Image Representation Using Binary Space Partitioning (BSP) Trees

Hayder M. Radha ***
*Riccardo Leonardi**
*Bruce Naylor**
*Martin Vetterli***

** AT&T Bell Laboratories*
2L313 Crawfords Corner Rd, Holmdel, New Jersey 07733
(201) 949-2799

*** Center for Telecom. Research and Dept. of Elec. Eng.*
Columbia University
New York, New York 10027
(212) 854-3109

1. Introduction

Representation of two and three-dimensional objects by tree structures has been used extensively in solid modeling, computer graphics, computer vision and image processing. (See for example [Mantyla] [Chen] [Hunter] [Rosenfeld] [Leonardi].) Quadtrees, which are used to represent objects in 2-D space, and octrees, which are the extension of quadtrees in 3-D space, have been studied thoroughly for applications in graphics and image processing. 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 [Mantyla]). For example, BSP trees provide an effective tool in determining visible surfaces for polygon rendering and ray tracing applications [Naylor]. To our knowledge, no work has yet been reported on using the BSP tree representation for image processing or computer vision applications.

In this work we present a Hough transform-based method for generating the BSP tree representations of images in 2-D space. It is expected that BSP tree-based segmentation of images will provide an effective tool for achieving (1) high compression rates for image coding applications, (2) ideal representation for natural object manipulation (set operation, affine transformation) in computer graphics applications, and (3) compact representation of objects for computer vision applications.

In the next sub-section, we explain the BSP tree representations of polyhedra. The rest of the paper is organized as follows: Section 2 describes a hierarchical recursive algorithm for building a BSP tree on an arbitrary image. Section 3 discusses some simulation results and gives a rough estimate of the efficiency of this representation for image compression. Finally, Section 4 summarizes the main results of this work.

1.1 The BSP Tree Representation

A BSP tree can be formed in 2D by using lines to recursively partition the 2D space. Figure 1a shows a BSP tree induced partitioning of the plane and 1b shows the corresponding binary tree. The root node represents the entire plane. A binary partitioning of the plane is formed by the line labeled u, resulting in a negative halfspace and a positive halfspace. These two halfspaces are represented respectively by the left and right children of the root. A binary partitioning of each of these two halfspaces may then be performed, as in the figure, 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.

For any node of the tree, the corresponding region is defined by the intersection of the set of open halfplanes determined by each line associated with a node on the path to that region. Since a halfplane is a convex set, and the intersection of convex sets yields a convex set, each region of the tree is convex. The result then is a hierarchical decomposition of 2-space into a binary tree of convex regions, represented combinatorially by a graph forming a binary tree. This scheme just outlined can be easily extended to arbitrary dimensions if we simple use the concept of hyperplanes as the generalization of lines by which to partition a D-dimensional space.

The primary use of BSP trees to date has been to represent polytopes, i.e. polygons in 2D and polyhedra in 3D. This is accomplished be simply associating with each cell of the tree a single boolean attribute *classification* ::= { *interior*,*exterior* }. If, in figure 1, we assign to cells 1 and 5 the value *interior*, and to the rest *exterior*, we will have determined a concave polygon of six sides. This method, while conceptually very simply, is capable of representing the entire domain of polytopes. Moreover, the algorithms that use the BSP tree

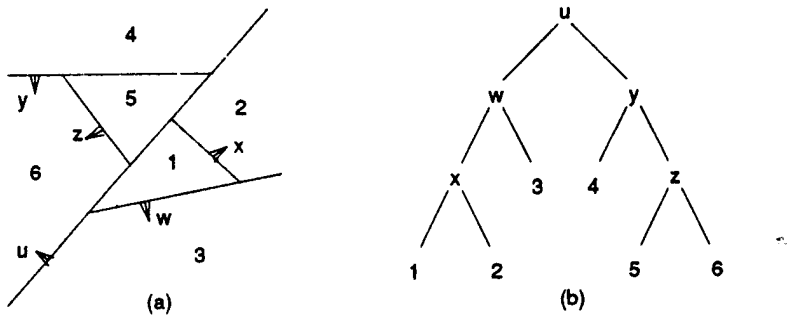


Figure 1. Partitioning of a 2D BSP Tree (a), and its binary tree (b).

representation are simple and uniform, i.e the algorithms for dealing with a triangle are identical to those dealing with a collection of arbitrary polytopes.

As an introduction to BSP tree algorithms, we show how to solve one of the simplest of problems, classifying a point with respect to a polygon. An arbitrary point can be classified with respect to a BSP tree by performing a binary search on the tree [Naylor 87]. Location of a point P with respect to a hyperplane H is given by the dot product :

```
CASE P*H
        < 0 -> InNegativeHalfspace
        > 0 -> InPositiveHalfspace
        = 0 -> OnHyperplane
```

By successively determining in which of the two children the point lies, one will arrive at the cell containing the point. If the point lies on a hyperplane, both subtrees are searched. A point is on the boundary of the set if and only if its epsilon neighborhood is found to include both interior and exterior cells. Thus, the boundary of the set lies on the hyperplanes. It is often the case that the boundary is represented explicitly, in 2D by line segments and in 3D by convex polygons, where each segment/polygon is stored at the internal node whose region contains them. Many BSP tree algorithms are known, including affine transformations, set operations, visualization/rendering, and metric properties (see e.g. [Naylor 90b]).

