JOURNAL OF VISUAL COMMUNICATION AND IMAGE REPRESENTATION
Vol. 2, No. 3, September, pp. 201–221, 1991

Binary Space Partitioning Tree Representation of Images

HAYDER RADHA,*,† RICCARDO LEONARDI,* MARTIN VETTERLI,†,‡ AND BRUCE NAYLOR*

*AT&T Bell Laboratories, 2L313 Crawfords Corner Road, Holmdel, New Jersey 07733; and †Center for Telecommunications Research and Department of Electrical Engineering, Columbia University, New York, New York 10027

Received April 26, 1991; accepted May 7, 1991

1. INTRODUCTION

Tree-based representations of two- and three-dimensional objects have been used extensively in solid modeling, computer graphics, computer vision, and image processing. (See, for example, [26, 10, 18, 39, 23].) Quadtrees and octrees, which are used to represent objects in 2-D and 3-D spaces, respectively, have been studied thoroughly for graphics and image processing applications.

Binary space partitioning (BSP) trees have been used in computer graphics applications as an efficient representation of polyhedra in D-dimensional space (a polyhedron is defined as a boundary with only planar faces [26]). For example, BSP trees provide an effective tool in determining visible surfaces for polygon rendering and ray tracing applications [29]. For 2-D applications, the BSP tree approach partitions the 2-D space, which surrounds the objects to be presented, by arbitrarily oriented lines. Recently (see [37, 38]), we proposed using the BSP tree representation for image processing and computer vision applications.

In this paper we present a Hough transform-based method for generating the BSP tree representations of images in 2-D space. As will be shown, BSP tree-based segmentation of images provides an effective tool for achieving high compression rates for image coding applications. Preliminary results show that compression ratios in the range of 40–100 are achievable. Moreover, the method of representation using arbitrarily oriented lines allows one to perform affine transformations (e.g., rotation and translation) on images very easily. Therefore, and due to the simple data structure (a binary tree) resulting from the binary partitioning, these affine transformations can be implemented using simple tree traversal algorithms.

In Section 2, we explain the BSP tree representations of polyhedra, and in Section 3 we provide a high-level description of image representation using BSP trees. The rest of the paper consists of two main sections: Section 4 describes a Hough transform-based, recursive algorithm for building a BSP tree of an arbitrary image, and Section 5 introduces a multiresolution approach for implementing the Hough transform-based method described in Section 4. In each of these two sections (4 and 5) we show some simulation results. Section 6 derives some expressions for the computational complexity of our proposed methods, and Section 7 explains the potential of the BSP tree representation approach for image coding applications. We conclude the paper with a summary in Section 8.

2. THE BSP TREE REPRESENTATION

The binary space partitioning tree (BSP tree) is an abstract data type that provides a representation of a D-dimensional space through the use of recursive subdivision. For the 2-D case, the subdivision is created using arbitrarily oriented lines. Any straight line h creates two halfplanes, which are distinguished as the negative and positive halfplanes h^- and h^+, reflecting whether the dot product of a point p with h is negative or positive. The recursive partitioning is a local operation that takes as input an unpartitioned region R, initially the entire 2-D plane, and a line h, selected according to various criteria, that intersects R, and produces as output two new regions formed by partitioning R by h into two half-regions, R^- and R^+. The two half-regions can then be similarly partitioned by some additional lines, and so on recursively until a termination criterion is met (see Fig. 1). This results in a hierarchy of regions in which the lowest level unpartitioned regions, called cells, form a partitioning of the 2-D plane.

The data structure used to represent this partitioning is a binary tree. The root of the tree represents the whole 2-D space, and it is labeled with the line used to partition it. The two resulting half-regions are represented by two child nodes, which may be partitioned as well. A left edge can be interpreted as representing the action of performing an intersection with h^- and the right edge with h^+. Thus, every region in the tree is defined by the intersection of halfplanes denoted by the edges on the path from

‡ Partially supported by the National Science Foundation (NSF) under Grant ECD-99-1111.

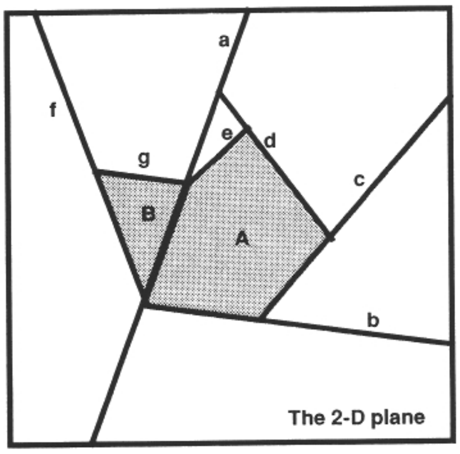


FIG. 1. A binary space partitioning of the 2-D plane.

the root to that region's node. Because a halfplane is a convex (unbounded) region, and since the intersection of convex sets is a convex set, then every region (corresponding to a leaf or nonleaf node) in the tree is convex. Each leaf node corresponds to a cell of the partitioning, while each internal node corresponds to a partitioned region that is the union of the cells at its leaf nodes.

For example, Fig. 1 shows a BSP tree-induced partitioning of the plane and Fig. 2 shows the corresponding binary tree. The root node represents the entire plane. A binary partitioning of the plane is formed by the line labeled **a** resulting in a negative halfplane and a positive halfplane. These two halfplanes are represented respectively by the left and right children of the root. A binary partitioning of each of these two halfplanes may then be performed, as in Fig. 1, and so on recursively. When, along any path of the tree, subdivision is terminated, the leaf node will correspond to an unpartitioned region, called a cell. As shown in the figure, the two cells **A** and **B** define a concave object in the 2-D space. In other words, Fig. 2 shows a BSP tree representation of this 2-D object.

For those readers familiar with k-d trees [3], BSP trees can be viewed as generalizing this data type by allowing the partitioning lines to be arbitrarily oriented, rather than restricting them to being orthogonal to a coordinate

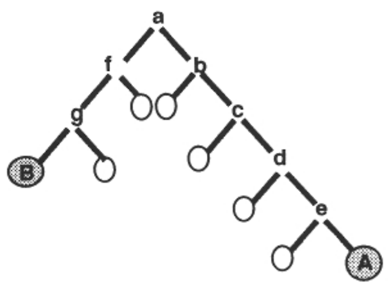


FIG. 2. The BSP tree representation of the partitioning in Fig. 1.

axis (BSP trees were, however, developed independently, beginning in 1969). They differ from quadtrees and octrees (2^D trees) both in the use of arbitrary lines and in arity of the tree: a BSP tree's data structure is a binary tree, not a 2^D-ary tree. The freedom to use any line can lead to smaller and more exact representations, at the cost of needing to perform a more expensive computation to choose the lines.

An important consequence of supporting arbitrary orientations is that any affine transformation (translation, scaling, rotation, or shearing) can be applied to a BSP tree by simply transforming the line equations (by a vector–matrix product). The orthogonality required of 2^D trees prohibits rotations by any angle that is not a multiple of 90° without reconstructing the tree. And for 2^D trees, the restriction to subdividing a region into equal parts also necessitates for all translations and most scalings that a new tree be constructed. Consider a quadtree representation of an image in which one quadrant is black and the remainder are white, which is a tree composed of five nodes. Translation by a distance of 1 pixel causes a new tree to be generated whose size is on the order of the number of pixels required to represent the boundary of the square.

These difficulties with affine transformations arise from the fact that 2^D trees are based upon representing a discrete plane (defined on a regular lattice), which was originally motivated by discrete image representation/generation. In contrast, BSP trees were developed explicitly in the context of continuous space and so the ease with which they admit affine transformations arises immediately from this fact.

The principal application of BSP trees has been for representing polytopes, the dimension-independent entity of which polyhedra and polygons are the 3-D and 2-D instances, respectively. If each cell of the BSP tree is classified as either "in" the set or "out" of the set, then the union of the in-cells determines the polytope (since all regions are formally considered to be open sets, a closure must be taken after the union).

More generally, each region of the tree may have an arbitrary set of attributes associated with it, or equivalently, a single aggregate attribute. The membership attribute required for polytopes is the simplest example of this: a constant boolean-valued function indicating in or out. However, for representing images, an intensity or color attribute is needed and possibly a transparency/opacity attribute to be used for compositing.

Therefore, the BSP tree partitioning of the 2-D space can be used to represent a piecewise continuous function whose domain is the entire plane. All discontinuities in this function are restricted to lie on the partitioning lines, a condition met during tree construction. For example, in the case of a polygon, all of the edges of the polygon must lie on the partitioning lines, since these edges are exactly the points of discontinuity. Thus the BSP tree structure provides the discontinuous part of the function, so to speak, and pieces together the various continuous functions, each restricted to certain cells of the tree. The value of the piecewise function at a single point p (located within a cell C) can be obtained by evaluating the corresponding continuous function (defined over the cell C) at p.

It is important to note that all the 2-D space ideas presented above can be easily extended to higher dimension spaces. For example, the BSP tree representation of a 3-D space can be achieved by using planes for the recursive partitioning.

Besides affine transformations, most other work with BSP trees has focused on the problem of rendering polyhedra and merging of BSP trees. In fact, BSP trees were originally developed as a solution to the visible surface problem: a view-dependent traversal of a tree generates a visibility priority ordering of the polyhedral faces [14, 29]. They also can be used to provide ray tracing of polyhedra [30], clipping to a (nonconvex) window [31], computing shadows [11], and global illumination of diffuse surfaces [15]. It is also possible to convert between a BSP tree representation and a boundary representation of polyhedra (in both directions) and to perform set operations between the two types [43]. Algorithms for merging two BSP trees (representing two objects) were also presented [32]. It was shown that BSP tree merging yields boolean set operations between the two objects.

3. BINARY PARTITIONING OF IMAGES

Image segmentation using binary space partitioning provides an efficient representation for image coding applications (see Section 7). Moreover, the *arbitrarily oriented* partitioning lines and simple data structure (a binary tree) of the BSP representation make it very useful for manipulation of images.

The BSP approach partitions the desired image, recursively, by straight lines in a hierarchical manner. First, a line is selected (based on an appropriate criterion) to partition the whole image into two subimages. Using the same criterion, two lines are selected to split the two subimages resulting from the first partitioning. This procedure is repeated until a terminating criterion is reached. The outcome of this recursive partitioning is a set of (unpartitioned) convex regions which are referred to as the *cells* of the segmented image. A good segmentation is obtained when the pixel values within each cell are *homogeneous*. A cell is considered homogeneous if its pixel intensities can be modeled using a smooth (or continuous) function. This desired feature (i.e., homogeneous cells) can serve as a *terminating* criterion.

As explained in the previous section, this recursive partitioning generates a binary tree which is referred to as the *binary space partitioning tree* representation. In this case, the nonleaf nodes of the BSP tree are associated with the partitioning lines, and the leaves represent the cells (unpartitioned regions) of the image.

It is important to note that *every* node in the tree (i.e., not only the leaves) represents a convex region of the image. However, a convex region of a nonleaf node is partitioned by the node's line into two other convex regions as explained above. In addition to a geometrical description (defined by the partitioning lines of its ancestors), a convex region of the image's BSP tree representation may have one or more attributes. An example of such attributes is a zero-order (i.e., the mean value) or higher-order polynomial model of the pixel intensities within the region.

There are two important aspects of the BSP representation approach. The first aspect (and most critical one) is the criterion used for selecting the partitioning lines. The other significant aspect is the criterion used to terminate the recursive partitioning and generate an unpartitioned region (cell). We refer to the first aspect as the *line-selection criterion,* and to the second one as the *terminating criterion*. The line-selection and terminating criteria determine the *efficiency* and *accuracy* of the BSP tree representation, respectively. These two criteria are not independent of each other since a given line-selection strategy influences the choice of the terminating criterion as explained below.

3.1. *Boundary-Based Partitioning*

In this work, we based the partitioning on the image boundary information, i.e., edges. A selected line, which partitions a given region of the image, has to pass through the *strongest* edge that lies within that region. The strength of an edge is measured, in this case, by the number of connected, collinear boundary points lying on the edge. Therefore, the first line selected to partition the whole image passes through the largest number of collinear connected edge points in the image. The same process is applied to the two subimages resulting from partitioning the image. In other words, for each subimage the selected line has to pass through the largest number of connected edge points that lie in that subimage. This boundary-based, line-selection criterion provides very good segmentation, and generates efficient representation as shown in the next section.

Our terminating criterion is based on the number of edge points in a given region. In other words, if this number is lower than some threshold, then the recursive partitioning terminates and the region under consideration becomes a cell of the BSP tree representation. The lower the threshold value, the more accurate the representation. By using this terminating criterion, the resulting cells have, in general, *homogeneous* pixels. It should be clear that the terminating criterion is influenced by the boundary-based, line-selection strategy explained above.

