QUADTREE MESHES

William T. Verts
COINS Department
University of Massachusetts
Amherst, MA 01003

Professor Francis S. Hill, Jr.
ECE Department
University of Massachusetts
Amherst, MA 01003

ABSTRACT

Quadtrees have long been a favorite data structure for reducing the memory storage requirements of bilevel images and for representing those images hierarchically. In general, a quadtree requires far less storage than the corresponding unencoded image. Unfortunately, storage requirements depend critically on the offset of the image within its sampling grid; quadtrees are variant with respect to translation. Reducing the amount of storage required by a quadtree implementation is strongly related to reducing its sensitivity to translation. Techniques that address these issues include Linear Quadtrees, Quadtree Normalization, the Quadtree Medial Axis Transform, and Quadtree Forests. The Translation Invariant Data structure is a related non-Quadtree technique based on medial axis transforms. This paper presents a translation invariant representation that maintains both the hierarchical properties and spatial coherence of each object in an image. Each image object is allocated its own quadtree, then those objects are interconnected with a meshwork (such as a Delaunay Triangulation) based on object centers. The geometry of the meshwork allows each object to be translation independent of all others (they may overlap), and allows the composite image to be of arbitrary size. The properties of this technique are explored and applications to cartography and extensions are discussed.

INTRODUCTION

As reviewed by Samet (Samet; 1984), there are several types of quadtrees with different properties and applications. For this paper we focus on the most widely known type, the *region quadtree* (hereafter referred to simply as a "quadtree").

What is a Quadtree?

Quadtrees are tree data-structures useful for storing bilevel images. Extensions that allow quadtrees to capture gray-scale images will be considered later. An image to be represented by a quadtree is an N by N square array of pixels, where N is a power of two ($N = 2^L$, for some positive L). Each pixel may be either *black* or *white*; black pixels represent part of an image object, white pixels are background. Following convention, we call the two levels black and white, but in general the image element may have any two discrete values.

Each node in a quadtree is either a leaf node or an internal node. Leaf nodes of the quadtree correspond to a region of the image that contains a single color: all pixels are black or all pixels are white. Internal nodes, with pointers to four descendants, represent regions that are a mix of black and white pixels. The region covered by any internal node is the union of the four regions covered by its descendants (a perfect tiling; there are no gaps and no overlap). Other auxiliary information may also be stored in each node, such as a pointer to its ancestor, or some statistical information about the image in its subtree (a *gray* level representing the average color of the region covered by the node, for example).

How is a Quadtree Formed?

Most images contain some mixture of black and white areas. If the image is entirely black or entirely white, the quadtree captures the entire image with a single leaf node. Otherwise, it is divided into quarters, where each quarter corresponds to one subtree of the quadtree root. Each of the four descendants of the root corresponds to a quadrant of the image, called the NorthWest, NorthEast, SouthWest and SouthEast quadrants, respectively. The process is recursively repeated on each quadrant until the subimage contains a single color. See Figure 1.

Attractive Properties of Quadtrees

If the recursive decomposition of the image stops before the pixel level is reached, the corresponding quadtree will typically require less storage than that required by the original (uncompressed) image. Local coherence in the image tends to reduce the number of unique points that are stored.

Quadtrees also encode images hierarchically; if gray information is stored at internal nodes, coarse approximations of an image can be constructed by examining all nodes at a single level in a quadtree; every level covers (tiles) the entire image area. More detail appears as deeper levels of the quadtree are examined. When a branch of the quadtree ends in a leaf the size and color of that leaf are used in all lower levels.

Problems With Quadtrees

Normally, there are several *objects* in the scene captured by the image. The number of nodes required by a quadtree can change drastically by shifting those objects within the image (translation variance). Adding points to and deleting points from a image can also drastically change the number of nodes in the corresponding quadtree.

Image objects have no internal *coherence*, and may be split among several quadrants during the formation of the quadtree.

White areas are explicitly stored in the quadtree. Image objects are fully described by the black nodes of the quadtree; the white nodes are redundant. Techniques to eliminate or reduce the number of white nodes are outlined in the next section, but in general some must be stored.

