**SEMAINE INTERNATIONALE DE L'IMAGE ELECTRONIQUE /*INTERNATIONAL ELECTRONIC IMAGE WEEK***

**Deuxième colloque image/*Second Image Symposium***

traitement, synthèse, technologie et applications / *image processing computer generated images, technology and applications*

**Nice, Avril 1986**

# STRUCTURES HIERARCHIQUES DE DONNEES
# HIERARCHICAL DATA STRUCTURES*

Hanan Samet

Computer Science Department, University of Maryland,
College Park, Maryland 20742 USA

Robert E. Webber

Computer Science Department, Rutgers University,
New Brunswick, New Jersey 08903 USA

## RESUME

Une revue de l'utilisation de structures de données hiérarchiques, telles que "quadtrees" ou "octrees", est présentée, pour les applications graphiques. Ces structures de données hiérarchiques se divisent en deux classes - celles qui groupent les objets d'une façon hiérarchique et celles qui décomposent hiérarchiquement un "espace-image". Bien que les hiérarchies d'objets soient utiles pour construire les interfaces d'un logiciel ou d'un système graphiques, il est préférable de définir les problèmes algorithmiques en termes de hiérarchie "espace-image". De telles techniques se sont révélées utiles pour de nombreuses applications graphiques, y compris le calcul de parcours de rayons lumineux et l'intersection de surfaces courbes. Dans de nombreux cas, les hiérarchies "espace-image" sont l'équivalent géometrique d'un tri et de ce fait, il est vraisemblable qu'elles resteront centrales dans les futures applications graphiques.

## SUMMARY

An overview is presented of the use of hierarchical data structures, such as the quadtree and octree, in computer graphics applications. These hierarchical data structures are subdivided into two classes - those that group objects in a hierarchical manner and those that hierarchically decompose an image space. While object hierarchies are useful in designing interfaces to a graphics package or device, it is preferable to address algorithmic issues in terms of an image-space hierarchy. Such techniques have found use in many tasks in computer graphics including ray tracing and intersecting curved surfaces. In many cases the image-space hierarchies are the geometric equivalent of sorting and hence can be expected to remain central to graphics applications in the future.

*The support of the National Science Foundation under Grant DCR-83-02118 is gratefully acknowledged.

STRUCTURES HIERARCHIQUES DE DONNEES
HIERARCHICAL DATA STRUCTURES

Hanan Samet and Robert E. Webber

## 1. INTRODUCTION

Computer graphics applications require the manipulation of two distinct data formats: raster and vector. The raster format enables modeling the graphics image as a collection of square cells of uniform size (called pixels). A color is associated with each pixel. In contrast, instead of modeling the display screen directly, the vector format models the ideal geometric space that is to be represented on the display screen. Vector data consists of points, line segments, filled polygons, and polyhedral solids. In addition to processing these two formats of data directly, in computer graphics applications we also are concerned with the problem of conversion between these formats.

Both data formats have obvious representations. These representations are minimal in the sense of just providing sufficient structure to allow updating. For the raster format, the obvious representation is as a two-dimensional array of color values. For the vector format, the obvious representation is as a linked list of data items. While these representations are suitable for medium range applications, once the scene being modeled becomes significantly larger than the display grid, major logistic problems arise that require more complicated data structures to efficiently manipulate the scene's contents. There are two approaches to handle the logistics problems. One approach, based on object space hierarchies [9], is beyond the scope of this paper. The other approach, based on image space hierarchies, is typified by hierarchical data structures such as quadtrees and octrees.

In the remainder of this paper we review the application of hierarchical data structures such as the quadtree and octree in computer graphics. Section 2 contains a general discussion of their properties. Sections 3 and 4 describe algorithms using quadtrees and octrees respectively. Section 5 concludes with a brief discussion of some other applications of these data structures as well as hardware implementations. For more references and details on hierarchical data structures, see [48, 56].

## 2. PROPERTIES OF QUADTREES AND OCTREES

The quadtree data structure [29] is constructed in the following manner. If the entire image space has a simple description, then it does not require any further hierarchical structure. If this is not the case, then the image space is partitioned into four disjoint congruent square regions whose union covers the original image space. Each of these new image spaces is treated as if it was isolated and for each one the question is raised as to whether or not it has a simple description. This decomposition technique is referred to as a regular decomposition in order to distinguish it from decomposition approaches that vary the size of the subregions formed from the original regions.

The test for determining whether or not an image space has a simple description is called the leaf criterion. There are many plausible leaf criteria. For the purposes of this paper, we consider quadtrees constructed from two different leaf criteria (one for handling raster data and the other for handling vector data). For raster data, we use the quadtree built from the criterion that no space can contain more than one color. This works for raster data because the raster grid is built of regions that only contain one color and hence the hierarchy need never decompose to a level lower than that of these pixels. As an example, consider the region shown in Figure 1a which is represented by the $2^3 \times 2^3$ binary array in Figure 1b. The 1's correspond to pixels that are in the region (black) and the 0's correspond to pixels that are outside the region (white). The resulting squares for the array of Figure 1b are shown in Figure 1c. This process can be represented by a tree of degree 4 (i.e., each nonleaf node has four sons) such that the root node corresponds to the entire array. Each son of a node represents a quadrant (labeled in order NW, NE, SW, and SE) of the region represented by that node. The leaf nodes of the tree correspond to those blocks for which no further subdivision is necessary. Figure 1d is the tree representation for Figure 1c.

