Rochester Institute of Technology
RIT Digital Institutional Repository

Theses

3-18-1991

Quadtree algorithms for image processing

Jim Benjamin Isaac

Follow this and additional works at: https://repository.rit.edu/theses

Recommended Citation
Benjamin, Jim Isaac, "Quadtree algorithms for image processing" (1991). Thesis. Rochester Institute of Technology. Accessed from

This Thesis is brought to you for free and open access by the RIT Libraries. For more information, please contact repository@rit.edu.

QUADTREE ALGORITHMS FOR IMAGE PROCESSING

by
Jim Isaac Benjamin

(Sr. Project Engineer, Xerox Corporation, Webster, New York)

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

MASTER of SCIENCE in

Computer Engineering

Approved by:

Ronald G. Matteson, Ph.D. (Committee Chairman)
Associate Professor, Department of Computer Engineering

Stanislaw Radziszowski, Ph.D. (Committee Member)
Associate Professor, School of Computer Science and Information Technology

Conger W. Gable, Ph.D. (Committee Member)
Vice President, Strategy and Planning, Xerox Engineering Systems

Roy S. Czernikowski, Ph.D. (Department Head)
Professor, Department of Computer Engineering

DEPARTMENT OF COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK
MARCH 1991

TITLE OF THESIS: Quadtree Algorithms for Image Processing.

I, Jim Isaac Benjamin, hereby grant permission to the Wallace Memorial Library of the Rochester Institute of Technology to reproduce this thesis in whole or in part. Any reproduction will not be for commercial use or profit.

DATE: 4/18/91

ACKNOWLEDGEMENTS

In the preparation of this thesis, I received excellent help from many individuals. Thus in the category of support an encouragement, I would like to thank my expectant-wife Kay, who kept our home running smoothly while I became possessed with the preparation of this document. I would also like to extend special thanks to our parents for sharing their love with us.

A second category of help was administrative and technical. I am extremely fortunate to have worked with Ronald Matteson, Ph.D. (Associate Professor, Rochester Institute of Technology, Department of Computer Engineering), in the preparation of this thesis. His expertise in the area of Image Processing Algorithms provided the right advise for the research content, as well as the hardware and software development.

A third category of help was from my thesis committee. I am indebted to Stanislaw Radziszowski, Ph.D. (Associate Professor, Rochester Institute of Technology, School of Computer Science and Information Technology) and Conger Gabel, Ph.D. (Vice President, Strategy and Planning, Xerox Engineering Systems), each reviewed this thesis, and their suggestions were acknowledged.

The final category is sponsorship. I would like to acknowledge the Rochester Institute of Technology, Department of Computer Engineering which provided an environment in which many ideas were developed and a major portion of the work was accomplished. I am also grateful to the Department's support throughout my graduate level studies. Last but certainly not least, I am equally grateful to Xerox Corporation for providing the resources and encouragement to pursue this endeavor.

Table of Contents

Table of Tables

Table of Figures

ABSTRACT

The issue of constructing a computer-searchable image encoding algorithm for complex images and the effect of this encoded image on algorithms for image processing are considered. A regular decomposition of image (picture) area into successively smaller bounded homogeneous quadrants is defined. This hierarchical search is logarithmic, and the resulting picture representation is shown to enable rapid access of the image data to facilitate geometric image processing applications (i.e. scaling, rotation), and efficient storage. The approach is known as quadtree (Q-Tree) encoding. The applications in this thesis are primarily to grayscale pixel images as opposed to simple binary images.

1. INTRODUCTION

1.1 HISTORY AND BACKGROUND OF Q-TREES:

The motivation for this investigation is to explore the usage of a quadtree (Q-Tree) data structure for image processing, compaction, segmentation, progressive transmission, and storage applications. The methods and algorithms discussed were collected from a wide variety of literature, and are, of course, not limited to this presentation. A quadtree data structure is defined as the recursive division of an image into four disjoint congruent square regions called quadrants, whose union covers the original image space.

There are three general methods to the representations of a quadtree encoded image data base:

(1) a tree structure that uses pointers

(2) listing nodes in preorder or postorder traversal of the tree structure, representing them as NW, NE, SW, SE. Nodes can be terminal (identified by a color), or non-terminal.

(3) locational codes

The scope of this paper focuses on approach (2). Methods (1) and (3) are covered in great detail by **[Samet-4]** and **[Gargantini-1,2]**.

A very important issue involved with the processing of images is called **segmentation**. The **segmentation** task involves identifying subsets or extracting objects of interest from a scanned (digitized) image. Horowitz and Pavlidis **[Pavlidis-1]** show how to segment a picture using traversal of a quadtree. The topic of segmentation will be discussed further in section 3. Other typical requirements for the processing of images are geometric transformations, progressive transmission, and compression. It is shown in this paper that the Q-Tree algorithms selected can help facilitate these requirements.

Section 2 is devoted to the description of the Q-Tree (Regular Decomposition) image processing algorithm. This data structure and its potential for efficient handling of image processing functions will be discussed. Definitions of terms are developed to facilitate later discussion of the Q-Tree data structure.

In section 3 the Complexity Theorem is stated along with several Image Processing algorithms that may be applied to the Quadtree data structure.

In section 4 the Image Processing algorithms are applied to complex portions of several images. Computational examples will be discussed, along with

measurements of storage, and computation time. These results will be compared with similar measurements on unencoded images.

Section 5 summarizes some key advantages and disadvantages of the use of the Q-Tree hierarchical data structure and suggest future areas of exploration.

The Appendix provides the complete C-language source code for "**Videorw.c**", "**Xform.c**" and "**Thesisqt.c**" programs.

At this time we will briefly introduce the concept of Q-Tree (regular) decomposition which will be further detailed in section 2.

The concept of the decomposition algorithm applied to binary images is stated as follows:

1. Represent a digitized image as spatial subsets of different sizes marked either "intermediate node" or "final node".
2. Discard picture elements (pixels) that belong to "final node" subsets [**Klinger**].

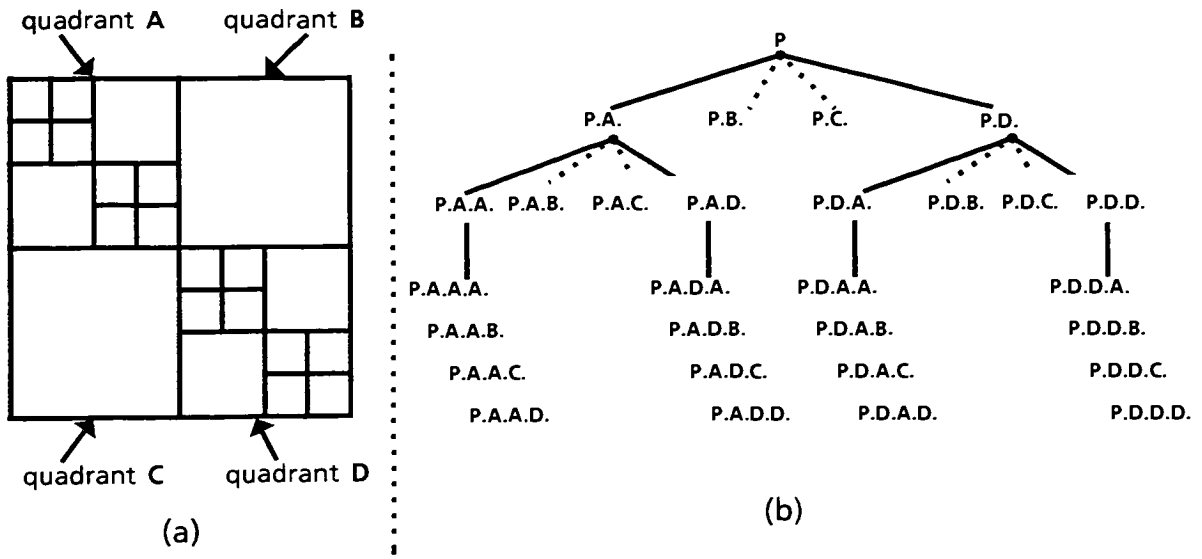


FIG. 1.1. (a) A diagonally oriented object, and (b) Q-tree representation of (a), with **P** at the root and the reduced picture of **P** at the leaves.

A variation to this concept is implemented in this Thesis in the following manner: First, the entire digitized image (a square array of gray level or light intensity values)

is a quadrant. Secondly two-possibilities exist when the Q-Tree algorithm looks at a quadrant:

1. The present level is a large homogeneous area. Such an area may eliminate the investigation of all possible levels below the present or parent node, thus this area is stored as a "final node" in the data structure without loss of image information.
2. Nonhomogeneous information is found in the quadrant. Many lines, vertices, and regions with diverse textures are examples which may be found. Such an area is classified as a mixed node and must be investigated at all possible levels below the present or parent node. Thus this area should be saved in the data structure as an "intermediate node".

Thus, if the algorithm fails to make a decision about a picture quadrant, it is subdivided and then each of the four subquadrants is processed by the same procedure. The subdivision process is applied recursively until either no failures occur or else the quadrant size becomes equal to the smallest resolvable point of the image (i.e. one pixel) [**Klinger**]. **Fig. 1.1** and **Fig. 1.2** are illustrations of this concept.

The process of regular decomposition is thus a logarithmic search for picture areas where there is data ("intemediate node" & "final node") present (i.e. the amount of space required to store a Q-Tree data structure, is proportional to the log of the image diameter n ($\log 2^n = n$), where n is also a representation of levels within the data structure. The algorithm builds a tree by hierarchically examining a picture's contents. Each area is assigned an importance based on how informative it is judged to be.

As mentioned above the prime motivations for the development of the ***Q-Tree*** data structure are the reduction in processing time of image processing primitives, the enabling of progressive transmission and compaction (reduction of storage space / transmission time) with respect to gray-scale images.

One should note however, that there are some pathological cases; for example the encoding of a checkerboard image with the Q-Tree data structure would not be very efficient, as shown in **Fig. 1.3**.

If image compression is of primary interest, there are other methods such as run length encoding, adaptive hierarchical coding and bintree encoding. Run length coding is discussed briefly in section 2; however the other aforementioned methods are beyond the scope of this Thesis.

