Operations on Images Using Quad Trees

GREGORY M. HUNTER, MEMBER, IEEE, AND KENNETH STEIGLITZ, MEMBER, IEEE

Abstract—A quad tree for representing a picture is a tree in which successively deeper levels represent successively finer subdivisions of picture area. An algorithm is given for superposing N quad trees in time proportional to the total number of nodes in the trees. Warnock-type algorithms are then presented for building the quad tree for the picture of the boundary of a polygon, and for coloring the interior of such a polygon. These algorithms take $O(v+p+q)$ time, where v is the number of polygon vertices, p is the polygon perimeter, and q is a resolution parameter. When the resolution q is fixed, these algorithms are asymptotically optimal.

Index Terms—Cartography, computer-aided design, encoded raster graphics, layout, pattern recognition, picture compaction, picture processing, pyramid, quad tree, Q-tree, rope, scan-line encoding, space planning.

I. Introduction

THIS paper is concerned with the efficient construction and use of quad trees for pictures. We begin by giving rough characterizations of these objects to prepare for some discussion which may show why the subject is of interest.

Pictures: A picture is a square array of colored points. Although we use the word "picture," we mean any two-dimensional array of information: a map, a floor plan, the relationship of machine parts in a plane, etc. Likewise, the color of a point may in fact be any information to be associated with a point in a two-dimensional grid.

Quad Trees: A quad tree is a tree whose nodes are either leaves or have four children. The children are ordered: 1, 2, 3, and 4.

Quad Trees for Pictures: A quad tree for, or representing, a picture is a quad tree whose leaves represent areas of the picture. Each leaf is labeled with the color of the area of the picture which it represents. Each node is associated with a square region of the picture. The root is associated with the entire picture. Besides the root, each other node is associated with one of the four quadrants of its parent's square. The ith child is associated with the ith quadrant of its parent's square. No parent node may have all its descendant leaves the same color. Since a parent and its descendants represent the same region of the picture, if all the descendants have the same color, the picture can be more compactly represented by coloring the parent, and removing all the children (see Fig. 1).

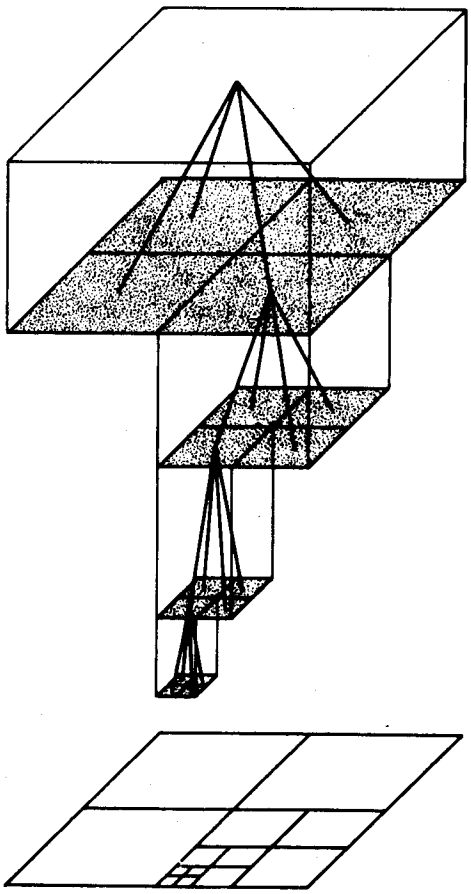


Fig. 1. A picture and its quad tree.

The square associated with a node may be called the square for that node, or the quadrant for that node. When speaking loosely, we may refer to a node, meaning its quadrant. More generally, we may refer to items in the quad tree and mean their counterparts in the picture, or vice versa.

Klinger and Dyer [1] provide a good bibliography of the history of quad trees. Their paper reports experiments on the degree of compaction of picture representation which may be achieved with tree encoding. Their experimental pictures include block letters and photographs. Their experiments show that tree encoding can produce memory savings. Pavlidis [2] reports on the approximation of pictures by quad trees. Horowitz and Pavlidis [3] show how to segment a picture using traversal of a quad tree. They segment the picture by

Manuscript received March 31, 1978; revised September 14, 1978. This work was supported by the National Science Foundation under Grant GK-42048 and the U.S. Army Research Office, Durham, NC, under Grant DAAG29-75-G-0192.

G. M. Hunter is with Decisions and Designs, Inc., McLean, VA 22101.

K. Steiglitz is with the Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ 08540.

0162-8828/79/0400-0145$00.75

polygonal boundaries. Tanimoto [4] discusses distortions which may occur in quad trees for pictures. Tanimoto [5, p. 27] observes that quad tree representation is particularly convenient for scaling a picture by powers of two. Quad trees are also useful in graphics and animation applications [6], [7], which are oriented toward construction of images from polygons and superposition of images. Encoded pictures are useful for display [8], especially if the encoding lends itself to processing.

The model of computation which we use in the analysis of the time and space requirements of our algorithms involves a RAM, or random access machine, for which storage or retrieval of data from memory requires constant time. Coordinate words may be considered to be real numbers or a fixed number of bits, depending upon the analysis. Time is measured in steps, or operations upon pairs of words, such as comparison and multiplication, requiring constant time. If an algorithm requires $f(x_1, x_2, x_3, \cdots, x_n)$ time or space, it is defined to be $O(g(x_1, x_2, x_3, \cdots, x_n))$ if and only if $f(x_1, x_2, x_3, \cdots, x_n) \leqslant kg(x_1, x_2, x_3, \cdots, x_n)$ for some constant k and all x_1, $x_2, x_3, \cdots$, and x_n greater than some fixed integer. An algorithm is asymptotically optimal when its requirements are, in the limit for large input parameters $x_1, x_2, x_3, \cdots$, and x_n, proportional to those of the best algorithm performing the same function. Aho *et al.* [9] provide a good explanation of this kind of analysis.