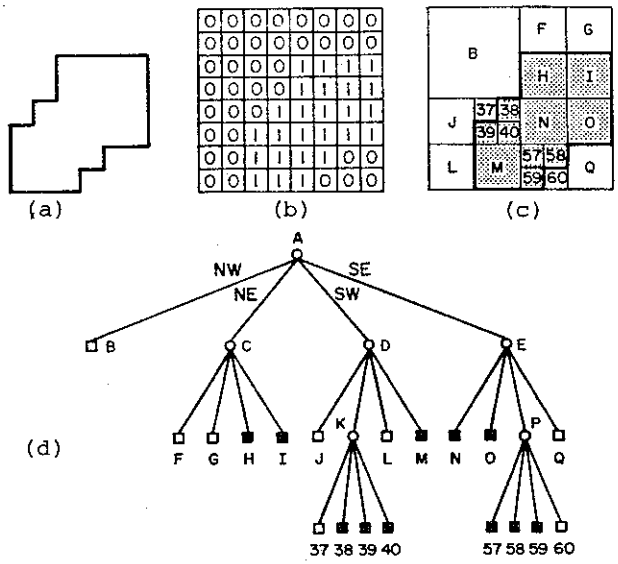


Figure 1. (a) A region, (b) its binary array, (c) its maximal blocks, and (d) the corresponding quadtree.

The quadtree can also be represented without using pointers. One such representation is in the form of a preorder tree traversal (i.e., depth-first) of the nodes of the quadtree [28, 37]. Another pointer-less representation is in the form of a collection of the leaf nodes comprising it. Each leaf node is encoded by a number termed a *locational code* corresponding to a sequence of directional codes that locate the leaf along a path from the root of the quadtree (e.g., [16]).

The octree is a three-dimensional analog of the quadtree. The principle behind octrees is that if the objects within a cubical volume are sufficiently complex, then the volume is recursively subdivided into eight congruent disjoint cubes (called octants) until the complexity is sufficiently reduced. As in the raster quadtree, the leaf criterion for raster octrees is homogeneity, i.e., a volume element is represented by a leaf if it is entirely one color [20, 23, 32]. As an example, consider the solid shown in Figure 2a whose octree representation is given in Figure 2b.

Many raster quadtree and octree algorithms are simply preorder traversals of the structure and thus their execution time is generally a linear function of the number of nodes in the structure. Hunter [20] has shown that for a simple polygon (i.e., nonself-intersecting edges) of perimeter $p$ (measured in pixel widths) on a $2^n \times 2^n$ grid, the number of nodes in the quadtree is $O(p+n)$. An interpretation of this result is that as the resolution of the image is doubled, the perimeter doubles (ignoring fractal effects), and hence the number of nodes will double. On the other hand, the number of pixels in the binary array

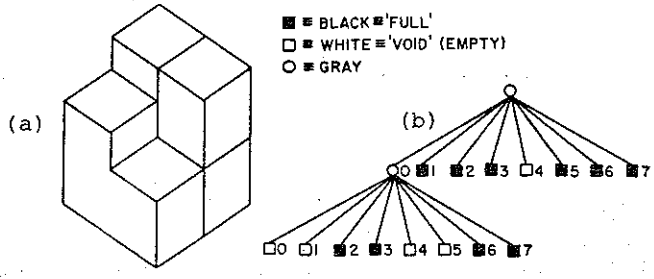


Figure 2. (a) Example object and (b) its octree.

representation quadruples. These results also apply to the octree (i.e., the number of nodes is proportional to the surface area measured in voxels) [32] and can be generalized to $k$ dimensions as follows:

> the size of the $k$-dimensional quadtree of a set of $k$-dimensional objects is proportional to the size of the $(k-1)$-dimensional interfaces between these objects.

The other type of data that we want to represent is vector data. There are a number of useful leaf criteria [52] for representing vector data using quadtrees. These criteria differ in the degree of the complexity of the image-space description versus the size of hierarchy (i.e., the number of nodes in the quadtree). The criteria chosen depends on whether we prefer a large number of simple leaf nodes or a smaller number of more complicated leaf nodes (where it is understood that the expense of processing a leaf is proportional to the complexity of the information stored in the leaf). Below, we present a leaf criterion that results in many simple leaf nodes, but which minimizes the complexity of the description of algorithms.

(1) There can be at most one vertex in an image space.
(2) If there is a vertex in the image space, then all line segments in the image space must share that vertex.
(3) If there are no vertices in the image space, then at most one line segment passes through the image space.

For our purposes, vertices occur at the endpoints of line segments and at any place where two line segments intersect. For a related data structure, see the edge quadtree [57].

The vector quadtree leaf criteria can also be generalized to form vector octree leaf criteria to represent polyhedra. Octree data structures have been used where the octree decomposition was performed as long as the number of primitives in a leaf node exceeded a predefined bound [19, 25, 69]. The problem with such an approach is that there are some features that cannot be exactly represented, thus requiring a maximum depth truncation. One way to avoid the information loss from a maximum-depth cutoff is to permit a variable number of primitives to be associated with each octree leaf node. The vector octree analog [4, 5, 15, 35] of the vector quadtree consists of leaf nodes of type face, edge, and vertex, defined as follows. A face node is an octree leaf node that is intersected by exactly one face of the polyhedron. An edge node is an octree leaf node that is intersected by exactly one edge of the polyhedron. A vertex node is an octree leaf node that is intersected by exactly one vertex of the polyhedron.