BSP trees partition space into convex sub-domains, i.e the cells, so that one can define a separate continuous function over each sub-domain. Thus, one may construct a discontinuous, but piecewise-continuous, function over the 2D-space. Note however, that this does not preclude the function being chosen to be continuous on any subset of a hyperplane. Such piecewise continuous functional representation is suitable for images [Kunt 85].

2. A Hough Transform-Based Method for BSP tree Generation

As explained above, the BSP tree approach partitions the space (that surrounds the desired object to be presented) by hyperplanes passing through the boundary of the object. This is an easy task when the shape, size, and other attributes of the object are known. On the other hand, partitioning an image that consists of several objects of unknown shapes and sizes would 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 binary partition on the image contour information. Two steps will be needed for generating such a BSP tree representation (see Figure 2): (1) extract the boundary locations of each object in the image. (2) determine the linear characteristics of these boundaries, in order to match them with a minimum set of straight lines.

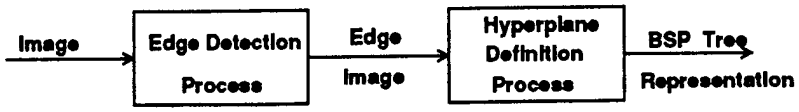


Figure 2

Each hyperplane which is associated with (1) a non-leaf node of the BSP tree, and (2) a convex region (sub-space) of the original image should fit a maximum number of boundary points. The partitioning hyperplane of a given sub-space has to match the largest set of collinear edge points in that sub-space. In addition, it is desirable to include a connectivity weighting of these edge points [Wang]. We emphasize on

the notion of connectivity since the human visual system is particularly sensitive to linear segments of connected points [Hubel]. Edge points that lie on a hyperplane and yet do not belong to one connected edge will be referred to "uncorrelated" collinear edges.

As a consequence, the hyperplanes associated with high level nodes (close to the root) in the BSP tree should 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.

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, hyperplanes are built on collinear edge segments.

The Hough transform is a very useful tool to extract hyperplanes and other geometrical features in edge images [Hough, Duda]. In the sequel, as we focus on images in 2-dimensional space, we shall use the words *hyperplane* and *line* interchangeably. The concepts presented below can be easily extended to spaces of higher dimensions.

Our BSP tree generation method starts by applying the Hough transform (HT) on all edge points in the image, and selects the cell with the maximum number of votes (corresponding to the hyperplane that passes through the maximum number of edge points) to partition the image. Then, the HT of one of the two regions (half spaces) resulting from the division is computed. Again, a search for the cell with maximum number of votes is performed. Now, the hyperplane corresponding to this cell will partition the selected half space. The same thing is done to the other half space. This process is repeated till a termination criterion is reached. This criterion could be one or both of the followings: (1) The area of the region to be partitioned is smaller than a certain threshold (2) the number of boundary points within the region is lower than some other threshold. These thresholds will determine the degree of accuracy of the representation. The lower the threshold values, the largest the number of nodes in the tree and the more accurate the image description.

It is important to note that a hyperplane 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 sub-regions of R_μ, and are associated with the right and left children of μ, respectively. Sh_μ represents a sub-hyperplane (of h_μ) that lies in R_μ. After selecting h_μ as the partitioning hyperplane of R_μ, it is crucial to eliminate all edge points $\in Sh_\mu$ (i.e., that lie on the hyperplane) and process just those points that belong to the half-spaces 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 node in the tree, (3) select each hyperplane as the peak of the corresponding Hough transform. The sub-sections below describe in detail each one of these steps.

2.1 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 (i.e., the set of straight lines used to partition the original image) is 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 [Torre] [Micheli]:

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

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

where L_2 is the set of square integrable functions. The Gaussian function satisfies the above conditions, and therefore is used as a pre-differentiation 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., [Shah] [Torre] [Bergholm]). An edge focusing algorithm was introduced by [Bergholm] to achieve both the filtering of high frequency noise and good localization of the original edges. In this work, we limited ourselves to a Gaussian function with *small* σ as a pre-differentiation filter.

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) [Marr] [Canny 86]. Although both methods are equivalent, it has been shown that the maximum gradient approach is less sensitive to noise [Micheli].

2.1.1 The Gradient Edge Detector

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.

We have selected the Sobel operator for computing the gradient image since it performs better than other numerical differentiation operators (e.g., Roberts) based on Abdou and Pratt's figure of merit [Abdou] [Pratt].

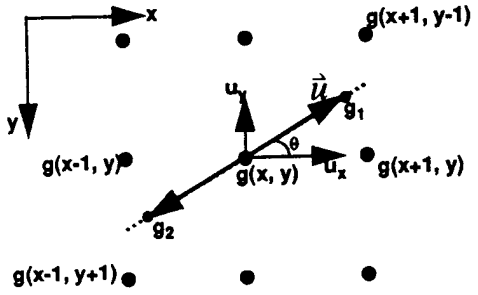
To locate edges from the gradient magnitude image $g_m(x, y)$, one can estimate the directional derivate $\frac{d}{d\vec{g}} g_m(x, y)$ at every point (x, y):

$$\frac{d}{d\vec{g}} g_m(x, y) = \cos\theta_g \frac{\partial}{\partial x} g_m(x, y) + \sin\theta_g \frac{\partial}{\partial y} g_m(x, y)$$

where θ_g is the direction of the gradient vector $\vec{g}$ at (x, y). By making the substitutions $\cos\theta_g = g_x / g_m(x, y)$ and $\sin\theta_g = g_y / g_m(x, y)$ in the above equation and equating the result to zero, one can show that the maximum gradient points of $g_m(x, y)$ in the direction of $\vec{g}$ satisfy the following equation:

$$\frac{d}{d\vec{g}} g_m(x, y) = g_x^2 g_{xx} + 2 g_x g_y g_{xy} + g_y^2 g_{yy} = 0$$

