], and shading and texture gradients [HORN75] yield partial 3D descriptions. One such description, called a $2\frac{1}{2}$D sketch, is a data structure that contains



(a) (b) (c)

Figure 1. An example of a 3D digital image. (a) CT scan of abdominal section of a patient with region of interest (spine) marked by a rectangle, (b) *top*: four consecutive slices containing only the region of interest marked in (a), *bottom*: corresponding binary slices, with elements containing bone indicated bright, all other elements indicated dark, (c) shaded surface 3D display of spine represented in (a) and (b). [Adapted from Raven Press, "The Use of three-dimensional computer display in the study of disk disease," G. T. Herman and C. G. Coin, *J. Comput. Assist. Tomography* 4, 4 (Aug. 1980), 564–567.]

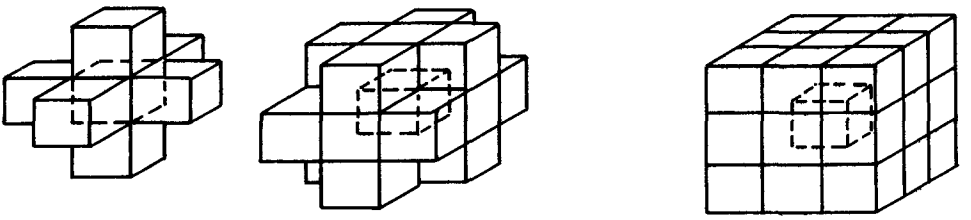
information about the orientation (relative to the viewer) of small patches of surfaces spaced evenly over the visual field [MARR78, BAJC80]. Space planning for robot movement [EAST70] and volumetric modeling for mechanical engineering design are related problems where 3D data structures are called for. Surveys for representations that deal more generally with computer vision, computer graphics, and rigid solid objects have been done by Shirai [SHIR78], Badler and Bajcsy [BADL78], and Requicha [REQU80], respectively.

In the application areas discussed so far, computer representations are needed both for displaying images on a graphics screen and for automatic analysis of images. In addition to cross-sectional images, it is useful to display shaded 3D reconstructions of objects; the reconstructions are especially valuable because very little is known about the ability of human beings to imagine 3D objects when presented with a series of cross-sectional images. Shaded reconstruction requires the application of a sequence of 3D image-handling algorithms to the 3D intensity array; a typical sequence is image segmentation, boundary detection, hidden surface removal, and shading. Algorithms for performing each of the steps of 3D display in real time are facilitated by careful design of data structures for image data. Computer recognition and description of objects in 3D images require more sophisticated spatial-knowledge representation schemes.

2. DIGITIZATION

Images that are produced by sensing objects through a form of radiant energy, for example, by reflection, transmission, or emission, are inherently continuous. Computer representation of 3D images requires a sampling of the volume to extract a discrete set of values. Although numerically created images, such as CT, are obtained as a discrete set of values, a discretization process has to be introduced at some stage.

One method of volume sampling is to use a regularly spaced array of points ($j\delta$, $k\delta$, $l\delta$), whose coordinates are multiples of some unit distance δ. A point $V \equiv (V1, V2, V3)$ of this array is referred to as a *digital point*. We associate with each digital point V those points $(x1, x2, x3)$ of continuous space satisfying $Vi - \delta/2 \leq xi < Vi + \delta/2$ (for $i = 1, 2, 3$); we refer to the resulting unit cube volume element as *voxel* V (the term voxel is short for "volume element," analogous to pixel for "picture element" in two dimensions). Note that each digital point uniquely specifies a voxel and that each voxel contains exactly one digital point. This method yields a *cellular* or *polyhedral-close-packed tessellation* of the volume. Among the five Platonic solids, tetrahedron, cube, octahedron, pentagonal dodecahedron, and icosahedron, only the cube yields a close-packed tessellation; this differs from the 2D case in which three tesselations (triangular, square, and hexagonal) are possible. It should, however, be noted



(a) (b) (c)

Figure 2. Neighbors of a digital element in a cube array: (a) 6-neighbors (face neighbors); (b) 18-neighbors (face and edge neighbors), (c) 26-neighbors (face, edge, and vertex neighbors).

that when the restriction of identical Platonic solids is removed, other 3D close packings are possible [HILB52].

The relative positions of voxels is of importance in deriving certain representations. For this purpose we define the concepts *neighborhood* of a voxel and *distance* between voxels.

2.1 Neighbors

Each voxel has three kinds of neighbors, 6 abutting a *face* of the unit cube at distance δ, 12 abutting an *edge* at distance $\delta\sqrt{2}$, and 8 abutting a vertex at distance $\delta\sqrt{3}$. Similar to *square* array sampling of planes [ROSE76], we can group these together as 6-, 8-, and 26-neighbors (Figure 2). We say that if not more than n of their components differ by 1 and the rest are identical, then the two voxels V and U are n-adjacent, denoted by the relation R_n for $1 \leq n \leq 3$. Thus if $Di \triangleq |Vi - Ui|$, then

$$R_n = \{(V, U) \mid V \neq U, Di \in \{0, 1\}, \sum_i Di \leq n\}.$$

Thus the 6-, 18-, and 26-neighbors of V are precisely its 1-, 2-, and 3-adjacent voxels. An equivalent definition of R_n in terms of Boolean functions is

$$R_n = \{(V, U) \mid Di \in \{0, 1\} \text{ and } Fn = 1\}$$

where $F1 = D1 \cdot \overline{D2} \cdot \overline{D3} + \overline{D1} \cdot D2 \cdot \overline{D3} + \overline{D1} \cdot \overline{D2} \cdot D3$, $F2 = \overline{D1} \cdot D2 + \overline{D2} \cdot D3 + D1 \cdot D3$, and $F3 = D1 + D2 + D3$. We use the symbol $\mathcal{N}_n(V)$, that is, the n-neighbors of V, to denote the set $\{U \mid (V, U) \in R_n\}$, whose cardinality is $\sum_{j=1}^{n} 2^j \binom{3}{j}$.

2.2 Distance

The Euclidean distance between voxels V and U is defined as $d_E(V, U) = (\sum_i (Di)^2)^{1/2}$. Thus the Euclidean distance between the opposite corners of an $N \times N \times N$ cube is $N\sqrt{3}$ and the distance between opposite corners of a face of the cube is $N\sqrt{2}$. A disadvantage of the Euclidean distance is that distance is not always integer valued; modifications such as round(d_E), $\lfloor d_E \rfloor$, $\lceil d_E \rceil$, and d_E^2 violate the requirement of a metric. Thus we define the following non-Euclidean metrics.

Absolute Metric. $d_6(V, U) = \sum_i Di$; that is, the voxels at absolute distance of 1 from V are just its 6-neighbors. The distance between opposite corners of an $N \times N \times N$ cube is $3N$ and that between opposite corners of a face of the cube is $2N$. Voxels at distance $\leq t$ from V (analogous to a sphere with d_E) form a cube with faces of side length $t\sqrt{2}$ inclined at $\pm 45°$ to each axis.

Maximum Metric. $d_{26}(V, U) = \max_i \{Di\}$; that is, the voxels at absolute distance of 1 from V are just its 26-neighbors. The distance between opposite corners of an $N \times N \times N$ cube is N, which is equal to the distance between opposite corners of a face. Voxels at distance $\leq t$ from V form a cube centered at V and with side length $2t$.

There does not appear to be any simple metric function corresponding to $d_{18}(V, U)$; for this reason a choice of either 1- or 3-adjacency is preferable in neighborhood definitions.

2.3 Digital Image

A 3D *digital image* is a mapping that associates each voxel with a value; the value is usually of one of the types: real, integer,

or binary. When the 3D digital image is a sampled 3D continuous function, sampling is followed by *quantization*, a process which enables image representation with finite precision. Some problems require a third step, solving the *correspondence* problem of identifying the same object in adjacent slices. This may require a subtle solution, particularly when there is no precise control of image registration [REDD78]. When the 3D image consists of a folio of 2D slice images, the between-slice resolution tends to be less than the within-slice resolution, in which case the intermediate slice images are usually obtained by *interpolation* (in a typical example, 40 CT slices 1.5 mm apart were interpolated to create 74 estimated slices 0.8 mm apart [ARTZ81]).

2.4 Digital Binary Image

A 3D digital binary image is defined by a function $f(V)$, also called the *characteristic function*, whose domain is the set of all voxels V and range is the set $\{0, 1\}$. The set of voxels $S = \{V \mid f(V) = 1\}$ is referred to as the *object* and the set $\bar{S} = \{V \mid f(V) = 0\}$ is referred to as the *background*.

The high-level topological and geometrical representations that are discussed later in this paper utilize digital binary images. In some application areas, the input data are in the form of a 3D binary array, and in others they are in the form of a multivalued (gray level) 3D image (which is quite often processed to yield a binary array). We briefly discuss each of these cases.

In applications such as space planning, the characteristic function f is specified by a 3D binary array with value 1 representing *full* and value 0 representing *void*. Such a 3D array is obtained from continuous space using a mapping such as the following: if Q is a point set in 3D Euclidean space, then $f(V) = 1$ if the points of V have a nonempty intersection with the points of Q, and $f(V) = 0$ otherwise. An alternative mapping is to have $f(V) = 1$ only if more than half the points of V are in Q. With an appropriately defined mapping I, the set S of all voxels that have value 1 is said to be the digitization of Q and denoted as $S = I(Q)$.

A CT image, on the other hand, is usually in the form of a multivalued 3D array. The process of isolating object voxels from background voxels in such an array is known as *segmentation*. As in the case of 2D image processing [ROSE76], segmentation procedures for 3D images can be grouped into *region-based* and *boundary-based* methods.

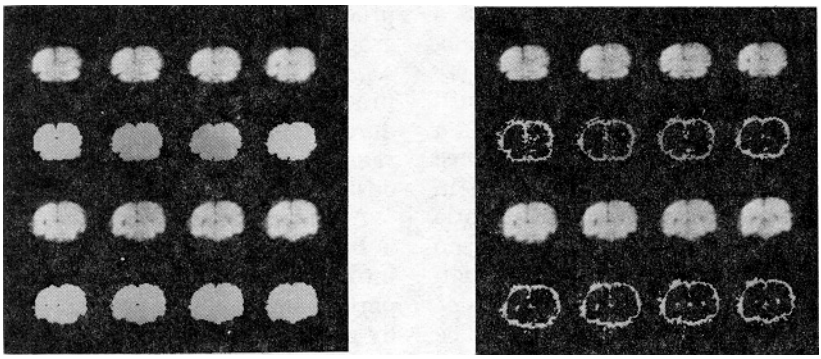
A region-based method of segmentation is one that proceeds by assigning a voxel into object or background on the basis of partitioning an appropriate feature space by means of a decision function. When the feature used is only the value associated with the voxel, then the procedure is known as *thresholding*. If $f(V)$ represents the value of the image at V and $\mathscr{D}$ is the range of $f(V)$, then the thresholding operation is defined by the characteristic function

$$f(V) = \begin{cases} 1, & \text{if } f(V) \in \mathscr{D}' \subseteq \mathscr{D}, \\ 0, & \text{otherwise.} \end{cases}$$

Thresholding is effective when there is high contrast between object and background values and little clutter. An example of segmentation of serial section images using two different sets $\mathscr{D}'$ is shown in Figure 3.