II. A Superposition Algorithm

We now have enough rough definitions to present an algorithm which indicates the usefulness of quad tree encoded pictures. The next section contains more precise definitions. We begin with a preliminary algorithm to clean up a quad tree which has four siblings colored the same.

The Condensation Algorithm: Given a tree satisfying the requirements of a quad tree for a picture, except that there may be cases of four identically colored sibling leaves, a quad tree for a picture may be formed by recursively removing these siblings and giving their color to their parent. Time linear in the number of input nodes is required. The method is to use a postorder traversal [10].

Starting at the root, traverse the given tree as follows. Recursively, traverse the subtree rooted in each child, recording the color of the subtree, if it is only one node, at the root. Then, visit the root. If the root is the parent of four leaves all colored the same, color the root their color and remove them.

We next consider the important operation of superposing one tree on another. The Superposition of an occluding picture $P1$ over a background picture $P2$ of the same size as $P1$ is defined as $P1$ with points colored transparent replaced by the corresponding points of $P2$.

Pairwise Superposition Algorithm: This algorithm has, as input, the tree encodings of two pictures of the same size. The output is the tree for the picture which is the superposition of one picture over the other. Input specifies which tree represents the occluding picture. Input also designates a color to be considered transparent.

The general idea of the algorithm is that we can traverse both trees in parallel, modifying the bottom tree when necessary. The algorithm proceeds as follows. Traverse the two trees in parallel. When a leaf is visited in either tree, there are three alternatives:

Case 1: The traversal is visiting leaves in both trees. If the leaf in the upper tree is transparent, do nothing to the lower tree. If the leaf in the upper tree is opaque, replace the leaf in the bottom tree with the leaf in the upper tree. Continue the traversal.

Case 2: The traversal is visiting a leaf in the upper tree and a parent in the lower tree. If the upper tree leaf is transparent, do nothing. If the leaf in the upper tree is opaque, replace the lower tree parent and its descendants (without traversing them by changing pointers) with the upper tree leaf. Continue the traversal as if the descendants in the lower tree already had been visited.

Case 3: The traversal is visiting a parent A in the upper tree and a leaf B in the lower tree. Replace B with A and its subtree with all transparent leaves given the color of B. After traversing the subtree, continue the parallel traversal.

Definition: Super (x, y) is the pairwise superposition algorithm applied to upper tree x and lower tree y.

Lemma: If y is already in memory to start, and the output need not be read out of memory but may be formed there and left there, Super (x, y) is O(number of nodes in x). If input and output must be read in and out, respectively, Super(x, y) is O(number of nodes in x and y).

Proof: Observe that in each case of traversal interruption, work is proportional to a portion of the upper tree unique to that interruption. Q.E.D.

We now consider the *superposition* of n pictures of the same size and given in an order $1, 2, 3, \cdots, n$, each picture except number one having associated with it a color considered *transparent* for that picture, where lower numbered pictures are considered *lower*, and higher numbered pictures are considered to be *higher*. This is defined as the picture which at each point is colored the same as the corresponding point of the highest picture not transparent at that point. All colors not transparent in a tree are defined to be opaque in the tree.

The N-Tree Superposition Algorithm: Start with the bottom tree. Using the pairwise superposition algorithm, superpose the next highest tree. Over the resulting tree, superpose the next highest tree. Repeat until the highest tree has been superposed on the superposition of all the lower trees. After all the pairwise superpositions have been completed, apply the condensation algorithm, and the n-tree superposition algorithm terminates.

Theorem: The N-tree superposition algorithm may be performed in time and space linear in the number of nodes in all the trees.

Proof: After tree 1 has been constructed from the input, Super (n, Super (n - 1, Super (n - 2, $\cdots$ Super (3, Super (2, 1)) $\cdots$) yields the superposition of the n-trees. It is clear that each Super operation can be accomplished in time and space linear in the size of the upper tree. Summing the sizes of each

upper tree in each such operation and the size of tree 1 gives the sum of the sizes of all the trees. Q.E.D.

We note that similar techniques can be used to find the union, intersection, and other operations on pictures. The reader is referred to [11] for details. Next, we consider the complexity of quad tree representations of polygons.

III. More Definitions

We begin with more precise definitions of previous terms, then add some new terms.

Pictures: A picture is an ordered pair $P = (C, A)$, where C is a finite set of colors, and A is a square array of pixels (picture elements, points of the picture), each an element of C.

Trees for Pictures: A tree T is a tree for the picture $P = (C, A)$, if there is a mapping M from the leaves of T into disjoint subsets of A whose union is A. Each leaf is labeled with a color shared by all pixels in the subset of A, which is the image of the leaf under the mapping M. No parent node has all descendant leaves colored identically.

"**" Denotes exponentiation.

Quad Trees for Pictures: T is a (depth-q) quad tree for a picture $P = (C, A)$, if T is a tree for P under mapping M, T is a quad tree, A is a $2^{**}q \times 2^{**}q$ array, and q is a positive integer. Let K be a mapping from nodes of T to square portions of A, and let K map the root of T to A. Let K map the ith child of a node to the ith quadrant of the K-image of the parent. A quadrant includes the points on its boundary. M is the restriction of K to leaves of T. Note that T has at most $q + 1$ levels (see Fig. 2).

Polygons: A polygon is a list of coplanar points, called its vertices, listed as pairs of coordinates. The edges of a polygon are line segments between consecutive vertices and between the first and the last vertices. Polygons are simple; that is, their edges do not intersect. Each edge intersects only two vertices: its endpoints. The vertices of polygons have non-negative coordinates.