If the pixels within each cell are estimated using a smooth 2-D function (e.g., the average value of the pixel intensities or a low-order polynomial), then this boundary-based partitioning leads to a piecewise continuous approximation of the image function. Moreover, each smooth function, defined over a given cell, can serve as an attribute of that cell. It is important to note that the discontinuities of the piecewise continuous approximation lie on the selected partitioning lines, and these lines, as explained before, pass through the image edges.

Other line-selection and terminating criteria can be used to generate efficient BSP tree representations of images. For example, a straight line can be selected to partition a given region of the image if the line minimizes some error function defined over that region. Based on this error-minimization criterion, we are currently developing a least-square-error method for constructing BSP trees. In this paper, however, our work is based on boundary information only.

3.2. *Multiresolution-Based Partitioning*

Depending on the criteria used to build a BSP tree of an image, it is desirable that the resulting tree contains a *multiresolution description* of that image. In a multiresolution description, the nodes close to the root of the tree represent the coarse (low-resolution) information of the image, whereas the nodes close to the leaves contain the fine details (high-resolution information). We refer to a BSP tree with such a description as a *multiresolution BSP tree*. It is desirable to construct a multiresolution BSP tree representation of a given image since traversing the tree from top (the root node) to bottom (the leaves) is equivalent to zooming at that image. However, in order to guarantee that the recursive binary partitioning generates a multiresolution BSP tree, one needs to use a *multiresolution approach* when constructing the desired tree.

The multiresolution approach derives a pyramid of different resolution images from the original image [25]. This method is similar to the scale-space approach proposed in [49] as well as the Laplacian pyramid scheme [6]. The bottom of the multiresolution pyramid is the original image, and the top is the most coarse image. Therefore, by moving from the base to the top of the pyramid the image representation becomes more coarse (i.e., fine details are eliminated). Moreover, an image at a given level of the pyramid contains the information represented by all images above that level. For example, the original image contains the information represented by all other images in the multiresolution pyramid.

When constructing a multiresolution BSP tree, one has to start with the top of the pyramid. In other words, by applying the desired line-selection and terminating criteria on the lowest resolution image, a BSP tree (which represents the most coarse details of the original image) is first generated. This coarse BSP tree is then used as an initial guess to construct a finer tree from the next higher resolution image in the pyramid.

When expanding a coarse tree into a finer tree, two steps are needed. First, all nonleaf nodes of the coarse tree are mapped into the same number of nonleaf nodes in the finer tree. (This mapping requires, in general, some focusing procedure as explained in Section 5.) Second, using the higher resolution image, the coarse tree's leaves (cells) are expanded by new BSP subtrees. These subtrees represent finer details of the original image. The same procedure, i.e., expanding a coarse tree into a finer tree, is repeated until one reaches the base (original image) of the pyramid.

It is important to note that the multiresolution-based BSP tree generation method is independent of the particular approach used to construct the BSP tree. In this work, for example, we applied the multiresolution method to our boundary-based BSP tree generation scheme. In other words, we first construct a coarse tree using the boundary information of the lowest resolution image in the pyramid. This coarse tree is then used as an initial guess to construct a finer tree from a higher resolution image. In this case, the cells of the coarse tree are expanded into BSP subtrees using the boundary information of the higher resolution image.

In addition to generating a *multiresolution* BSP representation, this hierarchical approach provides a significant reduction in the computational expense. This is because the pyramidal method takes advantage of computations that take place at a coarse level of the pyramid when performing similar computations at a finer resolution level. Moreover, using the multiresolution approach gives better segmentation as shown in Section 5.

4. A HOUGH TRANSFORM-BASED METHOD FOR BSP TREE GENERATION

As explained above, our proposed BSP tree approach partitions the domain of the desired image by straight lines passing through the image boundaries. This is an easy task if the image contains a few simple objects of known shapes and sizes. On the other hand, partitioning an image that consists of several objects of unknown shapes and sizes may require the segmentation of the various objects in the scene. Without performing the difficult segmentation process, one way to solve the problem is to base the partitioning on the image contour information. Two steps will be needed for generating such a BSP tree representation (see Fig. 3): (1) extract the boundary locations of each object in the image, and (2) determine

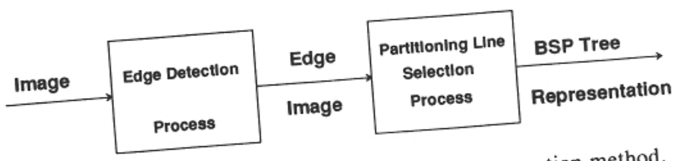


FIG. 3. Stages of a boundary-based BSP tree generation method.

the linear characteristics of these boundaries, in order to match them with a minimum set of straight lines.

Each straight line, which is represented by a nonleaf node of the BSP tree, partitions a convex region of the original image. The partitioning line of a given region has to match the largest set of collinear edge points in that region. In addition, it is desirable to include a connectivity weighting of these edge points [47]. We emphasize the notion of connectivity since the human visual system is particularly sensitive to linear segments of connected points [17]. Edge points that lie on a straight line and yet do not belong to one connected edge will be referred to as *uncorrelated* collinear edges.

As a consequence, the straight lines associated with high-level nodes (close to the root) in the BSP tree should (in general) fit the boundaries of large objects whereas low-level nodes (close to the leaves) represent fine details and small objects. This defines a natural hierarchical description of the image. However, in order to guarantee that the resulting tree is a *multiresolution BSP tree* (as defined in Section 3.2), one needs to use the *multiresolution approach* which is explained in detail in Section 5.

It is clear that the boundary points of the various objects in the image can be located by an edge detection process. The performance of the edge extraction can significantly influence the tree structure. After a reliable edge detection process, straight lines are built on collinear edge segments. The Hough transform is a very useful tool to extract straight lines and other geometrical features in edge images [16, 13].

Our BSP tree generation method starts by applying the Hough transform (HT) on all edge points in the image, and selects the HT cell[1] with the maximum number of votes (corresponding to the line that passes through the maximum number of edge points) to partition the image. The selected line partitions the image into two subimages. The same process is repeated on each subimage. In other words, the HT of one of the two subimages is computed, and a search for the cell with the maximum number of votes is performed. Now, the line corresponding to this cell partitions the selected subimage. The same thing is done to the other subimage. This process is repeated until a termination criterion is reached. This criterion could be one or both of the following: (1) the area of the region to be partitioned is smaller than a certain threshold, and (2) the number of boundary points within the region is lower than some other threshold. These thresholds determine the degree of accuracy of the representation. The lower the threshold values, the larger the number of nodes in the tree and the more accurate the image description.

It is important to note that a line h_μ (associated with a node μ and a convex region R_μ) classifies each point $\mathbf{x} \in R_\mu$ into one of the following three convex sets,

$$h_\mu^+ = \{\mathbf{x} \in R_\mu : \mathbf{x} \cdot \mathbf{a}_\mu > \rho_\mu\}$$
$$h_\mu^- = \{\mathbf{x} \in R_\mu : \mathbf{x} \cdot \mathbf{a}_\mu < \rho_\mu\}$$
$$sh_\mu = h_\mu \cap R_\mu = \{\mathbf{x} \in R_\mu : \mathbf{x} \cdot \mathbf{a}_\mu = \rho_\mu\},$$

where $\mathbf{a}_\mu$ is a unit vector normal to h_μ, and ρ_μ is the normal distance between h_μ and the origin. Here h_μ^+ and h_μ^- represent the positive and negative subregions of R_μ, and are represented by the right and left children of μ, respectively. sh_μ represents a subline (of h_μ) that lies in R_μ. After selecting h_μ as the partitioning line of R_μ, it is crucial to eliminate all edge points $\in sh_\mu$ (i.e., that lie on the line) and process just those points that belong to the subregions h_μ^+ and h_μ^-. Otherwise, these points (that are $\in sh_\mu$) may contribute to one or both of the regions, and subsequently an infinite recursion may occur.

To summarize, we propose a BSP tree generation method based on the following: (1) perform an edge detection process; (2) compute a series of Hough transforms, one for each nonleaf node in the tree; and (3) select each line as the peak of the corresponding Hough transform. The subsections below describe in detail each one of these steps.

4.1. *The Edge Detection Process*

As mentioned above, the input to this Hough transform-based BSP tree generation method is an edge image which can be represented as a binary function $E(x, y)$. The structure of the resulting tree and the set of straight lines selected to partition the original image are strongly dependent on $E(x, y)$. $E(x, y)$ can be obtained from the original image through a numerical differentiation process followed by a simple thresholding strategy. However, it is well known that differentiation amplifies high-frequency noise. Edge detection is a mildly ill-posed problem which can be transformed into a well-posed problem by convolving the original image (before differentiation) with a smoothing filter $f(x, \sigma)$ whose Fourier transform $F(\omega, \sigma)$ satisfies the conditions [44, 28]

$$\lim_{\sigma \to 0} F(\omega, \sigma) = 1$$

$$\omega F(\omega, \sigma) \in L_2,$$

[1] The HT cell should not be confused (and has no relation) with the BSP tree cell. The HT cell is defined in Section 4.2.

where L_2 is the set of square integrable functions. The Gaussian function satisfies the above conditions, and therefore is used as a predifferentiation filter in most edge detection methods. Gaussian filtering, however, introduces uncertainty in the actual edge location. The larger the standard deviation σ, the better the smoothing at the expense of poorer edge localization. This dilemma has been studied and analyzed in the literature (e.g., [40, 44, 4]). An edge focusing algorithm was introduced in [4] to achieve both filtering of high-frequency noise and good localization of the original edges. In this section, we limit ourselves to a Gaussian function with *small* σ as a predifferentiation filter. In Section 5, however, we use a line focusing algorithm that takes into account the shift in the edge points due to filtering. See Section 5 for more details.

After filtering the original signal, one can locate edge points in the filtered image by looking for zero crossings of the second derivative or maxima of the first derivate (gradient) [27, 8]. Although both methods are equivalent, it has been shown that the maximum-gradient approach is less sensitive to noise [28].

In this work our edge detection process consists of (1) filtering the image with a Gaussian function, (2) computing the gradient magnitude and direction of the filtered image, (3) locating the maxima of the gradient magnitudes in the direction of the gradient vector $\vec{g}$, and (4) performing adaptive thresholding. To locate the maxima of the gradient magnitudes, we have used a nonmaximum gradient suppression algorithm proposed by Canny [7]. For the adaptive thresholding step, we used a hysteresis thresholding strategy with both low and high thresholds [8]. (See [37] for more details about our implementation of the gradient edge detector.)

4.2. *The Hough Transform*

The standard Hough transform maps each edge point (x_e, y_e) into a curve in the Hough space. It acts as a voting process, where (x_e, y_e) votes for all lines passing through it. These lines are represented by two parameters in the Hough space (also known as the parameter space). Here we use the (θ, ρ) parameterization to represent straight lines in the Hough space. θ is the direction (with respect to the x axis) of the vector $\mathbf{n}$ normal to the line (θ, ρ), and ρ is the distance between the line and the origin [13]. With this parameterization, the line equation is given by

$$\rho = x\cos(\theta) + y\sin(\theta). \qquad (4.2.1)$$

The HT at a point (θ, ρ) in the Hough space can be expressed as

$$ht(\theta, \rho) = \sum_x \sum_y \delta(\rho - x\cos\theta - y\sin\theta)E(x, y), \qquad (4.2.2)$$

where $E(x, y)$ is a 2-D binary function representing the edge image, and $\delta(x) = 1$ for $x = 0$ and zero otherwise. In practice, the Hough space is divided into a finite number of cells[2] organized as an accumulation array, where each cell is associated with a discrete pair of the coordinates (θ, ρ).

The standard HT is computed as follows. First, θ is sampled at N_θ values between zero and 180°. For each sample of θ, ρ is calculated and quantized to one of N_ρ possible levels, and the corresponding cell in the HT space is incremented by one vote. Therefore, an edge point (x_e, y_e) in the image space is mapped into a sinusoidal function in the (θ, ρ) parameter space as shown in Fig. 4. The intersection of two sinusoidal functions (corresponding to two edge points) gives the (θ, ρ) parameterization of the straight line that connects the two points in the image space. In other words, the element of the accumulation array that receives v votes represents a straight line that passes through the same number (v) of edge points in the image space. For the example shown in Fig. 4, the line $(\pi/4, \sqrt{2})$ receives three votes from the three collinear edge points which lie on that line. The detection of straight lines in the edge image can be accomplished by searching for (θ, ρ) cells that possess a large number of votes (*peaks*) in the parameter space. The HT literature is plentiful. A comprehensive survey can be found in [19].

Improvements have been suggested to this simple approach as the standard HT is sensitive to background noise and texture in the image [5, 47]. Here, we use two methods designed to eliminate some of the HT background noise. One common method is the usage of gradient direction information about the edge points. On the basis of a simple probabilistic model, we have shown in [37] that using gradient direction information significantly reduces the range of θ values that need to be considered when computing the Hough transform.