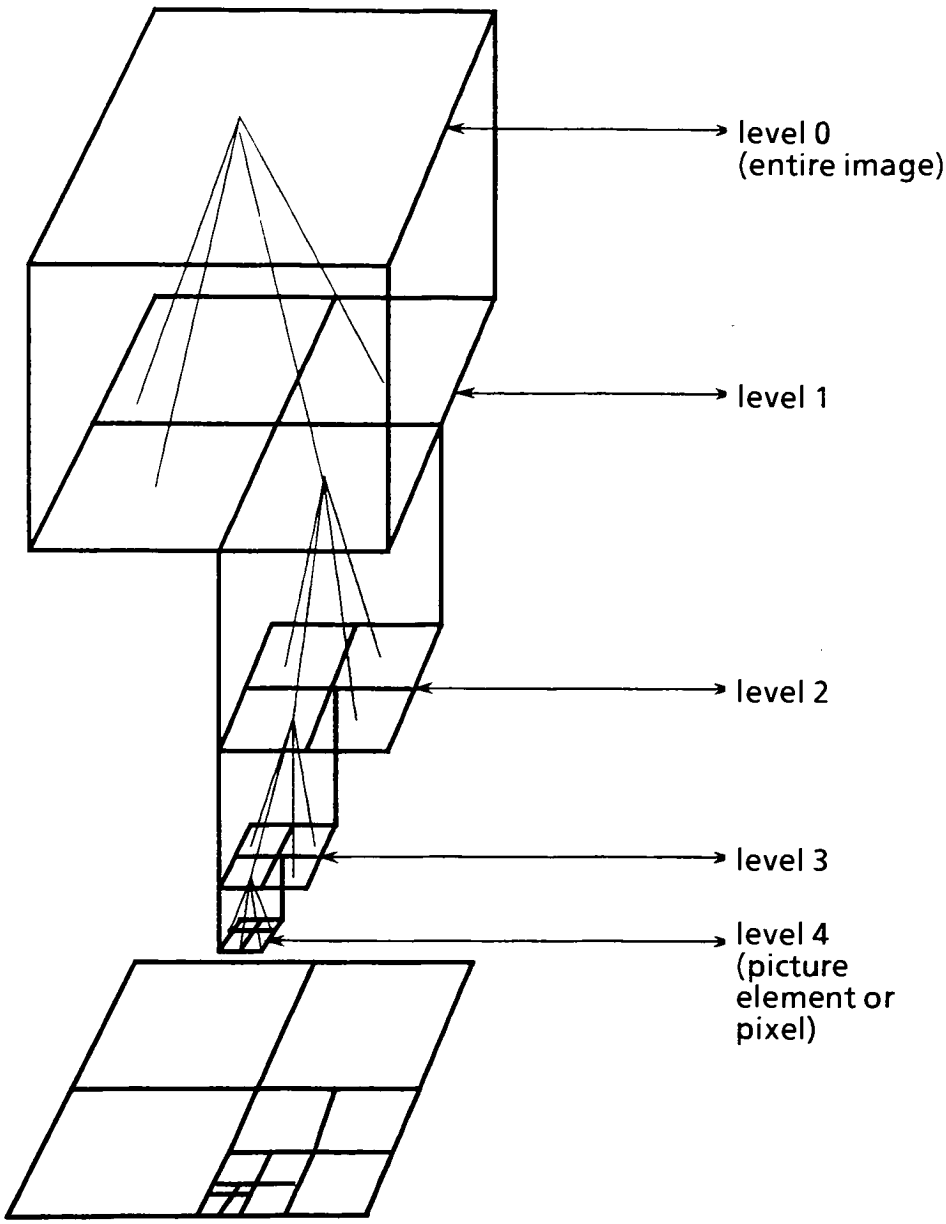


FIG. 1.2. A picture of a Q-tree.The entire image corresponds to level zero, and single pixels correspond to level n [**Hunter-1**].

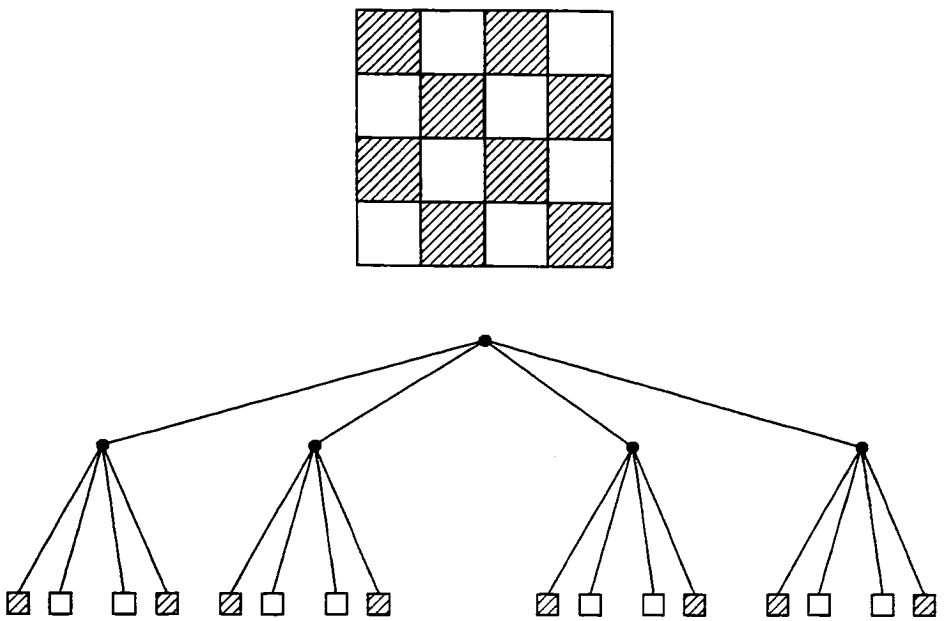


FIG. 1.3. The worst case for a quadtree of a given depth in terms of storage requirements occurs when image corresponds to a checkerboard pattern [**Sam**et-2].

2. METHODS OF IMAGE ENCODING

2.1 RUN LENGTH ENCODING:

The array is a common form of data structure for image encoding. However, for large images the amount of storage required is often deemed excessive [**Samet-1,2**]. Many ways have been devised for reducing the size of the image database, by encoding the data in some way. As an example run length coding is one method for reducing the number of required memory cells. The principle is to collect runs of adjacent picture elements of equal brightness and to encode the length of the run and the brightness. Run length codes are particularly useful for raster-like devices, such as television. Another application of run length coding is the generation of halftone pictures on graphic computer terminals, in this application run length is used to improve the limited capacity of the refresh memory. Arnemann presented a solution to this display problem, runs were collected and represented by one basic pattern which covered several picture elements [**Arnemann**]. An example of such a pattern is shown in **Fig. 2.1**.

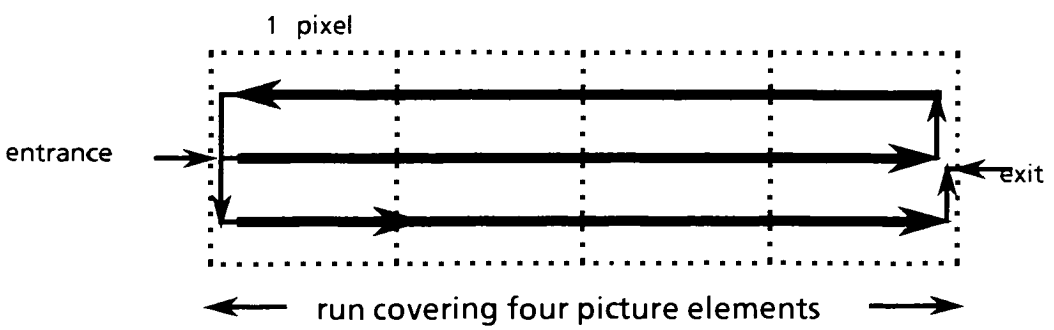


FIG. 2.1. Basic pattern for run length coded halftone

However, for large images the amount of storage required is still deemed excessive. The interested reader is encouraged to consult [**Arnemann and Limb**], for an in-depth analysis of run length coding applications to halftones display (i.e. television).

2.2 Q-TREE ENCODING:

Another data structure which can be used to reduce the size of the image database is Q-Tree. Q-Tree encoding also results in less computation time for image processing than doing it on the unencoded image. A common approach in preparing an image for Q-Tree encoding, is to threshold the picture itself. Intensity of gray level is the coarse picture importance parameter. [This is the natural approach in several cases: automatic character recognition (black characters, white page); reconnaissance photographs (clouds whiter than terrain)]. The fixed threshold technique is the simplest to implement in that the decision rule is simply stated:

$$\text{If } I_{xy} > T, \text{ then } P_{xy} = R,$$
$$\text{else } P_{xy} = 0.$$

The threshold T is usually in the neighborhood of $R/2$ [**Jarvis**] Often, a high-contrast text or line image can be successfully displayed by this technique; however, fixed thresholding of continuous tone subjects generate output images consisting of featureless bands of lit cells that bear only an outline resemblance (see **Fig. 4.3**) to the original (see **Fig. 4.4**). A threshold, $T = 128$ was used to produce **Fig. 4.3**.

A slight improvement over the fixed threshold is to replace T by equally distributed random numbers over the range 0 to R with a new random number generated for each I_{xy} [**Jarvis**].

The contribution of this paper is in the area of gray-scale images. Thus gray images with 256 gray-levels are thresholded using the following rule:

$$\text{if } (0 <= I_{xy} <= 31), \text{ then } P_{xy} = BLACK,$$
$$\text{else if } (32 <= I_{xy} <= 63), \text{ then } P_{xy} = GRAY1,$$
$$\vdots$$
$$\text{..else if } (192 <= I_{xy} <= 223), \text{ then } P_{xy} = GRAY6,$$
$$\text{else } P_{xy} = WHITE.$$

where, **BLACK = 0, GRAY1 = 63, . . . ,GRAY5 = 191, GRAY6 = 223, WHITE = 255**

The above is an example of thresholding an image that has 256 gray levels reduced to 8 gray levels. A different leaf criterion could be used; e.g. any number of leaf levels from 2 to 256 could be used for the leaf criterion.