Pictures of Polygons: A picture $P = (C, A)$ is a picture of polygon G if the following conditions hold. Coordinates of G are between 0 and $2^{**}q$, inclusive. Array A is $2^{**}q \times 2^{**}q$. Pixels are square. $A(1, 1)$ represents a pixel with corners $(0, 0), (0, 1), (1, 0)$, and $(1, 1)$. A $(2^{**}q, 2^{**}q)$ is a pixel with corners $(2^{**}q, 2^{**}q), (2^{**}q - 1, 2^{**}q - 1), (2^{**}q - 1, 2^{**}q)$, and $(2^{**}q, 2^{**}q - 1)$. Other pixels in A are mapped into the coordinate system in the natural way. All pixels inside the polygon have one color, "interior." Pixels outside the polygon are colored "exterior." Pixels intersecting the polygon are colored interior.

Quad Trees for Polygons: T is a quad tree for polygon G if t is a quad tree for picture P and P is a picture of polygon G.

In a quad tree for a polygon, nodes are divided into three exclusive classes: interior, exterior, and boundary nodes (see Fig. 3).

Boundary nodes are those intersecting the boundary of the polygon. It may be that only the boundary of the node intersects the polygon. That is, the polygon may touch the node only on the edges of its square.

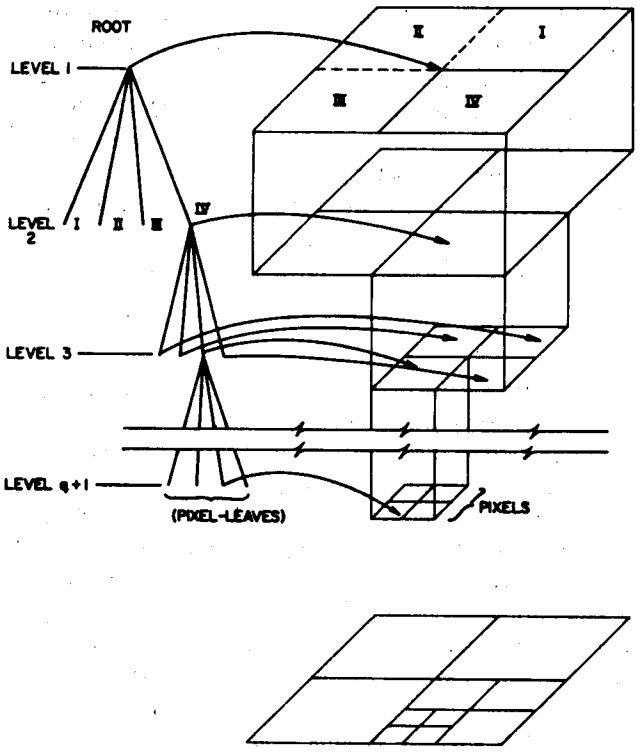


Fig. 2. A quad tree for a picture, and examples of mapping K.

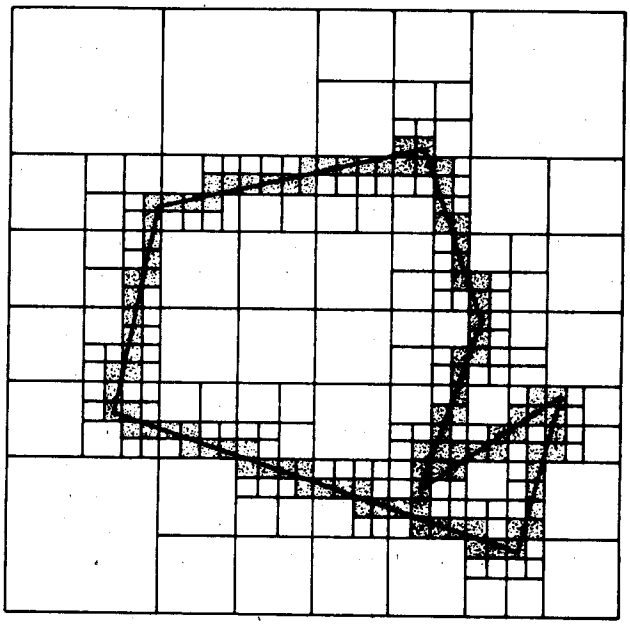


Fig. 3. A polygon and the leaves of its picture. The boundary leaves are shaded.

Interior nodes are those containing only points interior to the polygon.

Exterior nodes are nodes wholly outside the polygon.

Complexity Parameters: v is the number of vertices in a polygon. q is the base 2 logarithm of the length of a side of a picture. q limits the depth of the quad tree. If we assume that

the number of bits in the coordinates of polygonal vertices is fixed, it is not meaningful to let q increase indefinitely. We will assume that if the number of bits in polygonal coordinates is fixed, so is picture resolution, and so is q. Our analyses will make explicit the effect of varying q. p is the smallest integer no less than the perimeter of a polygon. p is measured in pixel widths, assuming there is a picture associated with the polygon.

CEILING (X) is the smallest integer no less than X.

IV. The Complexity of the Quad Tree for a Polygon

The Parenthood Theorem: In a quad tree for a polygon, only boundary nodes may have children.

Proof: Assume that an interior or exterior node has children. Interior nodes have all interior descendants; exterior, exterior descendants. Therefore, all descendant leaves of interior nodes are colored interior; of exterior, colored exterior. But no parent has all descendant leaves colored identically, and the assumption is contradicted. Q.E.D.

Lemma: Any curve entering a square, touching each of its four quadrants, and then exiting must be at least as long as the side of the square.

Proof: (see Fig. 4.) Assume that the side of the square is of length 2. Let the curve enter the square $WXYZ$ on side XY at point A, first intersect PR at point B, and exit $WXYZ$ at D. $AB \geq 1$. If B is the center point, $BD \geq 1$. Therefore, we may assume that B is closer to R than to P, as shown, without loss of generality. If D is on WZ, XY, or XW, then $BD \geq 1$. Therefore, we may assume that D is on YZ. Since the curve must touch some point C in I after reaching B, $CD \geq 1$. Q.E.D.