It is extremely difficult to add points to an image outside of the (2^N by 2^N) sampling grid. If there is space available the image can be shifted over to allow the new points to be added. Shifting will not be possible in all cases; when it is possible the quadtree will be affected by translation variance.

Some images can not be easily compressed. In cases where every pixel must be present in the quadtree (as in a checkerboard) more storage space is required than that by the uncompressed image, due to the overhead of the quadtree structure. In a checkerboard every pixel differs from its neighbors; no spatial optimization can take place.

ALTERNATIVE METHODS

Several methods have been proposed to reduce both storage and translation variance in representing images.

LINEAR QUADTREES

One method used to reduce the overhead associated with quadtrees is to store them as Linear Quadtrees (Gargantini; 1982). A linear quadtree is a pointer-less representation that stores only the black nodes of a quadtree. There is one entry in the linear quadtree for each black leaf in the full quadtree; each entry is a coded path from the root to the leaf. The symbols in the coded path are in a "quaternary" (base 4) code, where symbols

0, 1, 2, and 3 stand for the NW, NE, SW, and SE descendants of a node, respectively. The number of symbols along the coded path corresponds to the depth of the leaf in the quadtree. The linear quadtree for the image in figure 1 is: 00, 010, 012, 020, 021, 03, 10, 12, 203, 21, 30.

Linear quadtrees have some nice properties. They can be transmitted via a text-only link because of the ease in which the quaternary codes can be converted into ASCII characters. They can readily be expanded back into their original images, and it is easy to perform Boolean compositing operations (union, intersection, etc.) between two linear quadtrees.

QUADTREE NORMALIZATION

Quadtree normalization is a technique used to reduce the number of nodes in a quadtree by shifting the image around in the image grid. It has been shown (Li, Grosky, Jain; 1982) that the optimum placement of an N by N pixel image can be computed in time $O(N^2 \operatorname{Log}_2 N)$. Their algorithm takes an image and returns X and Y shift factors to obtain the optimum (minimum) number of quadtree nodes. The optimum placement of the image within the grid may require that the grid size be doubled (quadrupling the grid area).

Normalization is also used in (Chien, Agarwal; 1983) to simplify the recognition of a class of objects (jet planes in their paper) that may be in any size, placement, or orientation on an image grid. By normalizing their list of expected objects with respect to size, principle axes and centroids, they use the quadtree representations as "shape descriptors" to perform pattern matching. Note that the entire image is not converted to a quadtree, only objects within the image. This "decoupling" of objects from their background is an important tool used in the Quadtree Mesh technique described below.

QUADTREE MEDIAL AXIS TRANSFORM

Samet (Samet; 1983 & 1985) describes a technique for transforming one quadtree into another quadtree with fewer nodes (additional information must be present in each node, however). The new quadtree contains both black and white nodes as before, except that associated with each black node is an integer *radius* which indicates the size of the black square. The square defined by the radius may or may not cover a larger area than the subtree node would otherwise cover. The radius will never be smaller than the "natural" size of the black square. As the radius grows larger and larger, fewer and fewer nodes must be kept in the tree structure.

The radius may indicate that the corresponding square extends beyond the borders of the image, and for this reason it is assumed that all space outside of the image boundary is black.

Asking if a particular pixel is white or black involves more than simply traversing the tree; a white node in the quadtree may be white, but it may also be covered (partially or totally) by a nearby large-radius black node. Neighboring cells in the tree must be examined to see if there is any overlap.

The worst possible outcome for the Quadtree Medial Axis Transform (QMAT) is that the resulting quadtree will be identical to the original quadtree. The result will never contain more nodes than the original, but it often contains significantly fewer nodes. This also means that a QMAT tends to be less sensitive to shift than its quadtree counterpart.

QUADTREE FORESTS

In (Jones, Iyengar; 1981) an idea is presented that offers space savings by breaking a quadtree into a list (or forest) of trees where each tree is a small section of the original structure.

A normal quadtree is labeled according to a simple recursive algorithm as follows: any black leaf is considered to be *good*; any internal (gray) node with two or more good descendants is also considered to be good; all other nodes are *bad.*

Once the quadtree has been labeled, the forest is formed by detaching subtrees at their highest good point from the labeled structure. If the root node has been labeled as good, the result is a forest of a single tree (the original structure).