For the implementation of the above equation, we have used a non-maximum gradient suppression algorithm proposed by Canny [Canny 83]. In this algorithm, a gradient magnitude $g_m(x, y)$ at position $\vec{p} = (x, y)$ is marked as a maximum if its value is larger than the gradient magnitudes g_1 and g_2 at $\vec{p}_1 = \vec{p} + \vec{u}$ and $\vec{p}_2 = \vec{p} - \vec{u}$, respectively, where $\vec{u}$ is a unit vector in the direction of the gradient $\vec{g}$. If $\vec{p}_1$ (or $\vec{p}_2$) does not lie on one of the discrete points of the image 2-dimensional grid, then g_m at $\vec{p}_1$ and $\vec{p}_2$ will have to be interpolated from the nine-pixel neighborhood surrounding $\vec{p}$. An example of this algorithm is shown in Figure 3. The interpolation is performed by weighting the gradient magnitudes at the two nearest neighbors.



$$g_1 = \frac{u_y}{u_x} g(x+1, y-1) + \frac{u_x - u_y}{u_x} g(x+1, y)$$

$$g_2 = \frac{u_y}{u_x} g(x-1, y+1) + \frac{u_x - u_y}{u_x} g(x-1, y)$$

Figure 3

To eliminate irrelevant maxima in the gradient magnitude image, a thresholding strategy with hysteresis is chosen. We derive a low threshold T_l from the histogram of the maxima in the gradient image. T_l is selected to be 80% of the cumulative histogram. Then, a high threshold $T_h = \beta T_l$ is set, where $2 \leq \beta \leq 3$. We use both T_l and T_h to perform thresholding with hysteresis on the maxima gradient points as explained in [Canny 86]. Our implementation of this method is as follows:

1. All maxima gradient points with gradient magnitudes above T_h are marked as *strong* edges.
2. Any point (in the maxima gradient image) that is (a) connected to a strong edge, and (b) higher than T_l, is marked as a strong edge. Points that are higher than T_l but not connected to a strong edge are marked as *low* edge points.

The low edge points are removed from the final edge image.

2.2 The Hough Transform

The standard Hough transform (HT) maps each edge point (x_e, y_e) into a curve in the line parameter 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) parametrization to represent straight lines in the Hough space. θ is the direction (with respect to the x-axis) of the vector $\vec{n}$ normal to the line (θ, ρ), and ρ is the distance between the line and the origin [Duda]. With this parametrization, the line equation is given by:

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

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

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

where $E(x, y)$ is a 2-dimensional 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 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 degrees. 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. 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 (Figure 4). The detection of straight lines in the edge image can be accomplished by searching for (θ, ρ) cells that posses large number of votes (*peaks*) in the parameter space. The HT literature is plentiful. A comprehensive survey can be found in [Illingworth].

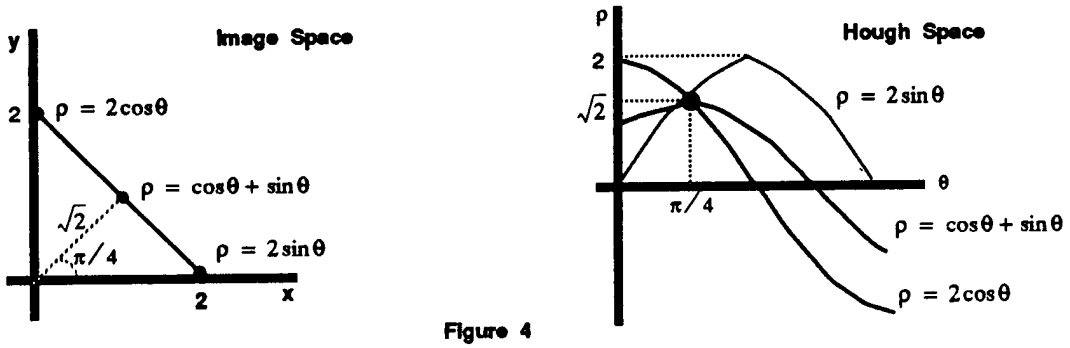


Figure 4

Improvements have been suggested to this simple approach as the standard HT is sensitive to background noise and texture in the image [Wang] [Brown]. Here are some possible solutions.

2.2.1 Using Gradient Direction Information

It is shown in this sub-section that the use of gradient direction information significantly reduces the range of θ values need to be considered when computing the Hough transform.

If an edge point (x_e, y_e) belongs to a connected edge which lies on the straight line (θ, ρ) in the image space, then it is easy to see that the gradient direction θ_g at (x_e, y_e) is the same as the angle θ of the line (θ, ρ). Here we assume that (x_e, y_e) is not a corner, trihedral, or any crossing point.