Consider a grid of squares, each having sides of length n. Let G be a polygon of length p lying on the grid. We will decompose G into a sequence of curves. Chose some point where G exits one square to enter another. Define the first curve in G as extending from this point until four squares have been intersected and we cross into a different, fifth, square. At this point, begin the next curve in G, intersecting four new squares not counting those intersected by any previous curve and extending up to the point where G enters a fifth square not previously intersected. G is composed of a sequence of such curves, plus one final curve taking us back to the point where we started. This final curve may intersect fewer than a full complement of four previously nonintersected squares.

Theorem: Each curve defined above, except the final curve, must be at least n in length.

Proof: Call the four squares which define the curve, A, B, C, and D. If the curve is less than n in length, no two of A, B, C, and D can be n or more units apart at their closest points. Since any two squares which do not touch must be separated by at least n at their closest points, all four squares must touch one another. This means that the squares are arranged into a larger 2×2 square of squares. According to the lemma above, any curve entering such a larger square, touching each of the four component squares, and exiting the larger square must have length at least $2n$. Q.E.D.

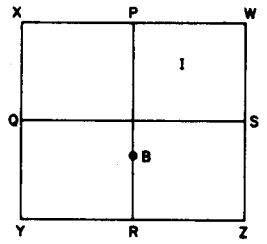


Fig. 4. Square $WXYZ$ and its quadrants.

Corollary: Since only four new squares are added by each curve, and each curve has length at least n, a polygon with a perimeter no longer than p can intersect no more than 4 CEILING (p/n) squares.

We will find, for each level of the quad tree for a polygon, an upper bound on the number of nodes at that level. Then, we will sum the bounds for all levels to get an upper bound to the number of nodes in the tree.

Let the root be level 1 of a quad tree for a polygon. At level k, the grid of quadrants has side $2^{**}q$ and each square has side $2^{**}(q - k + 1)$. Then, the last corollary implies that the polygon may intersect no more than 4 CEILING $(p/2^{**}(q - k + 1))$ quadrants at level k. Call this expression $B(k)$.

Since only nodes intersecting the boundary may have children (the parenthood theorem), no more than $B(k)$ nodes at level k have children. Since each node on level $k + 1$ is the child of a node on level k, there can be no more than $4B(k)$ nodes on level $k + 1$.

Since there are, at most, the root and four children of the root on the first two levels and no nodes beyond level $q + 1$, we have in the entire tree no more nodes than

$$5 + \sum_{k=2}^{q} 4B(k) \leq 5 + \sum_{k=2}^{q} 16 \text{ CEILING } (p/2^{q-k+1})$$

$$\leq 5 + \sum_{k=2}^{q} 16\,(1 + p/2^{q-k+1})$$

$$\leq 5 + \left(16 \sum_{k=2}^{q} 1\right) + 8 \sum_{k=2}^{q} p/2^{q-k}$$

$$\leq 5 + 16(q - 1) + 8 \sum_{i=0}^{q-2} p/2^{i}$$

$$\leq 16q - 11 + 8 \sum_{i>0} p/2^{i}$$

$$\leq 16q - 11 + 16p.$$

We have just proved the *tree complexity bound theorem*. There are no more than $16q - 11 + 16p$, i.e., O $(p + q)$ nodes in the quad tree for a polygon. The bound holds for trees which would be quad trees except for cases of boundary nodes with all children colored identically. This is because our analysis, having nothing to do with color, assumed only that all parent nodes touch the boundary. See [11] for examples which approach this bound, and Tanimoto [5] for another complexity measure.

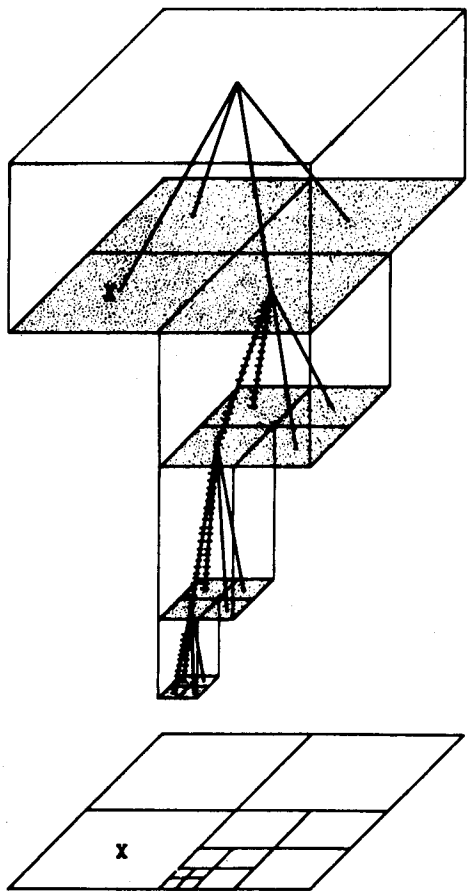


Fig. 5. The adjacency tree for leaf X is shown with hashed edges.