2.2.1 Tree-structures and Digitized Images.

Digitized images are commonly represented as square arrays, where each element of the array contains some information about the corresponding area of the image space being viewed. It is well known that such an array can be thought of as a special case of a tree structure **[Knuth]**. For example, **Figs. 2.2** and **2.3** give two representations of a *4 X 4* matrix, the latter explicitly displaying the relationships of elements in the same row in a tree. However, this tree does not display all the structure of the matrix (column relationships omitted), and a similar tree based on column relations omits row information. Since picture elements are usually obtained from horizontal raster scan lines, row-oriented data structures and algorithms utilizing line by line 'slices' are common in image processing **[Klinger]**.

A(1,1)	A(1,2)	A(1,3)	A(1,4)
A(2,1)	A(2,2)	A(2,3)	A(2,4)
A(3,1)	A(3,2)	A(3,3)	A(3,4)
A(4,1)	A(4,2)	A(4,3)	A(4,4)

FIG. 2.2. *4 X 4* matrix A.

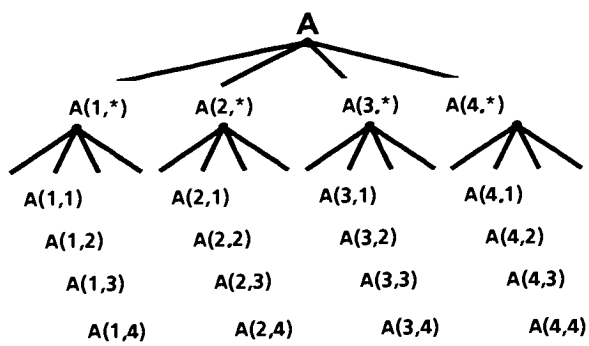


FIG. 2.3. Tree of matrix A decomposed by rows

However, general properties of image classes are unlikely to be represented in linear (row) form, since key geometrical, topological, structural, and metric constraints are usually involved. Because a data structure should reflect prior knowledge about properties contained within the data base, representations which facilitate computer search for picture properties in areas would be preferable. A tree structure which directly reflects areas (rather than rows) for the above *4 X 4* matrix is shown in **Figs. 2.4** and **2.5** [**Klinger**].

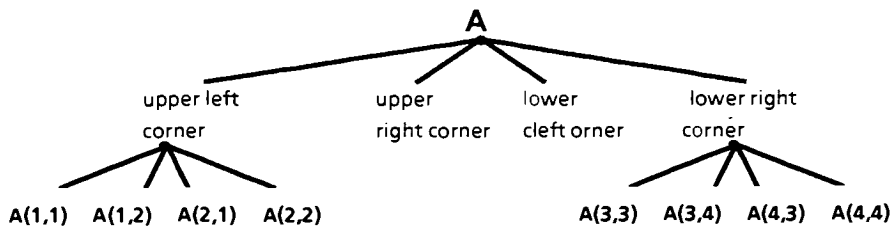


FIG. 2.4. Tree partition of matrix A by areas

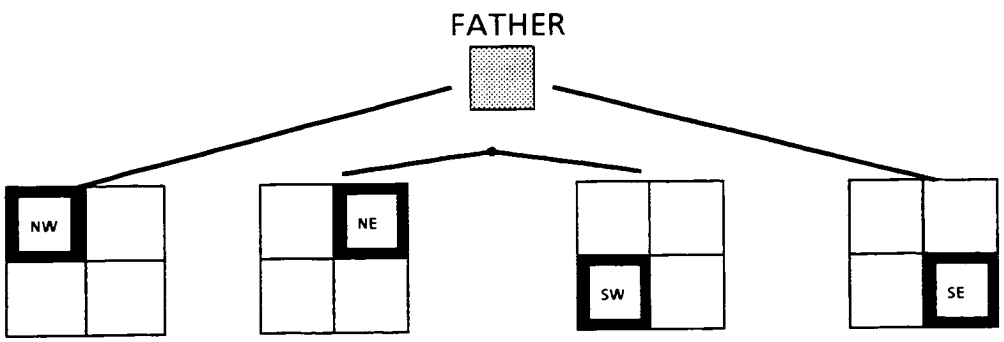


FIG. 2.5. One level of Q-tree (regular Decomposition).

This structure can be conveniently represented for computation by Dewey decimal [**Knuth**] (library classification) notation:

1 A
1.1 upper left corner
1.1.1 A(1,1)
1.1.2 A(1,2)
1.1.3 A(2,1)
1.1.4 A(2,2)

1.2 upper right corner

1.4.4 A(4,4)

This leads to a decision to build algorithms which traverse an encoded image's quadtree by preorder [**Knuth**]. Preorder is defined as a listing of the root node first followed by the root node's descendants. Thus, visiting the nodes of the tree by preorder permits viewing of all successors, because the root node will be examined first followed by all subtrees of the root. Since successors represent smaller picture zones, the tree and a preorder traversal algorithm break the overall scene analysis task into the solution of a series of subproblems. Since the postorder approach allows for the viewing of each level, this algorithm is used to derive the results for this presentation. This approach allows for the progressive processing / transmission of images in successively greater detail, allowing the process to be terminated at any "satisfactory" level. We'll see some examples of this later.

2.2.2 Tree Structures and Q-tree (Regular Decomposition) Algorithms

The data structure of **Fig. 2.4** is the regular decomposition of the picture area by successive partitioning into quadrants. The result is a tree where each father node has four successor nodes. These are ordered arbitrarily by: upper-left (NW), upper-right (NE), lower-left (SW), lower-right (SE). This is illustrated in **Fig. 2.5** for a single level of decomposition and in **Fig. 2.6** for six levels, and formalized by the following definitions [**Klinger**].

Q-tree: A ***Q-tree*** is a finite set of nodes which is either empty or consists of a quadrant and at most four disjoint ***Q-Trees*** [**Klinger**]. This recursive definition of ***Q-Tree*** is analogous to Knuth's definition of binary tree [**Knuth**], except here each node has exactly four equal-sized subtrees (quadrants) or children. The Q-Tree data structure successively subdivides a bounded region until homogeneous blocks are obtained.

Root node: **Root node** represents the entire array [**Samet-4**].

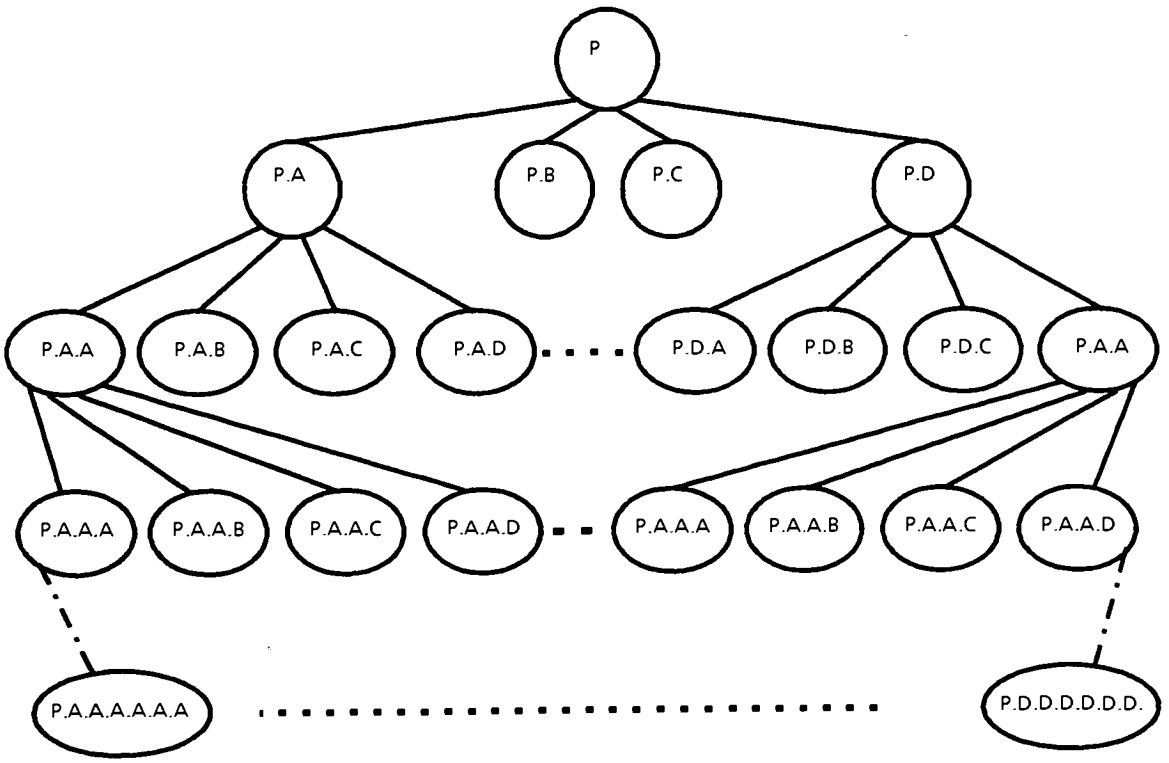
Quadrant: A ***quadrant*** is represented by one of the four sons of a node [**Samet-4**]. Quadrants are nodes which can be labeled by *NW, NE, SW, SE.*

Picture: A ***picture*** is a quadrant labeled *P.*

Leaf quadrant: A ***leaf quadrant*** is a terminal node corresponding to a block of the array for which no further subdivision is necessary [**Samet-4**].

Reduced picture: A ***reduced picture*** is the set of all leaf quadrants in a ***Q-Tree*** whose root is labeled *P*. 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 descendants have the same color, the picture can be more compactly represented by coloring the parent, and removing all the children [**Hunter-1**].

Regular Decomposition: A ***regular decomposition*** of a picture *P* is a ***Q-tree*** with *P* at the root and the reduced picture of *P* at the leaves [**Klinger**].



NUMBER OF NODES AT LEVEL $i = 4^i$

MAXIMUM TREE DEPTH FOR 64 X 64 ARRAY = 6

TOTAL NUMBER OF NODES IN TREE = $\sum_{i=0}^{6} 4^i$ = 5461 NODES

FIG. 2.6. Complete Quad tree to level 6: Leaf nodes are pixels if P is digitized to a 64 X64 array.

Informally, a quadrant is a square image area, characterized by relative position, size, and display value (intesity). The absolute location of a quadrant in the picture is obtainable from its Dewy decimal label [Klinger and Knuth] or its position in the pre (post)-order search.

To compute the poperties of the four successor subquadrants we may apply Rule 1 describe below, which divides each side of the quadrant in half and assigns new vertices to the subquadrants forming integer-values of boundary line positions and hence disjoint subquadrants (see Fig. 2.7) [Klinger]. Where, $I(i,j)$ is defined as the intensity value at desired pixel location.