Without the use of this apriori information (about the edge point gradients in the image space), it can be shown (under certain assumptions) that it is necessary to compute the HT for *all* θ values within 180 degrees (e.g., from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$). This is true even with apriori knowledge about the probability density function of the gradient vector. To show this, let us assume that the x and y components of the gradient vector $\vec{g}$ are two jointly distributed Gaussian random variables $\mathbf{g_x}$ and $\mathbf{g_y}$, respectively. The marginal probability distribution functions are assumed to have zero mean and a standard deviation σ_g. Since the gradient direction angle $\theta_g = arctan(g_y / g_x)$, we define the random variable $\mathbf{z} = \frac{\mathbf{g_y}}{\mathbf{g_x}}$. It can be shown that $\mathbf{z}$ has the following distribution function [Papoulis]:

$$F_z(z) = 0.5 + \frac{1}{\pi} arctan\frac{z - r}{\sqrt{1 - r^2}}$$

where r is the correlation coefficient between $\mathbf{g_x}$ and $\mathbf{g_y}$. The probability density function of the random variable $\Theta = arctan(\theta)$ can be easily obtained from $F_z(z)$:

$$f_\Theta(\theta) = \frac{1}{\pi} \frac{\sqrt{1 - r^2}}{1 - r\sin(2\theta)}$$

It is important to note that $f_\Theta(\theta)$ is independent of the standard deviation σ_g. Here we assume that $\mathbf{g_x}$ and $\mathbf{g_y}$ are uncorrelated (i.e., $r = 0$) which leads to the uniform density $f_\Theta = 1/\pi$. Therefore, without any apriori gradient direction information, and regardless of the amount of dispersion of the random variables $\mathbf{g_x}$ and $\mathbf{g_y}$ around their zero mean, all θ values are *equally* likely under the above assumptions.

Given the gradient direction at (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 radians proposed in [Princen]. (However, the θ range (around $\hat{\theta}_g$) required for computing the HT, can be determined through a probabilistic model. This model will be shown in a future work.)

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 [Brown]. The improvement for incorporating gradient direction information is shown in Section 3.

2.2.2 Weighting the HT Votes

Real life images may contain a large number of *uncorrelated* collinear edge points that do not belong to the same edge (object boundary). An example of this scenario is shown in Figure 5. By using the standard HT, these uncorrelated collinear points will cause the occurrence of false peaks in the Hough space [Wang]. One way to solve this problem is to associate a weight for each vote that an edge point cast (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 that lies on the line (θ, ρ).

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-dimensional non-negative 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:

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

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 $\pi/4 \le \theta \le \pi/2$ and $-\pi/2 \le \theta \le -\pi/4$:

$$x(t) = t + x_e \qquad -d\sin(\theta) \le t \le d\sin(\theta)$$

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

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

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

For other values of θ, the x coordinate is expressed as a function of y. This is done to avoid aliasing when implementing the line integration on a discrete sampling lattice. In such a case, the following equations were used to compute $v_e(\theta, \rho)$:

$$y(t) = t + y_e \qquad -d\cos(\theta) \le t \le d\cos(\theta)$$

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

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

The above integrations were implemented by incrementing t by 1 within the specified ranges. The performance of such technique are explicitly described in Section 3.

2.2.3 Post Filtering the Hough Space

The effect of the parameter space resolution (i.e., $\Delta_\theta = \frac{\pi}{N_\theta}$ and Δ_ρ) on both the accuracy and computational complexity of the HT has been

studied thoroughly in the literature [Veen] [Svalbe]. If large quantization increments are used, there will be a large degree of inaccuracy in the location of straight lines in the image space. On the other hand, by using very small quantization increments Δ_ρ (i.e., oversampling) for a given Δ_θ, a peak for a straight line could be extended over several cells in the ρ direction. We can express the maximum number of HT cells over which a peak can be distributed in the ρ direction as follows [Veen]:

$$n_\rho = \frac{l \sin\left(\frac{\Delta_\theta}{2}\right)}{\Delta_\rho} + 2$$

where l is the line length. Based on this peak distribution phenomenon one may assume that a better result can be achieved by filtering (smoothing) the Hough space (in the ρ direction) with a linear (averaging) filter whose size $S_f = n_\rho$. If the image dimension is S_I, then the maximum length of a line in the image space is $l_{max} = \sqrt{2}S_I$. Therefore, in a worst case scenario, the maximum filter size is:

$$S_f = \frac{\sqrt{2}S_I \sin\left(\frac{\Delta_\theta}{2}\right)}{\Delta_\rho} + 2$$

This approach was tested using different filter sizes, but did not provide any improvement to the standard HT. The main reason for the failure of this method is its dependency on the line size l. For example, a given filter (with size S_f) that is capable of smoothing a distributed peak due to a line L with length l, may cause other peaks corresponding to much smaller lines (than L) to be merged into one major false peak in the Hough space. Moreover, it is difficult to estimate the true length of a line consisting of a set of disconnected segments.

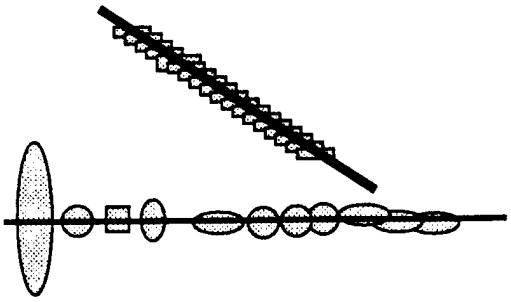


Figure 5

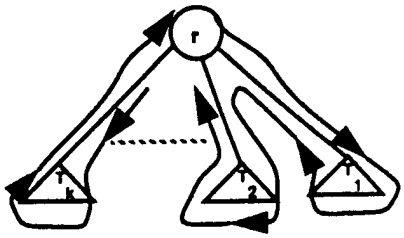


Figure 6

2.3 A Preorder Tree Traversal Algorithm

We outline three methods that are commonly used for traversing tree-like data structures. Let T be a tree with root r and subtrees $T_1, T_2, \ldots, T_k$ as shown in Figure 6 (for binary trees, $k = 2$). The *preorder*, *inorder*, and *postorder* traversal of a tree are as follows [Aho]:

1. The *preorder traversal* or (*listing*) of T is the root r followed by the preorder traversal of the nodes of T_1, then the preorder traversal of the nodes of T_2, and so on, up to the traversal of T_k in a preorder manner.
2. The *inorder traversal* or (*listing*) of T is the inorder traversal of the nodes of T_1 followed by the root r, then the inorder traversal of the nodes of $T_2, \ldots, T_k$, each subtree inorder.
3. The *postorder traversal* or (*listing*) of T is the postorder traversal of the nodes of T_1 followed by the inorder traversal of the nodes of T_2, and so on, up to the traversal of T_k in postorder all followed by the root r.

A useful way to visualize these 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. For the preorder (postorder) case we list a node the first (last) time we pass it. For inorder traversing, we list a leaf the first time we pass it, but list an intermediate (interior) node the second time we pass it, where the root node is considered as an interior node.

It is clear that the preorder and postorder traversings follow top-bottom and bottom-up approachs, respectively. In our case we are attempting to generate a tree, by first searching for the "best" hyperplane that can be used to partition the edge image into two halfspaces, and associating this hyperplane with the root node of our BSP tree. Since the same process (searching for the "best" hyperplane) will be repeated on one of the two halfspaces after generating the root node, it is clear that we are attempting to generate our tree in a *preorder* manner.

3. Simulation Results and Compression Aspects

3.1 BSP Segmentation Performance

The algorithm described in the previous section was implemented in C, and tested on a 256x256 original image with 256 gray levels (Figure 7). The image was selected 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 are particularly noticeable.

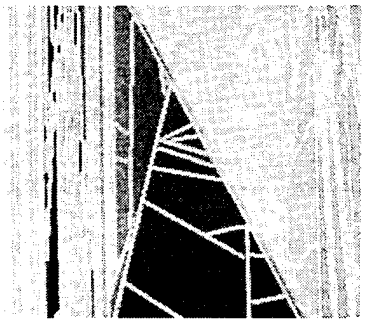
We first applied the algorithm on the edge image of Figure 8a. The edge image was obtained (without any filtering) using a non-linear edge detector proposed in [Radha] in combination with a thinning process. In this case we computed the HT using the standard method, i.e. considering all θ values between zero and 180 degrees. Therefore, under this scenario we did not use any gradient 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.

Original Image

Figure 7

Edge Image

Figure 8a



BSP Labelled Image
using standard Hough transform

Figure 8b

It is clear that this BSP tree does not represent the original image (except for the face region) due to 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 maximum gradient edge detector described in Section 2.1. These edge images were derived from a Gaussian-filtered version of the original image. Figure 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 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 Figure 9c as an input to our BSP tree algorithm.

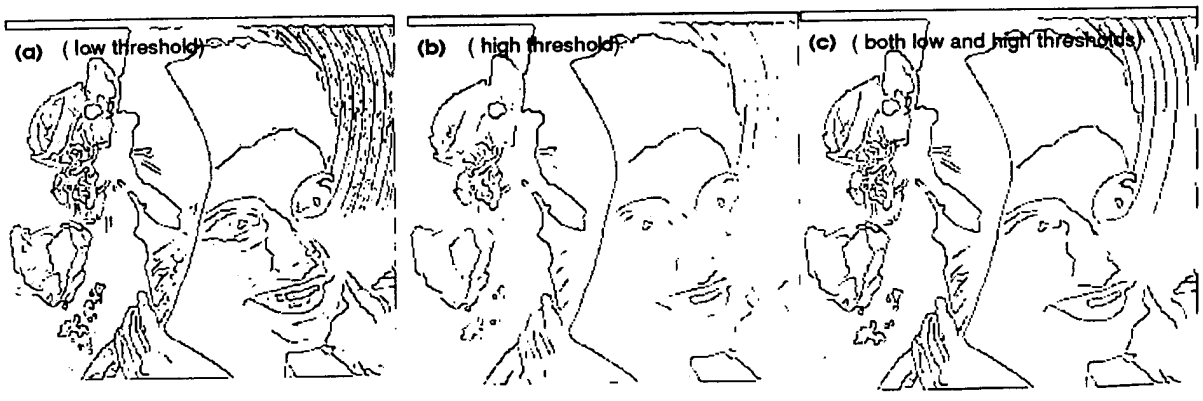
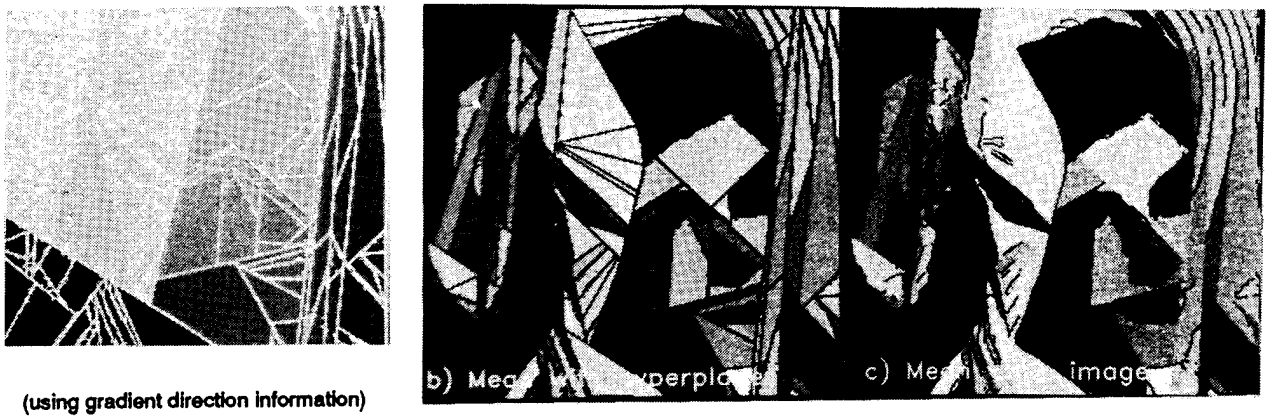