Boundary-based segmentation proceeds in two steps by detecting local edges using some form of 3D spatial differentiation, and by grouping local edges into boundary contours that separate object voxels from background voxels. A number of 3D edge operators have been defined for this purpose. They are based on the principle that the magnitude of the gradient can be estimated from the directional derivative of $f(V)$ along three orthogonal directions. If ∇i, $i = 1, 2, 3$, are the three derivative magnitudes, then the gradient magnitude is given by the Euclidean norm $(\sum_i \nabla i^2)^{1/2}$, which is approximated by $\nabla 1 + \nabla 2 + \nabla 3$. One approach to defining the directional derivatives [LIU77] is to use the cross operator of ROBE65 along mutually perpendicular planes. In this case, using the notation $V_{ijk} = (V1 + i, V2 + j, V3 + k)$, we get

$$\begin{aligned} \nabla 1 &= |f(V_{000}) - f(V_{011})| \\ &\quad + |f(V_{010}) - f(V_{001})|, \\ \nabla 2 &= |f(V_{000}) - f(V_{101})| \\ &\quad + |f(V_{100}) - f(V_{001})|, \\ \nabla 3 &= |f(V_{000}) - f(V_{110})| \\ &\quad + |f(V_{100}) - f(V_{010})|. \end{aligned}$$



(a) (b)

Figure 3. Multivalued 3D images can be represented as binary 3D images by applying an image segmentation procedure. In both (a) and (b) rows 1 and 3 show (top view) slices 14–21 of a 64-slice NMR reconstruction of a human brain. Rows 2 and 4 are slices of the binary image obtained by thresholding. The range $\mathscr{D}'$ for (a) was chosen to include all voxels within the cortex, and a much narrower range was chosen for (b). [Adapted with permission from G. T. Herman, D. M. Kramer, P. C. Lauterbur, A. M Rudin, J. S. Schneider, and J. K. Udupa, "The three-dimensional display of nuclear magnetic resonance images," *Proc. 9th SPIE Conf. Applied Optical Instrumentation in Medicine*, March 22–25, 1981]

Other 3D edge operators have been defined as generalizations of the 2D Hueckel operator [ZUCK81] and the 2D Prewitt operator [MORG80a]. A more detailed discussion of the problem of 3D image segmentation is given by Altschuler et al. [ALTS81].

3. DATA STRUCTURES

In 3D image analysis it is necessary to represent object information, such as CT number, which corresponds to different object points. But methods which store every object space point are frequently excessive in their space requirements. For example, if an object can be modeled by dividing it into 1000 parts along each axis, then the internal representation requires 1 billion points. As in the case of 2D image representation, more efficient methods of representation can be made possible by exploiting special properties of the data.

In designing methods for storing high-resolution data, a number of trade-offs, involving, among others, computation necessary to access the data, overhead storage needed, and visual quality of display generated, must be considered. We present four different methods here. The first is the well-known *dope vector* method used by compilers in the linearization of multidimensional arrays. The other methods are essentially tree data structures which are suited for 3D region representation. The latter methods are called *marginal*, *symmetric recursive*, and *asymmetric recursive indexing*, each of which requires successively more overhead storage, less computation, and more storage efficiency with data homogeneity. In the following elaborations we assume that X, Y, and Z are the three coordinates which are divided into N equal parts, giving a maximum of N^3 data elements to be stored.

3.1 Dope Vectors

This is an exhaustive method of representation which assumes that each data element requires the same amount of storage. The data elements occupy consecutive locations (addresses) with the first subscript X varying most rapidly, Y next, and Z least rapidly. In order to find the descriptor for index (X, Y, Z), six comparisons are made to determine whether X, Y, and Z are within their respective upper and lower bounds; if so, the address of the descriptor is calculated using the coefficients (or *strides*) Di from the formula $D0 + (X*D1) + (Y*D2) + (Z*D3)$. If A is the first address and indices are in ranges $L1 \leq X \leq U1$,

$L2 \le Y \le U2$, $L3 \le Z \le U3$, then the strides Di are defined as $D0 = A - [L3*(U2 - L2 + 1)*(U1 - L1 + 1)] - [L2*(U1 - L1 + 1)] - L1$, $D1 = 1$, $D2 = U1 - L1 + 1$, and $D3 = [(U2 - L2 + 1)*(U1 - L1 + 1)]$. The indexing program only needs to store the four so-called *dope* vectors: $(D0)$, $(U1, L1, D1)$, $(U2, L2, D2)$, and $(U3, L3, D3)$. For example, in the case of a $2 \times 3 \times 4$ array with $(L1, L2, L3) = (1, 1, 1)$, $(U1, U2, U3) = (2, 3, 4)$, and $A = 0$, the dope vectors are (-9), $(2, 1, 1)$, $(3, 1, 2)$, and $(4, 1, 6)$. Thus the location of element (X, Y, Z) of this array is given by $-9 + X + 2Y + 6Z$.

This method is useful when all elements must be stored, as is the case with highly fluctuating data. Accessing an element requires two multiplications and three additions; a parallel implementation can reduce the cost to one multiplication and two additions. Local operations are simple to implement, since if a voxel is accessed at address α, then its n-adjacent voxels are obtained by adding (or subtracting) the strides to (from) α. It follows that data in cross sections can also be easily obtained.

3.2 Marginal Indexing

This method is based on the idea of using a linked set of tables to store the data. In order to find element (X, Y, Z), X is used as an index into the first table to get a pointer to a second table. Then Y is used as an index into the second table to get a pointer to a third table. Each block in the third table contains the data for the given values of X and Y and varying values of Z. By using Z as the index, the proper element can be found.

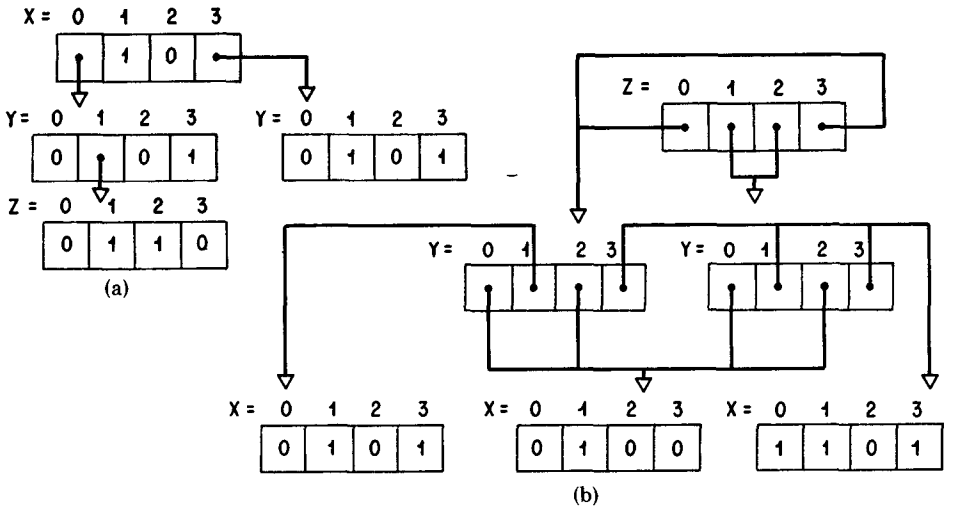
The method may be thought of as specifying a slice of the volume by X at the first level (root) of a three-level tree, then a strip of this slice by Y (at the second level), and finally an element of this strip by Z (at the third level). When all data elements of a slice are identical, data are stored at that level (either in the table or via a pointer) and no pointer proceeds from the first level to the second. Similarly, when a strip is homogeneous, no pointer proceeds from the second level to the third level. If two or more slices are identical but nonhomogeneous, then a single copy of the slice is stored, in effect, with multiple pointers pointing to the same slice. Similarly, when two or more strips are identical, a single copy of the strip is maintained. An example of this method of indexing is shown in Figure 4 for the case of binary-valued data elements. The storage required by marginal indexing is sensitive to the ordering of indices. However, the maximum storage required for pointers is $N + N^2$; thus the storage complexity of the method is dominated by the N^3 required for data.

Accessing data in cross-sectional slices and strips of slices is quite simple with marginal indexing. Finding the values of voxels n-adjacent to (X, Y, Z) is best done by making independent accesses through the data structure for appropriate combinations of $X \pm 1$, $Y \pm 1$, and $Z \pm 1$.

3.3 Symmetric Recursive Indexing

This method differs from marginal indexing in the method of partitioning the volume. The cubic space is subdivided into eight subcubes (octants) of equal volume. Each of these octants will either be homogeneous (e.g., uniform attenuation) or have some nonuniformity. The heterogeneous octants are further divided into suboctants. This procedure is repeated as long as necessary until we obtain blocks (possibly single voxels) of uniform properties. The method is thus a 3D version of the 2D quad-tree data structure, which latter has been expounded by Tanimoto [TANI80].

Recursive indexing can be modeled by a tree of degree 8, variously referred to as an oct-tree [JACK80], octal-tree [SRIH80], and octree [MEAG80]. Each nonterminal node of an oct-tree has eight successors and the leaves of an oct-tree correspond to data elements. In order to access a single point (X, Y, Z), the binary representations of X, Y, and Z are obtained as $x_0x_1, \ldots, x_{M-1}$, $y_0y_1, \ldots, y_{M-1}$, and $z_0z_1, \ldots, z_{M-1}$, respectively, where $M = \log_2 N$. At the top level of the tree is a table of eight elements, one for each octant. The index of this table, called *son-type*, is obtained by concatenating the high bits of the X, Y, and Z coordinates. Thus $x_0y_0z_0$ selects an octant from the table. If that octant is to be further subdivided, then $x_1y_1z_1$ selects a suboctant, and so on. The oct-tree representation of an object



	Storage	
Index ordering	Pointers	Data
XYZ	3	13
XZY	10	10
YXZ	5	15
YZX	16	12
ZXY	12	16
ZYX	12	12

(c)

Figure 4. Marginal indexing of a 4 × 4 × 4 binary array: (a) linked list using index order *XYZ*; (b) equivalent list using index order *ZYX*; (c) storage requirements for 6 permutations of indices.

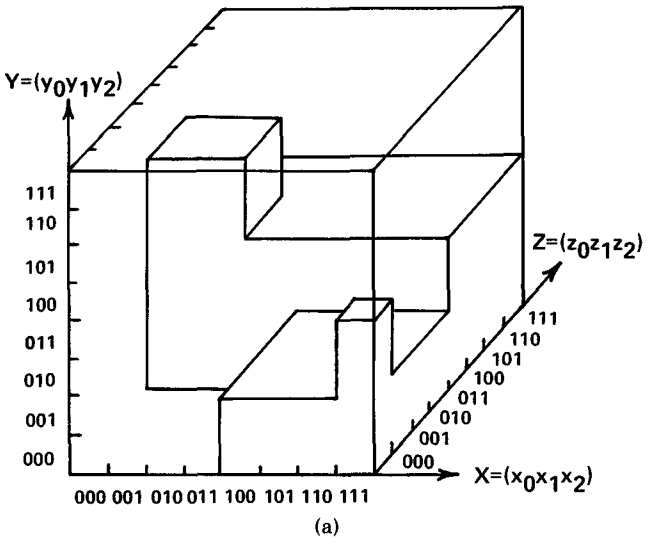
is shown in Figure 5 where, again, the elements are binary valued. The equivalent list representation of this tree is (010(10000000)(110011(00001010)0)100).