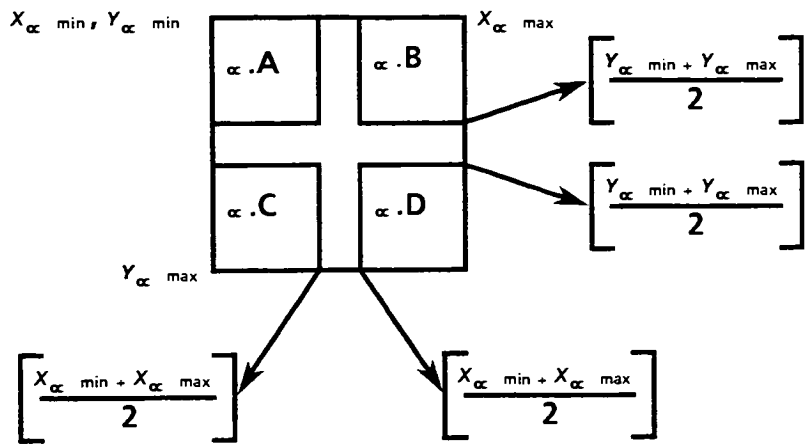


FIG. 2.7. Boundaries of four subquadrants of quadrant α defined by Rule 1.

RULE 1. Let α be the label of a quadrant whose properties are :

$$C_\alpha = \langle I(i,j), x_\alpha \text{ min}, x_\alpha \text{ max}, y_\alpha \text{ min}, y_\alpha \text{ max} \rangle.$$

Then the root-labels and properties of its four subquadrants are:

1. α.A

$$C_{\alpha.A} =$$

$$\left\langle \sum_{j = y_{\alpha.A \text{ min}}}^{y_{\alpha.A \text{ max}}} \sum_{i = x_{\alpha.A \text{ min}}}^{x_{\alpha.A \text{ max}}} I(i,j),\ x_\alpha \text{ min},\ \left[\frac{X_\alpha \text{ min} + X_\alpha \text{ max}}{2}\right],\ y_\alpha \text{ min},\ \left[\frac{Y_\alpha \text{ min} + Y_\alpha \text{ max}}{2}\right] \right\rangle$$

2. $_{\alpha}$.C

$C_{\alpha.C} =$

$$\left\langle \sum_{j=y_{\alpha.C\,min}}^{y_{\alpha.C\,max}} \sum_{i=x_{\alpha.C\,min}}^{x_{\alpha.C\,max}} I(i,j),\ x_{\alpha}\ min,\ \left[\frac{X_{\alpha\ min}+X_{\alpha\ max}}{2}\right],\ \left[\frac{Y_{\alpha\ min}+Y_{\alpha\ max}}{2}\right],\ y_{\alpha}\ max \right\rangle$$

3. $_{\alpha}$.B

$C_{\alpha.B} =$

$$\left\langle \sum_{j=y_{\alpha.B\,min}}^{y_{\alpha.B\,max}} \sum_{i=x_{\alpha.B\,min}}^{x_{\alpha.B\,max}} I(i,j),\ \left[\frac{X_{\alpha\ min}+X_{\alpha\ max}}{2}\right],\ x_{\alpha}\ max,\ y_{\alpha}\ min,\ \left[\frac{Y_{\alpha\ min}+Y_{\alpha\ max}}{2}\right] \right\rangle$$

4. $_{\alpha}$.D

$C_{\alpha.D} =$

$$\left\langle \sum_{j=y_{\alpha.D\,min}}^{y_{\alpha.D\,max}} \sum_{i=x_{\alpha.D\,min}}^{x_{\alpha.D\,max}} I(i,j),\ \left[\frac{X_{\alpha\ min}+X_{\alpha\ max}}{2}\right],\ x_{\alpha}\ max,\ \left[\frac{Y_{\alpha\ min}+Y_{\alpha\ max}}{2}\right],\ y_{\alpha}\ max \right\rangle$$

A common program which performs regular decomposition implements the definitions:
A picture *P* to be processed begins as an ***n X n*** digitized image; Rule 1 is applied to it recursively and it is decided how the four successors (quadrants) **Q-Trees** should be added to the data structure. If quadrant = "terminal", then y's successor **Q-Tree** with root labeled x consists of the single leaf quadrant x. If quadrant = "not terminal", then y's successor **Q**-Tree with root labeled x is a **Q**-Tree consisting of at least four nodes; here, Rule 1 will be applied to quadrant x as it was to its parent y. The final result is a tree structure with *P* as the root node, "not terminal" quadrants making up the nonterminal nodes and "terminal"quadrants comprising the leaf nodes.

The data structure can also be described as follows: each node is represented as a record of type node containing six fields. The first five fields contain pointers to the node's father and its four sons, which correspond to the four quadrants:

a) If the node is a leaf node, it will have four pointers to the empty record (i.e., NIL).
b) If P is a pointer to a node and I is a quadrant, these fields are referenced as FATHER(P) and SON(P,I), respectively [**Samet-1**].
c) The sixth field, NODETYPE, describes the contents of the block of the image that the node represents:(i.e. black, white, gray$_i$ or mixed).

*Encode**quadtree** **Algorithm.***

From the discussions above the actual format of the algorithm used for this Thesis is as follows: the image is encoded using a breadth-first scheme [**Knuth**], all nodes (i.e., father, sons) are stored as a byte of data, and data beneath homogeneous regions are not stored (as mentioned before this facilitates the amount of space required for a given image). The NODETYPE described in this algorithm is either, BLACK, WHITE, GRAY1-6, or MIXED. The pseudo-code is illustrated below (see **"Thesisqt.c" in Appendix for complete C-language program listing**).

```
Encodequadtree (x, y, width)
{
    open output file to contain
        encoded quadtree image;
    determine maximum-level;
    for (level = 0; level < = maximum-level; level + 1)
    {
        if (level = 0){
            determine node (father) color-value;
            write value to output file;
        }
        else{
            perform recursive subdivision of image;
            keep track of ancestors (i.e., father, sons, brothers);
            write out values of each to output file;
        }
    }
}                                   /* end of Encodequadtree */
```

Decodequadtree Algorithm.

The decoding algorithm for a quadtree is analogous to the preceeding encoding algorithm. The pseudo-code is illustrated below, (**see "Thesisqt.c" in Appendix for complete C-language program listing**).

```
Decodequadtree (x, y, width)
{
    determine rotation;
    open input file that contains
        encoded quadtree image;
    determine maximum-level;
    for (level = 0; level < = maximum-level; level + 1)
    {
        if (level = 0)
        {
            read Encoded quadtree file;
            Display node value at current level;
        }
        else
        {
            determine width from scale factor;
            for (each element at level)
            {
                Determine quadrant position;
                Display node value at current level;
            }
        }
    }
}                                   /* end of Decodequadtree */
```

3. IMAGE PROCESSING APPLICATIONS

3.1 COMPLEXITY THEOREM:

We will now examine Q-tree algorithms used to implement linear transformations and their complexity.

The following theorem illustrate the complexity of the transformation algorithms:

Let ***p*** be the polygon perimeter.
Let ***q*** be the resolution (depth) of the perimeter.
Let ***m*** be the number of polygons in the input tree.
Let ***n*** be the number of nodes in the input tree.
Let ***s*** be the scaling factor.

Theorem 1.2. *Time and space of O(n + sp + mq) are sufficient for the quad tree transformation algorithm.*

The proof of **Theorem 1.2** is covered in great detail in [**Hunter-2**].

The corollary to **Theorem 1.2** is as follows:

Corollary 1.2. *Since m is no larger than n, time and space of O(nq + sp) are sufficient, or O(n + p) if the resolution q and the scale factor s of T are fixed* [**Hunter-2**].

Thus the Q-Tree data structure can produce approximately 33 percent more nodes than pixels [**Pavlidis-2**].

Aside from its implications on the storage requirements the quadtree complexity theorem also has a direct impact on the analysis of the execution time of algorithms. In particular, most algorithms that execute on a quadtree representation of an image instead of an array representation have an execution time that is proportional to the number of nodes in the image rather than the number of pixels.

Thus the cost of the transformation is simply the cost of visiting each node of the quadtree and creating a copy with the appropriate new data.

3.2 QUADTREE ROTATION ALOGRITHM:

For example a clockwise rotation by 90 degrees, the SW, NW, NE, and SE sons become NW, NE, SE, and SW sons rotated respectively, at each level of the quadtree [**Samet-3**]. An example of clockwise 90° rotation is illustrated in **Fig. 3.1**.

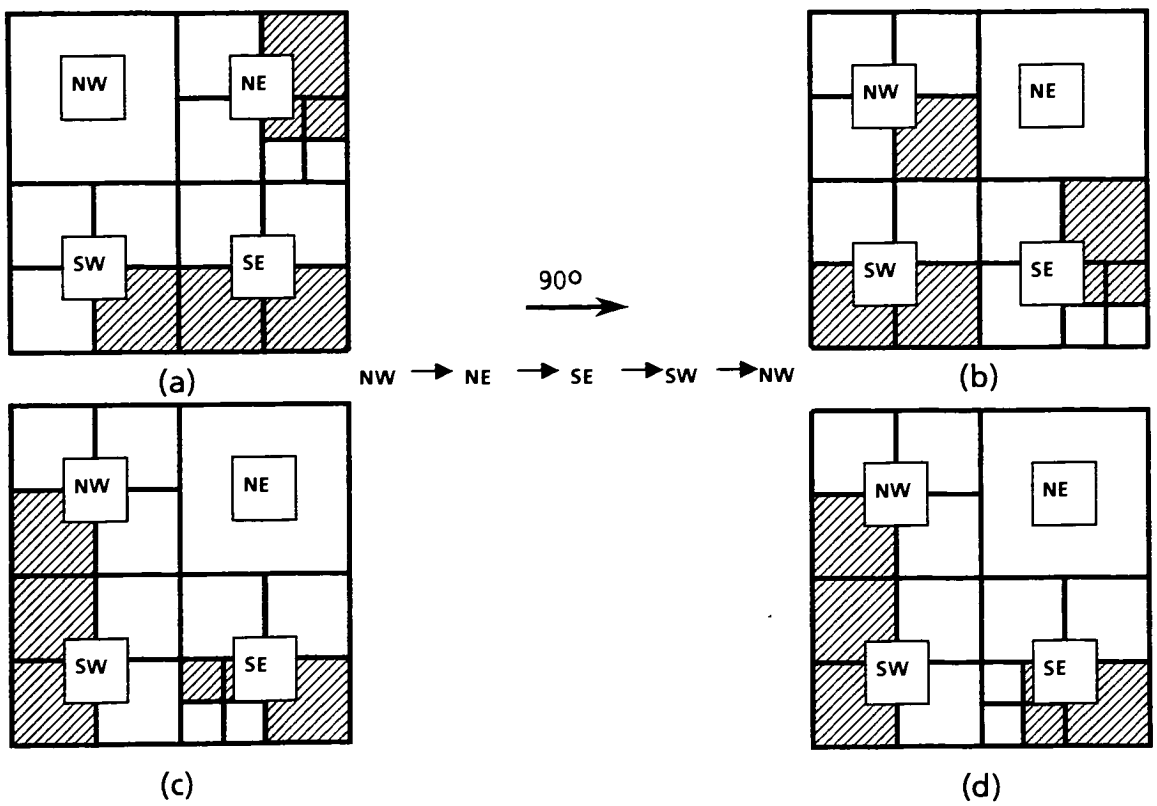


FIG. 3.1. Illustration of clockwise 90° rotation: (a) original image, (b) level 1 90° rotation, (c) level 2 90° rotation, (d) level 3 90° rotation.