The vector octree techniques have also been extended to handle curvilinear surfaces. Primitives including cylinders and spheres have been used in conjunction with a decomposition rule that limits the number of distinct primitives that can be associated with a leaf node [14, 69]. Another approach extends the concepts of face node, edge node, and vertex node to handle faces represented by biquadratic patches. The use of biquadratic patches enables a better fit with fewer primitives than can be obtained with polygonal faces, thus reducing the size of the octree [36]. The difficulty in organizing curved surface patches by using octrees lies in devising efficient methods of calculating the intersection between a patch and an octree node. Observe that in this approach we are organizing a collection of patches in the image space, in contrast to decomposing a single patch in the parametric space by use of quadtree techniques as discussed in Section 3.5.2.

## 3. ALGORITHMS USING QUADTREES

In this section we briefly describe how a number of basic graphics algorithms can be implemented using quadtrees. In particular, we focus on set operations and image transformations, polygon coloring, display, and quadtree construction. We also expand on the concept of neighbor finding which serves as a basis of many algorithms using quadtrees and octrees.

### 3.1. SET OPERATIONS AND TRANSFORMATIONS

The basic set-theoretic operations on quadtrees were first described by Hunter and Steiglitz [20, 21] for pointer-based quadtrees. Gargantini [18] and van Lierop [61] later investigated these operations for pointer-less quadtrees. Gargantini [18] raises the issue of performing these operations on quadtrees that are not aligned. Hunter and Steiglitz [20, 22] and Peters [38] consider the problem of performing an arbitrary linear transformation on an object represented by a quadtree.

In many applications the entire quadtree must be traversed. For example, for binary images, the intersection of two quadtrees that are aligned yields a black node only when the corresponding regions in both quadtrees are black. This operation is performed by simultaneously traversing three quadtrees. The first two trees correspond to the trees being intersected and the third tree represents the result of the operation. At each step in the traversal one of the following actions is taken:

(1) If either input quadtree node is white, then the output quadtree node is white.
(2) If both input quadtree nodes are black, then the output quadtree node is black.
(3) If one input quadtree node is black and the other input quadtree node is gray (i.e., an internal node), then the gray node's subtree is copied into the output quadtree.
(4) If both input quadtree nodes are gray, then the output quadtree node is gray, and these four actions are recursively applied to each pair of corresponding sons. Once the sons have been processed, we must check to see if they are all leaf nodes of the same color in which case a merge takes place. Note that for the intersection operation, a merge of four black leaf nodes is impossible and thus we must only check for white leaf nodes.

The worst-case execution time of this algorithm is proportional to the sum of the number of nodes in the two input quadtrees. Note that because of the first action, it is possible for the intersection algorithm to visit fewer nodes than the sum of the nodes in the two input quadtrees.

The union operation can be easily implemented by applying DeMorgan's law to the above intersection algorithm. When the set-theoretic operations are interpreted as boolean operations, union and intersection become "or" and "and" operations, respectively. Other boolean operations, like "xor" and set-theoretic operations such as set-difference are coded in an analogous manner with linear-time algorithms. Similar algorithms can also be devised for performing these operations on quadtrees that are not aligned. Shifting and rotation can be shown to be special cases of intersecting unaligned quadtrees where one of the quadtrees corresponds to a black square (possibly rotated) in the desired position. Since all of these algorithms are based on preorder traversals, they will execute efficiently regardless of the specific encoding used for the quadtree. Note that clipping is a special case of the intersection operation where one of the input quadtrees corresponds to a black region that represents the display screen's location and size, thereby making it easy to implement using quadtrees.

Besides using the quadtree for the traditional graphics operations of translation (shifting) and rotation which are mentioned above, the quadtree also can be used to scale an image. Scaling by powers of two is simple and can be used in the progressive transmission of images which facilitates browsing a database of images. One successful approach [54, 58] is to transmit the nodes of a raster quadtree in breadth-first order, so that large

leaf nodes are seen first.

### 3.2. BOTTOM-UP NEIGHBOR FINDING

Many quadtree algorithms involve more than just traversing the tree. In particular, in several applications we must perform a computation at each node that depends on the values of its adjacent neighbors. Thus we must be able to locate these neighbors. There are several techniques for achieving this result. One approach makes use of the coordinates and size of the node whose neighbor is being sought to compute the location of a point in the neighbor and then accesses it by following a path from the root of the quadtree. For a $2^n \times 2^n$ image, this can require $n$ steps. An alternative approach, and the one we describe below, uses father links and computes a direct path to the neighbor by following links in the tree. This method is termed *bottom-up neighbor finding* and has been shown to require an average of no more than four links to be followed for each neighbor that is sought [46, 55].

In this section we shall limit ourselves to neighbors in the horizontal and vertical direction that are of size equal to or greater than the node whose neighbor is being sought. For neighbors in the diagonal direction, see [46]. Finding a node's neighbor in a specified horizontal or vertical direction requires us to follow father links until a common ancestor of the two nodes is found. Once the common ancestor is located, we descend along a path that retraces the previous path with the modification that each step is a reflection of the corresponding prior step about the axis formed by the common boundary between the two nodes. For example, when attempting to locate the eastern neighbor of node 38 (i.e., node N) in Figure 1, node A is the common ancestor of nodes 38 and N, and the eastern edge of the block corresponding to node 38 is the common boundary between node 38 and its neighbor. The main idea behind bottom-up neighbor finding can be seen by examining more closely how the nearest common ancestor of a node, say $A$, and its eastern neighbor of greater than or equal size, say $B$, is located. In particular, the nearest common ancestor has $A$ as one of the easternmost nodes of one of its western subtrees, and $B$ as one of the westernmost nodes of one of its eastern subtrees.