Figure 9

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 BSP tree representation of the original image) of Figure 10a illustrates a significant improvement when compared with the image of Figure 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 hyperplanes used to partition the image.



(using gradient direction information)

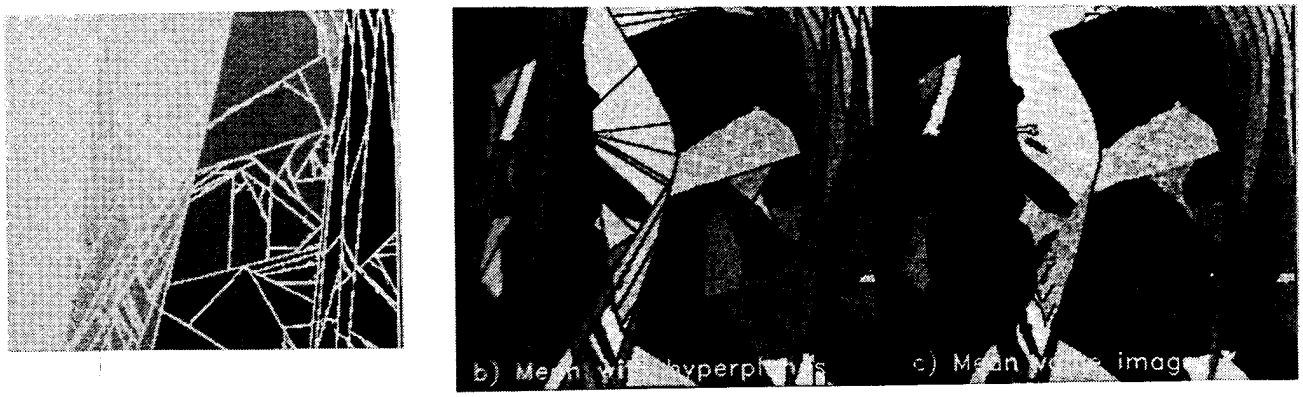
(a) BSP labelled image

(b) Mean value image with hyperplanes

(c) Mean value image

Figure 10.

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



using gradient information and line integration

(a) BSP labelled image

(b) Mean value image with hyperplanes

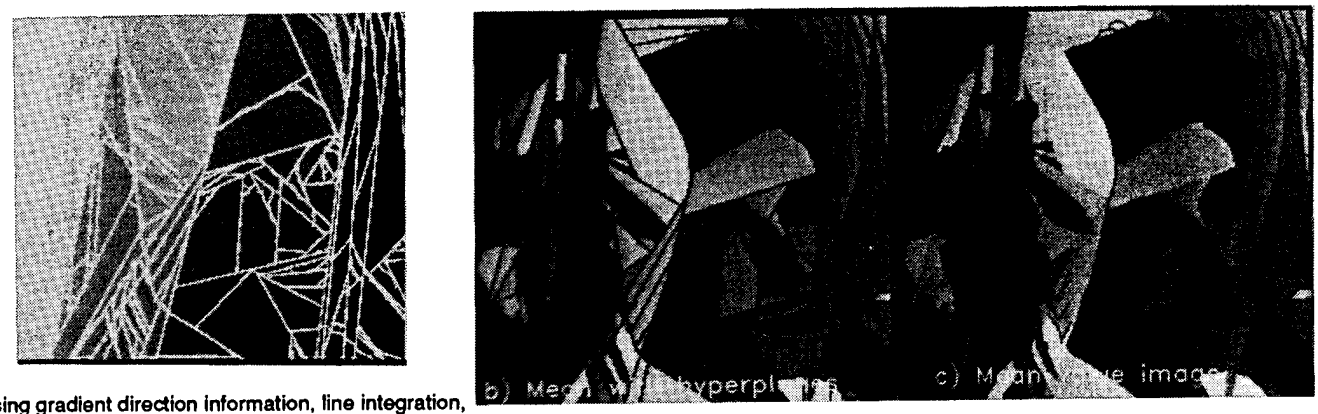
(c) Mean value image

Figure 11.

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 hyperplane (θ_p, ρ_p) associated with a parent node μ_p has to be a minimum distance (in the Hough space) from its child hyperplane (θ_c, ρ_c). This minimum distance requirement can be expressed as follows:

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

where θ_d and ρ_d are the desired minimum distances in the θ and ρ directions, respectively. The results of using this strategy is shown in Figure 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).



(using gradient direction information, line integration, and parent-child minimum distance strategy)
(a) BSP labelled image

(b) Mean value image with hyperplanes

(c) Mean value image

Figure 12.

Figure 13 shows mean value images resulting from using this strategy for the two cases (1) $\theta_d > \rho_d$ (Figures 13a and b), 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 figure 12, we set $\theta_d=\rho_d=10$, for figures 13a and 13b, $\theta_d=2\rho_d=20$, and for figures 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 Figure 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 Figure 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.