In section 4 the above rotation algorithm is compared to the homogenous rotation transformation algorithm (**see "xform.c" in Appendix for complete C-language program listing**).

3.3 QUADTREE SCALING ALGORITHM:

To make an image represented by a quadtree half the size that it was originally, we need only create a new root and give that root three white sons and one son that was the original quadtree. To make the quadtree twice as big, we choose one of the subtrees to serve as the new root (e.g. the SW subtree), thus eliminating the remaining three subtrees. If a particular portion of the quadtree is to be doubled or halved in size, then a shift operation may have to be performed for the purpose of alignment. See **Fig. 3.2** for an illustration of scaling algorithm. In section 4 this scaling algorithm is compared to the homogenous scaling transformation algorithm (see **"xform.c" in Appendix for complete C-language program listing**).

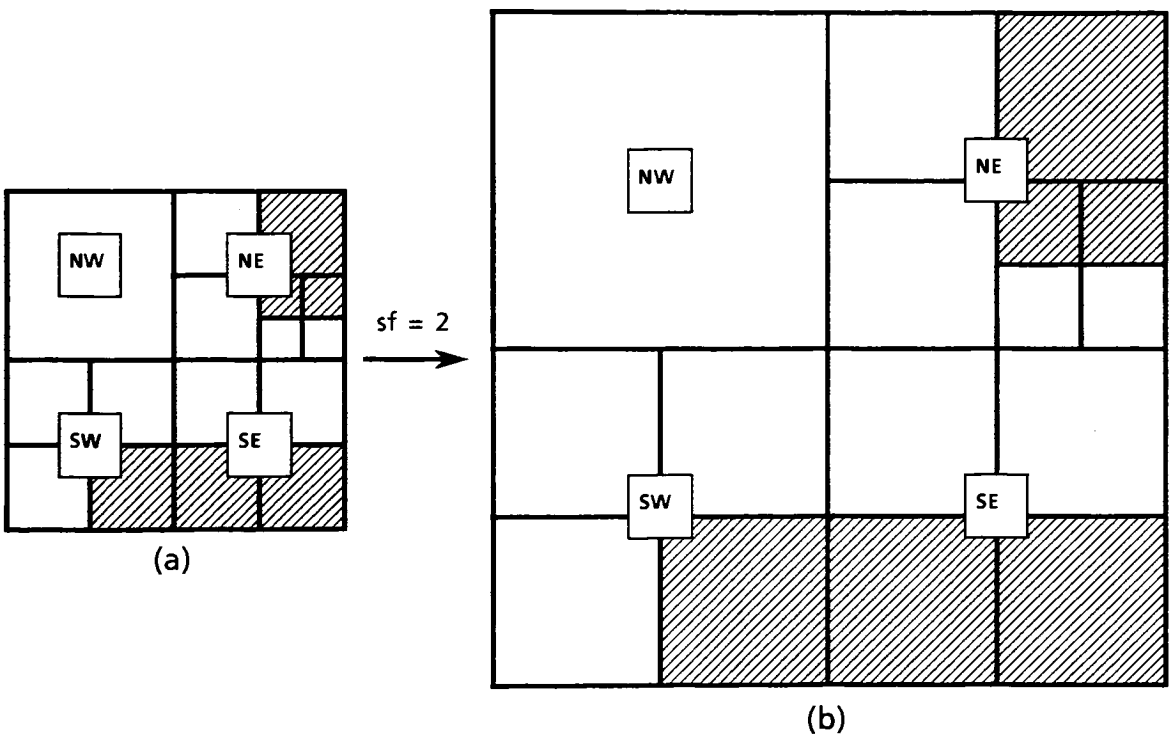


FIG. 3.2. Illustration of scaling algorithm for a 256 x 256 pixel region: (a) original 256 x 256 pixel region, (b) original region scaled by a factor of 2 to produce a 512 x 512 region.

3.4 PROGRESSIVE TRANSMISSION ALGORITHM:

Progressive transmission of images represented by quadtrees can be achieved by taking advantage of the above techniques for scaling by powers of two. Progressive transmission of an image enables the receiver to preview a reduced-resolution version of the image before seeing it in its entirety. This application facilitates browsing through a database of images. If the nodes of a raster quadtree are transmitted in breadth-first order, large leaf nodes are seen first [**Samet-3**]. This is one of the primary motivations for the breadth-first ordering of tree traversal.

3.5 SEGMENTATION AND QUAD TREE DATA STRUCTURE:

While edge detection and thresholding focus on the difference of pixel values, region growing looks for groups of pixels of similar brightness. In its simplest form, the method starts with one pixel, and then examines its neighbors in order to decide whether they have similar brightness. If they do, then they are grouped together to form a region. In this way regions are grown out of single pixels. **Fig. 4.20** illustrates the concept segmentation by extraction of pixels equal to gray5 (160-191). More advanced forms of segmentation do not start with pixels but with a partition of an image into a set of smaller regions. A uniformity test is then applied to each region, and if the test fails the region is subdivided into smaller elements. This process is repeated until all regions are uniform. Then regions are grown out of smaller regions, rather than pixels. The details of implementation depend strongly on the data structures used for representing an image. This particular application of segmentation is exactly suited for quad tree data structure [**Gonzalez**], and are commonly referred to as 'split-and-merge' algorithms.

Although the split-and-merge algorithm was not implemented in this Thesis, this algorithm is an extension of the use of region growing and uniformity tests for image segmentation. One starts with the nodes (squares) at some intermediate level of a quad tree. If a square is found to be nonuniform, then it is replaced by its four subsquares (split). Conversely, if four squares are found that form a uniform square, they are replaced by it (merge). This process may continue recursively until no further splits or merges are possible.[**Samet-2**].

4. DISSCUSION OF RESULTS

In the experimental analysis several gray-scale images were examined. These images covered a range of 256 gray-levels. The following list of equipment was used to carry out the experimental analysis:

- Frame Grabber PCVISION-PLUS:

  Frames

  Two frames by 512 x 512 x 8 pixels each (512k Bytes of memory)

  Pixel locations

  horizontal - 0< = X < = 511

  vertical - 0< = Y < = 511

  Area size ranges

  horizontal - 1< = X < = 512

  vertical - 1< = Y < = 512

  Banks - the banks are divided up in a 64K by 8 matrix, due to the 64K segment size of the IBM type of personal computer, each frame grabber memory is divided into four strips,

  Memory A - 0 to 3

  Memory B - 4 to 7

  Strip selection - bank desired is selected by loading 0, 1, 2, or 3; or 4, 5, 6, or 7 into the block select register, PC I/O address bits 5 -7.

  Memory Addressing - Frame grabber memory is addressed by selecting segment D000x, and offset 0 through FFFFx. The bank selected by the block select register will then be addressed.

- IBM compatible PC/AT-286 (6MHZ):

  IBM DOS ver. 4.0

  1.2 MB Floppy disk drive

  80 MB Hard disk

  Enhanced Graphics Adapter

  1MB RAM

  Math co-processor chip 80287

- Video Input is from a TV camera using standard RS-170 television signal.
- Laser Printer
- Microtek MS-200A 200 spi scanner

With the list of equipment above and a full-screen editor called **Epsilon**, the program listings in the **Appendix** were developed using the C - Language Microsoft Quick-C complier ver. 5.1. **Fig. 4.1** and **Fig. 4.2** describes the system's Hardware and Software Overview respectively.

The objectives of the experimental analysis were as follows:

1) Perform Q-tree encoding, as described
2) Homogeneous transformations (i.e. rotate, scale):
   Compare execution time of rotation and scaling functions on unencoded and encoded versions of images.
3) Determine compaction ratio of Q-tree gray-scale images
4) Segmentation:
   extract all parts of an image whereby nodes = $gray_i$

These four objectives were performed on the following common gray-scale images, U of R chapel, Boat, Forest and Girl. **Fig. 4.3 and 4.4** illustrates, the concept of selecting threshold value equal to half the maximum number of gray levels. **Fig. 4.5 - 4.7** are the Original .img 256 gray levels image of Boat, Forest and Girl respectively.

Fig. 4.8 - 4.11 are the Original .img 256 gray levels image reduced to 8 gray levels image of Boat, Forest and Girl respectively.