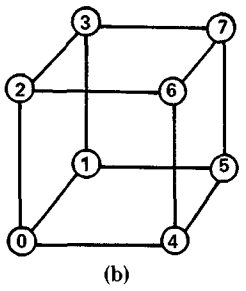
The maximum overhead storage required by the method of recursive indexing is given by

$$\sum_{i=1}^{M-1} 8^i = \frac{N^3 - 8}{7}.$$

As a comparison with marginal indexing, consider the case where $N = 128$, or each axis is divided into 128 equal parts. In this case there are a maximum of N^3, or approximately 2.1 million data items. Marginal indexing requires a maximum of 16,512 pointers, or less than 2 percent of the data, whereas recursive indexing requires 299,592 pointers, which is less than 15 percent of the data. When the arrays are sparse, these overheads can be expected to be significantly lower. As in the case of marginal indexing, the storage required by recursive indexing can be further minimized by detecting and maintaining a single copy of all isomorphic subtrees.

The method of recursive indexing is particularly suitable for hardware implementation owing to the simple bit manipulations necessary to determine pointers at each level. An algorithm for generating the oct-tree representation of a binary image from the quad-trees of its slices is given by Yau and Srihari [YAU81a]; the method allows voxels to have "don't-care" values, which allows the development of a compact data structure even in the case of a volume whose sides are not a power of 2. This algorithm was applied to a 64 × 64 × 64 binary array representing the human brain (8 of whose slices are shown in Figure 3a). The oct-tree, with 2539 nonterminal nodes over 6 levels, was constructed in 72 seconds on a CDC Cyber 174 computer. (Since this oct-tree can be completely represented using 8 12-bit fields per nonterminal node, the 243,744 bits needed for the oct-tree reflects savings over the 262,144 bits needed for the array.)





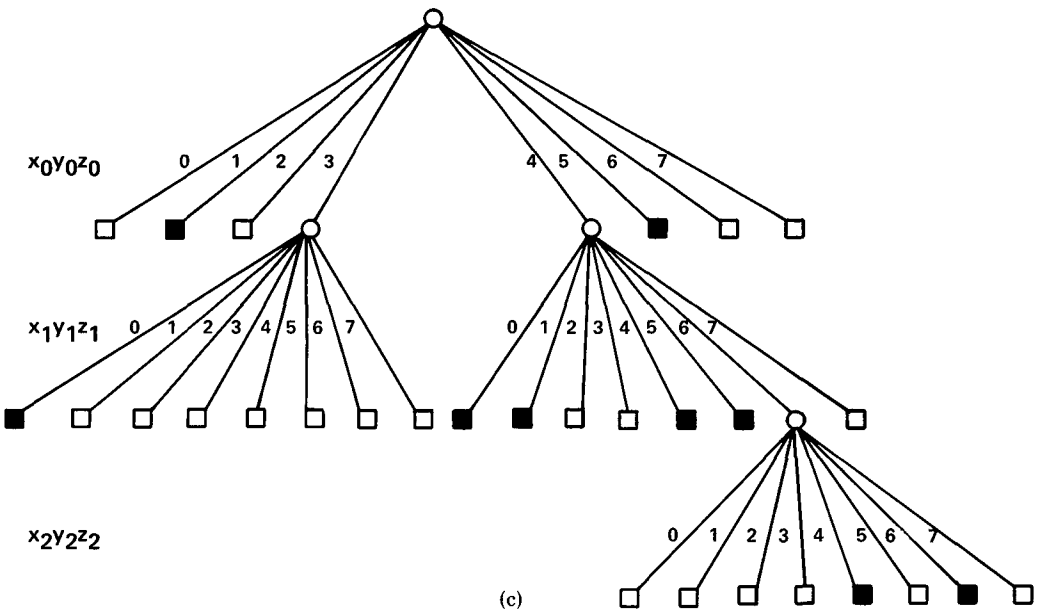


Figure 5. Recursive indexing of a binary-valued 3D object: (a) object having $8^3 = 512$ data items, (b) octant indices as specified by concatenating corresponding index bits, (c) oct-tree requiring 37 data items including pointers.

Operations with Oct-Trees. Certain operations on images are simple to implement with recursive indexing. For instance, clockwise *rotation* by 90° along an axis perpendicular to, and centered at, the midpoint of the *YZ* cross section is done by permuting the son-types at every node according to the permutation {(0, 2), (1, 0), (2, 3), (3, 1), (4, 6), (5, 4), (6, 7), (7, 5)}. *Scaling up* by 2 is done by deleting the root node and making one of its 8 sons the new root. Similarly, *scaling down* by 2 is done by making the root node to be one of the sons of a new root node.

Translation of an oct-tree-represented object by integer distances along the three axes is somewhat more involved. The goal of a translation algorithm is to convert both a *source* oct-tree, representing a 3D object, and a movement vector into a *target* oct-tree representing the translated object. The basic strategy is to generate node values of the target tree in a postorder traversal (in which the sons of a node are visited first) by simultaneously traversing the source tree. Each target node is compared with a list of the source tree's nodes whose octants overlap the target node's octant; thus a target node may derive its value from source nodes that terminate at a higher level. Further details of this translation algorithm are given by Jackins and Tanimoto [JACK80].

Since an oct-tree maintains object voxels in a spatially presorted format, a *hidden-surface view* can be generated without searching or sorting. If octants are visited and displayed in the proper sequence, as determined by the location of the viewer, no octant can obscure the view of an octant later in the sequence. Thus if voxels are displayed such that later voxels overwrite earlier voxels on the screen, a hidden-surface view will be generated [MEAG80]. Other operations involving spatial sorting, such as convex hull computation (see Section 5.3.3), can be performed efficiently with recursive indexing [YAU81b].

3.4 Asymmetric Recursive Indexing

One way to extend recursive indexing is to employ some knowledge about the particular volume being represented so that its data structure can be more concise. In this method, the space to be subdivided is broken into rectangular parallelepipeds rather than into cubes. Division of the space is done with planes perpendicular to the *X*, *Y*, and *Z* axes, but the planes are not equally spaced and there can be a different number of planes along each axis. Because the subdivision is variable along the axes, storage can be saved by intelligent subdivision. For example, if a very small object is to be represented in the middle of a large empty volume, the equal subdivision model will have to traverse many levels of the tree before it gets to the detail of the object. Using unequal subdivision, two closely spaced planes along each axis will exactly single out the object so that the next level of subdivision can begin at the proper detail.

Point accessing is more difficult with this model because each level of subdivision needs three vectors that indicate the location of the *X*, *Y*, and *Z* planes which make up the subspaces. The *X*, *Y*, and *Z* components of the desired point must be located in these vectors so that the indices can be used to select the proper subspace.

The concept of asymmetric recursive indexing is illustrated in Figure 6, which is an oct-tree of the object in Figure 5a. In this model, at each nonterminal node, three indices representing the positions of the partitioning planes are placed. Let (X_{jk}, Y_{jk}, Z_{jk}) represent the indices of the partitioning planes at the kth nonterminal node of level j, where $0 \le k \le 7$, $0 \le j \le \log_2 N - 1$. Then point (X, Y, Z) is accessed by selecting a branch at each nonterminal node by the concatenated bits $x_j y_j z_j$, where

$$x_j = \begin{cases} 1 & \text{if } X \ge X_{jk}, \\ 0 & \text{otherwise;} \end{cases}$$

$$y_j = \begin{cases} 1 & \text{if } Y \ge Y_{jk}, \\ 0 & \text{otherwise;} \end{cases}$$

$$z_j = \begin{cases} 1 & \text{if } Z \ge Z_{jk}, \\ 0 & \text{otherwise.} \end{cases}$$

3.4.1 Distributed Indexing

A variation of asymmetric recursive indexing—one more suitable for continuous

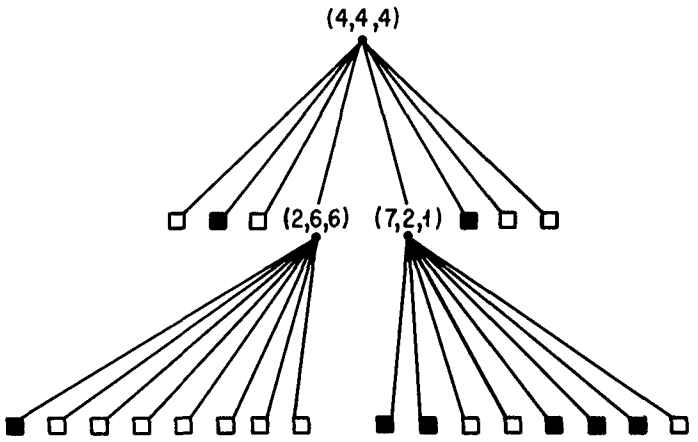


Figure 6. Oct-tree of object in Figure 5a obtained by asymmetric recursive indexing. Each nonterminal node has three indices that specify the location of partitioning planes.

spaces—is to break down the volume into rectangular parallelepipeds that are not necessarily aligned with an axis. The parallelepipeds can be any size and orientation, but any point that is not in one of these parallelepipeds is empty. In this scheme, therefore, one finds a point's attributes by determining which parallelepiped it occupies. If the point is found in a parallelepiped, then the next level of detail is examined. This level has another set of parallelepipeds that isolate the nonempty parts within it. Sublevels in this model operate in the coordinate space of the parallelepiped that encloses them, so accessing of points is done recursively. The implementation of this method, as proposed by Reddy and Rubin [REDD78], is as follows. The system is presented with a point (X, Y, Z) that lies in the object space. At the top level of the structure there are N transformation matrices, T_1 through T_N, and each is a 4×4 transformation that converts the object space point into the coordinate system of its parallelepiped. In this new system, the point $(0, 0, 0)$ is at one corner of the subspace, and the point (UX, UY, UZ) is at the diagonally opposite corner. The object space point is within parallelepiped i if, after transforming (X, Y, Z) through T_i to become (X', Y', Z'),

$$0 \leq X' \leq UX_i \wedge 0 \leq Y' \leq UY_i \wedge 0 \leq Z' \leq UZ_i.$$

If none of the parallelepipeds at a given level is found to contain the requested point, then that point is reported to be empty. If the point falls within one of the parallelepipeds, then there is a possibility that it is nonempty. To find out, all of the subparallelepipeds within this new object space must be searched. The algorithm is recursive and the point (X', Y', Z') is extracted from the subspace. This recursion continues until the level at which there is no more detail of the object is reached. At this level a point that falls within a parallelepiped is nonempty and the properties of the points are extracted.

3.4.2 Dynamic Indexing

A data structure that has been proposed for representing multidimensional accumulator arrays, such as those encountered in implementing the Hough transform (see DUDA73, p. 335), is known as *dynamic indexing* [O'ROU81]. It could also be applied to the representation of 3D images.

Here the space is divided into rectangular parallelepipeds by means of planes perpendicular to the axes. Unlike the oct-tree data structure, a given rectangular parallelepiped is divided into only two halves (*upper* and *lower*) by means of a plane which is perpendicular to, say, the X axis, and positioned midway along the extent of the parallelepiped along that axis. The two resulting parallelepipeds are divided recursively until each parallelepiped is uniform. The

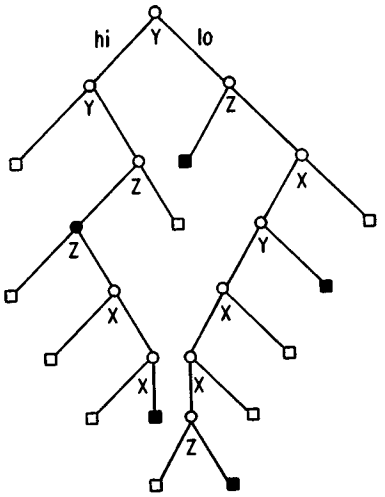


Figure 7. Binary tree of object in Figure 5a obtained by dynamic indexing. Each nonterminal node has an index that specifies the direction of the partitioning plane. The left and right subtrees, respectively, describe the upper and lower halves of the space with respect to the plane.

resulting data structure is in the form of a binary tree, where each nonterminal node has an index that specifies the direction of the plane as X, Y, or Z. The left subtree can be regarded as describing the upper half of the parallelepiped and the right subtree as describing the lower half of the parallelepiped. Using this convention, the binary tree shown in Figure 7 is an exact representation of the object in Figure 5a.