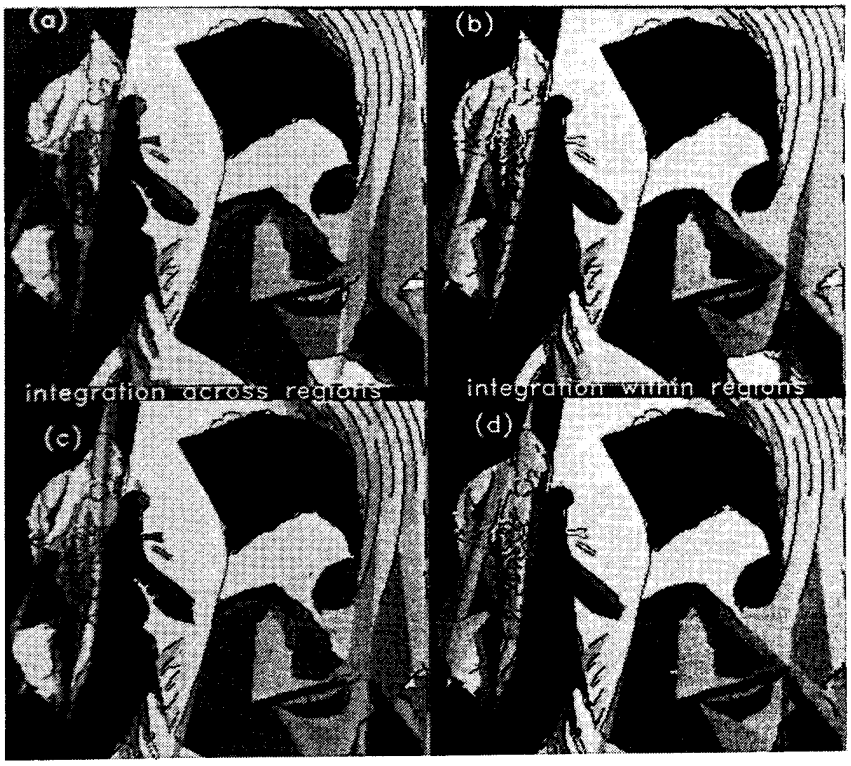


Figure 13.

3.2 BSP Trees for Image Compression

Recently, image compression was given a new life by designing more representative messages of typical image sources. Among the most promising techniques, let us mention pyramidal coding [Burt], anisotropic nonstationary predictive coding [Wilson], sketch based representation of images [Carlsson], contour-texture coding [Kunt 85], [Kunt 87], fractal based coding [Jacquin]. Significant improvements were made possible thanks to models that could handle the non-stationary behavior of images. Within the contour-texture approach, segmentation based techniques have been given a great 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 accurately region boundaries. Even with a very limited number of regions, the boundary information represents 75% of the total cost. With more rigid partition of the images such as quadtree based representation, the tree structure is simple to encode at the expense of a large number of regions (cells). BSP tree have the advantage of providing a compact representation of images in terms of number of polygons (of the order of 150 for images of Figure 11 to 13), 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 M hyperplane equations assigned to each internal node of the tree and a luminance model for each convex polygonal domain defined by a leaf in the tree. In what follows, we assume 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 hyperplane equations are parametrized by the two parameters θ and ρ. Without any entropy consideration, the encoding can be optimized by noticing that each hyperplane 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 hyperplane with 2 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 hyperplane. Given the synthetic quality of the reconstructed images when the number of polygons is kept low, each polygon was assigned the mean value of the pixels it contains. 5 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 image of Figure 12 was coded using 0.055 bit per pixel. This represents a compression ratio of about 150 to 1.

4. Conclusion and Future Work

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 effect, we have shown the potential of this representation for High Compression Image Coding (see Section 3). For Computer Graphics purposes, we are considering this approach for easy object manipulation of natural scenes. Finally, Image Understanding or Feature Point Classification could benefit from this representation.

In a future work, we will show a sub-band based, hierarchical method for generating the BSP tree from several scale-space images derived from an original image. This new approach will include the introduction of a *hyperplane focusing* algorithm used to combine the trees representing the different scale-space images. The theoretical development of the optimum (in the minimum mean square error sense) binary segmentation of 2-dimensional domains is also underway.

References

[Abdou] I. E. Abdou and W.K. Pratt, "Quantitative Design and Evaluation of Enhancement/Thresholding Edge Detectors," *Proc. IEEE*, Vol. 67-5, May 1979.

[Aho] Aho, A. V., Hopcroft, J. E., and Ullman, J. D., *Data Structures and Algorithms*, Addison-Wesley, 1983.

[Bergholm] F. Bergholm, "Edge Focusing," *IEEE trans. Pattern Anal. Machine Intell.*, vol. PAMI-9, pp. 726-741, Nov. 1987.

[Brown] C.M. Brown, "Inherent Bias and Noise in the Hough Transform," *IEEE trans. Pattern Anal. Machine Intell.*, vol. PAMI-5, pp. 87-116, 1983.

[Burt] P.J. Burt and E.H. Adelson, "The Laplacian Pyramid as a Compact Image Code", IEEE Trans. on Commun., vol. COM-31, pp. 532-540, Apr. 1983.

[Canny 83] J. F. Canny, "Finding Edges and Lines in Images," Artificial Intell. Lab., M.I.T., Tech. Report 720, 1983.