### 3.3. CONSTRUCTING QUADTREES

Before we can operate on images represented by quadtrees, we must first build the quadtrees. This involves being able to convert between a number of different data formats and the quadtree. In this section we briefly describe the construction of raster quadtrees from vector and raster data. The construction of vector quadtrees from either type of data can be performed in an analogous manner.

The algorithm for building a raster quadtree from a two-dimensional array can be derived directly from the definition of the raster quadtree [43]. When building a quadtree from raster data in raster scan order [44] we use the bottom-up neighbor-finding algorithm described in Section 3.2 to move through the quadtree in the order in which the data is encountered.

Building a raster quadtree from vector data is more complicated than from raster data since a list of line segments has no inherent spatial ordering. A top-down algorithm for producing a raster quadtree from vector data takes as input a list of line segments. This list is recursively clipped against the region, say $R$, represented by the root of the current subtree of the quadtree. If no line segments fall within $R$, then a white leaf node is created. If $R$ is of pixel size and contains at least one line segment, then a black leaf node is created. Otherwise, a gray node corresponding to $R$ is created, and the algorithm is recursively applied to each of its four children using the list that has been clipped.

Alternatively, we could use a bottom-up approach to building the raster quadtree from vector data. First, we must convert the line segments into a list of pixel-to-pixel steps (also known as chain codes) using a traditional line drawing algorithm [41]. Next, we follow the path formed by the chain codes of the line segments creating black pixel-sized leaf nodes [42]. This is done by using the bottom-up neighbor finding algorithm of Section 3.2. Average-case analysis for the execution time of the chain code to raster quadtree algorithm has shown it to be linear in the length of the chain code [42]. Moreover, by preprocessing the chain code, it has been shown that the worst-case analysis of this algorithm is also linear in the length of the chain code [64]. Similarly, the use of neighbor finding to construct chain codes from quadtrees is possible [13].

### 3.4. POLYGON COLORING

Another raster operation that can be efficiently implemented in quadtrees using neighbor finding is the seed filling approach to polygon coloring. The classic seed-filling algorithm [41] has as its input a starting pixel location and a new color. The algorithm propagates the new color throughout the polygon containing the starting pixel location. When using arrays, this algorithm is coded by a recursive routine that checks if the color of the current pixel is equal to that of the original color of the start pixel. If yes, then its color is set to the new color and the algorithm is applied to each of the current pixel's four neighboring pixels (for a 4-connected region). By using bottom-up neighbor finding the array implementation of this algorithm can be adapted to quadtrees. Another approach to coloring a region is to color the border of the region and then move inward from smaller to larger quadtree nodes [20, 21]. This algorithm could also be implemented using bottom-up neighbor finding.

A more general version of polygon coloring is connected-component analysis. Here, the task is to take a binary image and recolor each of the distinct black regions so that each region has a unique color. The general approach is to traverse the quadtree in preorder and attempt to propagate different colors across the different regions. We discuss three techniques for propagating the colors. The first technique is to perform the quadtree-based seed-filling polygon-coloring algorithm described above whenever a new region is encountered during the traversal. The second technique consists of a three stage algorithm [45, 51]. The first stage propagates the color of a node to its southern and eastern neighbors. This may result in the coloring of a single connected component by more than one color in which case the equivalence of the two colors is noted. These equivalences are merged in the second stage. The third stage updates the colors of all nodes of the quadtree to reflect the result of the second stage. The third technique [64] is a modification of the second technique and avoids the second stage of merging equivalences. Each time the border of a new region is encountered, the preorder traversal is interrupted and the border of the region is traced and colored using bottom-up neighbor finding. At the end of the trace, the preorder traversal is resumed.

Both the second and third techniques described above use a special kind of neighbor finding, i.e., they perform a preorder traversal of a quadtree and are interested in some of the neighbors of each node in the traversal. For this approach top-down neighbor finding can be used to produce improved worst-case results [24, 47, 50, 64]. Top-down neighbor finding is based on the observation that the neighbor of a node is either 1) a sibling of the node, or 2) a child of a neighbor of the node's father. Thus, the neighbors of a node can be maintained as parameters to the function that is performing the preorder traversal of the quadtree. The same idea can be used for efficiently calculating the perimeter of a region represented by a quadtree [24].

STRUCTURES HIERARCHIQUES DE DONNEES
HIERARCHICAL DATA STRUCTURES
Hanan Samet and Robert E. Webber

### 3.5. QUADTREE HIDDEN-SURFACE ALGORITHMS

Probably one of the most basic graphics operations is the conversion of an internal model of a three-dimensional scene into a two-dimensional scene that lies on the viewplane for the purpose of display on a two-dimensional screen. This is known as the hidden surface operation. While there are many mappings that are abstractly possible between a three-dimensional space and a two-dimensional space, we are interested in a mapping that closely models classical optics. Each pixel of the viewplane determines a pyramid that is formed by the set of all rays originating at the viewpoint and intersecting the viewplane within the boundary of the pixel. A color is assigned to each pixel that corresponds to the color of the object that is closest to the viewpoint while also lying within the pixel's pyramid. Thus the hidden surface task [59] is basically a sorting task.

There are three approaches to this task that are relevant to this discussion. First, quadtrees can be used to model the viewplane. Second, the parametric space of the surface of a three-dimensional object can be modeled by a quadtree. Finally, the notion of a quadtree can be directly extended to model the three-dimensional scene (which is discussed in the context of octrees in Section 4). The first two approaches assume a vector data representation of a three-dimensional scene.