Each entry in the forest contains a pointer to the section of the tree that has been labeled as good, and also the level and path information that serves to position the tree section within the original image. The path key is very similar to the quaternary codes of the linear quadtree.

This technique does not eliminate white nodes from the resulting tree structures, but it does tend to reduce the amount of white space that is stored.

The worst possible outcome for a quadtree forest occurs when the image is a kind of checkerboard where each and every 2x2 pixel area contains at most one black pixel. The quadtree is fully developed to the pixel level, and the corresponding forest consists of one entry for each (1x1) black pixel (Gautier, Iyengar, Lakhani, Manohar; 1985).

This technique suffers from many of the same problems that normal quadtrees suffer from; they are shift variant and cannot take advantage of object coherence within an image. Shifting an image within an image grid will require a new forest to be developed. The list of subtrees is also presented as a linear list, with no topological relationships between the locations of trees in the forest.

COMPACT QUADTREES

In (Jones, Iyengar; 1983) one more technique is developed to reduce the amount of storage with a standard pointer-based quadtree. Instead of containing in each node one color field, one pointer for each of the four descendants (all Nil in case of a leaf node) and one pointer for the ancestor, each *metanode* contains one ancestor pointer (MFATHER), one descendant pointer (MSONS), one brother pointer (MCHAIN), and the colors of all four quadrants; black, white, or gray. A gray value in the color list implies that there is a metanode attached to the descendant pointer. If more than one gray appears in the color list, the additional metanodes will be chained via the brother link off of the descendant node.

The combination of keeping all four colors and fewer pointers in each node reduces both the number of nodes required and the overall number of pointers.

TRANSLATION INVARIANT DATA STRUCTURE

The final technique for reducing the size of an image representation is not a quadtree technique at all, but it has some attributes that make it worthy of discussion. The Translation Invariant Data Structure (TID) (Scott, Iyengar; 1986) is based on the medial axis transform of an image, and operates by breaking the image down into a list of black maximal squares along with their location and radius. The squares may or may not overlap, and the union of all squares is the original image.

A TID is translation invariant by virtue of the fact that the locations in the list can be considered to be relative to some origin; change the origin and the objects in the image have moved, but without disturbing the relative positions of the objects. The locations in the list can also be modified according to which of several objects is moving relative to the others.

The storage requirements for a TID are no worse than for any of the quadtree techniques outlined above, and may be considerably better. Forming a TID from a region that is R rows by C columns is of the order O(RC Log(Min(R,C))) (Scott, Iyengar; 1986). See also (Gautier, Iyengar, Scott; 1985).

One drawback of using TID is that any hierarchy of subimages or of objects is lost. No square has any priority over any other, and no coarse resolution images can be quickly extracted as in the case of the quadtree (or any of its variants). This technique might also be extended to use rectangles instead of squares for obtaining better matches of the objects being captured.

QUADTREE MESHES

The previous techniques all address one or more of the problems associated with quadtrees. What we present here is a synthesis of several of the aforementioned techniques, with some new considerations thrown in.

What is a Quadtree Mesh?

A Quadtree Mesh is a collection of quadtrees that may be placed anywhere in the plane, with a geometrical meshwork (graph) applied to the origins of the quadtrees. The quadtrees may overlap, and may contain conflicting information about a particular subregion.

In general, each quadtree contains the image of a single object or of a group of objects that belong together (figure 3). These *object quadtrees* are connected together with a meshwork. The meshwork may be as simple as a linear list (with all its inherent disadvantages) or it may be some form of optimal mesh such as a Delaunay Triangulation (figure 2).

Each entry in the mesh contains the following information: a pointer to the quadtree itself, the power of two that describes the side length of the square area covered by the quadtree, and the *image origin* (coordinates of the upper left corner of the square, for example). Additionally, it is very useful in geographic applications to keep the coordinates of the primary *point of interest* of the area, relative to the origin point. This reference point can be the origin point itself, the image center, the center of mass of the image object, or even some point completely outside of the image. In figure 2, the reference point in each object is the NorthWest pixel of the four pixels that surround the object center. The positions of the reference points are used when deciding how the quadtrees are to be connected together in the mesh.