As in the case of other asymmetric indexing methods, and unlike the symmetric oct-tree, the binary tree representation of a given image is nonunique, unless other criteria for determining the directions of the partitioning planes are imposed. Since the space is divided into only two halves at each level, the binary tree can be expected to have more levels than the oct-tree.

4. TOPOLOGICAL REPRESENTATIONS

Fundamental to the definition of high-level representations of digital images are the topological concepts of connectedness and mathematical relationships between volume subsets (containment, adjacency, etc.). In this section we discuss definitions of connectivity, components, and holes in binary-valued 3D discrete images and consider algorithms for deriving certain topological representations. Some of these ideas are straightforward generalizations from the 2D case, for example, MINS72 and ROSE76; related ideas in discrete 3D computational topology are also given in GRAY70, MYLO71, PARK71, and YONE80.

4.1 Components

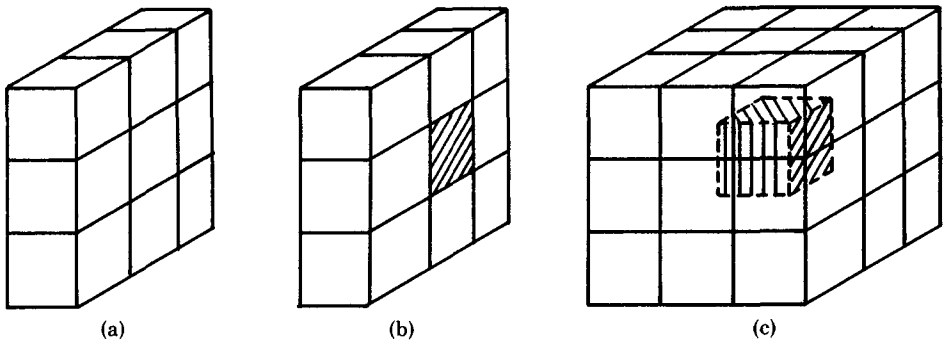
The concept of adjacency of voxels can be used to define connectedness of objects in 3D discrete space. Corresponding to different types of adjacency, we have different types of connectivity. If S is a finite set of voxels, then two voxels V and U are *n-connected* in S if there exists a sequence of n-adjacent voxels (or n-path) between V and U, $V = v^0, v^1, \ldots, v^m = U$, all in S, such that v^{p-1} is n-adjacent to v^p, for $1 \leq p \leq m$. Thus a sequence of object voxels that are 6-neighbors of each other is a 1-connected set, which is sometimes referred to as being *6-connected*. Similarly, a sequence of 26-neighboring object voxels is a 3-connected set or *26-connected*, and so on.

An *n-component* of S is an equivalence class of the partition induced by n-connectivity on S. Further, we say that an object is n-connected when it has a single n-component. An n-connected object is also referred to as a *digital solid*. In order to avoid the violation of certain topological invariants, the order of connectivity chosen for $\bar{S}$ should be $\bar{n} \neq n$; we limit the choice of $(n, \bar{n})$ to either (1, 3) or (3, 1).

4.2 Holes

In the case of 2D digital objects, holes of S are defined as finite components of $\bar{S}$. This definition in the 3D case is equivalent to the notion of a *cavity*, as in the case of the enclosed space of a hollow shell. Thus we define a cavity of S as a finite $\bar{n}$-component of $\bar{S}$. The *background* of S is simply the infinite $\bar{n}$-component of $\bar{S}$.

Another type of hole encountered in the 3D case is that of a *tunnel*, as in the case of the puncture in a doughnut. A tunnel (or handle) is said to be formed when we take a solid, say a sphere, make two circular holes in its surface, and join them by connecting the two ends of a tube to the two



Object	K (Components)	C (Cavities)	T (Tunnels (Genus))	V (Vertices)	E (Edges)	F (Faces)	η (Conn. No.)	χ (Euler Char.)
a	1	0	0	32	60	30	2	1
b	1	0	1	32	64	32	0	0
c	1	1	0	56	108	54	4	2

(d)

Figure 8. Three digital solids. (a) closed object, (b) object with a tunnel; (c) object with a cavity; (d) their topological invariants.

holes. This process yields a sphere with one handle. Additional handles can be added by the same process. A sphere with one handle can be continuously deformed into a surface that can be thought of as the surface of a doughnut. This surface, called a torus, can be mathematically described by rotating a circle which lies in a plane about a line in that plane that does not intersect the circle. A sphere with T handles can be put into direct correspondence with a torus with T tunnels. The number of tunnels present in an object is sometimes referred to as *handle number* or *genus* of the object [BARR70].

The genus of a digital solid can be determined by a property that holds for *netted* surfaces. A closed surface ϕ that can be divided into $\mathcal{F}$ *faces* by drawing $\mathcal{V}$ *vertices* and $\mathcal{E}$ connecting *edges* that are geodesics (shortest paths on the surface) so that faces are simply connected (i.e., each non-self-intersecting closed curve in the face can be deformed to become a point while lying in the face) is a closed netted surface. A topological property of such a netted surface [HILB52] is that

$$\mathcal{V} - \mathcal{E} + \mathcal{F} = 2 - 2T. \tag{4.1}$$

(From differential geometry [COXE69], it can also be shown that $2 - 2T$ is proportional to the *surface characteristic* of ϕ, which is defined by the integral of Gaussian curvature over ϕ.)

The surface of a digital solid S consists of square faces between the abutting voxels of S and $\bar{S}$. Such a surface is a closed netted surface whose faces are simply connected. It follows that the genus of S can be determined by counting the number of faces, edges (where two faces meet), and vertices (where four faces meet) on the surface of S. Equation (4.1) can be verified for the three 1-connected digital solids shown in Figure 8 by counting the number of faces, edges, vertices, and tunnels. It should be noted that sometimes an edge may locally belong to k surfaces, in which case the edge is counted k times; if S consists of two voxels

that touch at an edge, for example, the common edge belongs to two local surfaces and is therefore counted twice.

4.3 Euler Characteristic

The Euler characteristic of a 2D figure with K components and H holes is defined as $\chi = K - H$ (see DUDA73). Its 3D generalization is defined (see GRAY70) for a 3D scene with K components, C enclosed cavities, and T tunnels as

$$\chi = K - T + C. \tag{4.2}$$

It can be shown that χ is computable by a local operator, as follows.

In general, when an object consists of more than one component and closed cavities are present in components, then for each netted surface ϕ_1 which encloses an object or cavity, we have from (4.1)

$$\mathcal{V}i - \mathcal{E}i + \mathcal{F}i = 2 - 2Ti. \tag{4.3}$$

For the entire object one may define a *connectivity number*,

$$\eta = \sum_{i=1}^{K+C} (2 - 2Ti). \tag{4.4}$$

Combining (4.1), (4.3), and (4.4) and the fact that $\sum_i Ti = T$,

$$\eta = 2\chi = \sum_{i=1}^{K+C} (\mathcal{V}i - \mathcal{E}i + \mathcal{F}i). \tag{4.5}$$

Thus the consequence of (4.5) is that the Euler characteristic (and connectivity number) of a 3D object in cellular space is computable by a local operator that counts the number of faces, edges, and vertices. These concepts are also illustrated in Figure 8d.

Consider next an algorithm to compute connectivity number η of a 3D digital object. Counting the contribution of each object voxel to η is rather slow since it requires a procedure operating on 27 voxels for each object voxel; and the potentially fast method of enumerating all possible local configurations and storing their individual additive contributions to η is impractical due to the 2^{27} possible configurations. An alternative formulation is to visit each vertex of each object voxel exactly once and determine the contribution of the $2 \times 2 \times 2$ voxel neighborhood of the vertex to η. The 2^8 possible configurations can then be described by an 8-bit byte. The contribution to η by each vertex has to take into account the fact that faces and edges touching at that vertex also touch other vertices. Lobregt et al. [LOBR80] derive the table entries for the particular case where η needs to be evaluated in the $3 \times 3 \times 3$ neighborhood of an object voxel and then show them to belong to one of 22 basic different possibilities.

4.4 Adjacency Trees

The topology of a 2D black-and-white figure is given by its *adjacency tree*, a graph showing the containment relationship between the background, components, and holes. An algorithm for constructing the adjacency tree of a 2D figure, which is based on a single raster scan, is given by Buneman [BUNE69]. The adjacency trees of parallel 2D slices of a 3D object, together with adjacency information about components in successive slices, provides a simple topological representation. The adjacency of components in neighboring slices can be determined by maintaining a set of 2D coordinates corresponding to each component and checking for overlaps.

Let $\{\zeta_i\}$ be the forest of adjacency trees of parallel slices. The choice of order of connectivity in determining ζ_i is as follows: if $n = 1$ use 4-connectivity; otherwise use 8-connectivity. Let ijk represent the kth node (component of S or $\bar{S}$) at the jth level of ζ_i, and let $[ijk]$ represent the set of 2D coordinates (figure) corresponding to ijk. Then we define a *compatibility* relation between nodes as

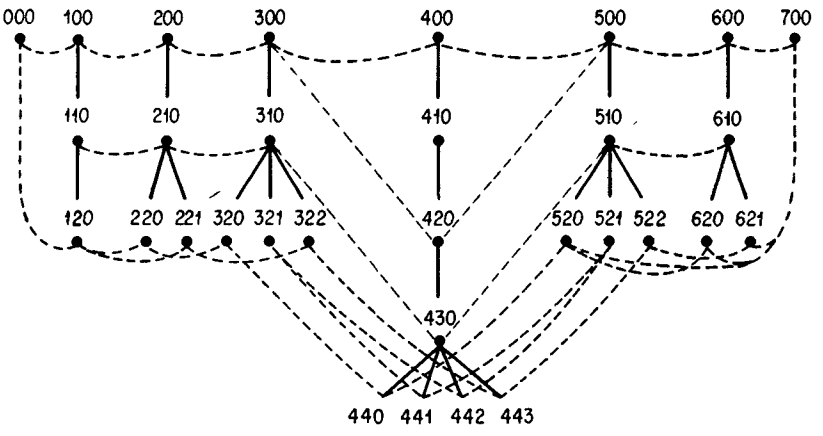
$$\psi = \{(ijk, i'j'k') \mid\mid i - i' \mid = 1, \text{ and } [ijk \text{ overlaps } [i'j'k']\}$$

where *overlap* between two figures implies the existence of a common point (for $n = 1$), a common 4-connected point (for $n = 2$), or a common 8-connected point (for $n = 3$).