Given the gradient direction θ_g at the edge point (x_e, y_e), ideally, one needs to compute the HT of (x_e, y_e) only for $\theta = \theta_g$. However, the uncertainty in $\hat{\theta}_g$ (which is a computed estimate of the actual gradient direction) makes it necessary to compute the HT for a small range of θ values centered around $\hat{\theta}_g$. Common edge operators (e.g., Sobel) are usually used to compute $\hat{\theta}_g$. Here we use the θ range ± 0.2 radian proposed in [35].

Limiting the Hough transform estimation process to a small range of θ values centered around the gradient direction is more computationally efficient and reduces background noise inherent to the Hough transform formulation [5]. The improvement for incorporating gradient direction information is shown in Section 4.4.

Another approach (which can improve the performance of the standard HT) is to associate the HT vote

[2] As noted before, the HT cells should not be confused with the BSP tree representation cells.

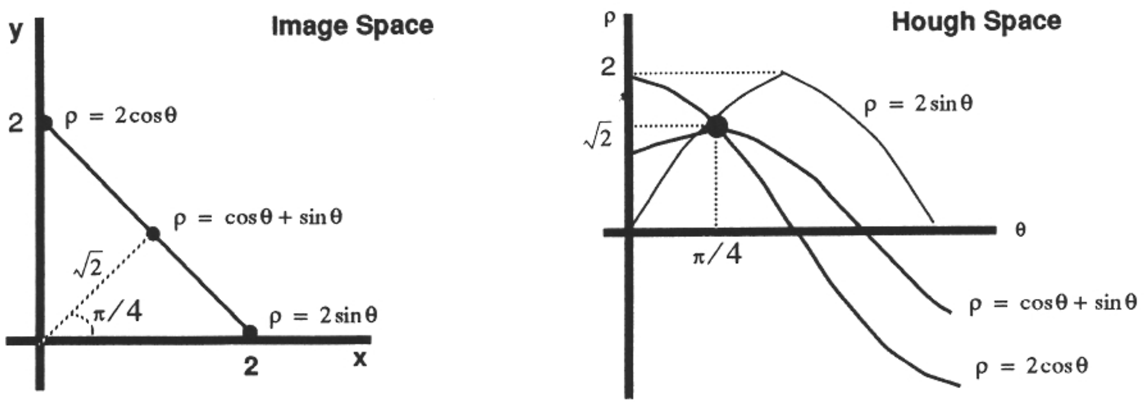


FIG. 4. An example of the Hough transform of three edge points.

that a given edge point (x_e, y_e) casts in the Hough space with an edge segment connectivity measure [47]. In this work, we introduce a modified HT method that weights the edge point votes using a line integral procedure. This new HT approach is explained in detail in the next subsection.

4.3. *The Modified Hough Transform*

Real-life images may contain a large number of *uncorrelated* collinear edge points that do not belong to the same edge (object boundary). Examples of this scenario are shown in Fig. 5. By using the standard HT, these uncorrelated collinear points will cause the occurrence of false peaks in the Hough space [47]. One way to solve this problem is to associate a weight for each vote that an edge point casts (in the parameter space) for a given cell (θ, ρ). If the edge point (x_e, y_e) is part of an edge segment e that lies on the line (θ, ρ), then the vote of (x_e, y_e) for (θ, ρ) should weight more heavily than other edge points which also lie on (θ, ρ) but do not belong to the edge segment e. Here we are assuming that e is either the only or most significant edge segment that lies on the line (θ, ρ).

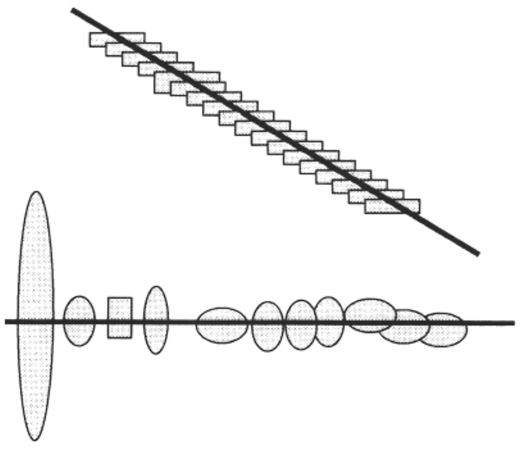


FIG. 5. Examples of uncorrelated, collinear boundary points.

To quantify this concept, we define the variable $v_e(\theta, \rho)$ as the line integral [along the line (θ, ρ)] of the product $E(x, y) \cdot w(|x - x_e|, |y - y_e|)$, where $E(x, y)$ is the edge image, and $w(r, s)$ is a 2-D nonnegative function that is symmetric and monotonically decreasing in all directions away from the origin. Therefore, instead of contributing only one vote to $ht(\theta, \rho)$ under the standard HT, an edge point (x_e, y_e) will increase $ht(\theta, \rho)$ by $v_e(\theta, \rho)$ under this technique:

$$ht(\theta, \rho) = \sum_{x_e} \sum_{y_e} \delta(\rho - x_e \cos\theta - y_e \sin\theta) \times E(x_e, y_e) v_e(\theta, \rho). \tag{4.3.1}$$

The line integration as performed is a natural way of measuring connectivity along the edge being considered. We chose $w(r, s)$ to be a Gaussian-shaped function with a standard deviation σ_l. For a given value of σ_l, the line integral is carried over a finite length $l = 2d$, where l is proportional to σ_l. The integral along the line (θ, ρ) can be expressed using the following parametric forms for x and y, assuming $\pi/4 \le |\theta| \le \pi/2$:

$$\begin{aligned} x(t) &= t + x_e & -d\sin(\theta) \le t \le d\sin(\theta) \\ y(t) &= -t\,\mathrm{cotan}(\theta) + y_e & -d\sin(\theta) \le t \le d\sin(\theta) \\ &= -[x(t) - x_e]\ \mathrm{cotan}(\theta) + y_e. \end{aligned}$$

Now $v_e(\theta, \rho)$ can be expressed as

$$v_e(\theta, \rho) = \sqrt{\mathrm{cotan}^2(\theta) + 1} \int_{-d\sin(\theta)}^{d\sin(\theta)} \times E[x(t), y(t)] e^{-[(x(t)-x_e)^2 + (y(t)-y_e)^2]/2\sigma_l^2}\, dt. \tag{4.3.2}$$

For other values of θ, the x coordinate is expressed as a function of y. This is done to avoid aliasing when imple-

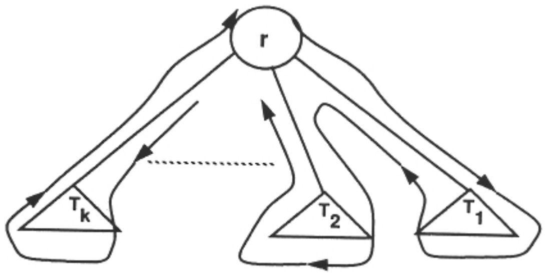


FIG. 6. An example of traversing a tree.

menting the line integration on a discrete sampling lattice. In such a case, the following equations were used to compute $v_e(\theta, \rho)$:

$$\begin{aligned} y(t) &= t + y_e & -d\cos(\theta) \le t \le d\cos(\theta) \\ x(t) &= -t\tan(\theta) + x_e & -d\cos(\theta) \le t \le d\cos(\theta) \\ &= -[y(t) - y_e]\tan(\theta) + x_e & \end{aligned}$$

$$\begin{aligned} v_e(\theta, \rho) = {} & \sqrt{\tan^2(\theta) + 1} \int_{-d\cos(\theta)}^{d\cos(\theta)} \\ & \times E[x(t), y(t)] e^{-[(x(t)-x_e)^2 + (y(t)-y_e)^2]/2\sigma_l^2}\, dt. \quad (4.3.3) \end{aligned}$$

The above integrations were implemented by incrementing t by 1 within the specified ranges. The performance of such technique is explicitly described in Section 4.4.

4.4. *Segmentation Performance of the HT-Based Method*

Similar to other tree-like data structures, when building a BSP tree one has to follow a systematic tree traversal approach. The *preorder, inorder,* and *postorder* algorithms are commonly used for traversing trees. (See [2] for the definitions of these traversal methods.) If T is a tree with root r and subtrees $T_1, T_2, \ldots, T_k$ (for binary trees, $k = 2$), then a useful way to visualize the different traversal schemes is to draw a path around the tree T starting from the root node and staying as close as possible to the nodes of the trees as shown in Fig. 6. In our case we are attempting to generate a tree, by first searching for the "best" line that can be used to partition the image into two subimages, and associating this line with the root node of our BSP tree. Since the same process (searching for the best line) will be repeated on one of the two subimages after generating the root node, we construct our desired BSP tree in a *preorder* manner.

This preorder traversal algorithm was implemented in the C language using the HT-based method (described above) to select the partitioning lines, and to construct the BSP tree representation of the desired image. The algorithm was tested on the 256 × 256 image shown in Fig. 7. This image was chosen due to its complexity. As seen from the figure, the image contains a fair amount of texture (e.g., the hat surface and the flowers) in addition to very low-contrast edges (the right side of the lady's face). Moreover, it represents a human face for which distortion is particularly noticeable.

We first applied the algorithm on the edge image of Fig. 8a. The edge image was obtained (without any filtering) using a nonlinear edge detector proposed in [36] in combination with a thinning process. In this case we computed the HT using the standard method, i.e., considering all θ values between zero and 180°. Therefore, under this scenario we did not use any gradient direction or edge connectivity information for the HT. Figure 8b shows the output which is a labeled, segmented image, where the darkest and brightest regions represent the right-most and left-most leaf nodes in the resulting BSP tree.

It is clear that this BSP tree does not represent the original image, except for some of the face features which were vaguely reproduced. This poor performance results from the occurrence of a large number of false peaks in the Hough space. These false peaks are caused by the large number of uncorrelated collinear texture points.

Figure 9 shows three edge images obtained with the gradient edge detector described in Section 4.1. These edge images were derived from a Gaussian-filtered version of the original image. Figures 9a and 9b show the resulting edges when only a low threshold T_l or a high threshold T_h is used, respectively. Figure 9c shows the edges obtained when both T_l and T_h are applied using the thresholding with the hysteresis algorithm. It is clear that significant improvements can be achieved by using the hysteresis approach versus a simple thresholding strategy. All the simulation results discussed below were obtained using the edge image of Fig. 9c as an input to our BSP tree construction algorithm.

Figure 10 shows the results of applying the algorithm and using the gradient direction information when computing the HT. The labeled image (which visualizes our

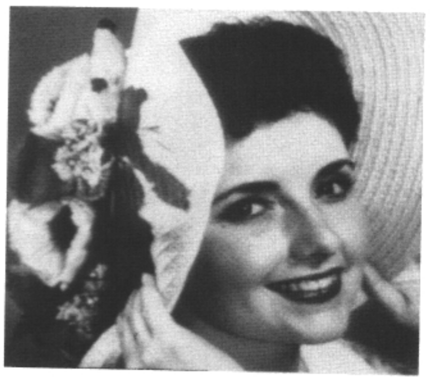


FIG. 7. The 256 × 256 original image with 8 bits per pixel.

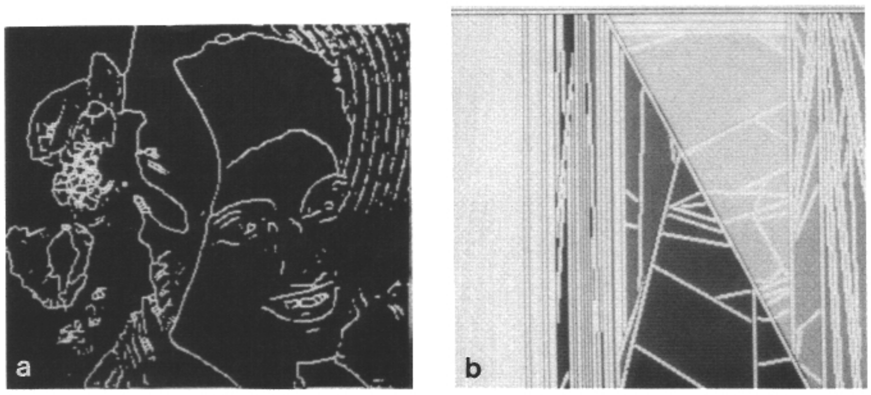


FIG. 8. (a) Edges of the original image of Fig. 7. The edges were detected using a robust edge detector and without filtering. (b) A labeled image illustrating the BSP tree representation of the original image. This representation was generated using the standard Hough transform and without any gradient direction information.