[Canny 86] J. F. Canny, "A Computational Approach to Edge Detection," *IEEE trans. Pattern Anal. Machine Intell.*, vol. PAMI-8, pp. 679-698, Nov. 1986.

[Carlsson] S. Carlsson, "Sketch based Coding of Grey Level Images", Signal Processing, Vol. 15, No.1, pp. 57-83, July 1988.

[Chen] H. H. Chen and T. S. Huang, "A Survey of Construction and Manipulation of Octrees", *Computer Vision, Graphics, and Image Processing*, 43, 409-431, 1988.

[Duda] R. O. Duda and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures", *CACM*, 15, 11-15, 1972.

[Hough] P. V. C. Hough, *Method and Means for Recognizing Complex Patterns*. U.S. Patent 3069654 (1962).

[Hubel] D.H. Hubel, "Eye, Brain, and Vision", Scientific American Library, 1988.

[Hunter] G. M. Hunter and K. Steiglitz, "Operations on Images Using Quadtrees", *IEEE Trans PAMI-1*, 1979, 145-153.

[Illingworth] J. Illingworth and J. Kittler, "A Survey of the Hough Transform", *Computer Vision, Graphics, and Image Processing*, 44, 87-116, 1988.

[Jacquin] 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 to IEEE Trans. on Acous., Speech, and Sig. Proc., Nov. 1989.

[Kunt 85] M. Kunt, A. Ikonomopoulos, M. Kocher, "Second Generation Image Coding Techniques", Proc. IEEE, vol. 73, pp. 549-574, Apr. 1985

[Kunt 87] M. Kunt, M. Benard, R. Leonardi, "Recent Results in High-Compression Image Coding", IEEE Trans. on Circ. and Syst., Vol. CAS-34, No. 11, pp. 1306-1336 (Nov. 1987).

[Leonardi] R. Leonardi and M. Kunt, "Adaptive Split-and-Merge for Image Analysis and Coding", *Proceedings of SPIE* Vol. 594 Image Coding, 1985

[Li] H. Li and M. A. Lavin, "Fast Hough Transform: A Hierarchical Approach" , *Computer Vision, Graphics, and Image Processing*, 36, 139-161, 1986.

[Mantyla] M. Mantyla, *An Introduction to Solid Modeling*, Computer Science Press, 1988.

[Marr] D. Marr and E. Hildreth, "Theory of Edge Detection," *Proc. Roy. Soc. London*, Vol. B-207, pp. 187-217, 1980.

[Micheli] E. De Micheli, B. Caprile, P. Ottonello, and V. Torre, "Localization and Noise in Edge Detection," *IEEE trans. Pattern Anal. Machine Intell.*, vol. PAMI-11, pp. 1106-1117, Oct. 1989.

[Naylor] B. Naylor, "A priori Based Techniques for Determining Visibility Priority for 3-D Scenes", Ph.D. Thesis, University of Texas at Dallas, May 1981.

[Naylor 90a] B. Naylor, "SCULPT: An Interactive Solid Modeling Tool", Graphics Interface '90, Halifax, Nova Scotia, May 1990.

[Naylor 90b] B. Naylor, J. Amanatides, and W. Thibault, "Merging BSP Trees Yields Polyhedral Set Operations", Computer Graphics, 24(4), (Siggraph '90), Aug. 1990.

[Papoulis] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill, New York, 2nd Edition, 1984.

[Pratt] W.K. Pratt, *Digital Image Processing*, Wiley, New York, 1978.

[Princen] J. Princen, H. K. Yuen, J. Illingworth, J. Kittler, "A Comparison of Hough Transform Methods," *Proc. IEE Intern. Conf. on Image Proc. and its Applic.*, pp. 73-77, July 1989.

[Radha] H.M. Radha, "A Class of Robust Edge-Preserving Filters for Image Restoration," *Proc. IEE Intern. Conf. on Image Proc. and its Applic.*, pp. 73-77, July 1989.

[Rosenfeld] A. Rosenfeld, "Quadtrees and Pyramids for Pattern Recognition and Image Processing", *5th Intl. Conf. Pattern Recog., Miami Beach, Fl., Dec. 1980.*

[Shah] M. Shah, A. Sood, and R. Jain, "Pulse and Staircase Models for Detecting Edges at Multiple Resolution," *IEEE Publication 0-8186-0685-1185/0000/0084*, pp. 84-95, 1985.

[Svalbe] I. D. Svalbe, "Natural Representations for Straight Lines and the Hough Transform on Discrete Arrays", *IEEE Trans. PAMI*, September 1989.

[Thibault] W. Thibault and B. Naylor, "Set Operations on Polyhedra Using Binary Space Partitioning Trees", SIGGRAPH, 1987.

[Torre] V. Torre and T. Poggio, "On Edge Detection," *IEEE trans. Pattern Anal. Machine Intell.*, vol. PAMI-8, pp. 147-163, Feb. 1986.

[Veen] T. M. van Veen and F. C. A. Groen, "Discretization Errors in the Hough Transform", *Pattern Recognition*, Vol. 14, 1981.

[Wang] Y. Wang, R. Leonardi, and S. K. Mitra, "The Connectivity Hough Transform and its Fast Implementation"

[Wilson] R. Wilson, H.E. Knutsson, and G.H. Granlund, "Anisotropic Nonstationary Image estimation and its Application: Part I - Predictive Image Coding", IEEE Trans. on Comm., Vol. COM-31, pp. 398-406, March 1983.