Next we show that $\{\zeta_i\}$ and ψ are sufficient to determine various topological properties. Define ijk to be compatible to $i'j'k'$ if and only if there exists at least one sequence of the form $a_0, a_1, \ldots, a_m$ such that $a_0 = ijk$, $a_m = i'j'k'$, and $(a_p, a_{p+1}) \in \psi$ for

Row number	Slice number				
	0 and 7	1	2 and 6	3 and 5	4
0	0000000	0000000	0000000	0000000	0111110
1	0000000	0000000	0000000	0000000	1000001
2	0000000	0011100	0011100	0011100	1011101
3	0000000	0010100	0010100	0010100	1010101
4	0000000	0010100	0011100	0011100	1011101
5	0000000	0010100	0011100	0010100	1010101
6	0000000	0010100	0011100	0010100	1011101
7	0000000	0010100	0011100	0010100	1010101
8	0000000	0010100	0011100	0011100	1011101
9	0000000	0010100	0010100	0010100	1010101
10	0000000	0011100	0011100	0011100	1011101
11	0000000	0000000	0000000	0000000	1000001
12	0000000	0000000	0000000	0000000	0111110

(a)



(b)

Figure 9. Adjacency tree representation· (a) data for slices 0–7; (b) the 8 adjacency trees of slices 0–7, where dotted lines indicate connectivity. The object consists of two components, one of which has a single tunnel, the other of which has two tunnels and a cavity, which in turn has a tunnel

$1 \leq p \leq m$. The resulting equivalence classes of $\{ijk\}$ represent the components and cavities of S; a compatible set of nodes of S is a component, and a compatible set of nodes of $\bar{S}$ is either a cavity or a background. For a given level j, two compatible nodes ijk and $i'j'k'$ $(i < i')$ are said to enclose $t - 1$ tunnels if there exist t sequences of compatible nodes with no two sequences $a_0a_1, \ldots, a_{m-1}a_m$ and $a_0b_1, \ldots, b_{m-1}a_m$ having $a_l = b_l$, $a_r = b_r$ for some l, r $(1 \leq l < r < m)$.

The forest of adjacency trees of eight slices of a 3D object with $n = 3$ is shown in Figure 9. The trees can also be represented in terms of list structures as $\zeta_0 = \zeta_7 = (0)$, $\zeta_1 = (((0)))$, $\zeta_2 = \zeta_6 = (((0)(1)))$, $\zeta_3 = \zeta_5 = (((0)(1)(2)))$, and $\zeta_4 = ((((((0)(1)(2)(3)))))$. The outermost parentheses indicate the background, and the corresponding pairs of inner parentheses alternately denote components and holes. For example, in $\zeta_2 = \zeta_6 = (((0)(1)))$, the outermost parentheses denote the background, the next set of parentheses denotes a single component, and the two sets of innermost parentheses indicate two holes. The label in the innermost parentheses represents a leaf node, which is either a component or a hole depending on whether it is nested an even or odd

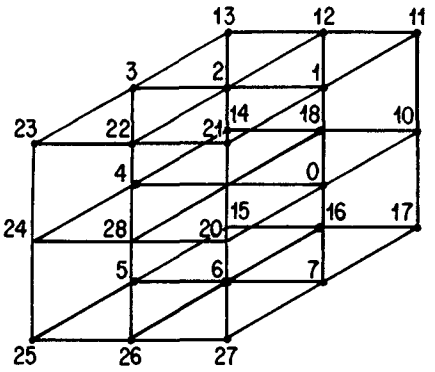


Figure 10. Chain code for digital space curves The nodes are numbered 0–7 and 10–18, 20–28 around the middle and the two ends, respectively (The central node of the middle slice touches nothing, and so it is not numbered.)

number of times. The set of nodes of the object has four equivalence classes. Equivalence classes {410} and {110, 210, 310, 430, 510, 610} correspond to components, equivalence class {321, 441, 442, 521} corresponds to a cavity, and the remaining equivalence class {000, 100, 120, 200, 220, 221, 300, 320, 322, 400, 420, 440, 443, 500, 520, 522, 600, 620, 621, 700} corresponds to the background. The object has four tunnels: a tunnel in the first component due to two sequences between 000 and 700, and one tunnel in the cavity due to two sequences between 321 and 521.

5. GEOMETRICAL REPRESENTATIONS

Representation methods that are concerned with details of shape information are referred to as *geometrical representations*. In this section we consider the exact representation of space curves, derivation and representation of surfaces, and the representation of shape by properties.

5.1 Space Curves

A *digital space curve* may be defined as a connected set of voxels all but two of which have exactly two neighbors in the set, while the exceptional two—the end voxels—each have exactly one neighbor in the set. The method of chain coding in 2D can be easily extended to digital space curves [FREE74]. From each data node there are 6, 18, or 26 possible directions to the next node depending on whether the order of adjacency n is 1, 2, or 3. Thus each possible direction can be uniquely designated by 3 bits for 1-connected space curves and by 5 bits for 2- or 3-connected space curves. One particular coding arrangement for the 26 possible directions of a 3-connected space curve is shown in Figure 10. Since each 26-neighbor of a given voxel can be designated by specifying for each coordinate whether it is incremented by 1, unchanged, or decremented by 1, each possible direction of a 3-connected space curve can also be specified by a 3-digit ternary number [ROSE80].

5.2 Surfaces

An important representation of a 3D object is by means of its 3D surface. Such a representation is necessary for shaded display on a graphics screen. We consider next the problem of exact representation of surfaces of objects in cellular space, where we assume that object voxels have value 1 and background voxels have value 0. Our discussion thus excludes methods of representing 3D manifolds by triangular facets [FUCH77] and interpolative patches [YORK80]; the former, however, does yield a fast algorithm that is used in many CT display programs [COOK80].

We have seen previously (Section 4.2) that the surface of an object S consists of voxel faces that are at the interface of S and $\bar{S}$. Since the voxel is a convenient primitive element for 3D data structures, it is useful to redefine the concept of surface in terms of voxels. This can easily be done since a face is uniquely defined by two abutting voxels. Thus the surface of an object S in 3D cellular space can be defined as the set of voxel pairs

$$\mathcal{S}[S] = \{(V, U) \mid V \in S, U \in \bar{S}, VR_1U\}.$$

If S is a finite n-connected object with K enclosed cavities, then $\mathcal{S}[S]$ can be uniquely partitioned into one *external* surface and K *internal* surfaces. An example of the external surface of an object is shown with shading in Figure 1c; a procedure for shaded display of voxel faces is described by Herman and Liu [HERM79].

The surface of a 3D object can be represented by specifying a set of constituent faces or indirectly by means of border voxels, graphs (whose nodes are faces and whose edges specify touching faces), and medial axes. In the following, we define these concepts and briefly describe algorithms for deriving such representations.

5 2.1 Borders

Determination of the surface of an object S, and the surface's partition, is facilitated by defining the *border* of S (which is a one-voxel-thick layer) as

$$B(S) = \{V \mid V \in S \text{ and } \mathcal{N}_{\bar{n}}(V) \wedge \bar{S} \text{ is nonempty}\}$$

where $(n, \bar{n}) = (1, 3)$ or $(3, 1)$ as before. Since

$$\{(V, U) \mid V \in B(S), U \in \bar{S}, VR_1U\} = \mathcal{S}[S],$$

it follows that the surface of S is uniquely specified by $B(S)$. More important, each n-component of $B(S)$ specifies either the external or one of the internal surfaces of S; or if resolution is low enough to separate two surfaces by fewer than two voxels, each specifies the union of the surfaces (the proof rests on showing that the border of an n-connected object without cavities is also n-connected). A detailed discussion of the mathematical properties of borders is given by Udupa et al. [UDUP79] and Morgenthaler and Rosenfeld [MORG80b].

A method of determining whether a voxel V is an element of $B(S)$ is to determine whether any element of $N_{\bar{n}}(V)$ belongs to $\bar{S}$. This leads to the following parallel algorithm for determining $B(S)$: if $\{V^i\}$ are the elements of $N_{\bar{n}}(V)$, then $V \in B(S)$ if and only if

$$\mathcal{f}(V) \cdot \overline{(\prod_i \mathcal{f}(V^i))} = 1$$

where $\prod$ denotes logical product. This is a fast method of determining the set union of all external and internal borders of the object.

In order to isolate a *single* external or internal border of an object, it is necessary to track connected voxels sequentially. An algorithm to detect a connected 3D border differs fundamentally from its 2D counterpart. A connected 2D border can be represented by a one-dimensional circular list of elements, where neighboring elements of the list represent adjacent border pixels; 2D border following is unique once the starting point and direction are specified [ROSE76]. Sequential traversal of the border of a 3D object requires a queue (or stack) to store "leads" to be followed; this is because there does not always exist a connected, non-overlapping traversal of the surface (or border) of a nonconvex object.

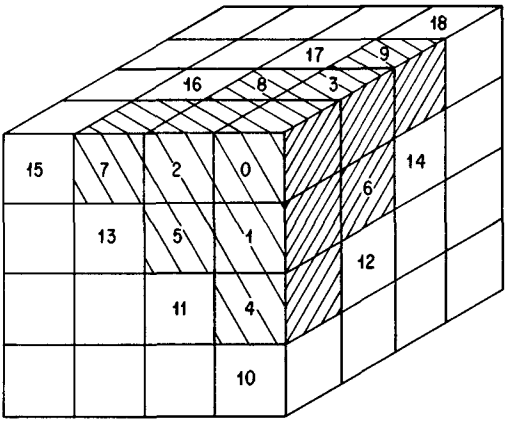


Figure 11. Example of border-following The initial voxel is labeled 0 Shaded faces correspond to the surface already traversed by the time the current voxel is 9, at this point voxels 10–18 are in the queue. The voxels were traversed in the order 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

A 3D border-following algorithm is illustrated with the aid of Figure 11 where object S is a $4 \times 4 \times 4$ cube, the initial voxel is V^0 which is the corner voxel labeled 0, and we chose $(n, \bar{n}) = (1, 3)$. Tracking proceeds by placing those voxels V of $N_n(V^0)$ that have value 1 and belong to $B(S)$ (because an element of $N_{\bar{n}}(V)$ has value 0), called $V^1 \cdots V^3$, into a queue Q and by marking V^0 as having already been traversed, say by changing its value from 1 to 2. Next the front element of Q, V^1, is removed as the current element of traversal. The elements of $N_n(V^1)$ having value 1 and belonging to $B(S)$ but not to Q, namely, $V^4 \cdots V^6$, are entered at the end of Q. Similarly, current element V^2 contributes V^7 and V^8 to Q, and so on. A detailed discussion of the complexity of this algorithm, its improvements, and its perform-

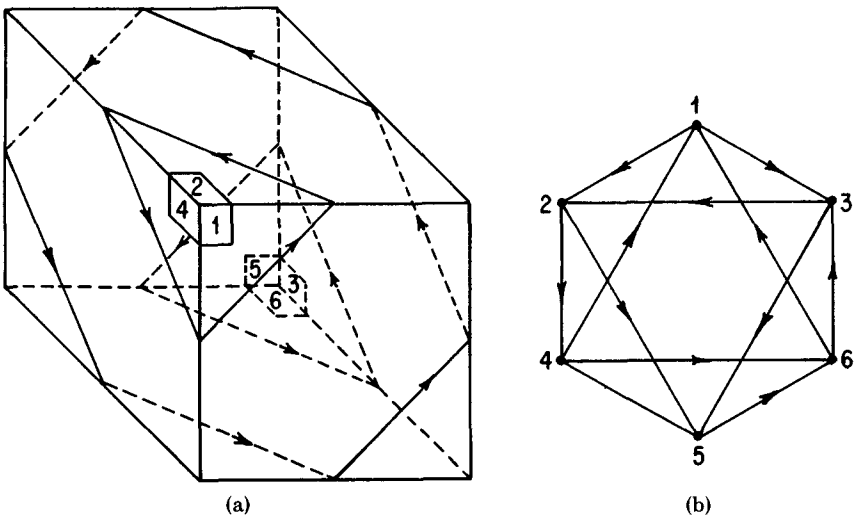


Figure 12. Directed graph representation of the surface of a 3D object: (a) each of the 6 types of voxel faces has 2 incoming and 2 outgoing edges as shown here for a single voxel; (b) the directed graph of the voxel in (a), where nodes correspond to voxel faces.

ance with CT data is given by Udupa et al. [UDUP79].