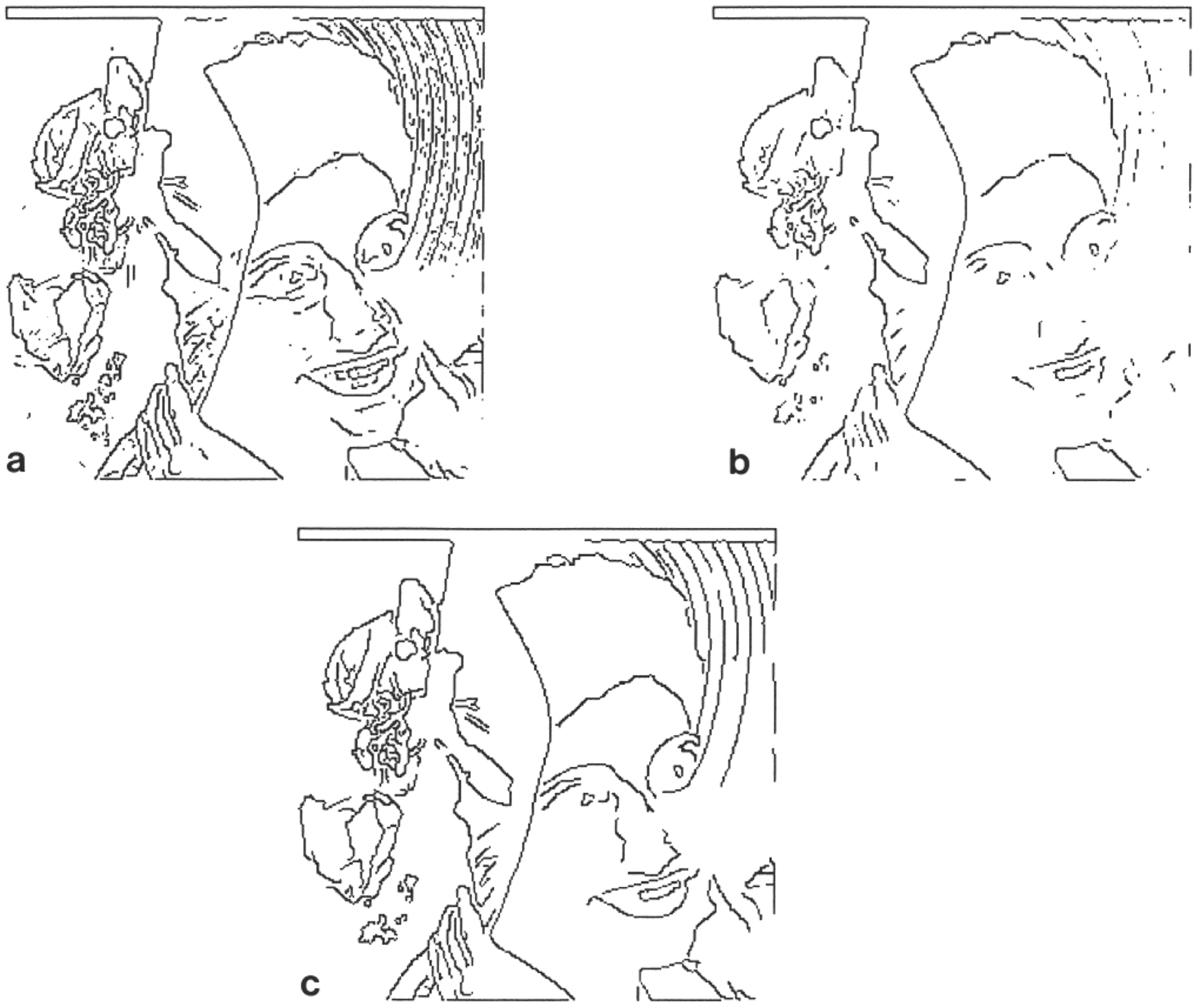


FIG. 9. (a) Edges of the original image using gradient edge detector and a simple thresholding strategy with a low threshold value. (b) Edges of the original image using gradient edge detector and a simple thresholding strategy with a high threshold value. (c) Edges of the original image using gradient edge detector and a hysteresis thresholding strategy with both low and high thresholds.

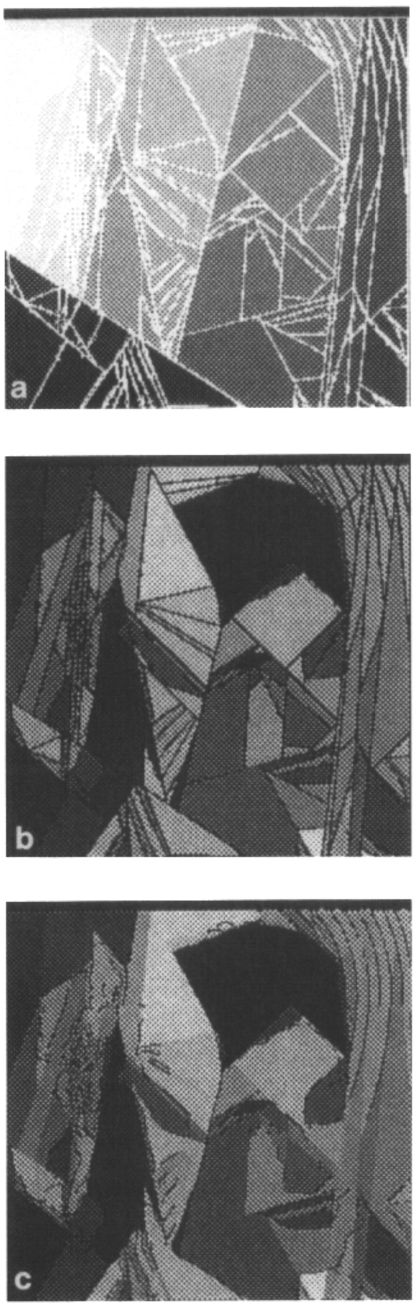


FIG. 10. BSP tree representation images resulting from applying the HT-based method on the original image of Fig. 7, and using gradient direction information when computing the Hough transform. (a) A labeled image showing the partitioning lines and the convex regions resulting from the binary partitioning. (b) A mean-value image with the partitioning lines, where each region is filled with its pixel intensities' mean. (c) A mean-value image without the partitioning lines.

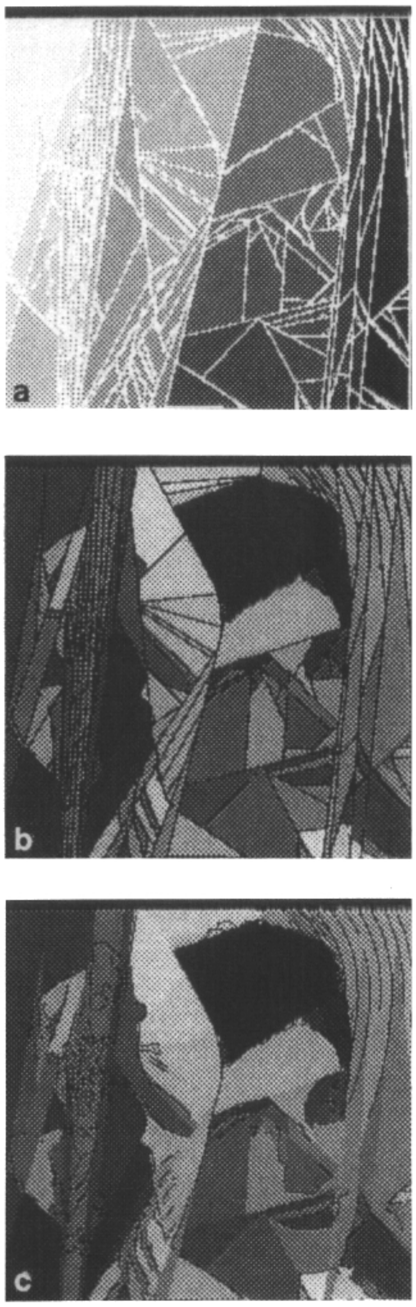


FIG. 11. BSP tree representation images resulting from applying the HT-based method on the original image of Fig. 7, and using the modified Hough transform. In the modified HT both gradient direction information and line integration are used. (a) A labeled image showing the partitioning lines and the convex regions resulting from the binary partitioning. (b) A mean-value image with the partitioning lines, where each region is filled with its pixel intensities' mean. (c) A mean-value image without the partitioning lines.

BSP tree representation of the original image) of Fig. 10a illustrates a significant improvement when compared with the image of Fig. 8b. This improvement is mainly due to the usage of gradient direction information in the HT. Figures 10b and 10c show the images resulting from filling the leaf node regions with the mean value of the corresponding regions of the original image. Figure 10b also shows the BSP tree lines used to partition the image.

Although the gradient direction information has helped in eliminating a large number of false peaks due to the uncorrelated collinear edge points, more improvements can be achieved when applying the line integration process (in combination with the gradient direction information) as shown in Fig. 11. The most obvious example of these improvements is the root node line. From the edge image of Fig. 9c, one would expect the straight line corresponding to the hat edge to be the first partitioning line. (The first selected line partitions the whole image into two segments and represents the root node of the binary tree.) This desired result is obtained in Fig. 11 (due to line integration) but not in Fig. 10. For the line integrals of eqs. (4.3.2) and (4.3.3), we have used $\sigma_l = 1$.

To further improve the quality of our BSP tree representation, we have used within our HT-based algorithm a *parent–child minimum distance* constraint. Under this approach the line (θ_p, ρ_p) associated with a parent node μ_p has to be a minimum distance (in the Hough space) from its child line (θ_c, ρ_c). This minimum distance requirement can be expressed as

$$|\theta_p - \theta_c| > \theta_d \quad \text{or} \quad |\rho_p - \rho_c| > \rho_d,$$

where θ_d and ρ_d are the desired minimum distances in the θ and ρ directions, respectively. The results of using this strategy are shown in Fig. 12. It is clear that the minimum distance constraint has improved the images by guiding the segmentation process in areas with high texture density (e.g., the flowers' region).

Figure 13 shows mean value images resulting from using this strategy for the two cases (1) $\theta_d > \rho_d$ (Figs. 13a and 13b) and (2) $\theta_d < \rho_d$. Both θ_d and ρ_d are measured in terms of the number of HT cells in the θ and ρ directions, respectively. In Fig. 12, we set $\theta_d = \rho_d = 10$, for Figs. 13a and 13b, $\theta_d = 2\rho_d = 20$, and for Figs. 13c and 13d, $\rho_d = 2\theta_d = 20$.

For each of the two cases we also show the results of performing the line integration process across and within the regions. Selecting a large minimum distance in the ρ direction (see Fig. 13d) has helped in partitioning some of the large polygons appearing as artifacts across the right side of the face, at the expense of poorer segmentation of the finer objects (e.g., the flower) in the image. On the other hand, performing the line integration across the regions provided a better segmentation of the finer objects (e.g., the flower and the lady's nose) but at the expense of introducing some artifacts on the larger object (the right side of the face). Overall, the segmentation of Fig. 12 provided the best results (see, for example, the preserving of the flower in the hat), as it uses good threshold values for θ_d and ρ_d.

The accuracy of the HT-based BSP tree representation is a function of the amount of boundary information used when constructing the tree. Therefore, a much more detailed representation can be achieved by increasing the number of edge points detected from the desired image. To accomplish that, we first derived an enhanced version (Fig. 14a) of the image in Fig. 7 by simply increasing the

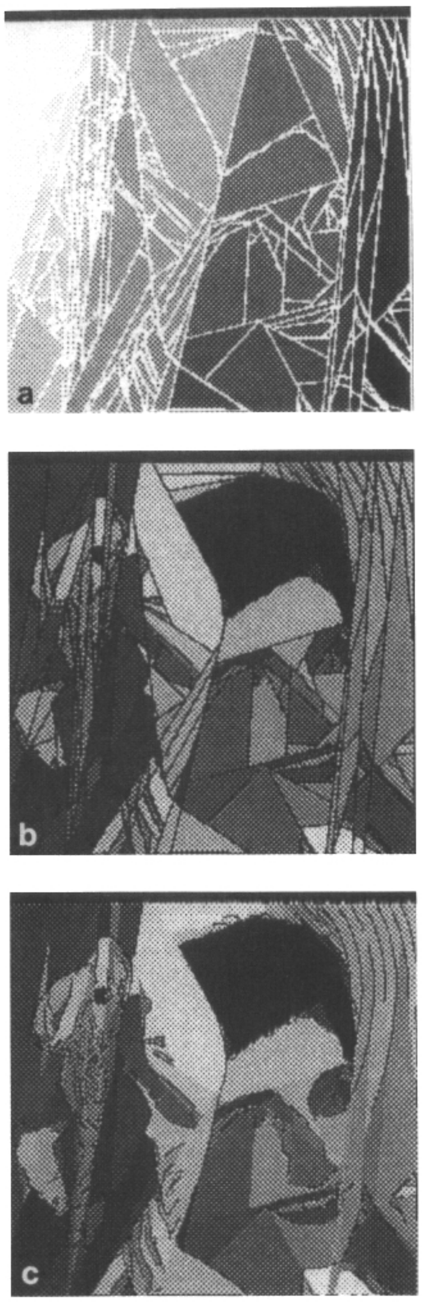


FIG. 12. BSP tree representation images resulting from applying the HT-based method on the original image of Fig. 7, and using the modified Hough transform and parent–child minimum distance strategy. (a) A labeled image showing the partitioning lines and the convex regions resulting from the binary partitioning. (b) A mean-value image with the partitioning lines, where each region is filled with its pixel intensities' mean. (c) A mean-value image without the partitioning lines.