Quadtree mesh images may be as large as necessary: because the origin and reference points may be anywhere in the plane, overall image size is bounded only by the integer precision and memory capacity of the computer system being used. The size of an image area is stored with each object quadtree, and that area can be as small as a single pixel or as large as the entire integer plane. The number of levels in each quadtree (its depth) can be easily determined from its size.

A quadtree mesh applied to a bilevel image effectively partitions that image into three values; black and white (inside at least one object quadtree), and *unknown* (outside all object quadtrees). For most purposes, unknown can be considered equivalent to white.

Advantages of a Quadtree Mesh

Quadtree meshes eliminate the problem of variance with respect to translation. Moving an object from once place to another entails changing only its origin point and does not affect the structure or contents of the corresponding object quadtree.

Each object quadtree need be only as large as necessary to capture its image object; although there is some white space stored, that amount of white space is relatively small. The implementation of the object quadtrees could be as Quadtree Forests, Normalized Quadtrees, QMATs, Compact Quadtrees, or any other method that reduces the overhead of storing a single quadtree. Each object quadtree could use a different method of storing the image depending on which method optimizes the image object best. The entries in the quadtree mesh would then need a field describing which of the methods is used in the corresponding object quadtree.

Four objects are shown in figure 3, partitioned according to how they would be described with quadtrees. Objects 1 and 2 are 4x4 regions that require object quadtrees of one leaf each (because object 1 is identical to object 2, only one copy needs to be constructed; the corresponding mesh nodes can share a single instantiation of the quadtree). Object 3 is a 4x4 region that requires an object quadtree with 10 leaves, and object 4 is a 7x7 region (normalized to the lower right corner of its 8x8 square) that requires an object quadtree with 40 leaves. Together, regardless of their placement within an image, the four object quadtrees require 52 leaves (51 if the quadtrees for objects 1 and 2 are shared). The two 16x16 images in figure 2 contain the four objects in different places. A quadtree describing the left image requires 142 leaves, and a quadtree describing the right image requires 103 leaves.

Searching

When looking for the color of a particular pixel, a search uses the meshwork to find the cluster of interest. The configuration of the meshwork can simplify many problems in geometry.

If object quadtrees describe areas of similar sizes, then finding the "correct" quadtree entails traversing the mesh to find the closest point of interest to the search pixel. If a Delaunay Triangulation is used as the mesh geometry, then finding the closest point is trivial: choose any point in the mesh as the current point, examine all points that neighbor (are connected to) the current point and set the current point to the one closest to the search pixel, and repeat until there are no closer points. The quadtree associated with the resulting mesh point is searched for the pixel value. By the properties of Delaunay Triangulations, the resulting mesh point will be one of the three points on the perimeter of the triangle enclosing the search pixel if the pixel is inside the convex hull of the mesh space, and it will be the closest hull point if the search pixel is outside. More than one object quadtree may contain the pixel, and a conflict between quadtrees may have to be resolved. Conflict resolution is addressed in the next section.

Once a pixel has been found, searching for adjacent pixels is fairly straightforward; if the new pixel is inside the current (just searched) quadtree, the same quadtree can be used. If it is not in the current quadtree, it may be in a "nearby" quadtree. The mesh is used to find the neighboring object quadtrees in the direction of the new point and those quadtrees are searched for the new point.

Extracting an image from the quadtree mesh is accomplished by searching for the colors of all pixels in a specified rectangular region. Extracting a coarse image in a quadtree mesh is possible because of the hierarchical nature of each object quadtree.

A Content Addressable Parallel Processor (Verts, Thomson; 1988) can be used to speed up search by assigning one processor to each quadtree mesh node. Each processor would contain the coordinates of the origin and the exponent of two which defines the side length N (which together can be used to determine the bounding box of the object quadtree), and the reference point (point of interest). To search for the color of a pixel all processors would compare, in parallel, the coordinates of the pixel with the bounding box of the quadtree assigned to that processor. Any processors not covering the desired pixel drop out of the search. If there is only one *responder* then the corresponding quadtree is searched for the pixel. If more than one responder remains then several quadtrees overlap the pixel and a conflict may exist.