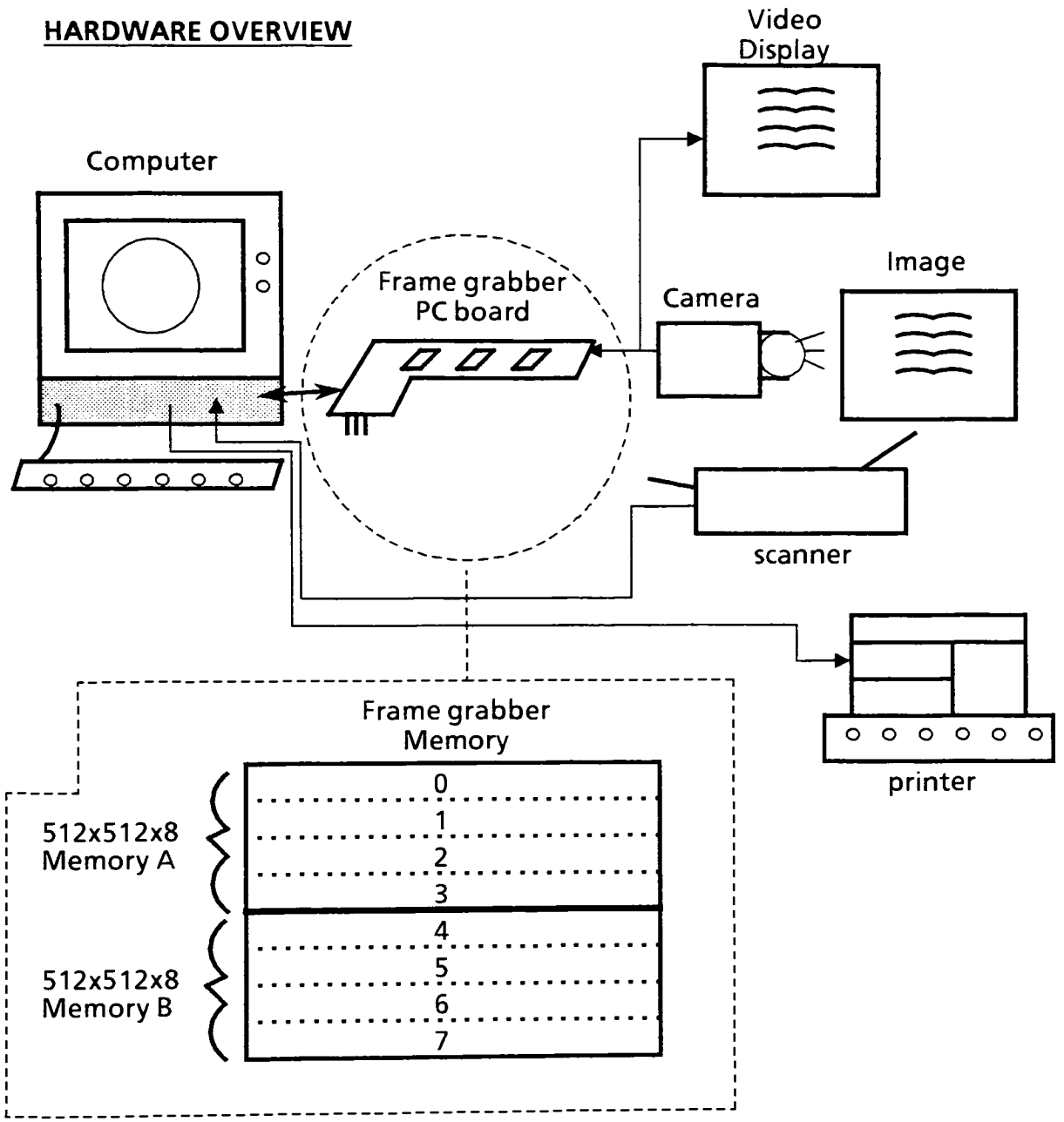


FIG. 4.1. Hardware Overview. Hardware and Memory block diagram used in the experimental analysis.

SOFTWARE OVERVIEW

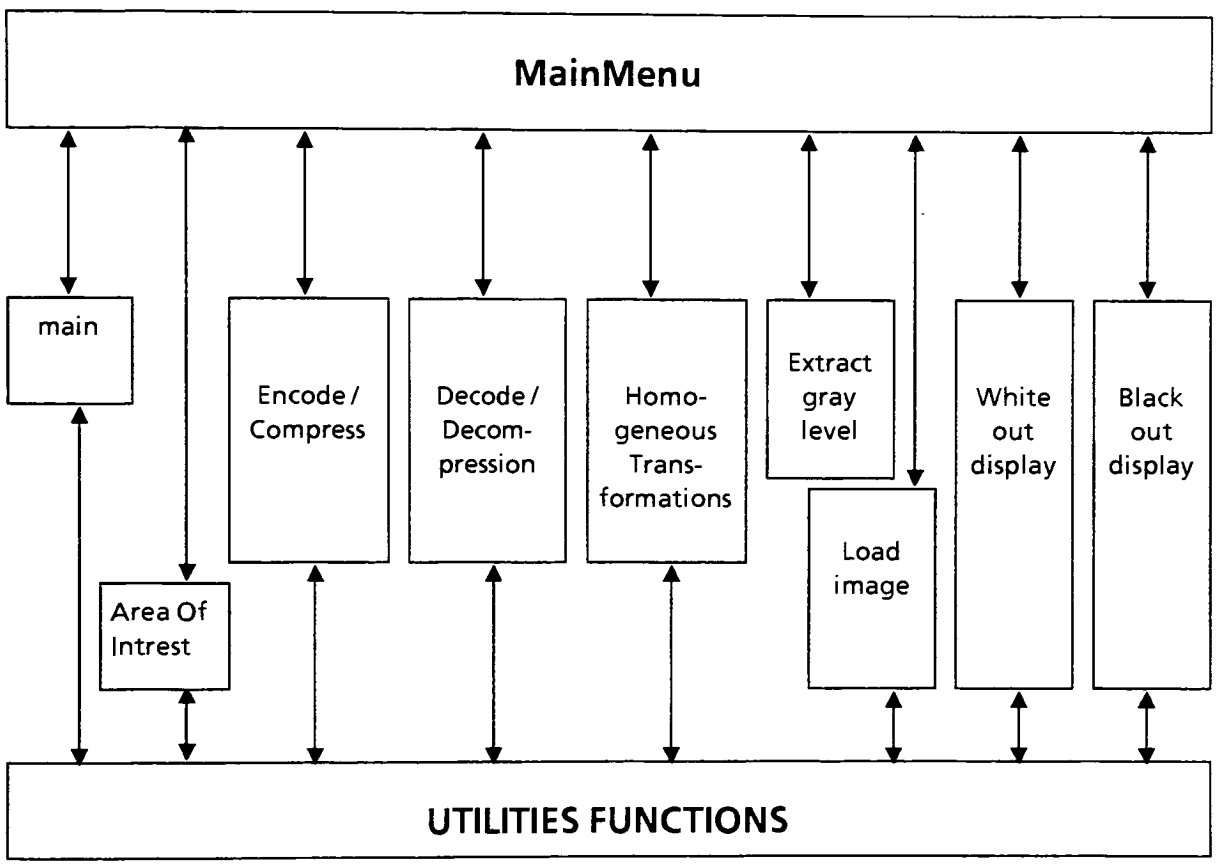


FIG. 4.2. Software Overview. A description of software modules relationships. **See Appendix for complete software listings of all modules.**

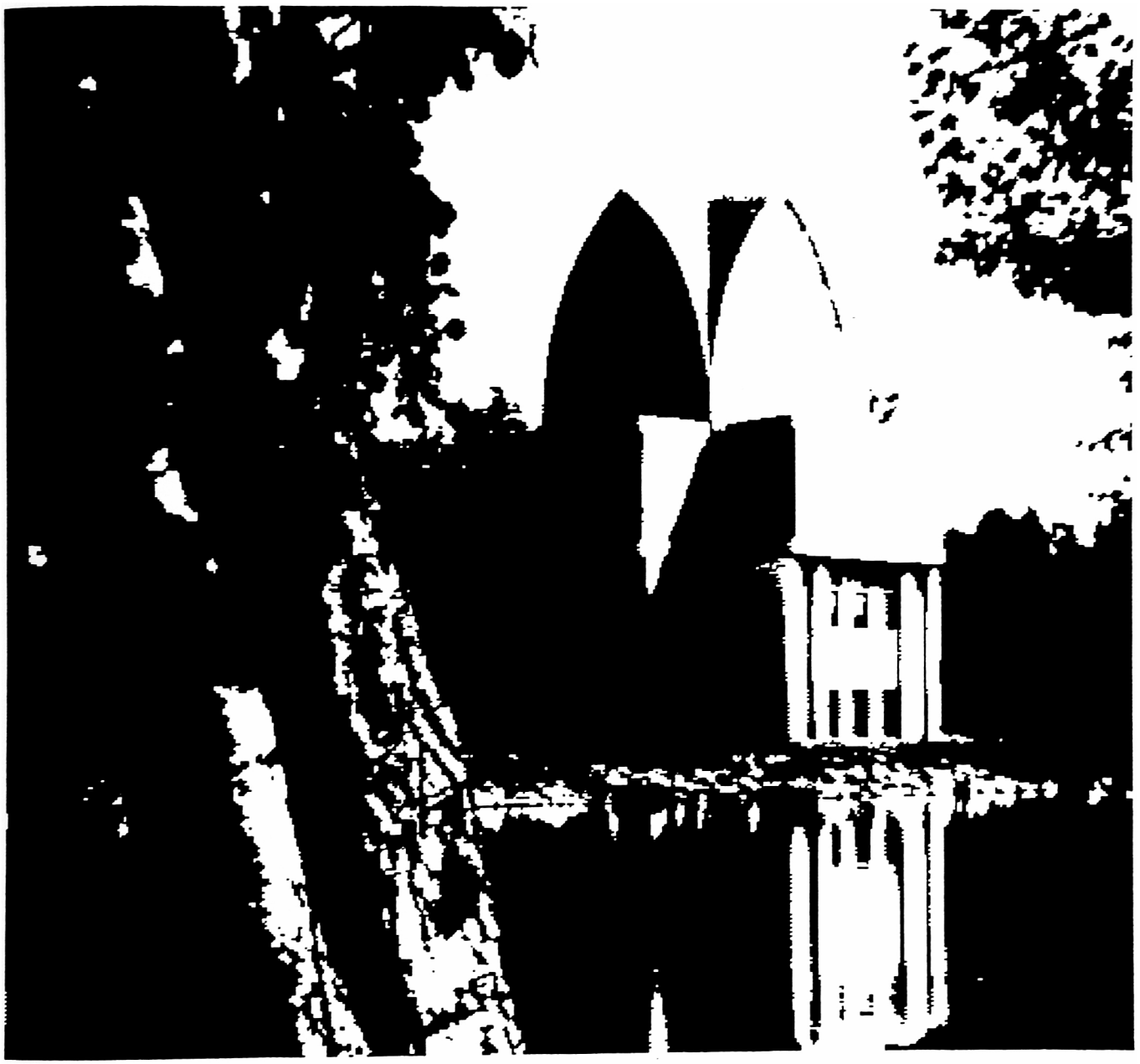


FIG. 4.3. Chapel.img with Threshold = 128.

```
number of lines:   476
number of pixels:  509
start line:        0
start pixel:       0
date:              02/20/91
```