#### 3.5.1. WARNOCK'S ALGORITHM

The usage of the quadtree for modeling the viewplane during the hidden surface operation was first described by Warnock [63]. The static presentation of Warnock's algorithm is as a quadtree with the following leaf criteria: 1) an empty leaf is valid; 2) a leaf representing a region the size of a pixel is valid; 3) a leaf containing a collection of polygons is valid only if one of the polygons occludes all the others over the region represented by the leaf.

Warnock's goal wasn't to construct a quadtree, but, instead, his algorithm yielded vector drawing commands for driving a display. Thus, rather than building a quadtree, the algorithm traverses the data in the same manner as a quadtree builder. However, when it comes to a a situation satisfying one of the leaf criteria described above, it takes the corresponding display action. These display actions are: 1) draw nothing since there are no lines in this region; 2) draw a point representing the border of the polygon occluding the upper lefthand corner of the pixel (if no such polygon exists, then draw nothing); 3) draw nothing since all the polygon borders in this region are occluded.

#### 3.5.2. DISPLAYING CURVED SURFACES

In this paper, vector data is usually viewed as consisting of straight line segments and polygons. However, the quadtree paradigm also has proven useful to researchers interested in the manipulation of curved features such as surfaces. Curved surfaces are often represented by a collection of bicubic surface patches. Determining the projected image of a patch is complicated by the fact that the perspective projection of a patch can have a complicated border. One early approach to displaying such surfaces was developed by Catmull [7]. The idea is to recursively decompose the patch into subpatches until the subpatches that are generated are so small that they only span the center of one pixel (or can be shown to lie outside the display region). The test for how many pixel centers are spanned by the patch (or whether or not the patch lies outside the display area) is based on the approximation of the patch by a polygon connecting the patch's corners. As was observed by Catmull, this can be generalized to other patch representations. Patch representations based on characteristic polyhedrons (e.g., Bezier and B-spline patches), allow these test decisions to be based on convex hulls that are guaranteed to enclose the patch.

As with Warnock's algorithm, Catmull's algorithm is oriented toward generating display commands and hence does not explicitly generate the quadtree structure although its processing follows the quadtree decomposition paradigm in the parametric space. Since more than one patch can span the same pixel center, the Catmull algorithm makes use of a z-buffer to keep track of the intensity/color of the patch that has most recently been found to be closest to the viewpoint.

Catmull's display algorithm has been adapted to handle a constructive solid geometry [40] representation of objects (i.e., objects composed as boolean combinations of primitive objects) for the case where the initial primitives are solids bordered by bicubic surfaces [6]. Instead of subdividing down to the pixel level everywhere, the subdivision is performed only until it has generated subpatches that are mutually disjoint. Two subpatches can be viewed as disjoint when the interiors of the convex hulls of their respective control points are disjoint. While this approach helps determine the actual intersection between two subpatches, it does not address the problem of determining which initial patches to compare. An octree approach to this problem [36] is described in Section 4.

### 4. ALGORITHMS USING OCTREES

The algorithms for performing basic computer graphics operations such as translation, rotation, scaling, and clipping on both raster and vector octrees are direct extensions of the algorithms discussed earlier for quadtrees. The techniques which were used in performing some of these operations (e.g., preorder traversal, rectilinear unaligned traversal, general unaligned traversal, bottom-up neighbor finding and top-down neighbor passing) can all be extended to deal with octrees once some additional bookkeeping information is maintained.

Building an octree is not an easy process from the point of view of the sheer amount of data that must be examined. A number of different approaches have been proposed. Clearly, the amount of work to construct a raster octree from an array representation of a three-dimensional image is quite costly due to the large number of primitive elements that must be inspected. This situation is alleviated, in part, by initially representing the data by using one of the more compact three-dimensional representations such as the boundary method or the CSG tree [40]. Tamminen and Samet [60] describe a method for building a raster octree from a boundary representation by use of connectivity labeling. In [53] they show how to build a bintree representation of a raster octree from a CSG tree (see also [68]).

An even more fundamental problem than building the octree is the acquisition of the initial boundary data to form the boundary representation. One approach is to use a three-dimensional pointing device to create a collection of samples from the surface of the object. Having collected the point data, it is then necessary to interpolate a reasonable surface containing the point data. Octrees have been used to determine this surface as a triangulation of the initial point data derived from sampling the surface of a three-dimensional object [39].

Alternative approaches to building an octree consist of taking a number of different views of an image [8, 62] or even range data [10]. This task can be viewed as the intersection of a collection of sweeps of two-dimensional silhouettes [8]. When using such methods we must be careful that the views are sufficient to describe the object in sufficient detail.

Once an octree has been constructed, it is natural to want to display it. For raster octrees, the most common display technique is the parallel projection [12]. Generalizations of the parallel projection to planes of arbitrary position and orientation are

described by Meagher [32] and Yau [70]. A more complicated display technique is to calculate the perspective projection. The perspective view of each opaque octree node can be calculated and overlaid onto the display quadtree in place of the parallel view of these opaque nodes. Binary tree variants of the raster octree also have been used as an intermediate step in the perspective projection of CSG trees [30].

One problem with displaying raster octrees is that there is little potential for using lighting models for shading since all of the faces meet at 90-degree angles. One approach to reducing this problem is described by Doctor and Torborg [12]. They suggest that the degree of shading of a node can be calculated as a function of the number of its transparent neighbors. Thus a node on the corner of an object surrounded by empty space will be brighter than another node on the surface of an object that has fewer transparent neighbors. This yields an interesting highlighting effect.