Resolving Differences in Overlapping Regions

Conflicts arise when two or more object quadtrees differ on the color of a particular pixel. Several techniques can be used to resolve differences. The simplest technique is to always return black for the pixel color. This is in effect taking the logical-OR of the overlapping regions; since at least two differ, one or more must be black. Another method would be to return the value encoded by that object quadtree which is closest to the search pixel (closest according to the mesh reference point).

Although the images are bilevel, the gray (internal) nodes of the quadtree may store an average that is between the black and white values. If so, and if a coarse resolution image is desired, the color of a particular pixel may be computed as the average of all the different values defined by overlapping object quadtrees. Alternatively, the darkest of the competing definitions could be chosen.

Deriving a Quadtree Mesh

Extracting the optimum quadtree mesh from a static image is the subject of ongoing research. Identifying unique, connected objects in the image is relatively simple, but the optimum mesh may involve breaking an object into pieces and assigning an object quadtree to each piece. For example, objects 1 and 2 overlap in the left-hand image of figure 2. An object quadtree applied to the combined figures would require a minimum of 13 leaves (normalized in an 8x8 grid), but object quadtrees for each object require just one leaf apiece.

Building the mesh is much simpler if the objects in an image are known beforehand. For example, constructing a Delaunay Triangulation of a set of N points in the plane has been shown to be of order O(N Log N) (Preparata, Shamos; 1985).

Problems With the Quadtree Mesh Technique

There are several problems with the Quadtree Mesh technique. Conflicting quadtrees have already been addressed.

When extracting the appropriate object quadtree from a static image, it is possible to *over-cluster* or *under-cluster* the image. Over-clustering can occur when there is one object quadtree for each black pixel in the image, regardless of object coherence within the image. Under-clustering can occur when the mesh consists of a single object quadtree, regardless of the complexity or number of objects in the image. The optimum quadtree mesh may indeed turn out to be one object quadtree for each pixel or one object quadtree for the entire image; the process for extracting the object quadtrees and deriving the mesh must be very careful.

If all object quadtrees in a mesh are similar in size, or are of a known maximum size, then it is fairly easy to identify those mesh neighbors that overlap a search pixel. If one object quadtree (on the periphery of the mesh, say) overwhelms the area encompassed by the entire rest of the mesh, then that object quadtree needs to be considered in all search problems, yet its reference point and position in the mesh indicate that it should be rarely involved in neighborhood searches.

EXTENSIONS

Gray Scale Quadtrees

Bilevel images are simple representations of areas: a pixel is either inside an object (black) or outside (white). Realistic images are not simply bilevel but are composed of shades of gray. When constructing the quadtree for a gray scale image it is difficult to determine when the four leaves of a quadrant can be replaced by one larger leaf; it is unlikely that there will be many areas in an image that all have the same gray level. If exact match is used, all four pixels in an area must have the same gray value to be collapsed into one.

Thresholding is a simple technique to convert a gray region into a black and white region; any pixel above a certain level is changed to black, anything below that level becomes white. This separates figures from their backgrounds in high contrast images, but too much information is lost for this to be useful in general.

In (Gonzalez, Wintz; 1987), a region is considered to be "homogeneous" (all one color) if at least eighty percent of the pixels in that region are within two standard deviations of its average gray level. If the homogeneity constraint is met, the region is assigned the average as its gray value.

An effective way to generate gray quadtrees is to specify an allowable error value such that the pixels in a region are considered to be all one color if no pixel deviates from the average by more than the error number. With an error value of one, for example, collapsing four leaves into one requires that all leaves differ from the average by at most one (plus or minus).

Gray Quadtree Meshes

Meshes can be formed from gray object quadtrees fully as easily as with bilevel object quadtrees. The same techniques apply when resolving overlaps as when dealing with the gray (internal) nodes of bilevel quadtrees. When two or more gray object quadtrees conflict, the color of the search pixel can be determined as either the average of the overlapping values, or as the darkest. Unknown areas (outside all mesh areas) are treated as white or as a "special" value not in the normal image.