Instead of determining the border voxels and collecting the faces that contribute to $\mathscr{S}(S)$, an alternative approach to determining a single internal or external surface is to track "connected" faces. That such an algorithm is possible is based on the fact that the surface of any closed digital object can be represented by a directed graph whose nodes correspond to faces, each node having exactly two incoming and two outgoing edges. The incoming and outgoing connected faces are defined as shown in Figure 12 for each of the six types of faces of a voxel. It can be shown that each internal or external surface of a digital object corresponds to a maximally connected subgraph of such a directed graph [ARTZ81]. Therefore sequential traversal of a closed surface of the object is equivalent to determining a binary spanning tree of the appropriate maximally connected subgraph.

5 2.2 Medial Axes

Compact boundary representations of a 2D figure are given by the chain code and by the *medial axis transformation* (MAT) [DYER80]. Of these two, only the latter can be generalized to the 3D case.

Surface points are specified in the MAT representation by a collection of overlapping spheres. Given the centers and radii of the maximal spheres, it is possible to reconstruct the boundary representation by choosing those points that are not in the interior [BLUM79]. An algorithm for performing such a conversion in continuous 3D space is given by O'Rourke and Badler [O'ROU79].

The MAT is defined in the discrete case by a collection of overlapping digital spheres (or blocks). A voxel $V \in S$ is said to belong to the *medial axis* $M(S)$ if

$$\text{card}\{U \mid (U, V) = \min_{W \in B(S)} d(V, W)\} > 1$$

where d is a 3D metric and card A is the cardinality of set A. This definition implies that $M(S)$ consists of those voxels of S whose distances from $\bar{S}$ are local maxima; that is, V is an element of S if and only if for all $U \in N_n(V) \wedge S$, $\min_{W \in B(S)} d(V, W) \geq \min_{W \in B(S)} d(U, W)$. Thus S is the union of maximal digital spheres centered at the voxels of $M(S)$. If d is the maximum value metric, then the digital sphere is a cube and the MAT representation is similar to the oct-tree representation; in fact, they are equivalent if we constrain the locations of centers of digital spheres to powers of 2.

It should be noted that although the locus of centers of a 2D medial axis consists of linear structures, its 3D counterpart is not necessarily a space curve; $M(S)$, for instance, may consist of one-voxel-thick layers. Thus the 3D medial axis may be referred to as a *medial* or *symmetric layer*.

5.3 Features

The application of pattern-classification algorithms to 3D digital images requires representations in the form of descriptive features. The types of features that can be extracted depend on whether we are dealing with a sampled function of three variables or with a discrete 3D object (although the former can be segmented into regions and each region can be considered an object). In this section we consider features for representing the *shape* of a 3D object. The challenge of shape description is that of finding properties that not only discriminate between different shapes but are also invariant to certain transformations (such as translation, rotation, and magnification, among others) of the low-level data structure. We discuss here the 3D counterparts of properties that are frequently used in 2D image recognition.

5.3.1 Metric properties

The *surface area* of an object S in the cellular space is the number of elements in its surface $\mathcal{S}(S)$, measured in units of the area of a single voxel face. Similarly, the *volume* of S is the number of elements in S, measured in units of the volume of a voxel. If S is the digital representation of an object in continuous space, then surface area and volume are sensitive to the resolution of digitization S. A method of estimating volume as a diagnostic feature in CT is discussed by Cook [COOK80].

5.3.2 Analytic Descriptions

Analytic shape description by means of *expansion of the intrinsic function* and *moments* are well known in two dimensions [DUDA73]. The intrinsic function of a closed 2D figure specifies boundary curvature as a function of arc length, and the coefficients of the Fourier series expansion of the periodic intrinsic function constitute the *Fourier descriptors* of the shape of the figure. The concept of Fourier descriptors generalizes to space curves [BADR80] but not to arbitrary closed 3D surfaces. One representation that is related to Fourier descriptors is applicable to any closed surfaces satisfying the property that every point of the surface is viewable from at least one point in the interior. Such surfaces are termed "museum-viewable" or "stellar." The distance (or radius) from the viewing point to such a surface is a function of the direction from the point to the surface. We can describe the direction in space in terms of position on a sphere centered about the viewing point, with the radius to the surface given by a continuous function on that sphere. Given a polar coordinate system on the sphere, the radius of a smooth stellar surface can be represented as a weighted sum of *spherical harmonics* (which are the basis functions in a Laplace series expansion). The method of spherical harmonics have been used in describing cardiac shape in ultrasound data [SCHU79].

Moments of a 3D object S can be defined as

$$A(P, Q, R) = \sum_{V} (V1)^P (V2)^Q (V3)^R \cdot f(V)$$

where $f(V) = 1$ if $V = (V1, V2, V3)$ is in S, and 0 otherwise. Thus $A(0, 0, 0)$ is simply the volume of the object. Dividing $A(1, 0, 0)$, $A(0, 1, 0)$, and $A(0, 0, 1)$ by $A(0, 0, 0)$ yields the coordinates of the centroid of the object. If we shift the coordinate system so that the origin is at the centroid of S, then the resulting moments are its central moments $M(P, Q, R)$. The first central moments $M(1, 0, 0)$, $M(0, 1, 0)$, and $M(0, 0, 1)$ are zero, and the second central moments $M(2, 0, 0)$, $M(0, 2, 0)$, $M(0, 0, 2)$, $M(1, 1, 0)$, $M(1, 0, 1)$, and $M(0, 1, 1)$ are the moments of inertia of S. The eigenvectors of the matrix of second central moments are the directions about which S has maximum and minimum moments of inertia. Ratios of the eigenvalues describe fatness or thinness of S in different directions. Sadjadi and Hall discuss the use of 3D moments in discriminating between a parallelepiped,

pyramid, and cylinder in continuous space [SADJ79].

5.3.3 Convexity

A method of describing the shape of an object is to state whether it is convex or not and to describe its convex deficiency (the space between the smallest convex object enclosing Q and Q itself). An object Q in Euclidean (continuous) space is said to be *convex* if the line segment joining any pair of points of Q lies within Q, or, equivalently, if every straight line meets Q at most once. The *convex hull* of Q, denoted $H(Q)$, is the smallest convex object that encloses Q. The *convex deficiency* of Q is the set difference $H(Q) - Q$. If $Q = \{V^i\}$ is a finite set of points in Euclidean space, then $H(Q) = \{x \mid x = \sum_i \alpha_i V^i$ where $\sum_i \alpha_i = 1$ and $\alpha_i \geq 0\}$ is a polyhedron; intuitively, $H(Q)$, the convex hull, is the polyhedron obtained by stretching a rubber sheet over points of Q. The design of efficient algorithms for determining the convex hull of a finite set of n points in Euclidean space has received considerable attention. One such algorithm based on the recursive "divide-and-conquer" principle, is as follows:

(1) Sort Q into a list (according to the first coordinate) and partition the list into two (nearly) equal subsets Q_1 and Q_2.
(2) Compute $H(Q_1)$ and $H(Q_2)$ recursively.
(3) Apply a merge algorithm to $H(Q_1)$ and $H(Q_2)$ to obtain $H(Q)$ and halt.

Since the merge step is of complexity $O(n)$ in 3D space [PREP77], the overall complexity of the algorithm is $O(n \log n)$.

Definition of convexity for objects in discrete space requires some care. It is reasonable to define a set of voxels S to be convex if there exists a convex point set Q in Euclidean space such that its digitization $I(Q) = S$ for some definition of I (see Section 2.4). Since this definition of convexity does not lend itself to the formulation of a finite procedure, an alternative definition is needed. One such definition invokes the concepts of simple solidness and of a semidigital point.

A *simple solid* S is a finite 1-connected set of voxels, having no pair of voxels V^1, $V^2 \in S$, such that the line segment joining the digital points of V^1 and V^2 is parallel to an axis and lies outside the voxels of S. A point (a, b, c) is said to be a *semidigital point* if at least one of its coordinates is an integer.

Let $H(S)$ be the Euclidean convex hull of the digital points of S. We define a simple solid to be convex if and only if every semidigital point on the faces of $H(S)$ is near a digital point of S; two points (a, b, c) *and* (a', b', c') are said to be near each other if $\max\{|a - a'|, |b - b'|, |c - c'|\} < 1$. This definition of digital convexity does not necessarily imply the existence of a convex set Q in Euclidean space such that $I(Q) = S$; however, this one implies that there exists a convex object Q in Euclidean space such that $I'(Q) = E(S)$ where $E(S)$ is the half-cell expansion of S obtained by assuming the corner points of voxels of S to be the digital points in a new lattice and $I'(Q)$ is the digital binary image of Q in this lattice [KIM80].

On the basis of this second definition of digital convexity, the digital convex hull $\mathcal{H}(S)$ of a set of voxels S is given by $\mathcal{H}(S) = S \cup \Delta S$, where ΔS is the smallest set of voxels such that $\mathcal{H}(S)$ is simple and that each semidigital point on the boundary of $\mathcal{H}(S)$ is near a digital point of $\mathcal{H}(S)$. An example of a digital object S, the Euclidean convex hull of the digital points of S, $H(S)$, and the digital convex hull $\mathcal{H}(S)$ is shown in Figure 13.

Yau and Srihari consider the computation of digital convex hulls from oct-tree representations by taking advantage of the implied sorting in the data structure [YAU81b]. The complexity of such a convex hull algorithm is related to the compactness of the oct-tree rather than to the number of points in the object.

6. STRUCTURAL REPRESENTATIONS

Topological and geometrical properties are abstract image representations which can be mapped into class descriptions [DUDA73]. With complex objects it is more useful for cognitive purposes to represent objects by structural descriptions of spatial organization. Thus we define *serial-section image understanding* as the process of producing, from a slice-by-slice image of the external world, a representation of spatial organization that is useful to a higher level

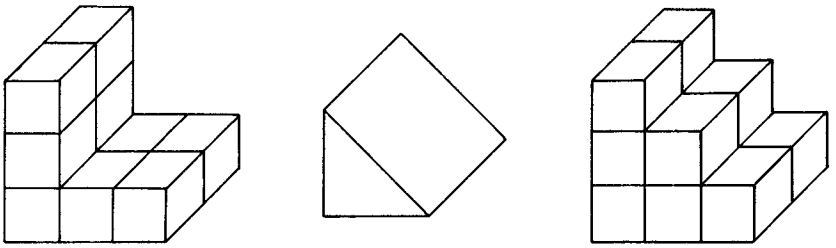


Figure 13. Digital convex hull representation: (a) object S; (b) Euclidean convex hull $H(S)$ of the digital points of S; (c) digital convex hull $\mathcal{H}(S)$.

human or machine cognitive processor (recognizer), uncluttered by irrelevant information. The goals of serial-section image understanding and of computer vision [MARR78] are identical; they differ, though, in the modality of input image data and therefore in the methods of deriving the representation.

Criteria for judging the effectiveness of a method of structural image representation are derivability, scope and uniqueness, and stability and sensitivity. *Derivability* is concerned with the ability to generate the representation, given the limits on image resolution, on storage and on computation time. *Scope* pertains to the class of shapes that the representation is suitable for; if the representation is to be used for recognition, it is also important that the description be *unique* for each member of that class. Within the above scope and uniqueness conditions, *stability* is a measure of how well we can capture more general (or less varying) properties of a shape, and *sensitivity* is a measure of our ability to distinguish finer shape characteristics.