While the above display techniques are suitable for computer-aided design, realistic modeling of lighting effects generally requires using some variant of raytracing [41]. Raytracing is a direct simulation of how light is propagated through the scene landing on the image plane. The quality of the displayed image is a function of the appropriateness of the equations for modeling light and the precision with which the scene was represented. Nevertheless, the amount of time required to display a scene is heavily influenced by the cost of tracing the path of the rays of light as they move backward from the viewer's eye, through the pixels of the image plane, and out through the scene. For example, Whitted [66] reports that as much as 95% of the total picture-generation time may be required to calculate points of intersection between rays of light and objects in a complex scene. Thus the motivation for using the octree in raytracing is to enable the calculation of more rays with a greater amount of accuracy. Since light modeling equations rely on the availability of accurate information about the location of the normal to the surface at the point of its intersection with the ray, vector octrees are generally more appropriate than raster octrees. This is especially true for vector octrees that are can represent curved, rather than planar, surfaces using either curved patches [36] or curved primitives [69].

Octrees have been used to speed up intersection calculations for raytracing [14, 19, 27, 25, 69]. Glassner [19] describes a method to do this using a linear octree (i.e., as a list of the locational codes of the leaf nodes) where the octree nodes are stored in a hash table rather than a list [17]. Thus, instead of using standard neighbor-finding techniques (either top-down or bottom-up) to move between the nodes that lie sequentially along a given ray, a neighboring node is located by calculating a point that would lie in the neighbor and then searching the octree for that point. This approach also has been applied to the pointer-based representation of octrees [14, 69]. For an analogous approach using the binary tree representation of octrees (known as the *bintree* [51]) see [27].

Although searching for the node containing a particular point can be done very efficiently, standard neighbor-finding techniques should be faster for more complicated scenes. The use of both top-down and bottom-up neighbor-finding for raytracing on an octree is discussed by Jansen [25]. However, more empirical results are required to evaluate the merits of the various neighbor-finding techniques for raytracing typical scenes. Nevertheless, the octree approach to raytracing seems promising. For example, Glassner [19] reports that tracing 597,245 rays in a particular scene of 1,536 objects required 42 hours and 12 minutes using non-octree raytracing techniques, while only 2 hours and 57 minutes were required when using octrees.

## 5. CONCLUDING REMARKS

As has been seen in the previous sections, quadtrees and octrees can be adapted to many tasks in computer graphics. They also are used in other applications some of which are briefly reviewed below. A quadtree heuristic is employed to calculate the nearest neighbor (see [1] for an exact computation) to reduce the amount of wasted pen motions in a plotting program [3]. Variants of quadtrees are used to represent points, lines, and areas in a geographic information system [49]. They also have been applied in finite element mesh generation [71].

An important advantage of quadtrees and octrees is that it is easy to update them to reflect changes in the scene that they are representing. Thus it is natural that they would prove useful in the representation of scenes that change over time due to the motion of objects within the scene. Ahuja and Nash [2] represent motion by updating an octree structure as the object is moved. Alternatively, Samet and Tamminen [53] view a changing three-dimensional scene as a four-dimensional object and use a four-dimensional bintree to represent the space-time object. Besides using octrees to represent motion, they also can be used to plan motion. Kambhampati and Davis [26] have developed a multiresolution path-planning heuristic for two-dimensional motion using quadtrees that could easily be extended to three-dimensional motion using octrees.

It is also anticipated that in the future quadtree and octree techniques will be incorporated in hardware designs. Many graphics displays accept filled rectangles as a display primitive (e.g., [65]) and thus the speed of displaying an image represented by a quadtree becomes proportional to the number of nodes in the displayed region (whereas pure raster displays would require the user to decompose the rectangle into pixels). Of course, when designing graphics display primitives, one issue is minimizing the number of bits that need to be transferred to represent a given primitive. Thus, while a general fill-rectangle primitive requires location, height, and width information in addition to color information, for a special purpose quadtree processor that expects a series of quadtree leaf nodes, only the width and color of the leaf nodes (deriving location from the position of the leaf in the list) need to be specified. Such an approach has been taken with at least one MC68000-based graphics display [67]. A more aggressive approach to quadtree hardware is to design a parallel computer where individual processors are connected like the nodes in a quadtree [11, 31, 63]. One such device that has proven useful in image processing is the pyramid machine [34]. An octree machine is described in [33].

## REFERENCES

1. D.J. Abel and J.L. Smith, A simple approach to the nearest-neighbor problem, *The Australian Computer Journal 16*, 4(November 1984), 140-146.

2. N. Ahuja and C. Nash, Octree representations of moving objects, *Computer Vision, Graphics, and Image Processing 26*, 2(May 1984), 207-216.

3. D.P. Anderson, Techniques for reducing pen plotting time, *ACM Transactions on Graphics 2*, 3(July 1983), 197-212.

4. D. Ayala, P. Brunet, R. Juan, and I. Navazo, Object representation by means of nonminimal division quadtrees and octrees, *ACM Transactions on Graphics 4*, 1(January 1985), 41-59.

5. I. Carlbom, I. Chakravarty, and D. Vanderschel, A hierarchical data structure for representing the spatial decomposition of 3-D objects, *IEEE Computer Graphics and Applications 5*, 4(April 1985), 24-31.

6. W.E. Carlson, An algorithm and data structure for 3D object

synthesis using surface patch intersections, *Proceedings of the SIGGRAPH'82 Conference*, Boston, July 1982, 255-264.

7. E. Catmull, Computer display of curved surfaces, *Proceedings of the Conference on Computer Graphics, Pattern Recognition, and Data Structure*, Los Angeles, May 1975, 11-17.