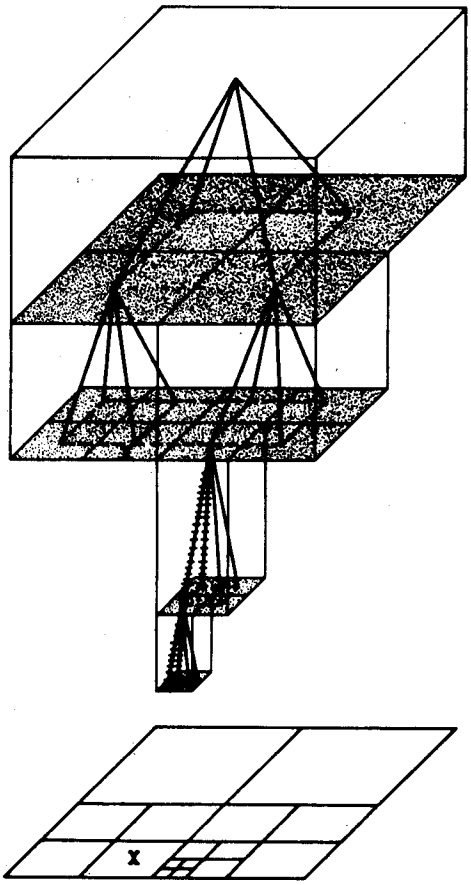


Fig. 6. The adjacency tree for leaf X is shown with hashed edges. Ropes are shown as dashed lines. Note that X and its leaf neighbors on the side of the adjacency tree are connected by moves along ropes and edges of the adjacency tree.

in the quad tree. The number of nodes in an adjacency tree is no more than twice the number of its leaves. Therefore, the number of nodes in adjacency trees is no more than four times the number of leaves in the quad tree. Q.E.D.

We may wish to rope the tree as we build it. If a tree grows by a sequence of additions of four children each to a sequence of leaves, the following technique applies.

The Roped Quad Tree Refinement Theorem: A leaf of a roped quad tree may be given four children and the ropes may be updated, all in constant time.

Proof: Give X four children. Connect neighboring children of X with (labeled) ropes. For each side of X, add or delete ropes as follows:

Case 1: X is roped on the side being considered. Leave the rope only if it leads to a leaf; otherwise, remove it. If the rope leads to a parent node, rope its children bordering X to neighboring children of X.

Case 2: X has no rope on the side being considered. If X has a neighbor on the side, it is larger than X. Do nothing.

Q.E.D.

Corollary: A tree may be roped in linear time.

The Roped Quad Tree Condensation Theorem: In a roped quad tree, the children of any given parent with four leaf-children may be removed and the resultant tree may be properly roped, all in constant time. Input to this process points to the given parent.

Proof:

Adding New Ropes: New ropes are needed when there are new cases of equal-sized neighbors, one of which is a leaf. No old leaves receive new neighbors in this process. The only new leaf is the ex-parent. All its neighbors of equal size which are leaves are already connected to it by ropes. All of its nonleaf equal-sized neighbors have children which are roped to the leaf-children to be removed. Therefore, the leaf-children are connected to children of nodes which should be roped to the given parent. Simply insert the new ropes.

Deleting Old Ropes: Since the only nonleaves in the new tree were nonleaves in the old tree, none of the ropes in the input tree need be removed except for those connected to the children removed. Remove the children and any ropes connected to them. Q.E.D.

We may wish to find neighbors more quickly than is possible by using ropes and adjacency trees.

The Collapsing Algorithm: Given a roped quad tree, neighbors will be connected to form what we will call a netted quad tree. This will be done in time linear in the number of leaves. Traverse the quad tree in time linear in the leaves of the quad tree. For each leaf X of the quad tree having an adjacency tree,

(A different perspective is obtained by measuring p in units of the length of the side of the picture. p shrinks by a factor of $2^{**}q$. With these units, the number of nodes in the quad tree for a polygon is $O(p2^{**}q)$. Comparatively, there are $2^{**}(2q)$ pixels in the array representing the picture as a grid. Varying q shows how changing resolution affects the number of nodes for a given, fixed image.)

V. A Warnock-Type Algorithm for Building the Quad Tree for a Picture

A Warnock-type algorithm is one which uses successive subdivision of the picture until each quadrant is simple, according to some criterion. Warnock's visible-surface algorithm is discussed by Newman and Sproull [12, pp. 297-312].

How may we, given a polygon as a list of vertices and a desired picture element size, build the quad tree for the polygon? How much time is required? We will first give a Warnock-type algorithm for which a quadrant is defined as simple if it has only one color. Thus, simplicity is equivalent to nonintersection with the polygonal boundary.

The Tree-for-a-Polygon Algorithm: Start with the root as the current node. Recursively create the four children of every nonpixel node whose square intersects the polygonal boundary. Color pixels and nonintersecting nodes to reflect their inclusion or exclusion by the polygon.

Theorem: Time $O(v(p+q))$ is sufficient to build the quad tree for a polygon. Space proportional to the input and output is sufficient.

Proof: It is enough to test each of the $O(p+q)$ nodes against the polygon. We can test each node for intersecting the boundary of the polygon in $O(v)$. We can then test each node not intersecting the boundary for inclusion in the polygon to determine the right color in $O(v)$. This may be done by following the polygon around an arbitrary point in the node's quadrant, integrating the angles traced. The integral is nonzero if and only if the polygon wraps around the point. Q.E.D.

The data structures introduced next will aid us in finding an algorithm that is asymptotically faster.

VI. Neighbors, Adjacency Trees, and Ropes

It will be useful later to find picture information for areas neighboring other areas.

Definitions: A pair of nodes are *neighboring*, or *bordering*, if their quadrants do not overlap areas but intersect on an entire side of one of them, not only on a corner. (Recall that a quadrant includes the points on its boundary.)

Boundary nodes are those intersecting the polygon, if only at a point. It follows that, given two boundary leaves, there is a sequence of boundary leaves from one of the given leaves to the other, each successor leaf being a neighbor to its predecessor. It might be said that the boundary leaves are pathwise connected.

Adjacency trees are used to provide paths from leaves to neighboring leaves: an *adjacency tree* is a binary tree formed from nodes and edges in the quad tree. For each side on which a leaf X of a quad tree has multiple neighboring leaves, there is an adjacency tree, with these neighbors as leaves. The nodes of the adjacency tree are these leaves, plus those of their ancestors in the quad tree which are no larger than X. These nodes and the edges between them in the quad tree form the adjacency tree (see Fig. 5). Note that the root of an adjacency tree for a leaf X is a neighbor of X of equal size.