Shirai has surveyed various methods of structural representation in the context of computer vision [SHIR78]. In what follows, we describe structural representations that seem promising for serial-section image analysis.

6.1 Generalized Cylinders

A large class of 3D objects, including bricks, pyramids, and vases, can be described by the so-called *generalized cylinder* (GC) representation [WINS76]. A GC is defined by a 3D space curve called the *axis* and by planar cross sections of arbitrary shape normal to the axis. Since there are an infinite number of GCs representing a single object, the following constraints help to assure uniqueness: (1) parametric form of cross-sectional shape (ellipse, for example), and (2) a sweeping function which describes how the cross section changes in shape as it moves along the axis. Given the GC representation, the surface of the object can be synthesized for the purpose of recognition.

In the case of complex shapes, the object has to be decomposed into simpler components before describing each component by a GC. Critical issues in GC representation are the following: segmentation into parts that can be simply described, selection of natural or useful axes, and definition of structural relationships among parts. Each of these tasks can only be approached using heuristic or interactive techniques.

Successful heuristics for 2D shape segmentation, or decomposition, have been based on identifying points of high *curvature* on the boundaries. The objective is to decompose a figure into near-convex parts. Generalization of this idea has to contend with the fact that a single 3D surface point can have different (normal) curvatures in different directions. A study of the relationship between the boundary curvature of an object and the curvature of the medial surface of the object, in continuous 3D space, has been done by Nackman [NACK80].

Selection of natural axes for GC representation has been attempted for computer vision by guessing from range data provided by a projected laser image [NEVA77]. More robust techniques are possible for serial-section images owing to complete spatial data. The medial axis of a 3D object is not

a suitable candidate for the natural axis, since it is not always a space curve and is extremely sensitive to minor indentations in the object's surface. Methods of topology preserving shrinking, or *skeletonization*, are more useful for representing an object's structure, particularly for elongated objects (see Section 6.2).

Structural relationships between segmented parts can be described by *attachment points*, which are like tinkertoys: parts have predefined points at which other pieces may be attached. For all primitive objects, cube, cylinder, and so on, the attachment points (base, top, side, and back) are predeclared. For nonprimitive objects, declaring a new attachment point involves specifying a transform or relative displacement with respect to the base of an object. This method can be used to describe complex objects by assemblies of GCs making use of axes inherent to the primitives. A program to translate such descriptions into polyhedral models and line drawings in an image synthesis mode is discussed by Agin [AGIN81], who gives examples of a screwdriver, a model airplane, and a chair.

A method of structural description that is related to the GC representation is the generalized blob model [SHAP80]. In this representation there are three kinds of 3D parts: sticks, plates, and blobs. Sticks are long thin parts that have significant length but very small width and depth, plates are flat wide parts that have two significant dimensions, and blobs are neither thin nor flat. All three kinds of parts are near convex; that is, a stick cannot bend very much, the surfaces of a plate cannot fold too much, and a blob can be bumpy but cannot have large concavities. A 3D object is then described by specifying the relative sizes of sticks, plates, and blobs and the constraints on how they are put together.

The derivation of generalized cylinder representations from serial section images have been considered for a bin of identical L-shaped mechanical parts [SRIH79], abdominal anatomy [SHAN80], and the human heart [SORO79].

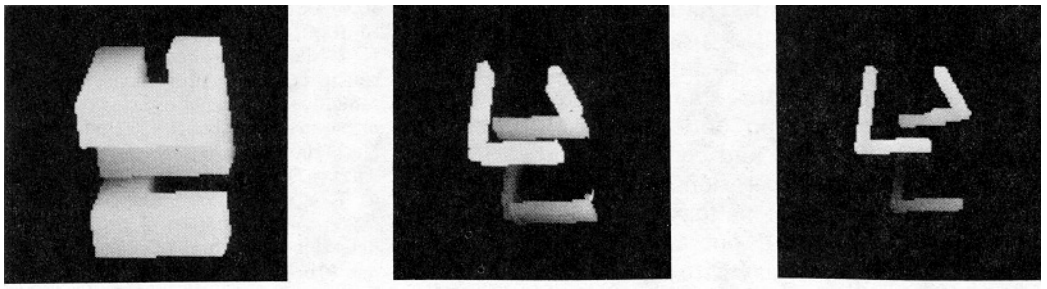
6.2 Skeletons

The skeleton of a 3D object is an intuitive concept that refers to a stick figure which captures the structure of the object. A 3D skeletonization (or thinning) algorithm may be based on the principle that a voxel V in an object S may be *deleted* as long as its deletion does not cause a change in the topology of S. The object that results when no voxel can be deleted is referred to as the skeleton of S.

The suggested skeletonization algorithm implies determining the relevant topological properties before and after the deletion of V. Because of computational considerations it is preferable to restrict the topological property to a local neighborhood of V. For example, one can determine within a $3 \times 3 \times 3$ neighborhood of V whether the deletion of V changes either the connectedness of remaining object voxels [SRIH79, TSAO81] or the connectivity number η [LOBR80]. (It should be noted that if we adopt the local criterion, there is no guarantee that the topology of the scene is unaltered after deletion.) However, such algorithms produce reasonably good results for elongated objects. In what follows we develop further details of a specific skeletonization algorithm.

Deletability alone does not yield a satisfactory skeletal axis, since a space curve will be eroded into a single voxel. To avoid this, criteria determining layer and end voxels are needed. Corresponding to the positive and negative directions of the three axes there are six types of border voxels. A voxel V in S is an *m-border* voxel, where m is a member of the set of directions $d = \{X, -X, Y, -Y, Z, -Z\}$ if V does not have a 1-adjacent neighbor within S in the m-direction. V is said to be a *k-layer* voxel, where k is a member of $\{X, Y, Z\}$, if V is both a k-border voxel and $-k$-border voxel. V is said to be an (m_1, m_2)-*border* voxel if it is both an m_1-border voxel and an m_2-border voxel (such that $|m_1| \neq |m_2|$); in other words, V does not have 1-adjacent neighbors in the m_1 and m_2 directions. Note that there are 12 distinct (m_1, m_2) pairs that specify diagonal directions. A voxel V in S is a *simple* voxel if its deletion does not affect the n-connectivity of object voxels in its $3 \times 3 \times 3$ neighborhood. Finally, a voxel V in S is an *end* voxel if there is only one object voxel n-adjacent to V.

Based on the above concepts, the follow-



(a) (b) (c)

Figure 14. The structure of 3D objects can be represented by their skeletons (a) object S represents a bin of L-shaped objects; (b) the skeletal stratum S' of S, which consists of one-voxel-thick layers; (c) the skeleton S'' which consists of space curves. The array of 16 × 16 × 22 voxels was skeletonized in 35 seconds on a Data General Eclipse computer. [From SRIH79.© IEEE 1979.]

ing two-step approach is suggested [SRIH79]. Algorithm S1 converts a set of object voxels S to a *skeletal stratum* S' which may contain both one-voxel-thick layers and branching space curves. Algorithm S2 erodes a skeletal stratum S' into a *skeletal* S'' which consists of branching space curves only.

Algorithm S1 (*Converts object S to skeletal layer S'*);

```
begin
  d := [x, −x, y, −y, z, −z]; (*direction vector*)
  repeat
    D := {φ};                          (*the null set*)
    for i := 1 to 6 do begin
      m := d(i);     (*the ith component of d*)
      k := |m|;      (*the absolute value of m*)
      in parallel for each V in S do begin
        if (V is an m-border voxel and
        V is not a k-layer voxel and
        V is simple) then
        include V in D end;
      S := S − D
    end
  until (D = {φ});
  S' := S
end;
```

Algorithm S2 (*Converts skeletal layer S' to skeleton S''*);

```
begin
  d' := [(x, y), (−x, −y), (x, −y), (−x, y), etc.];
                                   (*direction vector*)
  repeat
    D := {φ};
    for i := 1 to 12 do begin
      (m1, m2) := d'(i);
      in parallel for each V in S' do begin
        if (V is an (m1, m2)-border voxel
          and V is a simple voxel and
          V is not an end voxel) then
          include V in D end;
      S' := S' − D
    end
  until (D = {φ});
  S'' = S'
end;
```

An example skeletal representation S' and S'' of a simulated 3D object S, which was obtained using the above algorithms, is shown as a series of 3D displays in Figure 14. Although the skeleton may be considered intuitively acceptable in this example, additional criteria for deletability may be useful. One such criterion based on "checking planes" is suggested by Tsao [TSAO81] and has been shown to yield acceptable skeletal representations for connected L-shaped objects and a block letter A.

7. CONCLUSIONS

There exists a wide spectrum of methods for the computer representation of 3D digital images. Our survey has emphasized methods applicable to the analysis of images represented in 3D cellular (or voxel-based) space. These have included data structures for 3D arrays and methods that capture topological, geometrical, and structural properties of objects embedded in 3D cellular space. We have also discussed some algorithms associated with the derivation of such representations from serial-section images.

A number of hierarchical data structures for exploiting uniformities present in 3D digital images have been examined. The choice of the data structure should be based on careful consideration of the trade-off between storage space and computational complexity of typical operations on the data structure. Such operations may be a combination of graphical operations (e.g., shaded surface 2D display with hidden surfaces eliminated) and image-processing operations (e.g., 3D segmentation, translation, or rotation). In fact, the data structure may only be an intermediate step in deriving high-level 3D representations.

Although some of the methods for deriving topological and geometrical representations can be generalized from experience with digital pictures, computational issues become much more significant due to the exponential increase in the number of image elements with dimensionality. Structural representations are critical for the computer recognition and understanding of the spatial organization of 3D environments. The proposed frameworks for structural representation developed in the context of computer vision seem to be adequate, but those methods that derive such representations from serial-section images have many aspects that need further development. The variety of open problems in every aspect of representation of 3D spatial data, and in its significant applications, points to a fertile area of research in the coming years.

ACKNOWLEDGMENTS

Many of the ideas discussed in this paper have arisen as a result of discussions with my students and colleagues at SUNY/Buffalo. In particular, I am indebted to Gabor Herman, Mann-may Yau, and Gideon Yuval, who gave me comments on an earlier version of this paper, and to Gloria Calato who prepared the manuscript.

This work was supported in part by the National Science Foundation under Grant IST-80-10830 and in part by SUNY/Buffalo BRSG funds.

REFERENCES

AGIN81 AGIN, G J "Hierarchical representation of 3D objects using verbal models," *IEEE Trans. Pattern Anal. Mach Intell* **PAMI-3** (1981), 197-204.

ALTS81 ALTSCHULER, M. D., CENSOR, Y, HERMAN, G T., LENT, A., LEWITT, R. M., SRIHARI, S. N., TUY, H., AND UDUPA, J K. "Mathematical aspects of image reconstruction from projections," in A. Rosenfeld and L. Kanal (Eds), *Progress in pattern recognition*, North Holland, The Netherlands, to appear

ARTZ81 ARTZY, E., FRIEDER, G., AND HERMAN, G. T. "The theory, design, implementation and evaluation of a 3D surface detection algorithm," *Comput. Gr. Image Process.* **15** (1981), 1-24.