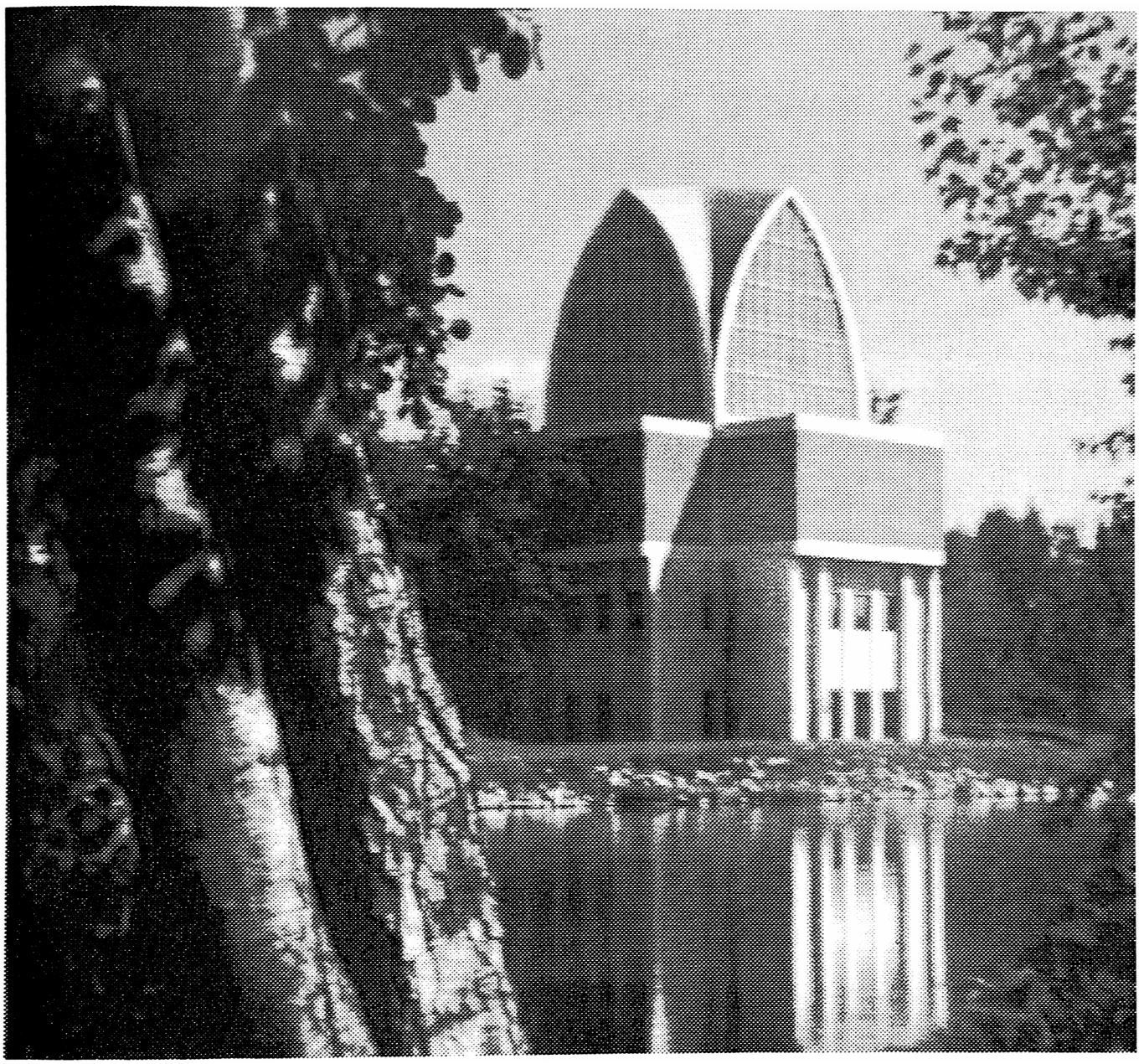


FIG. 4.4. Original Chapel.img 256 gray levels

number of lines: 476
number of pixels: 509
start line: 0
start pixel: 0
date: 02/20/91

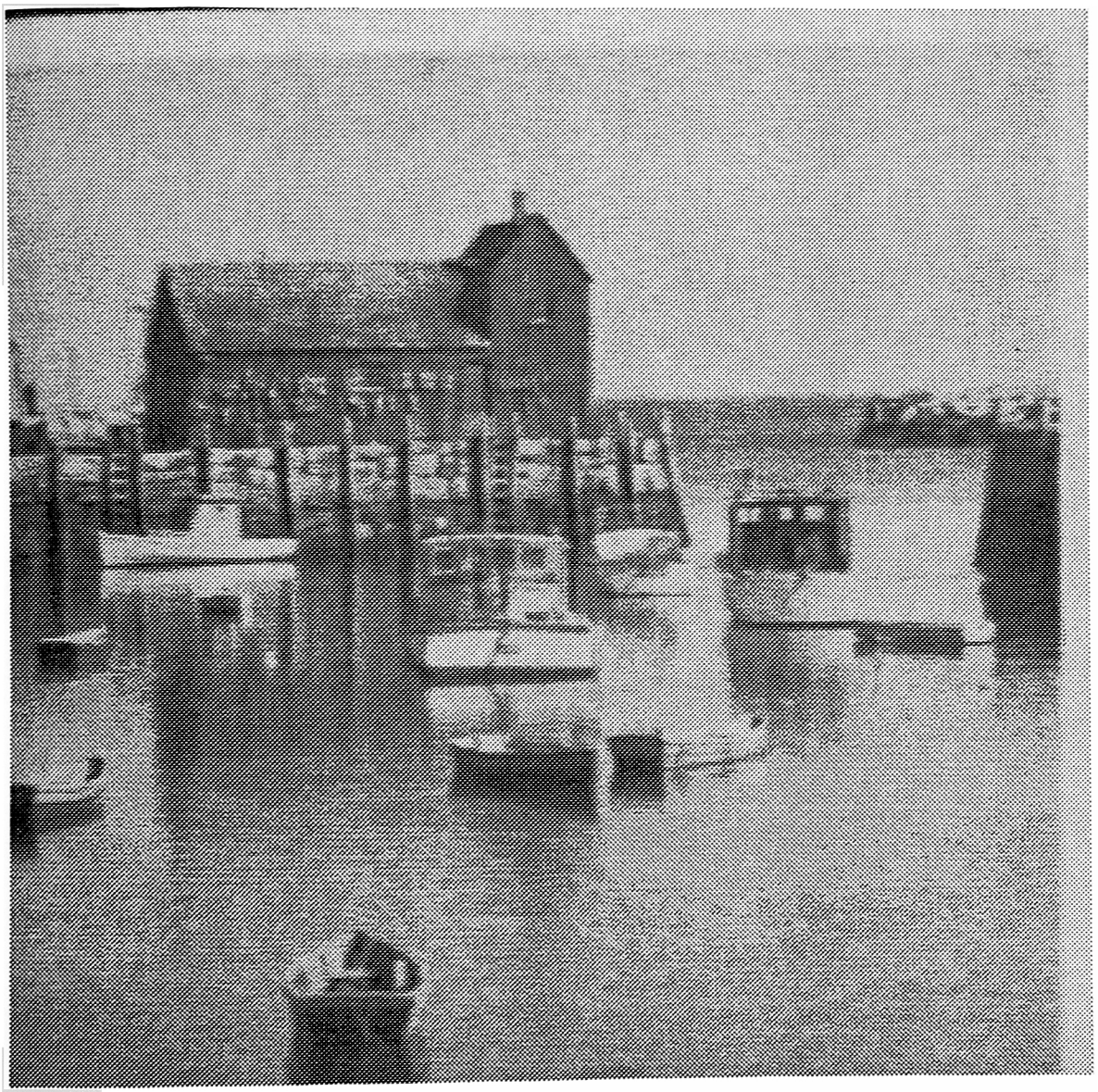


FIG. 4.5. Original Boat.img 256 gray levels.

number of lines: 441
number of pixels: 441
start line: 0
start pixel: 35
date: 02/20/91