A "rope" may be used to find the adjacency tree for a leaf, as follows. A *rope* is an edge between neighboring nodes of equal size, at least one of which is a leaf (see Fig. 6). Each end of a *labeled rope* is labeled with the side of the node on that end which borders the neighbor on the other end. Throughout this exposition, assume either that all ropes are labeled, or that each node is labeled with the coordinates of its sides. Thus, constant time is enough to determine the direction in which a rope leads.

A *roped* quad tree is a quad tree in which all possible ropes are present. Note that in a roped tree each leaf is roped to each of the roots of its adjacency trees.

The Neighbor-Finding Theorem: Given a roped quad tree and given a leaf, finding a neighboring leaf with a particular x coordinate and sharing a given horizontal side with the given leaf, or with a particular y coordinate and sharing a given vertical side with the given leaf, can be done in time linear in the number of levels separating the neighbors in the quad tree. The requested side must not lie on the boundary of the picture, and the x or y coordinate must intersect the given side. Otherwise, the specified neighbor does not exist.

Proof: We give an algorithm.

Case 1: If there is no rope from the leaf on the desired side, the desired neighbor must be larger. Ascend until the current node has a rope on the desired side. This rope leads to the desired neighbor.

Case 2: If there is a rope from the leaf on the desired side and it leads to a leaf, this leaf is the desired neighbor.

Case 3: If there is a rope from the leaf on the desired side and it leads to a nonleaf, descend. Consider only the descendants on the desired side of their parents. Branching to a selected child of a node selects one-half of the range of coordinates of the parent. Thus, branching into the correct range of coordinates at each step leads us closer to the desired neighbor. When continued descent of this sort is impossible, we are visiting the desired neighboring leaf. Q.E.D.

As we move through the tree, if we keep track of the picture location and size of the current node, we can compute the quadrant for any parent, child, or neighbor node which we visit next. This is possible because we can check to see which child is involved in a vertical movement and which side is involved in following a rope. Alternatively, as we build the tree, we may store with each node the inequalities representing the sides of the quadrant for the node. These inequalities determine inclusion in the quadrant.

The Adjacency Tree Size Theorem: The total number of nodes in all adjacency trees for a given quad tree is no more than four times the number of leaves in the quad tree. A node in multiple adjacency trees is counted once for each tree.

Proof: On two sides of any given quad tree leaf are siblings or siblings' children. Thus, a quad tree leaf can have, at most, two larger neighbors. Therefore, a quad tree leaf can be a leaf in only two adjacency trees. Thus, the number of leaves in adjacency trees is no more than twice the number of leaves

pause in the quad tree traversal while traversing each of X's adjacency trees, joining each of their leaves with pointers to X. While traversing the adjacency tree for side Y of X, construct a circularly linked list of two-way pointers to the neighbors of X on side Y, in order along the side Y, counterclockwise about X. Put two-way pointers between this list and side Y. Put two-way pointers between each X and each of its sides. The result of this process is called a *netted quad tree.* By the adjacency tree size theorem, the total cost of adjacency tree traversal is linear in the leaves of the quad tree.

For a netted tree, finding some neighbor on a given side of a given leaf requires constant time. Finding all the neighbors on a side requires time linear in the number of neighbors.

VII. Two Component Algorithms

The algorithms of this section will be used in the algorithm of the next section.

Finding a Point Inside a Polygon: Given a polygon as a sequence of vertices, we will find some inside point in $O(v)$ time. First find any leftmost vertex A. Find B, the leftmost vertex right of A. $A = (xa, ya)$, and $B = (xb, yb)$. Let $d = xa + 0.5(xb - xa)$. Bisect the angle at A. All points on the bisector to the right of A are on the interior of the polygon so long as they are to the left of any edges not touching A. Therefore, the intersection of the bisecting line and the line $x = d$ is a point inside the polygon.

Finding a Polygon's Edges' Insides: We will visit the edges of a polygon in order, determining on which side of each edge is the inside of the polygon. We require $O(v)$ time. We use the technique of finding a point inside a polygon to find a leftmost vertex and an associated inside point. Pick an edge having the vertex as endpoint. Determine on which side the inside point lies. This is the inside side of the edge. In constant time for each succeeding edge, find its inside knowing the inside of its predecessor. One method: bisect the angle formed by the two edges. A point on the bisector is on the inside of one edge if it is on the inside of the other.

VIII. The Outline Algorithm

The outline algorithm constructs the quad tree for a polygon, except that each boundary pixel is represented by a leaf; boundary leaves are one color, and interior and exterior leaves are another color. Optionally, the output tree is roped, and output may include a list of all boundary pixel sides which are wholly interior to the polygon, with pointers both ways between them and their pixels in the tree.

This algorithm is a Warnock-type algorithm in the sense that successive subdivision of quadrants continues until each quadrant is either nonboundary or a pixel.

Using the techniques of the algorithms for finding points inside polygons and the insides of edges, traverse the polygon. Whenever we find that the polygon touches only the boundary of a pixel, or goes through a corner, we make an arbitrarily small perturbation in the polygon so this does not occur.

Construct the root of the quad tree. From a starting vertex, follow the polygon until it enters, then exits, a pixel. Call this pixel X. In the quad tree, give four children to each leaf whose quadrant contains X, and continue to do so until X has been added to the tree. In this and all following addition of nodes to the quad tree, use the technique of the roped quad tree refinement theorem to keep the tree properly roped.

Each point at which the polygon enters or exits a pixel is on the boundary of the polygon. Therefore, following the boundary of the pixel from such an entry or exit point takes us immediately either inside or outside the polygon. Thus, each entry or exit point has an *inside* and an *outside* side on the boundary of the pixel.