8. C.H. Chien and J.K. Aggarwal, Reconstruction and matching of 3-d objects using quadtrees/octrees, *Proceedings of the Third IEEE Workshop on Computer Vision: Representation and Control*, Bellaire, MI, October 1985, 49-54.

9. J. H. Clark, Hierarchical geometric models for visible surface algorithms, *Communications of the ACM 19*, 10(October 1976), 547-554.

10. C.I. Connolly, Cumulative generation of octree models from range data, *Proceedings of the International Conference on Robotics*, Atlanta, March 1984, 25-32.

11. M. Dippe and J. Swensen, An adaptive subdivision algorithm and parallel architecture for realistic image synthesis, *Proceedings of the SIGGRAPH'84 Conference*, Minneapolis,, July 1984, 149-158.

12. L.J. Doctor and J.G. Torborg, Display techniques for octree-encoded objects, *IEEE Computer Graphics and Applications 1*, 1(July 1981), 39-46.

13. C.R. Dyer, A. Rosenfeld, and H. Samet, Region representation: boundary codes from quadtrees, *Communications of the ACM 23*, 3(March 1980), 171-179.

14. K. Fujimoto and K. Iwata, Accelerated ray tracing, *Proceedings of Computer Graphics'85*, Tokyo, 1985, T1-2, 1-26.

15. K. Fujimura and T.L. Kunii, A hierarchical space indexing method, *Proceedings of Computer Graphics'85*, Tokyo, 1985, T1-4, 1-14.

16. I. Gargantini, An effective way to represent quadtrees, *Communications of the ACM 25*, 12(December 1982), 905-910.

17. I. Gargantini, Linear octtrees for fast processing of three dimensional objects, *Computer Graphics and Image Processing 20*, 4(December 1982), 365-374.

18. I. Gargantini, Translation, rotation, and superposition of linear quadtrees, *International Journal of Man-Machine Studies 18*, 3(March 1983), 253-263.

19. A.S. Glassner, Space subdivision for fast ray tracing, *IEEE Computer Graphics and Applications 4*, 10(October 1984), 15-22.

20. G.M. Hunter, Efficient computation and data structures for graphics, Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ, 1978.

21. G.M. Hunter and K. Steiglitz, Operations on images using quad trees, *IEEE Transactions on Pattern Analysis and Machine Intelligence 1*, 2(April 1979), 145-153.

22. G.M. Hunter and K. Steiglitz, Linear transformation of pictures represented by quad trees, *Computer Graphics and Image Processing 10*, 3(July 1979), 289-296.

23. C.L. Jackins and S.L. Tanimoto, Oct-trees and their use in representing three-dimensional objects, *Computer Graphics and Image Processing 14*, 3(November 1980), 249-270.

24. C.L. Jackins and S.L. Tanimoto, Quad-trees, oct-trees, and k-trees - a generalized approach to recursive decomposition of Euclidean space, *IEEE Transactions on Pattern Analysis and Machine Intelligence 5*, 5(September 1983), 533-539.

25. F.W. Jansen, Data structures for ray tracing, in *Data Structures for Raster Graphics*, F.J. Peters, L.R.A. Kessener, and M.L.P. van Lierop, Eds., Springer Verlag, Berlin, 1986.

26. S. Kambhampati and L.S. Davis, Multiresolution path planning for mobile robots, Computer Science TR-1507, University of Maryland, College Park, MD, May 1985.

27. M.R. Kaplan, Space-tracing: a constant time ray-tracer, SIGGRAPH'85 Tutorial on the Uses of Spatial Coherence in Ray-Tracing, San Francisco, ACM, July 1985.

28. E. Kawaguchi, T. Endo, and J. Matsunaga, Depth-first expression viewed from digital picture processing, *IEEE Transactions on Pattern Analysis and Machine Intelligence 5*, 4(July 1983), 373-384.

29. A. Klinger, Patterns and Search Statistics, in *Optimizing Methods in Statistics*, J.S. Rustagi, Ed., Academic Press, New York, 1971, 303-337.

30. P. Koistinen, M. Tamminen, and H. Samet, Viewing solid models by bintree conversion, *Proceedings of the EUROGRAPHICS'85 Conference*, C.E. Vandoni, Ed., North-Holland, 1985, 147-157.

31. T. Kushner, A. Wu, and A. Rosenfeld, Image processing on ZMOB, *IEEE Transactions on Computers 31*, 10(October 1982), 943-951.

32. D. Meagher, Geometric modeling using octree encoding, *Computer Graphics and Image Processing 19*, 2(June 1982), 129-147.

33. D. Meagher, The Solids Engine: a processor for interactive solid modeling, *Proceedings of the NICOGRAPH '84 Conference*, Tokyo, November 1984.

34. R. Miller and Q.F. Stout, Pyramid computer algorithms for determining geometric properties of images, *Proceedings of the Symposium on Computational Geometry*, Baltimore, June 1985, 263-269.

35. I. Navazo, Contribució a les tècniques de modelat geomètric d'objectes poliedrics usant la codificació amb arbres octals, Ph.D. dissertation, Department de Metodes Informatics, Universitat Politechnica de Barcelona, Barcelona, Spain, January 1986.

36. I. Navazo, D. Ayala, and P. Brunet, A geometric modeller based on the exact octree representation of polyhedra, Department de Metodes Informatics, Universitat Politechnica de Barcelona, Barcelona, Spain, January 1986.

37. M.A. Oliver and N.E. Wiseman, Operations on quadtree-encoded images, *Computer Journal 26*, 1(February 1983), 83-91.

38. F.J. Peters, An algorithm for transformations of pictures represented by quadtrees, *Computer Vision, Graphics, and Image Processing 32*, 3(December 1985), 397-403.