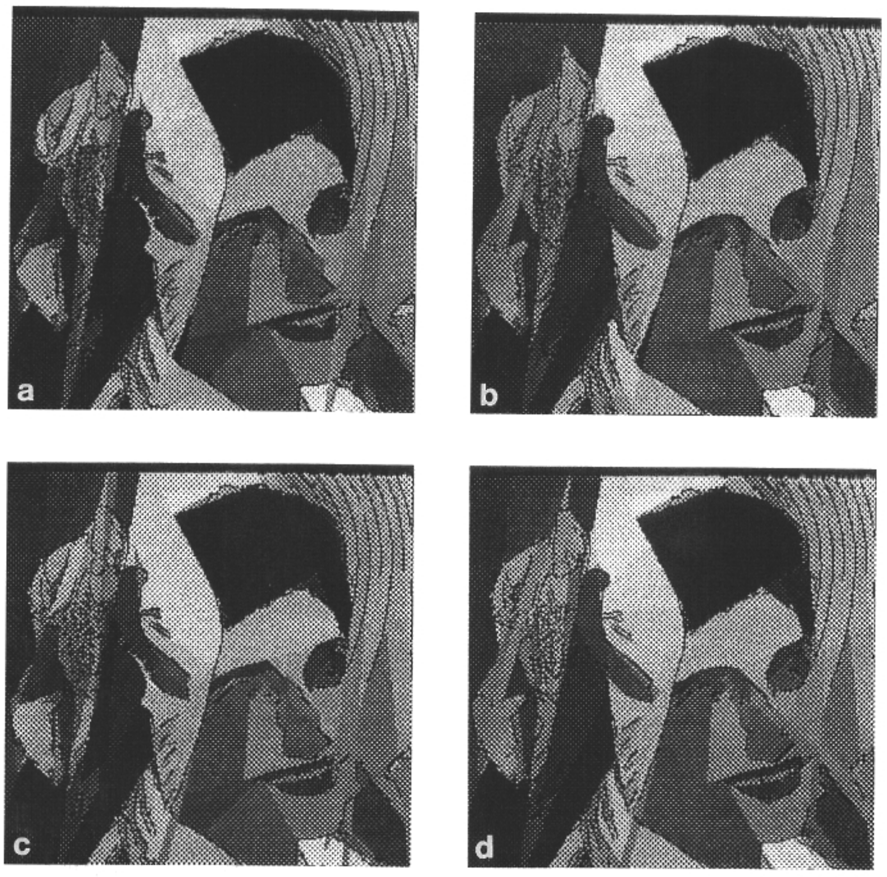


FIG. 13. Mean-value, BSP tree images resulting from applying the HT-based method on the original image of Fig. 7, and using the modified Hough transform and parent–child minimum distance strategy: (a) when performing line integration across the regions, and for $\rho_d = 10$ and $\theta_d = 20$; (b) when performing line integration within the regions, and for $\rho_d = 10$ and $\theta_d = 20$; (c) when performing line integration across the regions, and for $\rho_d = 20$ and $\theta_d = 10$; (d) when performing line integration within the regions, and for $\rho_d = 20$ and $\theta_d = 10$.

dynamic range of the gray levels in the image. By applying the HT-based method, one can generate the images shown in Figs. 14b–14d. It is clear from these figures that a much finer representation can be achieved by simply increasing the amount of boundary information.

5. THE MULTIRESOLUTION APPROACH

The HT-based method described in Section 4 is computationally intensive. A hierarchical implementation of the Hough transform is one solution to this problem as suggested in [47]. Another solution is to use a *multiresolution* method when building the desired BSP tree.

In this section, we describe a multiresolution-based, hierarchical method for generating the BSP tree from several scale-space images derived from an original image. As explained below, the multiresolution approach takes advantage of computations that take place at a coarse resolution level when performing similar computations at a finer (higher) resolution level. This new approach includes the introduction of a *line focusing* algorithm used to derive a tree representing an image (at a given resolution) from another tree representing a coarser image.

This strategy can provide a significant reduction in the computational expense, as explained below. In addition to the computational advantage, the hierarchical approach guarantees that the resulting tree is a *multiresolution* BSP tree. As explained in Section 2, traversing a multiresolution tree from top to bottom is equivalent to zooming at the image. Moreover, better segmentation is achieved using a hierarchical method.

The multiresolution approach derives a hierarchy (pyramid) of signals from the original signal [25, 6, 46] as shown in Fig. 15. We assign negative numbers to the coarse resolution levels since the original (highest resolution) signal starts at level 0.

A derived signal at a given resolution level L in the hierarchy contains the (original) signal's information of that level L and all other resolutions *lower* (or *coarser*) than L. For example, the image at level -1 in Fig. 15 contains the information of levels -1, -2, and -3. However, it does not contain the higher resolution (level 0) information. One can (for example) detect the boundaries

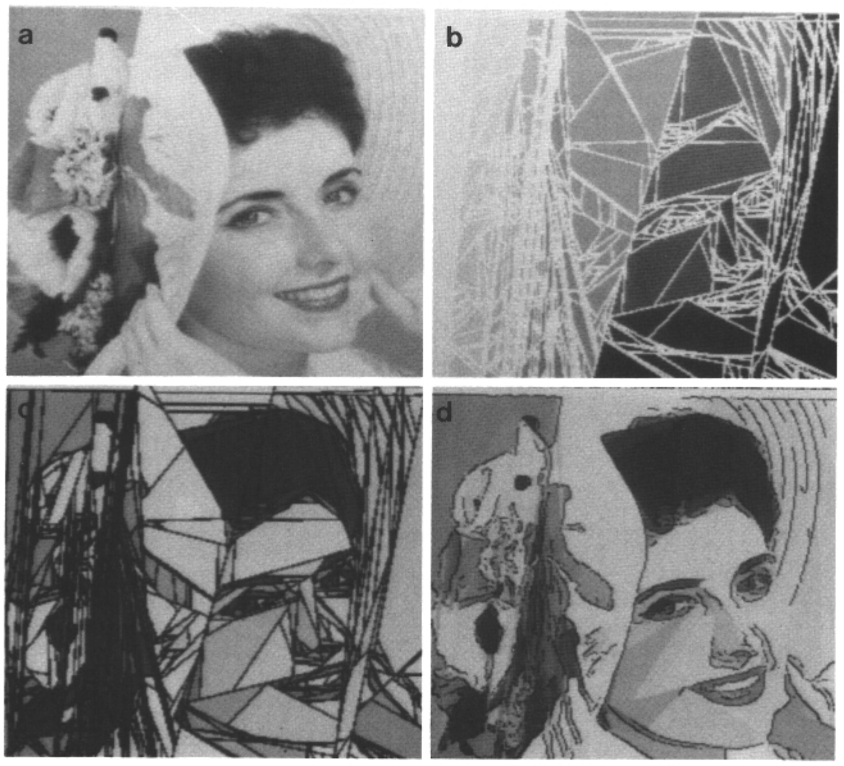


FIG. 14. Accurate BSP tree segmentation using more boundary information. (a) Enhanced version of the original image of Fig. 7. (b, c, d) BSP tree images resulting from applying the HT-based method on the image of (a), and using the modified Hough transform and parent–child minimum distance strategy. (b) A labeled image showing the partitioning lines and the convex regions resulting from the binary partitioning. (c) A mean-value image with the partitioning lines, where each region is filled with its pixel intensities' mean. (d) A mean-value image without the partitioning lines.

of large objects (coarse details) found in a scene by performing an edge detection process on a low-resolution image of that scene. The exact location of these coarse edges can then be detected using a high-resolution image of the scene. While this same low-resolution boundary information can also be detected from a high-resolution image of the same scene, performing the computation on the coarse image (in addition to refining the edge locations using the high-resolution image) can be significantly less intensive than performing the same computation directly on the high-resolution image.

In order to (1) reduce the computations required when generating a BSP tree representation of an image and (2) generate a multiresolution BSP tree, we construct the desired tree using this multiresolution approach. We first derive a hierarchy of images from the original image. Then, we build the BSP tree of the lowest (most coarse) resolution image in the pyramid using the HT-based method described in Section 4. We use this low-resolution tree as an initial guess to build another tree representing the next higher resolution image in the hierarchy. The same process (i.e., using a BSP tree representing an image at a given level of the pyramid as an initial guess to construct another tree representing the next higher resolution image in the pyramid) is repeated until we generate the BSP tree that represents the highest resolution (original) image. Below, we describe in detail our multiresolution approach for deriving a high-resolution-image BSP tree from another tree that represents a lower resolution image. Without loss of generality, we assume that the high-resolution image is the original (level 0) image. This method can be applied between any two successive levels of the pyramid.

5.1. *From a Coarse Tree to a Finer Tree*

Let I_0 be the original image, and E_0 be the edge image of I_0. From I_0, we derive the image I_m by (1) convolving I_0 with a 2-D Gaussian filter $g_{\sigma_m}(x, y)$ of variance σ_m^2, and (2) subsampling the filtered image using a decimation factor $M = 2^{-m}$ in both the x and the y directions, where $m \leq 0$ is the resolution level. An edge image E_m is then derived from I_m using a maximum-gradient edge detection process described in Section 4.1. By employing the HT-based method, a BSP tree T_m is then generated from the edge image E_m. [Every leaf node in T_m represents an unpartitioned, convex region of I_m. Every nonleaf node μ_m, which represents a convex region R_m in I_m, is associated with a line (θ_m, ρ_m) that partitions the region R_m.]

In Section 4, we derived a BSP tree T_0' (which repre-

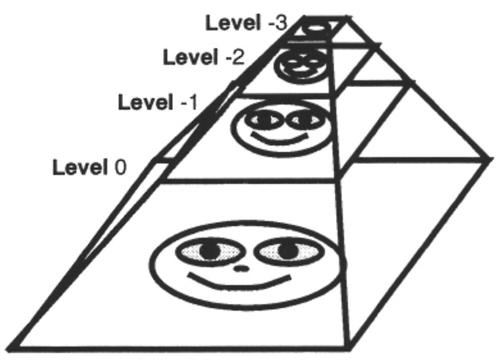


FIG. 15. A multiresolution pyramid.

sents the original image I_0) from the edge image E_0 only. It should be noted that E_0 contains the boundary information of I_0 at all resolution levels $L \le 0$.[3] The previous approach, however, is very computationally intensive. Here, the objective is to reduce the computational burden of the Hough transform-based method by deriving a BSP tree T_0 (which represents the original image I_0) from:

(1) the tree T_m which represents the image I_m, and contains the boundary information of the original image I_0 at all resolution levels $L \le m$;

(2) the boundary information of the resolution levels $m < L \le 0$ contained within the edge image E_0.

It is important to note that both T_0 and T_0' represent I_0 at all resolution levels $L \le 0$. In general, however, $T_0' \ne T_0$.

We derive T_0 in two stages: (1) a binary tree T_{m0} is derived using a preorder tree traversal of T_m and a line focusing algorithm (explained below), and (2) T_0 is then generated by replacing the leaf nodes of T_{m0} with new BSP subtrees which represent the boundary information at the resolutions $m < L \le 0$.

It should be clear that T_{m0} has the same shape, structure, and number of nodes as T_m. Moreover, and under ideal conditions, a line h_m associated with a node μ_m in T_m will have the same parametric representation (θ_m, ρ_m) as the line h_0 associated with the corresponding node μ_{m0} (which is derived from the node μ_m). Due to filtering, however, the edges of I_0 shift from their original positions. It was shown [4] that this shift is proportional to the scale-space parameter σ_m. We used this result to design a line focusing algorithm.