We define an *entry/exit item* as one entry point, the following exit point, a pointer to the pixel in the quad tree traversed between the entry and exit points, and a designation of the inside and outside sides of the entry and exit points on the boundary of the pixel. Each entry/exit item will be pointed to by the associated pixel in the quad tree.

Point X to an entry/exit item for X. Continue to traverse the boundary of the polygon, and for each entry and exit of a pixel, do two things:

1) The following steps add to the tree (if not already in the tree) Z, the pixel defined by the current entry and exit points. Thus, in the steps following, each pixel on the boundary of the polygon is added to the tree in the iteration of 1 which is based on the first entry and exit of that pixel.

Y, the leaf entered at the current entry point, is a neighbor of the last pixel exited by the polygon boundary and added to the tree in a previous iteration of 1. Starting from this last pixel, find its neighbor Y. If Y is a pixel, a rope leads from this last pixel directly to Y, and finding Y requires only constant time. If Y is a pixel, the remainder of 1 is omitted, since $Z = Y$ and thus Z already has been added to the tree. Assume that Y is not a pixel. Find Y using an adjacency tree and a rope. If Y is on level k, finding Y requires no more than $O(q - k)$ time. Since Y is not a pixel, Y must contain Z. Add four children to Y. Add four children to each leaf which is a descendant of Y and which contains Z, and continue to do so until Z has been added to the tree. Note that children are added only to nodes containing Z, which is entered by the boundary. Thus, children are added to boundary nodes only. This process of adding children requires adding $O(q - k)$ descendants of Y to the tree. Note further that in each interation of 1 that adds nodes to the tree, finding the next (nonpixel) leaf entered by the polygon requires time proportional to that required to actually add the nodes. Color leaves created according to whether they have been intersected by that portion of the boundary we have traversed, or not.

2) Create an entry/exit item containing the current entry and exit points, and install a pointer from Z to the item. Point the last entry/exit item to this new one.

When the traversal of the polygon has been completed back to the first entry point of X, the traversal of the polygon terminates.

The algorithm terminates now, unless the list of inside pixel sides is desired as an output. The following generates the optional list.

Noting that the entry/exit items form a linked list, and that there are pointers both ways between items and their pixels, scan the items for each pixel. This can require no more than time proportional to the number of entry/exit points for all

pixels $O(v+p)$. Identify the entry or exit point closest to each endpoint of each side of each pixel and lying on that side. Among these particular entry or exit points are some special points.

The special points which we will now define will make it easy to determine whether a pixel side not intersecting the boundary of the polygon is inside or outside the polygon. For each endpoint of a side not intersecting the boundary of the polygon, there is an entry or exit point which would be encountered first, starting from the endpoint and traveling away from the side along the boundary of the pixel. Call the two such points for the two endpoints of such a side, the *first-points* for the side (see Fig. 7).

Next, we will put into a list all the boundary pixel sides which are wholly interior to the polygon.

A side not intersecting the boundary of the polygon is inside the polygon if and only if it lies on the inside of each of its first points. Recall that entry/exit items tell us the inside side of each entry or exit point. For each pixel with an entry/exit item, find each side not intersected by the boundary of the polygon. Put into a list, with pointers both ways between it and its pixel in the tree, each such side lying on the inside of each of its first-points. The list now contains all sides wholly within the polygon which are sides of boundary pixels. Terminate the algorithm.

Theorem: The outline algorithm takes time and space $O(v+p+q)$.

Proof: Recall that each iteration of 1 that adds nodes to the tree takes time proportional to the number of nodes added, so that the total time taken by these iterations of 1 is proportional to the number of nodes in the output tree. The other iterations of 1 and the iterations of 2 take time proportional to the number of entry/exit items, and following the polygonal boundary takes no more than that plus $O(v)$. The number of entry/exit items is $O(v+p)$, and the number of nodes in the output tree is $O(p+q)$, so the whole algorithm takes $O(v+p+q)$ time. The space requirements are no worse than proportional to the time requirements.

Q.E.D.

The Modified Outline Algorithm Theorem: A modified outline algorithm can create in $O(v+p+q)$ time the quad tree for a polygon, except that the boundary leaves are one color, and the other leaves are another color. Optionally, the algorithm will output a roped tree, with a list of inside sides of boundary pixels, with two-way pointers between them and their pixels in the tree.

Proof: Let the modified algorithm consist of the outline algorithm followed by the condensation algorithm. It is apparent that condensation may proceed with each step modifying, in constant time, the list of inside sides to reflect the new tree. To output a roped tree, use the technique of the roped quad tree condensation theorem to modify the condensation algorithm so that each step yields a properly roped tree.

Since $O(p+q)$ nodes are produced, as above, by the outline algorithm, condensation requires no more time, and total time is $O(v+p+q)$, as for the outline algorithm. Q.E.D.

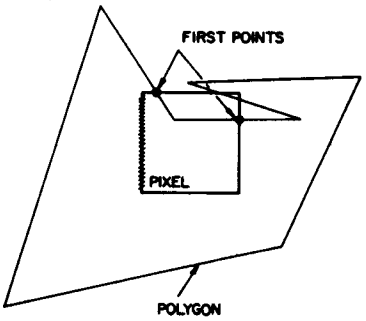


Fig. 7. A boundary pixel and the first-points of its hashed side.

We conclude this section with a comment on asymptotic optimality. For pictures of polygons, it must be that p is no more than v times $2^{**}(q+\frac{1}{2})$, or v times the length of the diagonal of the picture. Thus, if q is fixed, p is $O(v)$ and an $O(v+p+q)$ algorithm is $O(v)$. Since the time required just to read the input is proportional to v, such an algorithm is asymptotically optimal when the resolution q is fixed.