BADL78 BADLER, N., AND BAJCSY, R. "3D representation for computer graphics and computer vision," *Comput Gr* **12** (1978), 153-160

BADR80 BADRELDIN, A., WONG, A. K C., PRASAD, T., AND ISMAIL, M. A "Shape descriptors for *n*-dimensional curves and trajectories," in *Proc. Int. Conf Cybernic Society*, Cambridge, Mass , 1980, pp. 713-717.

BAJC80 BAJCSY, R. "Three-dimensional scene analysis," in *Proc. 5th Int. Conf. Pattern Recognition*, Miami Beach, Fla., 1980, pp. 1064-1074.

BARR70 BARRETT, L. K., AND YUST, C S. "Some fundamental ideas in topology and their application to problems in metallography," *Metallurgy* **3** (1970), 1-33.

BLUM79 BLUM, H. "3D symmetric axis coordinates an overview and prospectus," presented at the Workshop on Representation of 3D Objects, Philadelphia, Pa , 1979.

BUNE69 BUNEMAN, O. P "A grammar for the topological analysis of plane figures," *Mach Intell.* **4** (1969), 383-393.

COOK80 COOK, P. N. "3D reconstruction from serial sections for medical applications," Ph.D. dissertation, Univ. Missouri-Columbia, 1980.

COXE69 COXETER, H. S. M. *Introduction to geometry*, 2nd ed., Wiley, New York, 1969.

DEHO75 DEHOFF, R. T , AND GEHL, S M. "Quantitative microscopy of lineal features in three dimensions," in *Proc. 4th Int. Congress on Stereology*, 1975, pp 29-40.

DUDA73 DUDA, R. O., AND HART, P. E. *Pattern classification and scene analysis*, Wiley-Interscience, New York, 1973.

DYER80 DYER, C. R., ROSENFELD, A., AND SAMET, H. "Region representation boundary codes for quadtrees," *Comm ACM* **23** (March 1980), 171-179.

EAST70 EASTMAN, C. M. "Representations for space planning," *Comm. ACM*, **13** (April, 1970), 242-250.

FREE74 FREEMAN, H. "Computer processing of line-drawing images," *Comput. Surv.* **6** (March 1974), 57-93.

FUCH77 FUCHS, H., KEDEM, Z M., AND USELTON, S. P. "Optimal surface reconstruction from planar contours," *Comm. ACM* **20** (Oct 1977), 693-702.

GENN77 GENNERY, D. B. "A stereo vision system for an autonomous vehicle," in *Proc Int J. Conf Artificial Intelligence*, vol. 2, 1977, pp. 567–582.

GORD74 GORDON, R., AND HERMAN, G. T. "3D reconstruction from projections: a review of algorithms," *Int. Rev. Cytol* **38** (1974), 111–151.

GRAY70 GRAY, S. B. "Local properties of binary images in two- and three-dimensions," *IEEE Trans Comput.* **C-20** (1971), 551–561.

HERM79 HERMAN, G. T., AND LIU, H. K. "3D display of human organs from computed tomograms," *Comput Gr. Image Process* **9** (1979), 1–21.

HILB52 HILBERT, D., AND COHN-VOSSEN, S. *Geometry and the imagination*, Chelsea, New York, 1952

HORN75 HORN, B. "Obtaining shape from shading information," in P. H. Winston (Ed.), *The psychology of computer vision*, McGraw-Hill, New York, 1975

JACK80 JACKINS, C. L., AND TANIMOTO, S. L. "Oct-trees and their use in representing 3D objects," *Comput Gr. Image Process.* **14** (1980), 249–270.

KIM80 KIM, C. E., AND ROSENFELD, A. "Convex digital solids," Tech. Rep. TR-929, Computer Science Center, Univ. Maryland, College Park, 1980.

KRUG78 KRUGER, R P, AND CANNON, T M. "The application of computed tomography, boundary detection and shaded graphics reconstruction to industrial inspection," *Mater. Eval* **36** (1978).

LESK79 LESK, A. M. "Detection of three-dimensional patterns of atoms in chemical structures," *Comm. ACM* **22,** 4 (April 1979), 219–224.

LIU77 LIU, H. K. "Two and 3D boundary detection," *Comput. Gr. Image Process.* **6** (1977), 123–134.

LOBR80 LOBREGT, S., VERBEEK, P. W., AND GROEN, F. C. A. "3D skeletonization: principle and algorithm," *IEEE Trans. Pattern Anal Mach Intell* **PAMI-2** (1980), 75–77.

MARR78 MARR, D., AND NISHIHARA, H K "Representation and recognition of the spatial organization of 3D shapes," *Proc Royal Society of London* **B200** (1978), 269–274.

MEAG80 MEAGHER, D. "Octree encoding: a new technique for the representation, manipulation and display of arbitrary 3D objects by computer," Tech. Rep. TR-IPL-111, Dep. Electrical Systems Engineering, Rensselaer Polytechnic Inst, Troy, N. Y., 1980.

MINS72 MINSKY, M., AND PAPERT, S. *Perceptrons, an introduction to computational geometry*, MIT Press, Cambridge, Mass., 1972.

MORG80a MORGENTHALER, D. G., AND ROSENFELD, A. "Multidimensional edge detection by hypersurface fitting," Tech. Rep. TR-877, Computer Science Center, Univ. Maryland, College Park, 1980.

MORG80b MORGENTHALER, D. G, AND ROSENFELD, A. "Surfaces in 3D digital images," Tech. Rep. TR-940, Computer Science Center, Univ Maryland, College Park, 1980.

MYLO71 MYLOPOULOS, J. P., AND PAVLIDIS, T. "On the topological properties of quantized spaces," *J. ACM* **18,** 2 (April 1971), 239–254.

NACK80 NACKMAN, L. R. "Curvature relations in 3D symmetric axes," Tech. Rep. TR-80-011, Dep. Computer Sciences, Univ. North Carolina, Chapel Hill, 1980

NEVA77 NEVATIA, R, AND BINFORD, T. O., "Description and recognition of curved objects," *Artif. Intell.* **8** (1977), 77–98.

O'ROU79 O'ROURKE, J., AND BADLER, N. "Decomposition of 3D objects into spheres," *IEEE Trans. Pattern Anal Mach. Intell.* **PAMI-1** (1979), 295–305

O'ROU81 O'ROURKE, J. "Dynamically quantized spaces applied to motion analysis," Tech. Rep. TR-EE 81-1, Dep. Electrical Engineering, Johns Hopkins Univ., Baltimore, Md., 1981.

PARK71 PARK, C M., AND ROSENFELD, A. "Connectivity and genus in three dimensions," Tech Rep TR-156, Computer Science Center, Univ. of Maryland, College Park, 1971

POTE79 POTEMSIL, M "Generation of 3D surface descriptions from images of pattern illuminated objects," in *Proc IEEE Conf. Pattern Recognition and Image Processing*, 1979, pp 553–560.

PREP77 PREPARATA, F. P., AND HONG, S. J. "Convex hulls of finite sets of points in two and three dimensions," *Comm. ACM* **20,** 2 (Feb. 1977), 87–93.

REDD78 REDDY, D. R., AND RUBIN, S. "Representation of 3D objects," Tech. Rep TR-CMU-CS-78-113, Dep. Computer Science, Carnegie-Mellon Univ., Pittsburgh, Pa., 1978.

REQU80 REQUICHA, A. A. G. "Representation for rigid solids· theory, methods, and systems," *Comput. Surv.* **12,** 4 (Dec. 1980), 437–464.

ROBE65 ROBERTS, L. G. "Machine perception of 3D solids," in J. T. Trippett et al (Eds.), *Optical and electro-optical information processing*, MIT Press, Cambridge, Mass., 1965, pp. 159–197

ROSE76 ROSENFELD, A., AND KAK, A. C *Digital picture processing*, Academic Press, New York, 1976.

ROSE80 ROSENFELD, A. "Three-dimensional digital topology," Tech. Rep TR-936, Computer Science Center, Univ. of Maryland, College Park., 1980.

SADJ79 SADJADI, F. A., AND HALL, E L "Object recognition by 3D moment in-

variants," *Proc. IEEE Conf. Pattern Recognition and Image Processing*, 1979, pp. 327-336.

SCHU79 SCHUDY, R. B., AND BALLARD, D. H. "A computer model for extracting moving heart surfaces from four-dimensional cardiac ultrasound data," in *Proc. 6th Conf. Computer Applications in Radiology and Computer Aided Analysis of Radiological Images*, 1979, pp. 366-376.

SHAN80 SHANI, U. "A 3D model driven system for the recognition of abdominal anatomy from CT scans," in *Proc. 5th Int. Conf. Pattern Recognition*, Miami Beach, Fla., 1980, pp. 585-591.

SHAP80 SHAPIRO, L G., MULGAONKAR, P G, MORIARITY, J. D., AND HARALICK, R. M. "A generalized blob model for 3D object representation," in *Proc. IEEE Workshop Pictorial Data Description and Management*, 1980, pp. 109-116.

SHIR78 SHIRAI, Y. "Recent advances in 3D scene analysis, in *Proc. 4th Int. Conf. Pattern Recognition*, 1978, pp. 86-94.

SORO79 SOROKA, B. I., "Understanding objects from slices," Ph.D. dissertation, Dep. Computer Science, Univ. Pennsylvania, Philadelphia, 1979.

SRIH79 SRIHARI, S. N., UDUPA, J. K., AND YAU, M. "Understanding the bin of parts," in *Proc. IEEE Int. Conf. Cybernetics and Society*, Denver, Colo., 1979, pp. 44-49.

SRIH80 SRIHARI, S. N. "Hierarchical representations for serial section images," in *Proc. 5th Int. Conf. Pattern Recognition*, 1980, pp. 1075-1080.

TANI80 TANIMOTO, S. "Image data structures," in S. Tanimoto and A. Klinger (Eds.), *Structured computer vision*, Academic Press, New York, 1980.

TSAO81 TSAO, Y. F., AND FU, K. S "A parallel thinning algorithm for 3D pictures," *Comput. Gr. Image Processing*, to appear.

UDUP79 UDUPA, J. K., SRIHARI, S. N., AND HERMAN, G. T. "Boundary detection in multidimensions," *IEEE Trans. Pattern Anal. Mach. Intell.*, to appear.

WALT75 WALTZ, D. "Understanding line drawings of scenes with shadows," in P. H. Winston (Ed.), *The psychology of computer vision*, McGraw-Hill, New York, 1975.

WINS76 WINSTON, P. H. *Artificial intelligence*, Addison-Wesley, Reading, Mass., 1976.

YAU81a YAU, M., AND SRIHARI, S. N. "Recursive generation of hierarchical data structures for multidimensional digital images," in *Proc. IEEE Conf. Pattern Recognition and Image Processing*, Dallas, Tex., 1981, pp. 42-44.

YAU81b YAU, M., AND SRIHARI, S. N. "Digital convex hulls from hierarchical data structures," in *Proc. Canadian Man-Computer Communications Society Conf.*, Waterloo, Ont., Canada, 1981, pp. 163-171.

YONE80 YONEKURA, T., TORIWAKI, J., FUKUMURA, T., AND YOKOI, S. "On connectivity and Euler number of 3D digitized binary pictures," *Trans. IECE Japan* **E-63** (1980), 815-816.

YORK80 YORK, B. W., HANSON, A R., AND RISEMAN, E. M. "A surface representation for computer vision," in *Proc. IEEE Workshop Picture Data Description and Management*, Pacific Grove, Calif., 1980, pp. 124-129.

ZUCK81 ZUCKER, S. W., AND HUMMELL, R. A. "A 3D edge operator," *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-3** (1981), 324-331.

Received July 1980; final revision accepted September 1981