5.2. *Line Focusing Algorithm*

Let $h_0 = (\theta_0, \rho_0)$ be a straight line that: (1) partitions the region R_0 of the original image I_0, and (2) lies on several edge points from E_0. Let $p_0 = (x_0, y_0)$ be an edge point on h_0 as shown in Fig. 16. Filtering I_0 by a 1-D Gaussian kernel in the x direction causes p_0 to shift by a distance $|\Delta_x| \le c\sigma_m$, where c is some positive constant and σ_m is the standard deviation of the Gaussian filter. Filtering the resultant image by the same Gaussian kernel in the y direction causes p_0 to shift (further) a distance $|\Delta_y| \le c\sigma_m$. Under worst case conditions, either $\Delta_x = \Delta_y = +c\sigma_m$ or $\Delta_x = \Delta_y = -c\sigma_m$. Therefore, filtering I_0 by a 2-D Gaussian kernel causes p_0 to shift by a distance $|\Delta_{p_0}| \le \sqrt{2}c\sigma_m$. We denote by p_{m^+} and p_{m^-} the points $(x_0 + c\sigma_m, y_0 + c\sigma_m)$ and $(x_0 - c\sigma_m, y_0 - c\sigma_m)$, respectively (see Fig. 16).

If every point p_0 on h_0 moves to a new point p_{m^+} [or p_{m^-}], then it is trivial to show that these displaced points form a new line $h_m^+ = (\theta_{m^+}, \rho_{m^+})$, [or h_m^-], where $\theta_{m^+} = \theta_0$ [or $\theta_{m^+} = \theta_0$] and $\rho_{m^+} = \rho_0 + c\sigma_m(\cos\theta_0 + \sin\theta_0)$ [or $\rho_{m^-} = \rho_0 - c\sigma_m(\cos\theta_0 + \sin\theta_0)$].

Therefore, under a worst case scenario, the line h_m (at resolution m) corresponding to the line h_0 (at the higher resolution 0) has to be within the following region (the shaded strip in Fig. 16):

$$S_{h_m} = R_0 \cap H^+_{h_m^-} \cap H^-_{h_m^+}, \tag{5.2.1}$$

where $H^+_{h_m^-}$ and $H^-_{h_m^+}$ are the positive and negative half-planes of the lines h_m^- and h_m^+, respectively.

Similarly, one can show that given a line $h_m = (\theta_m, \rho_m)$ which partitions a region R_0 of I_0, its corresponding line h_0 has to be within the region

$$S_{h_0} = R_0 \cap H^+_{h_0^-} \cap H^-_{h_0^+}, \tag{5.2.2}$$

where $h_0^+ = (\theta_m, \rho_m + \Delta\rho_{\max})$, $h_0^- = (\theta_m, \rho_m - \Delta\rho_{\max})$, and

$$\Delta\rho_{\max} = c\sigma_m(\cos\theta_m + \sin\theta_m). \tag{5.2.3}$$

Therefore, by performing our modified Hough transform (as explained in Section 4.2) on all edge points within S_{h_0}, we can detect the parametric representation (θ_0, ρ_0) of the

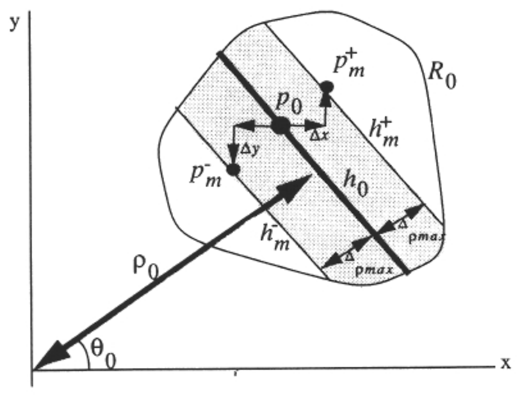


FIG. 16. The line focusing algorithm.

[3] Before proceeding, it is important to note that both m and L denote the resolution levels of the pyramid. However, for a given level m we use L as a variable representing all possible (including m) resolution levels.

line h_0. In this case (θ_0, ρ_0) are the coordinates of the modified HT peak.

It should be clear that the smaller $\Delta\rho_{\max}$ is, the less computation is required to detect the line h_0. Consequently, given $h_m = (\theta_m, \rho_m)$ and c, one can reduce $\Delta\rho_{\max}$ by selecting smaller values for σ_m. However, using small σ_m increases the amount of aliasing energy in the decimated image I_m. This represents a trade-off between the amount of: (1) computation required to detect h_0, and (2) aliasing energy one can tolerate in the decimated image.

Below we derive an expression for $\Delta\rho_{\max}$ in terms of a measure (F_a) of the aliasing energy of the 2-D Gaussian filter. Using this expression, one can quantify the relationship between the (minimum) desired search area of S_{h_0} and the amount of aliasing.

5.2.1. *Line focusing and aliasing*. To avoid aliasing when decimating a 1-D, discrete-time sequence by a factor M, the sequence has to be convolved (first) by a filter whose transfer function's cutoff frequency $\Omega_c \le \pi/M$ [12]. For 2-D signals the same principle is applied to the two orthogonal frequency axes.

The 2-D Gaussian function

$$g_{\sigma_m}(x, y) = \frac{1}{2\pi(\sigma_m)^2}\, e^{-(x^2+y^2)/2\sigma_m^2} \qquad (5.2.4)$$

has a transfer function

$$G_{\sigma_\omega}(x, y) = e^{-(\Omega_x^2+\Omega_y^2)/2\sigma_\omega^2}, \qquad (5.2.5)$$

where $\sigma_\omega = 1/\sigma_m$.

The Fourier transform of the Gaussian filter is a Gaussian-shaped function with infinite tails in both the Ω_x and the Ω_y directions. Therefore, using the Gaussian kernel as a predecimation filter makes aliasing inevitable. What can be done, however, is to reduce aliasing by using small (large) values of σ_ω (σ_m).

For a given (or desired) energy W under $G_{\sigma_\omega}(x, y)$, we define the Gaussian cutoff frequency Ω_{gc} as the radial frequency Ω_r satisfying the condition

$$W = \frac{1}{4\pi^2} \int_0^{2\pi} \int_0^{\Omega_{gc}} (e^{-\Omega_r^2/2\sigma_\omega^2})^2\, d\Omega_r(\Omega_r d\Omega_\theta), \qquad (5.2.6)$$

where $\Omega_r = \sqrt{\Omega_x^2 + \Omega_y^2}$, and $\Omega_\theta = \arctan(\Omega_y/\Omega_x)$.

It can be shown that

$$W = \frac{\sigma_\omega^2}{4\pi}(1 - e^{-\Omega_{gc}^2/2\sigma_\omega^2}). \qquad (5.2.7)$$

The total energy W_G under $e^{-(\Omega_x^2+\Omega_y^2)/2\sigma_\omega^2}$ can be expressed as $W_G = \sigma_\omega^2/4\pi$. Therefore, by dividing W by W_G, we get

$$\frac{W}{W_G} = (1 - e^{-\Omega_{gc}^2/2\sigma_\omega^2}). \qquad (5.2.8)$$

Solving for Ω_{gc},

$$\Omega_{gc} = \sigma_\omega \sqrt{\ln\left(\frac{W_G}{W_G - W}\right)}. \qquad (5.2.9)$$

Here we define the aliasing factor F_a as

$$F_a = \frac{W_G - W}{W_G}. \qquad (5.2.10)$$

It should be clear that F_a is a measure of the aliasing energy which is due to the infinite tails of the Gaussian-shaped transfer function $e^{-(\Omega_x^2+\Omega_y^2)/2\sigma_\omega^2}$. Using F_a, Ω_{gc} can be expressed as

$$\Omega_{gc} = \sigma_\omega \sqrt{\ln\left(\frac{1}{F_a}\right)}. \qquad (5.2.11)$$

Applying the condition $\Omega_c \le \pi/M$ to Ω_{gc}, one gets

$$\sigma_\omega \le \frac{\pi}{M\sqrt{\ln(1/F_a)}}, \qquad (5.2.12)$$

or

$$\sigma_m \ge \frac{M}{\pi}\sqrt{\ln\left(\frac{1}{F_a}\right)}. \qquad (5.2.13)$$

Using the minimum (permissible) value of σ_m for a given aliasing factor F_a, we can write $\Delta\rho_{\max}$ as

$$\Delta\rho_{\max} = (cM/\pi)\sqrt{\ln(1/F_a)}\,(\cos(\theta_m) + \sin(\theta_m)). \qquad (5.2.14)$$

Therefore, in order to find $h_0 = (\theta_0, \rho_0)$ it is sufficient to perform the Hough transform on every edge point (x_e, y_e) within the convex region R_0 of E_0 that satisfies the condition

$$\rho_m - \Delta\rho_{\max} \le \rho \le \rho_m + \Delta\rho_{\max}, \qquad (5.2.15)$$

where $\rho = x_e \cos(\theta_m) + y_e \sin(\theta_m)$.

As will be shown in Section 5.3, the algorithm provides very satisfactory results when tested on real images.

5.3. *The Multiresolution Approach Simulation Results*

The multiresolution approach described above was tested on the 256×256, gray-scale image shown in Fig. 14a. This image represents, in our case, the resolution level $m = 0$. Two scale-space images (I_{-1} and I_{-2}) at resolutions $m = -1$ $(M = 2)$ and $m = -2$ $(M = 4)$ were derived using $\sigma_m = 1$. Three edge images E_0 (Fig. 17g), E_{-1} (Fig. 17d), and E_{-2} (Fig. 17b) were derived from I_0,

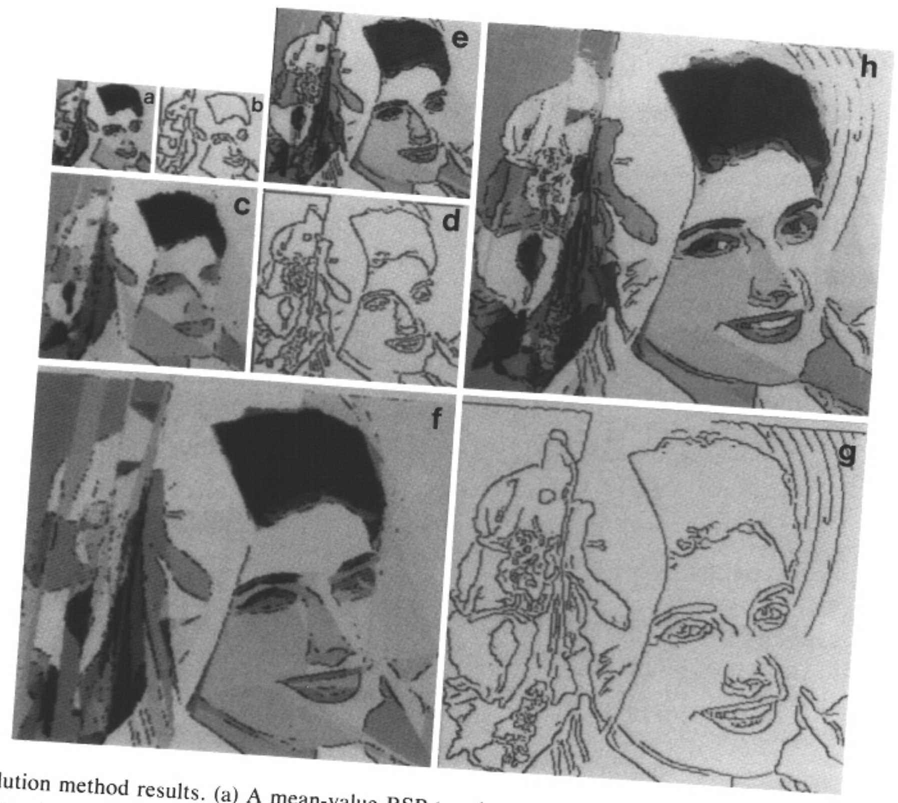


FIG. 17. The multiresolution method results. (a) A mean-value BSP tree image resulting from applying the HT-based method on a 64 × 64 image which represents the most coarse level ($L = -2$) of the multiresolution pyramid of the image in Fig. 14a. (b) Edges at resolution level $L = -2$. (c) A mean-value image resulting from applying the line focusing algorithm on the tree of (a). (d) Edges at resolution level $L = -1$. (e) A mean-value BSP tree image resulting from expanding the tree of (c) using the higher resolution edge information of (d). (f) A mean-value image resulting from applying the line focusing algorithm on the tree of (e). (g) Edges at the original resolution level $L = 0$. (h) A mean-value BSP tree image resulting from expanding the tree of (f) using the higher resolution edge information of (g).

I_{-1}, and I_{-2}, respectively, using a maximum-gradient edge detector.

In this section, we show the results as mean-value images, where each unpartitioned region (which corresponds to a leaf node of the tree) has been filled with the average value of the pixels' intensities within that region. We refer to these images as the BSP tree images.

First the HT-based method was applied on the edge image E_{-2} in Fig. 17b (of size 64 × 64). The resulting BSP tree image is shown in Fig. 17a. The corresponding tree T_{-2} and the edge image E_{-1} were used to derive the BSP tree image shown in Fig. 17c. By traversing T_{-2} and using the line focusing algorithm, the line equations at the resolution level $m = -1$ were determined. Therefore, Fig. 17c shows the details of resolution level $m = -2$ displayed at resolution $m = -1$. Figure 17e shows the result of expanding the leaves of T_{-2} using the boundary information of resolution $m = -1$ found in the edge image E_{-1}.