IX. The Coloring Algorithm

The coloring algorithm takes as input a roped quad tree for a polygon, except that the boundary and nonboundary leaves are colored two different colors and all boundary leaves are pixels, and a list of those sides of boundary pixels which are wholly interior to the polygon, the sides having pointers both ways between them and their pixels in the tree. Output is the quad tree for the polygon. In other words, the algorithm colors the interior nodes the same as the boundary nodes.

In outline, the coloring algorithm performs the collapsing algorithm, colors the interior leaves the same as the boundary leaves, and performs the condensation algorithm. The output is precisely the quad tree for the polygon.

Coloring Algorithm: Perform the collapsing algorithm to get a netted quad tree. For a netted tree, finding some neighbor on a given side of a given leaf requires constant time.

Put into a stack, with pointers back to their pixels, all the sides of the input list. The stack now contains all sides wholly within the polygon which are sides of boundary pixels.

For convenience, now call leaves colored with the boundary color, *colored*, and those colored as the exterior, *not colored*. Note that, at this point, interior leaves are not colored. We will color them by propagating color from colored leaves to neighboring leaves. Since there is a path to any interior leaf from a boundary leaf, all interior leaves will be colored.

Repeat the following until the stack is empty. Look at the side S which is on the top of the stack and the leaf P pointed to by S. If there is an uncolored leaf N bordering P on side S, color N. Also, push onto the stack, with pointers to N, all sides of N. Remove S from the stack.

When the stack is empty, perform the condensation algorithm to complete the coloring algorithm.

Theorem: The coloring algorithm requires time and space, at most, proportional to the size of its input. This is true even if inputs are of arbitrary resolution q.

Proof: The condensation and collapsing portions require time and space proportional to the number of nodes in the input tree. Pointer maintenance in the condensation phase requires time proportional to the number of pointers.

A simple bookkeeping argument shows that the total work is linear in the number of nodes in the input tree. Each side of each leaf is stacked, at most, once. Divide neighbor finding operations into two classes: those from leaves to smaller leaves, and those to nonsmaller leaves. The operations in each class can number no more than four times the number of leaves.

Q.E.D.

X. A Linear Tree-for-Polygon Algorithm

The Outline-and-Color Algorithm: Input is a polygon. Output is a quad tree for the polygon. The algorithm consists of the outline algorithm followed by the coloring algorithm.

Theorem: Summing the requirements for each part reveals that the outline-and-color algorithm requires $O(v+p+q)$ time and space. Recalling the argument after the modified outline algorithm theorem, this is asymptotically optimal time, when resolution q is fixed.

References

[1] A. Klinger and C. R. Dyer, "Experiments on picture representation using regular decomposition," *Comput. Graphics and Image Processing*, vol. 5, pp. 68–105, Mar. 1976.

[2] T. Pavlidis, "The use of algorithms of piecewise approximations for picture processing applications," *Ass. Comput. Mach. Trans. Math. Software*, vol. 2, pp. 305–321, Dec. 1976.

[3] S. L. Horowitz and T. Pavlidis, "Picture segmentation by a tree traversal algorithm," *J. Ass. Comput. Mach.*, vol. 23, pp. 368–388, April 1976.

[4] S. L. Tanimoto, "Pictorial feature distortion in a pyramid," *Comput. Graphics and Image Processing*, vol. 5, pp. 333–352, 1976.

[5] —, "A pyramid model for binary picture complexity," in *Proc. IEEE Comput. Soc. Conf. Pattern Recognition and Image Processing*, Rensselaer Polytechnic Institute, NY, June 6–8, 1977, pp. 25–28.

[6] G. M. Hunter, "Computer animation survey," *Comput. and Graphics*, vol. 2, pp. 225–229, 1977.

[7] N. Negroponte, "Raster scan approaches to computer graphics," *Comput. and Graphics*, vol. 2, pp. 179–193, 1977.

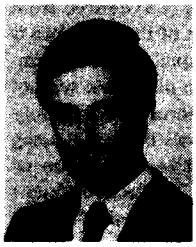
[8] G. M. Hunter, "Full-color television, from the computer, refreshed by run-length codes in main memory," Comput. Sci. Lab., Dep. Elec. Eng., Princeton Univ., NJ, Tech. Rep. 182, Apr. 21, 1975.

[9] A. Aho, J. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[10] D. E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, 2nd ed. Reading, MA: Addison-Wesley, 1973.

[11] G. M. Hunter, "Efficient computation and data structures for graphics," Ph.D. dissertation, Elec. Eng. Comput. Sci. Dep., Princeton Univ., NJ, June 1978.

[12] W. M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics*. New York: McGraw-Hill, 1973.



Gregory M. Hunter (S'70–M'78) was born in West Palm Beach, FL, on February 25, 1948. He received the B.A. degree in mathematics from Harvard University, Cambridge, MA, in 1970 and the M.S.E., M.A., and Ph.D. degrees in electrical engineering and computer science from Princeton University, Princeton, NJ, in 1972, 1973, and 1978, respectively.

For seven years, he was a Research Scientist with the RCA Laboratories, Princeton, NJ, where he specialized in electronic imagery, computer systems design, and human interfacing research. He is now with Decisions and Designs, Inc., McLean, VA.

Kenneth Steiglitz (S'57–M'64) was born in Weehawken, NJ, on January 30, 1939. He received the B.E.E. degree in 1959, the M.E.E. degree in 1960, and the Eng. Sc. D. in 1963, all from New York University, New York, NY.

Since September 1963 he has been with the Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ, where he is now Professor, teaching and conducting research in the computer and systems areas. He is Associate Editor of the *Journal of the Association for Computing Machinery* and the author of an introductory text on discrete systems.

Dr. Steiglitz is a member of Eta Kappa Nu, Tau Beta Pi, and Sigma Xi. He is Vice Chairman of the Digital Signal Processing Committee of the IEEE Acoustics, Speech, and Signal Processing Society.