An Application of Gray Quadtree Meshes

As a test problem, suppose that the "image" is a digital terrain model, where the "pixel" values represent elevations. The regions encoded by object quadtrees are areas that have known elevations; areas outside all object quadtrees have unknown elevation. The problem is to determine if a line of sight exists between two points in the model. A three dimensional variation of the Bresenham digital stepping algorithm can be used to determine all elevations along the line of sight between the two points. The next step is to examine each point to see if the elevation of the line of sight is above or below the elevation of that point in the model. Each successive point search uses the mesh neighborhood of the previous search. The hierarchical nature of the object quadtrees can be used to quickly derive a coarse model of the terrain, and a measure of the error used in extracting the quadtree from the true image will form a "fuzzy" region within which the true elevation is guaranteed to lie.

Octree Meshes, Gray Octrees and Beyond

The three dimensional analog to the quadtree is the octree. An octree mesh consists of (possibly overlapping) volumes connected together with a three-dimensional meshwork. A meshwork with properties similar to the two dimensional Delaunay Triangulation would define a set of tetrahedra, where any sphere that passes through all four non-coplanar vertices contains no other vertex.

Octrees need not be restricted to bilevel *voxels* (the three dimensional equivalent of a pixel that indicates whether a volume is filled or empty). A *gray octree* can be constructed in the same manner as a gray quadtree. Instead of color, the differing values can represent density, temperature, pressure, or some other multiple-valued phenomenon associated with volumes.

The meshwork, the data structures for decomposing space hierarchically, and the gray extensions to those data structures all have analogs in dimensions higher than three. While such structures are difficult to visualize, the mathematics are consistent.

CONCLUSIONS

The Quadtree Mesh representation has many of the benefits of quadtrees with few of the drawbacks. Multiple, overlapping quadtrees connected together in a meshwork, where each quadtree may be optimized for its subimage, maintain the translation independence and hierarchical representation of each object.

BIBLIOGRAPHY

Chien, C., Agarwal, J. K., 1983. A Normalized Quadtree Representation: Computer Vision, Graphics and Image Processing, Vol. 26 #3 (June 1984), pp. 331-346

Gargantini, I., 1982. An Effective Way to Represent Quadtrees: Communications ACM, Vol. 25 #12 (December 1982), pp.905-910

Gautier, N. K., Iyengar, S. S., Lakhani, N. B., Manohar, M., 1985. Space and Time Efficiency of the Forest of Quadtrees Representation: Image and Vision Computing, Vol. 3 #2 (May 1985), pp. 63-70

Gautier, N. K., Iyengar, S. S., Scott, D. S., 1985. Performance Analysis of TID: IEEE Computer Society Proceedings, Conference on Computer Vision and Pattern Recognition, San Francisco, CA, June 19-23, 1985, pp. 416-418

Gonzalez, R. C., Wintz, P., 1987. Digital Image Processing (second edition), Addison-Wesley, Reading, MA

Jones, L., Iyengar, S. S., 1981. Representation of a Region as a Forest of Quadtrees: IEEE Computer Society Proceedings, Conference on Pattern Recognition and Image Processing, Dallas, TX, August 3-5, 1981, pp. 57-59

Jones, L., Iyengar, S. S., 1983. Virtual Quadtrees: IEEE Computer Society Proceedings, Conference on Computer Vision and Pattern Recognition, Washington, DC, June 19-23, 1983, pp. 133-135

Li, M., Grosky, W. I., Jain, R., 1982. Normalized Quadtrees with Respect to Translations: Computer Graphics and Image Processing, Vol. 20 #1 (September 1982), pp. 72-81

Preparata, F. P., Shamos, M. I., 1985. Computational Geometry, Springer-Verlag, New York

Samet, H., 1983. A Quadtree Medial Axis Transform: Communications ACM, Vol. 26 #9 (September 1983), pp. 680-693

Samet, H., 1984. The Quadtree and Related Hierarchical Structures: ACM Computing Surveys, Vol. 16 #2 (June 1984), pp. 187-260

Samet, H., 1985. Reconstruction of Quadtrees from Quadtree Medial Axis Transforms: Computer Vision, Graphics, and Image Processing, Vol. 29 #3, March 1985, pp 311-328