Using the BSP tree T_{-1} (whose image is shown in Fig. 17e), we repeat the same process: (1) derive the BSP tree image of Fig. 17f which shows the details of resolution $m = -1$, and (2) expand the leaves of T_{-1} with new subtrees (representing the details at resolution $m = 0$) using boundary information within E_0 (Fig. 17g). The result of step (2) is shown in Fig. 17h.

By applying the HT-based method on E_0 directly, one gets the BSP tree image shown in Fig. 14d. By comparing this image with the BSP tree image of Fig. 17h, we can see that a better segmentation can be achieved using the multiresolution approach. This is an added value to the main speed advantage one can gain when employing this new approach. For example, using a Sun 4 workstation, it takes about 50 min to generate the image of Fig. 14d, whereas the corresponding image of Fig. 17h was generated (using the multiresolution approach) in about 10 min on the same machine.

6. COMPUTATIONAL ASPECTS

So far we have presented a boundary-based (or HT-based) method for constructing a BSP tree representation of an arbitrary image I_0. The method uses (as an input) an edge image E_0 of the desired original image I_0. We also described a multiresolution approach for constructing a multiresolution BSP tree representation of I_0 using a pyramid of images (and their corresponding edge images).

As mentioned above, one of the advantages of this multiresolution approach is the reduction in the computational expense of the HT-based method.

In this section we derive two computational complexity expressions: (1) for constructing a BSP tree T_0' using the HT-based method (i.e., building T_0' directly from E_0), and (2) for building a BSP tree T_0 when applying the multiresolution approach at one level of the hierarchical pyramid [i.e., building T_0 from (1) a low-resolution tree T_m, and (2) the high-resolution edge image E_0]. On the basis of these two expressions, we quantify the computational advantage of applying the multiresolution approach. First we develop a model for the computational complexity of building a (generic) binary tree T from an edge image E.

The time required to build T from E depends mainly on (1) the number of edge points P_E in E, and (2) the topology (i.e., the tree structure) of T. Due to the large number of possible topologies of a binary tree T with n ($\geqslant 1$) nodes, here we only consider two extreme cases of binary tree structures: (a) the complete (fully symmetric) and (b) fully asymmetric topologies. Figure 18 shows the fully symmetric and asymmetric binary trees for $n = 15$. These two cases, (a) and (b), represent the lower and upper bounds for the computational complexity of building T, respectively. We measure the computational complexity of both cases as the *total* number of edge points P_T needed to be processed in order to build the desired tree T.

Here we use N to denote the number of partitioning lines, and $\bar{p}$ as the average number of edge pixels that lie on a line h (detected from E). Therefore, $P_E = N\bar{p}$.

A *complete binary* tree T_c of depth d has all of its leaves at level d [42]. The total number of nonleaf nodes in T_c is $2^d - 1$. Therefore, the number of straight lines $N = 2^d - 1$, or $d = \log_2(N + 1)$. After detecting the root-node line (which requires processing all P_E edge points in E), one has to process (on the average) $P_E - \bar{p}$ edge points to detect the two lines (at level 1 of T) that are associated with the right and left children of the root node. Similarly, in order to detect the 2^j partitioning lines represented by the nodes at level j of the complete tree T_c, one has to process $P_E - \sum_{i=0}^{j-1} 2^i\bar{p} = P_E - (2^j - 1)\bar{p}$ edge points.

Therefore, the computational complexity of generating T_c can be expressed as

$$P_T = \sum_{j=0}^{d-1} [P_E - (2^j - 1)\bar{p}]. \tag{6.1}$$

This expression can be reduced to

$$P_T = \bar{p}[(N + 1)\log_2(N + 1) - N]. \tag{6.2}$$

A *fully asymmmetric* binary tree of depth d has d ($=N$) nonleaf nodes. There is one nonleaf node for each level between zero (the root node) and level $d - 1$. The nonleaf node at level $d - 1$ has two leaf children, whereas each of the remaining $d - 1$ nonleaf nodes has one leaf child and one nonleaf child. In order to detect the straight line associated with the nonleaf node at level j of a fully asymmetric binary tree T_a, it is required (on the average) to process $P_E - j\bar{p}$ edge points. Therefore, the computational complexity of generating T_a can be expressed as

$$P_T = \sum_{j=0}^{d-1} [P_E - j\bar{p}]. \tag{6.3}$$

This expression can be reduced to

$$P_T = \bar{p}[N(N + 1)/2]. \tag{6.4}$$

Using these results, one can estimate the time t_T required to build a BSP tree T of an image using the HT-based method as $t_T = t_{ht}P_T$, where t_{ht} is the time required to compute the HT of an edge point. Denoting by N_0, N_m, $\bar{p}_0$, and $\bar{p}_m$ the number of lines and the average numbers of edge pixels per line, respectively, in the edge images E_0 and E_m, it can be shown that the times $t_{T_0'}$ and t_{T_m} required to generate T_0' (from E_0) and T_m (from E_m) satisfy the inequalities

$$[d_m(N_m + 1) - N_m] \le \frac{t_{T_m}}{t_{ht(m)}\bar{p}_m} \le [N_m(N_m + 1)/2] \tag{6.5}$$

$$[d_0(N_0 + 1) - N_0] \le \frac{t_{T_0'}}{t_{ht(0)}\bar{p}_0} \le [N_0(N_0 + 1)/2], \tag{6.6}$$

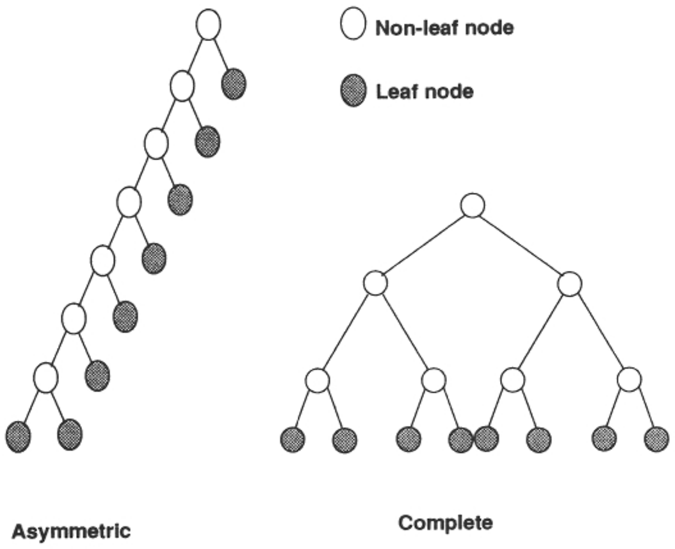


FIG. 18. Asymmetric and complete binary trees.

where d_0 and d_m are the depths of T_0 and T_m, respectively.

Now we can estimate the computational advantage of employing the multiresolution method. We define the computational advantage ratio as $R_{CA} = t_{T'_0}/t_{T_0}$, where t_{T_0} is the time required to build T_0 (from T_m and E_0).

To simplify the derivation we assume that every straight line h_0 detected from E_0 (when building T'_0) has a corresponding line h_m in T_m. In other words, E_0 contains boundary information at resolution levels $L \leq m$ only. Therefore, the number of lines N_m at resolutions $L \leq m$ is equal to N_0 above. (This is a reasonable assumption since: (1) the time t_{m-0} required to detect the partitioning lines at resolutions $m < L \leq 0$ is usually much smaller than the time required to detect the lines at resolutions $L \leq m$, and (2) t_{m-0} contributes to both $t_{T'_0}$ and t_{T_0}.) Consequently, $t_{T_0} = t_{T_m} + t_{focus}$, where t_{focus} is the time needed to perform the line focusing algorithm on the N_m lines. Since the line focusing computation takes place at resolution 0 (i.e., in E_0) we estimate $t_{focus} = t_{ht(0)} N_m \bar{p}_0$. Here, we neglect the *residual* edge points that satisfy the line focusing condition of Eq. (5.2.15), but do not belong to the line h_0.

Moreover, due to subsampling by the decimation factor $M = 2^{-m}$, we have $\bar{p}_0 = M\bar{p}_m$. Similarly, $t_{ht(0)} = Mt_{ht(m)}$. Under these assumptions, it can be shown that

$$\frac{M^2 \log_2 K}{M^2 + \log_2 K} \leq R_{CA} \leq \frac{M^2 K}{M^2 + K}, \tag{6.7}$$

where $K = (N_m + 1)/2$.

The expression for the lower bound of R_{CA} is based on the assumption that the number of nonleaf nodes N_m of a complete binary tree is much larger than the depth of the tree d_m. (This is a reasonable assumption for complex, real-life images.)

It is important to note that the lower bound of R_{CA} is *always* larger than 1, since M^2 and $\log_2 [(N_m + 1)/2]$ are always larger than 2 and 1, respectively. Therefore, using the multiresolution approach will always provide a computational advantage versus the direct application of the HT-based method.

For example, if $N_m = 255$ (the numbers of partitioning lines in the decimated images shown in Fig. 17 are of the same order of magnitude), then the lower bound of R_{CA} would be 2.6 and 4.9 when the decimation factor M takes on the values 2 and 4, respectively. For these same values of M and N_m, the upper bound of R_{CA} would be 3.9 and 14.2.

From Eq. (6.7), it is easy to show that both the lower and the upper bounds of R_{CA} increase when the number of partitioning lines increases. Therefore, the more complex the image, the more computational advantage one can gain from the multiresolution approach. The same is true (i.e., more computational advantage) for larger values of the decimation factor M.

Moreover, the upper bound of R_{CA} can be estimated by M^2 when $(N_m + 1)/2 \gg M^2$ which is the case in most images. In practice, the topology of the BSP tree of a typical image (as the one shown in this paper) is between the two extreme cases of complete and asymmetric trees. Therefore, for the average case the value of R_{CA} is of the order of M^2.

It is important to remember that (Eq. (6.7) quantifies the computational advantage one can gain for applying the multiresolution approach at *only one* level of the pyramid. Therefore, more reduction in the computational expense can be achieved when this multiresolution method is used at the different levels of the hierarchy.

Finally, it is crucial to note that the main objectives of the computational complexity expressions derived in this section are (1) to quantify (in terms of order of magnitudes) the different computational aspects of the HT-based and multiresolution BSP tree generation methods, and (2) to show that the multiresolution approach provides a clear reduction in the computational expense of the HT-based method.

7. BSP TREES FOR IMAGE COMPRESSION

Image compression has taken a new direction by designing more representative messages of typical image sources. Among the promising techniques, let us mention pyramidal coding [6] and contour-texture coding [21, 22]. Within the contour-texture approach, segmentation-based techniques have been given a great deal of attention. Images are described as a set of regions with their own luminance model (often of polynomial type). Limits were reached due to the high cost in describing region boundaries accurately. Even with a very limited number of regions, the boundary information represents 75% of the total cost. With more rigid partitioning of the images, such as a quadtree-based representation, the tree structure is simple to encode at the expense of a large number of regions (cells).

BSP trees have the advantage of providing a more compact representation of images in terms of number of polygons (of the order of 150 for images of Figs. 11 to 13, and 600 for the image in Fig. 14), with a simple and diversified description of the region shape.

The simple data structure of BSP trees makes the decoding of images represented by BSP trees particularly simple. Encoding an image using its BSP tree representation requires the description of the tree structure, the partitioning line equations assigned to each internal (nonleaf) node of the tree, and a luminance model for each convex polygonal domain defined by a leaf in the tree. Here, we use the average value of the pixel intensities within each unpartitioned polygon (cell) as the luminance model.

In what follows, we assume that the BSP tree contains M internal nodes (including the root) and N leaves. The tree structure can be encoded using 1 bit per node, i.e., a total of $M + N$ bits. The line equations are parameterized by the two parameters θ and ρ. Without any entropy consideration, the encoding can be optimized by noticing that each line is represented by two points. For each node in the tree, these points belong to the polygon that this node describes. This will however make the decoding of the entire BSP representation rather tedious. For simplicity, we choose to represent every line with two points on separate image borders.

If the image size is $2^k \times 2^l$, the first point requires $\max(k, l) + 2$ bits. The other point can appear at any location on one of the three remaining borders of the image. Its location can be encoded with less than max $(k, l) + 2$ bits as well. In our example, this corresponds to a cost of at most 20 bits per line. As mentioned before, each unpartitioned polygon was assigned the mean value of the pixels it contains. Five bits were sufficient to encode this mean value. Hence, the overall cost to reconstruct any image using a BSP encoding strategy is bounded by

$$C = 21M + 6N.$$

Using this strategy, the images of Figs. 12 and 14a were coded using 0.055 and 0.25 bit per pixel, respectively. This represents a compression ratio of about 150 (Fig. 12) and 40 (Fig. 14a) to 1. These results are preliminary and more work is underway to improve the quality and efficiency of the BSP tree representation of images.