39. J.L. Posdamer, Octal-tree spatial sorting and its applications, Computer Science Report WUCS-82-1, Washington University, St. Louis, MO, January 1982.

40. A.A.G. Requicha, Representations of rigid solids: theory,

methods, and systems, *ACM Computing Surveys 12*, 4(December 1980), 437-464.

41. D.R. Rogers, *Procedural Elements for Computer Graphics*, McGraw-Hill Book Company, New York, NY, 1985.

42. H. Samet, Region representation: quadtrees from boundary codes, *Communications of the ACM 23*, 3(March 80), 163-170.

43. H. Samet, Region representation: quadtrees from binary arrays, *Computer Graphics and Image Processing 13*, 1(May 1980), 88-93.

44. H. Samet, An algorithm for converting rasters to quadtrees, *IEEE Transactions on Pattern Analysis and Machine Intelligence 3*, 1(January 1981), 93-95.

45. H. Samet, Connected component labeling using quadtrees, *Journal of the ACM 28*, 3(July 81), 487-501.

46. H. Samet, Neighbor finding techniques for images represented by quadtrees, *Computer Graphics and Image Processing 18*, 1(January 1982), 37-57.

47. H. Samet and R.E. Webber, On encoding boundaries with quadtrees, Computer Science TR-1162, University of Maryland, College Park, MD, February 1982.

48. H. Samet, The quadtree and related hierarchical data structures, *ACM Computing Surveys 16*, 2(June 1984), 187-260.

49. H. Samet, A. Rosenfeld, C.A. Shaffer, and R.E. Webber, A geographic information system using quadtrees, *Pattern Recognition 17*, 6 (November/December 1984), 647-656.

50. H. Samet, A top-down quadtree traversal algorithm, *IEEE Transactions on Pattern Analysis and Machine Intelligence 7*, 1(January 1985), 94-98.

51. H. Samet and M. Tamminen, Efficient component labeling of images of arbitrary dimension, Computer Science TR-1480, University of Maryland, College Park, MD, February 1985.

52. H. Samet and R.E. Webber, Storing a collection of polygons using quadtrees, *ACM Transactions on Graphics 4*, (July 1985) (also *Proceedings of Computer Vision and Pattern Recognition 83*, Washington, DC, June 1983, 127-132).

53. H. Samet and M. Tamminen, Bintrees, CSG trees, and time, *Proceedings of the SIGGRAPH'85 Conference*, San Francisco, July 1985, 121-130.

54. H. Samet, Data structures for quadtree approximation and compression, *Communications of the ACM 28*, 9(September 1985), 973-993.

55. H. Samet and C.A. Shaffer, A model for the analysis of neighbor finding in pointer-based quadtrees, *IEEE Transactions on Pattern Analysis and Machine Intelligence 7*, 6(November 1985), 717-720.

56. H. Samet, Bibliography on quadtrees and related hierarchical data structures, in *Data Structures for Raster Graphics*, F.J. Peters, L.R.A. Kessener, and M.L.P. van Lierop, Eds., Springer Verlag, Berlin, 1986.

57. M. Shneier, Two hierarchical linear feature representations: edge pyramids and edge quadtrees, *Computer Graphics and Image Processing 17*, 3(November 1981), 211-224.

58. K.R. Sloan Jr. and S.L. Tanimoto, Progressive refinement of raster images, *IEEE Transactions on Computers 28*, 11(November 1979), 871-874.

59. I.E. Sutherland, R.F. Sproull, and R.A. Schumacker, A characterization of ten hidden-surface algorithms, *ACM Computing Surveys 6*, 1(March 1974), 1-55.

60. M. Tamminen and H. Samet, Efficient octree conversion by connectivity labeling, *Proceedings of the SIGGRAPH'84 Conference*, Minneapolis, July 1984, 43-51.

61. M.L.P. van Lierop, Transformations on pictures represented by leafcodes, Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 1984.

62. J. Veenstra and N. Ahuja, Octree generation from silhouette views of an object, *Proceedings of the International Conference on Robotics*, St. Louis, March 1985, 843-848.

63. J.E. Warnock, A hidden line algorithm for halftone picture representation, University of Utah Computer Science Tech. Report 4-5, May 1968.

64. R.E. Webber, Analysis of quadtree algorithms, Ph.D. dissertation, Computer Science Department, University of Maryland, College Park, MD, 1983 (see also Computer Science TR-1376).

65. D.S. Whelan, A rectangular array filling display system architecture, *Proceedings of the SIGGRAPH'82 Conference*, Boston, July 1982, 147-153.

66. T. Whitted, An improved illumination model for shaded display, *Communications of the ACM 23*, 6(June 1980), 343-349.

67. P. Willis and D. Milford, Browsing high definition colour pictures, *Computer Graphics Forum 4*, (1985), 203-208.

68. J.R. Woodwark and K.M. Quinlan, Reducing the effect of complexity on volume model evaluation, *Computer-aided Design 14*, 2(1982), 89-95.

69. G. Wyvill and T.L. Kunii, A functional model for constructive solid geometry, *The Visual Computer 1*, 1(July 1985), 3-14.

70. M. Yau, Generating quadtrees of cross-sections from octrees, *Computer Vision, Graphics, and Image Processing 27*, 2(August 1984), 211-238.

71. M.A. Yerry and M.S. Shepard, A modified quadtree approach to finite element mesh generation, *IEEE Computer Graphics and Applications 3*, 1(January/February 1983), 39-46.