Scott, D. S., Iyengar, S. S., 1986. TID – A Translation Invariant Data Structure for Storing Images: Communications ACM, Vol. 29 #5 (May 1986), pp. 418-429

Verts, W. T., Thomson, C. L., 1988. Parallel Architectures for Geographic Information Systems: Technical Papers, 1988 ACSM-ASPRS Annual Convention, Vol. 5 (GIS), St. Louis, MO, March 13-18, 1988, pp. 101-107

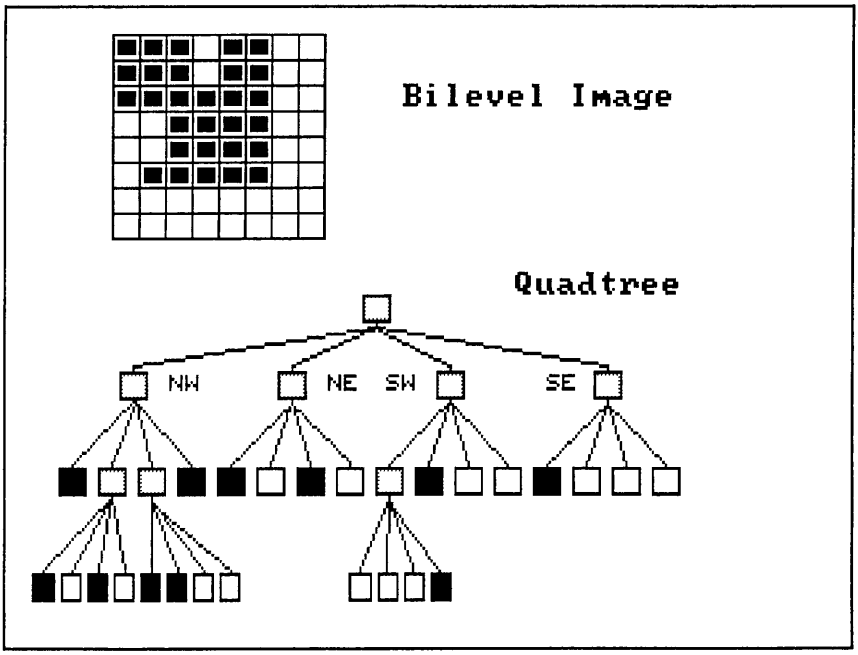

Figure 1: An image and its quadtree.

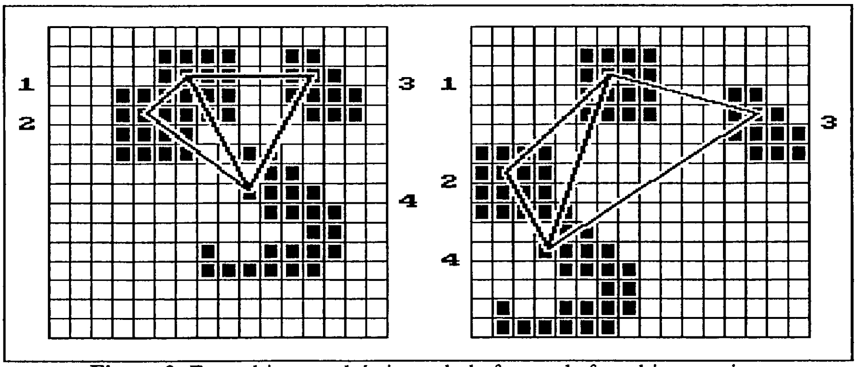

Figure 2: Four objects and their mesh, before and after object motion.

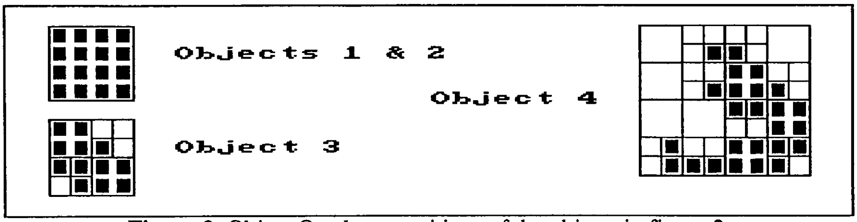

Figure 3: Object Quadtree partitions of the objects in figure 2.