One way to achieve higher compression ratios, for example, is to take into account the correlation between a given line, which is associated with a nonleaf node μ, and all other lines which are represented by the ancestors of μ. The compression numbers given above neglect this correlation model. In addition, one can generate a more accurate representation of the cells by using higher-order polynomials for the pixels within the unpartitioned polygons. The results shown in this work are based on a zero-order model (the mean value). In a future work, new BSP tree-based compression results will be presented using (1) the correlation among the partitioning lines, and (2) higher polynomial models for the pixel intensities within the unpartitioned polygons of the image.

8. CONCLUSION

In this paper we have introduced a Hough transform-based method for generating a BSP tree representation of an arbitrary image. A recursive algorithm (that implements the method) was successful in partitioning a complex image consisting of several objects (of different sizes and shapes), and in generating efficient segmentation using BSP trees.

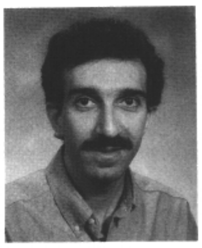
Due to the good segmentation performance of BSP tree representations of natural images, and the BSP tree efficient data structure, it is expected that a great variety of applications ranging from computer graphics to image processing and computer vision would take advantage of this approach. In particular, we have shown the potential of this representation for high-compression image coding. For computer graphics purposes, we are considering this approach for manipulation of objects found in natural scenes. Moreover, image understanding or feature point classification could benefit from this representation.

Finally, we showed a multiresolution method for generating the BSP tree from several scale-space images derived from an original image. This hierarchical approach included the introduction of a *line focusing* algorithm used to combine the trees representing the different scale-space images. As described above, the multiresolution approach provided a significant improvement in speed and segmentation performance.

REFERENCES

1. I. E. Abdou and W. K. Pratt, Quantitative design and evaluation of enhancement/thresholding edge detectors, *Proc. IEEE,* **67,** 5, May 1979.
2. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms,* Addison–Wesley, Reading, MA, 1983.
3. J. Bentley, Multidimensional binary search trees used for associative searching, *Comm. ACM* **18,** 9, Sept. 1989.
4. F. Bergholm, Edge focusing, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-9,** Nov. 1987, 726–741.
5. C. M. Brown, Inherent bias and noise in the Hough transform, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-5,** 1983, 87–116.
6. P. J. Burt and E. H. Adelson, The Laplacian pyramid as a compact image code, *IEEE Trans. Comm.* **COM-31,** Apr. 1983, 532–540.
7. J. F. Canny, *Finding Edges and Lines in Images,* Tech. Rep. 720, Artificial Intell. Lab., MIT, 1983.
8. J. F. Canny, A computational approach to edge detection, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-8,** Nov. 1986, 679–698.
9. S. Carlsson, Sketch based coding of grey level images, *Signal Process.* **15,** 1, July 1988, 57–83.
10. H. H. Chen and T. S. Huang, A survey of construction and manipulation of octrees, *Comput. Vision Graphics Image Process.* **43,** 1988, 409–431.
11. N. Chin and S. Feiner, Near real-time shadow generation using BSP trees, *Comput. Graphics* **23,** 3, Aug. 1989.
12. R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing,* Prentice–Hall, New York, 1983.
13. R. O. Duda and P. E. Hart, Use of the Hough transformation to detect lines and curves in pictures, *CACM* **15,** 11–15, 1972, 11–15.
14. H. Fuchs, Z. Kedem, and B. Naylor, On visible surface generation by a priori tree structures, *Comput. Graphics* **14,** 3, June 1980.

15. D. Fussell and A. T. Campbell, Adaptive mesh generation for global diffuse illumination, *Comput. Graphics* **24,** 4, Aug. 1990.
16. P. V. C. Hough, *Method and Means for Recognizing Complex Patterns,* U.S. Patent 306954, 1962.
17. D. H. Hubel, *Eye, Brain, and Vision,* Scientific American Library, New York, 1988.
18. G. M. Hunter and K. Steiglitz, Operations on images using quadtrees, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-1,** 1979, 145–153.
19. J. Illingworth and J. Kittler, A survey of the Hough transform, *Comput. Vision Graphics Image Process.* **44,** 87–116, 1988.
20. A. Jacquin, Image coding based on a fractal theory of iterated contractive Markov operators. Part 1. Theoretical foundations, and Part 2. Construction of fractal codes for digital images, submitted for publication.
21. M. Kunt, A. Ikonomopoulos, and M. Kocher, Second generation image coding techniques, *Proc. IEEE* **73,** Apr. 1985, 549–574.
22. M. Kunt, M. Benard, and R. Leonardi, Recent results in high-compression image coding, *IEEE Trans. Circuits and Systems,* **CAS-34,** 11, Nov. 1987, 1306–1336.
23. R. Leonardi and M. Kunt, Adaptive split-and-merge for image analysis and coding, *Proc. SPIE* **594** (Image Coding), 1985.
24. H. Li and M. A. Lavin, Fast Hough transform: A hierarchical approach, *Comput. Vision Graphics Image Process.* **36,** 1986, 139–161.
25. S. G. Mallat, A theory for multiresolution signal decomposition: The wavelet representation, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-11,** July 1989, 674–693.
26. M. Mantyla, *An Introduction to Solid Modeling,* Computer Science Press, Rockville, MD, 1988.
27. D. Marr and E. Hildreth, Theory of edge detection, *Proc. Roy. Soc. London B* **207,** 1980, 187–217.
28. E. De Micheli, B. Caprile, P. Ottonello, and V. Torre, Localization and noise in edge detection, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-11,** Oct. 1989, 1106–1117.
29. B. Naylor, *A Priori Based Techniques for Determining Visibility Priority for 3D Scenes,* Ph.D. thesis, University of Texas at Dallas, May 1981.
30. B. Naylor and W. Thibault, *Application of BSP Trees to Ray-Tracing and CSG Evaluation,* TR GIT-ICS 86/03, School of Information and Computer Science, Georgia Institute of Technology, 1986.
31. B. Naylor, SCULPT: An interactive solid modeling tool, in *Graphics Interface '90, Halifax, Nova Scotia, May 1990.*
32. B. Naylor, J. Amanatides, and W. Thibault, Merging BSP trees yields polyhedral set operations, *Comput. Graphics* **24,** 4, Aug. 1990.
33. A. Papoulis, *Probability, Random Variables, and Stochastic Processes,* 2nd ed., McGraw–Hill, New York, 1984.
34. W. K. Pratt, *Digital Image Processing,* Wiley, New York, 1978.
35. J. Princen, H. K. Yuen, J. Illingworth, and J. Kittler, A comparison of Hough transform methods, *Proc. IEE Intern. Conf. on Image Process. and Its Appl.,* July 1989, 73–77.
36. H. M. Radha, A class of robust edge-preserving filters for image restoration, *Proc. IEE Intern. Conf. on Image Process. and Its Appl.,* July 1989, 73–77.
37. H. Radha, R. Leonardi, B. Naylor, and M. Vetterli, Image representation using binary space partitioning (BSP) trees, *Visual Communications and Image Processing V Proceedings,* October 1990, Vol. 1360, Part One, pp. 639–650.
38. H. Radha, R. Leonardi, and M. Vetterli, A multiresolution approach to binary tree representations of images, *ICASSP Proceedings,* May 1991.
39. A. Rosenfeld, Quadtrees and pyramids for pattern recognition and image processing, *5th Internat. Conf. Pattern Recognition, Miami Beach, FL, Dec. 1980.*
40. M. Shah, A. Sood, and R. Jain, *Pulse and Staircase Models for Detecting Edges at Multiple Resolution,* IEEE Publication 0-8186-0685-1/85/0000/0084, pp. 84–95, 1985.
41. I. D. Svalbe, Natural representations for straight lines and the Hough transform on discrete arrays, *IEEE Trans. Pattern Anal. Mach. Intell.,* Sept. 1989.
42. A. M. Tenenbaum, Y. Langsam, and M. J. Augenstein, *Data Structures Using C,* Prentice–Hall, New York, 1990.
43. W. Thibault and B. Naylor, Set operations on polyhedra using binary space partitioning trees, *SIGGRAPH,* July 1987, 153–162.
44. V. Torre and T. Poggio, On edge detection, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-8,** Feb. 1986, 147–163.
45. T. M. van Veen and F. C. A. Groen, Discretization errors in the Hough transform, *Pattern Recognition* **14,** 1981.
46. M. Vetterli and K. M. Uz, Multiresolution techniques for digital television, submitted for publication.
47. Y. Wang, R. Leonardi, and S. K. Mitra, The connectivity Hough transform and its fast implementation, *ICIP Proc.,* Sept. 1989, 548–552.
48. R. Wilson, H. E. Knutsson, and G. H. Granlund, Anisotropic nonstationary image estimation and its application. Part I. Predictive Image Coding, *IEEE Trans. Comm.* **COM-31,** Mar. 1983, 398–406.
49. A. P. Witkin, Scale-space filtering, *International Joint Conference on Artificial Intelligence, Karlsruhe, 1983,* pp. 1019–1021.



HAYDER RADHA is a member of the Technical Staff at AT&T Bell Laboratories, Holmdel, New Jersey. He joined Bell Labs in 1986, and is currently working in the Pixel Machines Development Department. He received an honorary bachelor of science in electrical engineering from Michigan State University and a master of science from Purdue University's School of Electrical Engineering, West Lafayette, Indiana. During the academic year of 1985–1986 he was a research assistant at Purdue University. Since 1986, he has been pursuing his doctorate in the Center for Telecommunications Research and Department of Electrical Engineering at Columbia University, New York, New York. He is a member of the IEEE, SPIE, National Engineering Honor Society (Tau Beta Pi), and Electrical Engineering Honor Society (Eta Kappa Nu). His current research interests are image representation using computer vision and computer graphics techniques, hierarchical image coding, segmentation-based optimum image approximation, and parallel algorithms and architectures for image processing applications.

RICCARDO LEONARDI received his diploma and Ph.D. in electrical engineering from the Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland, in 1984 and 1987, respectively. In 1981, he spent one year at Carnegie–Mellon University, and in 1988, he was a post-doctoral fellow at the University of California, Santa Barbara. Since 1988, he has been performing research at AT&T Bell Laboratories in the field of visual communications. In 1991, he was appointed to coordinate the research and teaching activities of the Signal Processing Laboratory of EPFL. Dr. Leonardi has taught courses in the fields of digital signal processing, image processing, and video communications. He is a Contract Professor at Genova University. He has several interactions with Princeton University and Columbia University, where he supervised Ph.D. theses. His research interests include digital techniques for coding and transmission of video signals and images. He has done specific research for the Motion Picture Expert Group of the ISO. He has also been active in studying data structures for image representation; he is an expert on image segmentation techniques, using motion, texture, or luminance information. He is involved in the coordination and supervision of several European Economic Community projects for video compression. He is author and coauthor of several scientific publications. He is a member of EURASIP and IEEE.

BRUCE NAYLOR has been a member of the technical staff in research at AT&T Bell Laboratories in Murray Hill, New Jersey, since 1986. He previously was on the faculty of the School of Information and Computer Sciences at the Georgia Institute of Technology in Atlanta, Georgia. He received his B.A. in philosophy from the University of Texas at Austin in 1975, and his master's and Ph.D. in computer science from the University of Texas at Dallas in 1979 and 1981, respectively. Dr. Naylor's research interests include geometric modeling, computer graphics, interactive geometric design, computational geometry, and computer architectures for graphics/geometric computation.

MARTIN VETTERLI was born in Switzerland in 1957. He recieved the Dipl. El.-Ing. degree from the Eidgenössische Technische Hochschule Zürich, Switzerland, in 1981, the Master of Science degree from Stanford University, Stanford, California, in 1982, and the Doctorat ès Science degree from the Ecole Polytechnique Fédérale de Lausanne, Switzerland, in 1986. In 1982, he was a research assistant at Stanford University, and from 1983 to 1986 he was a researcher at the Ecole Polytechnique. He has worked for Siemens and AT&T Bell Laboratories. In 1986, he joined Columbia University in New York, where he is currently an associate professor of electrical engineering, a member of the Center for Telecommunications Research, and codirector of the Image and Advanced Television Laboratory. He is a senior member of the IEEE, a member of SIAM and ACM, a member of the MDSP committee of the IEEE Signal Processing Society, and on the editorial boards of *Signal Processing* and *Image Communication*. He served as European Liaison for ICASSP-88 in New York. He was recipient of the Best Paper Award of EURASIP in 1984 for his paper on multidimensional subband coding, and of the Research Prize of the Brown Bovery Corporation (Switzerland) in 1986 for his thesis. His research interests include multirate signal processing, wavelets, computational complexity, algorithm design for VLSI, signal processing for telecommunications, and digital video processing.