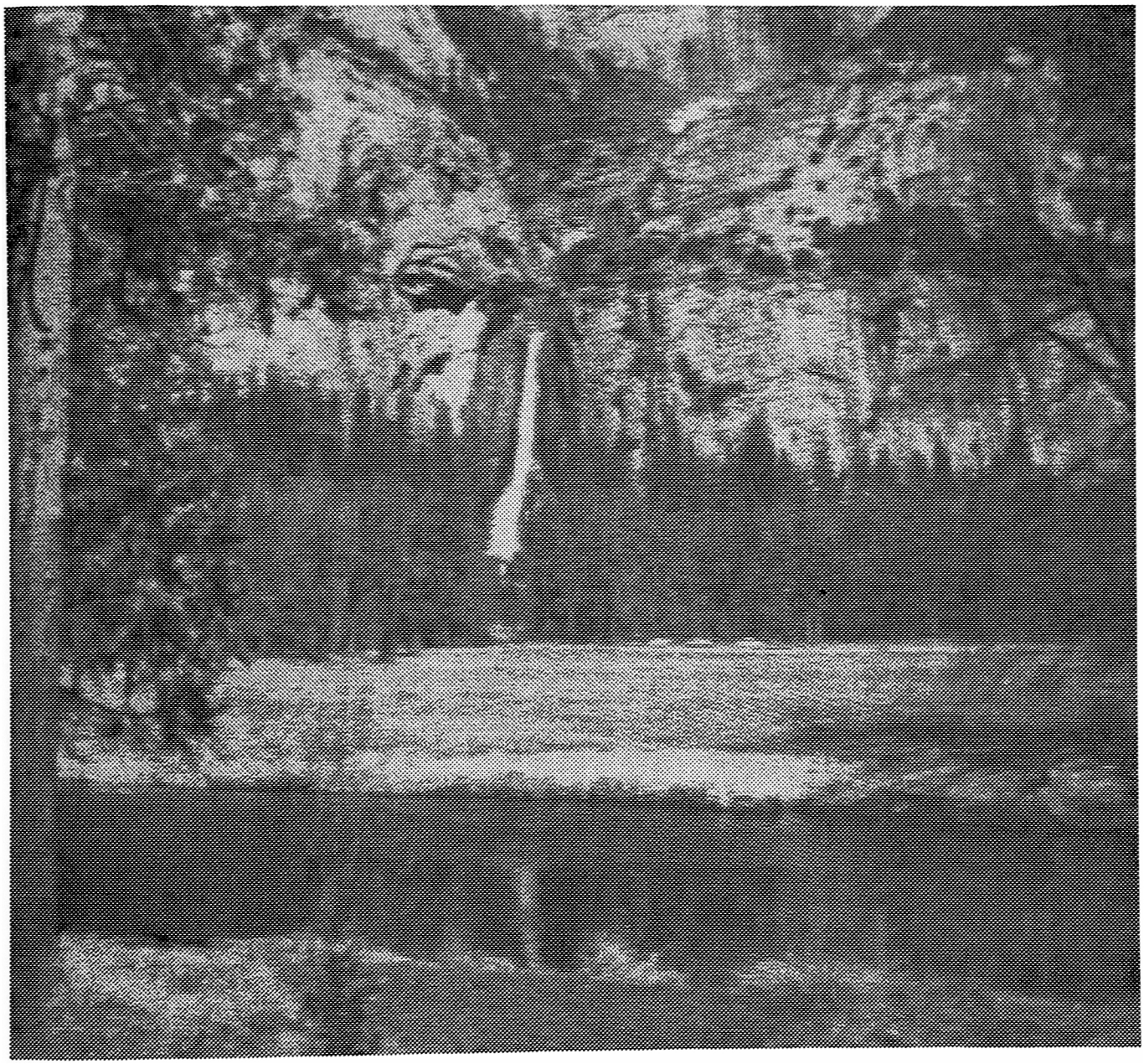


FIG. 4.6. Original Forest.img 256 gray levels.

number of lines: 476
number of pixels: 509
start line: 0
start pixel: 0
date: 02/20/91

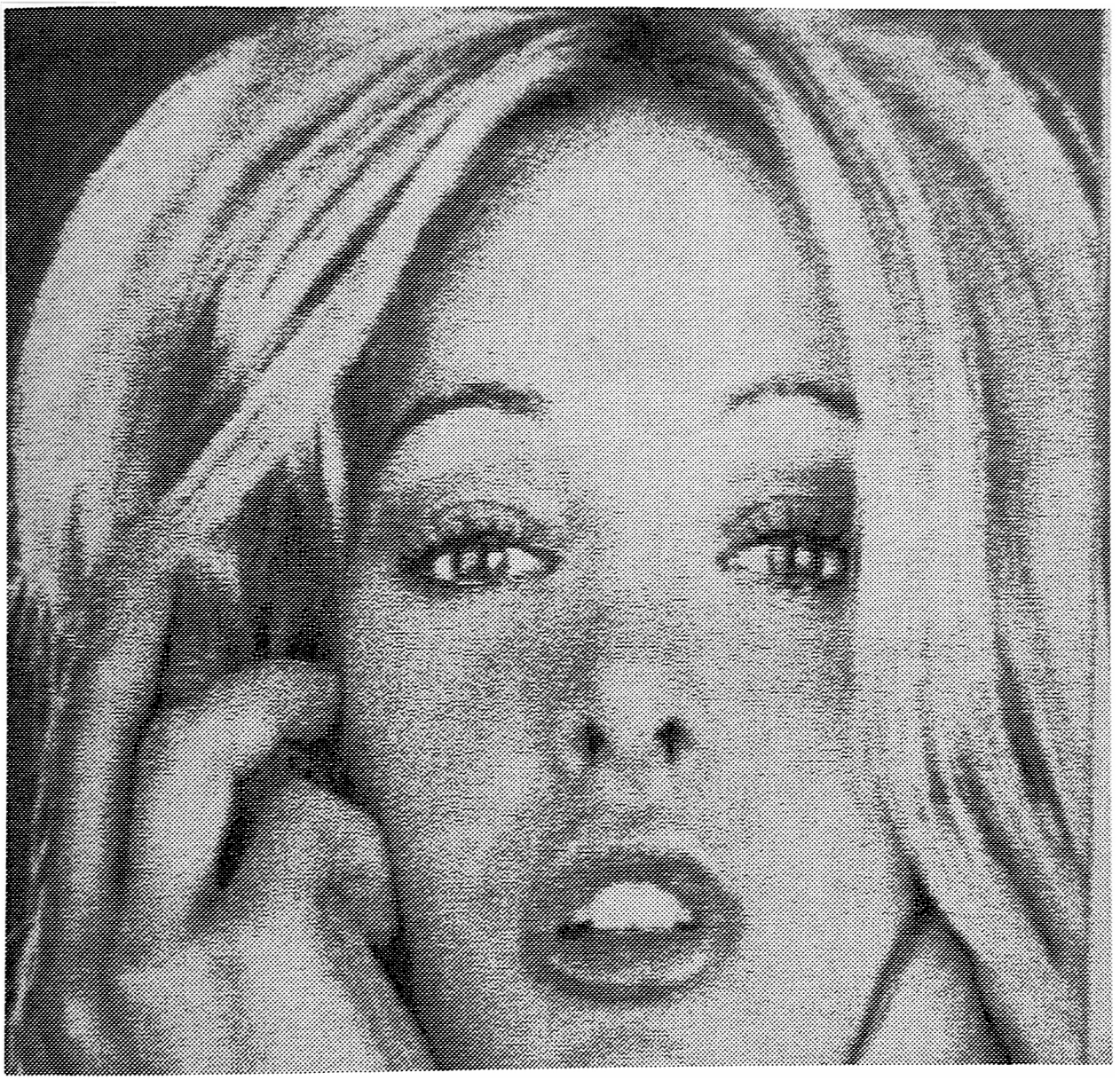


FIG. 4.7. Original Girl.img 256 gray levels.

```
number of lines:   456
number of pixels:  471
start line:        0
start pixel:       20
date:              02/20/91
```

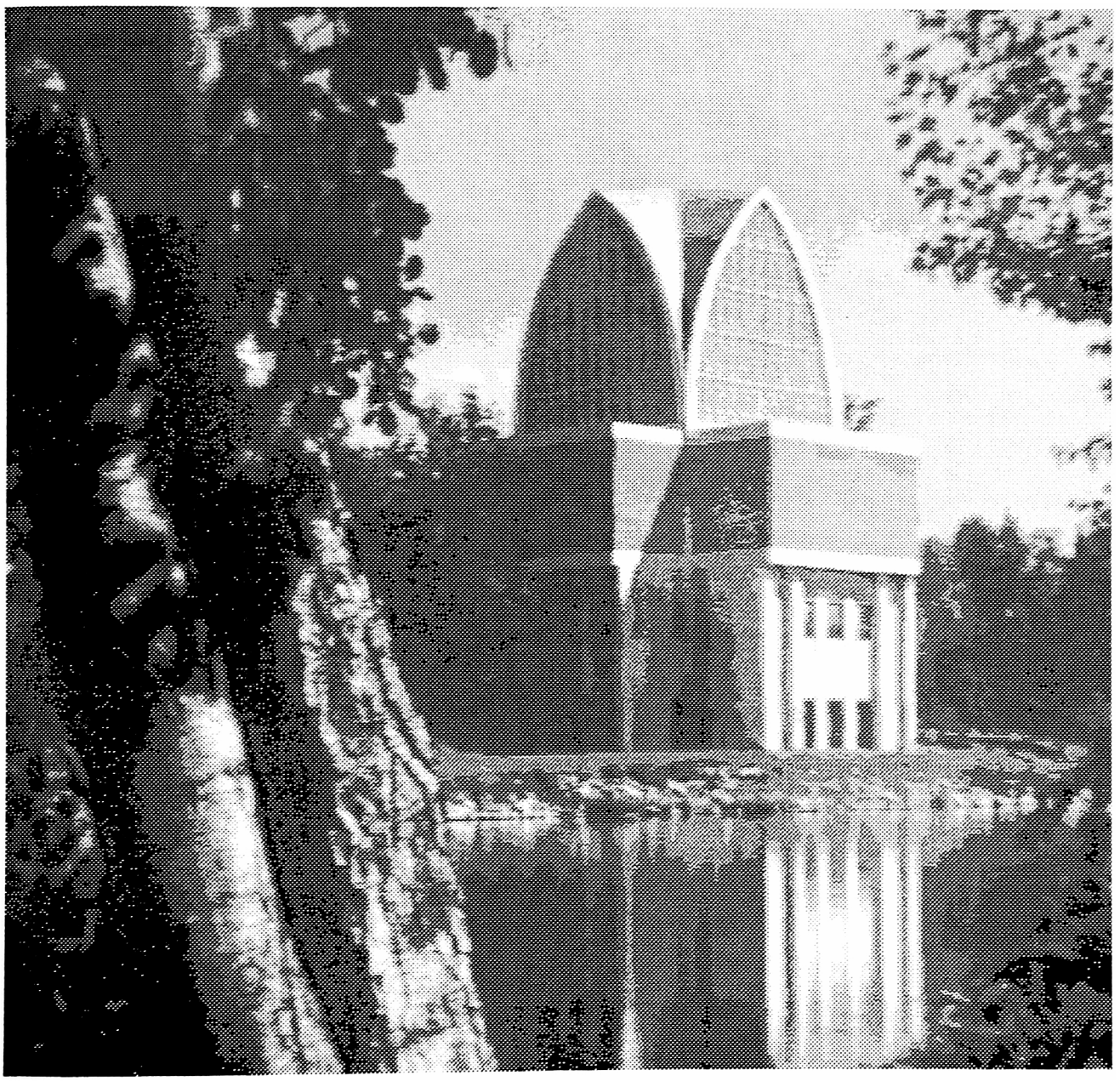


FIG. 4.8. Threshold Chapel.img from 256 to 8 gray levels.

number of lines: *476*
number of pixels: *491*
start line: *0*
start pixel: *0*
date: *02/20/91*

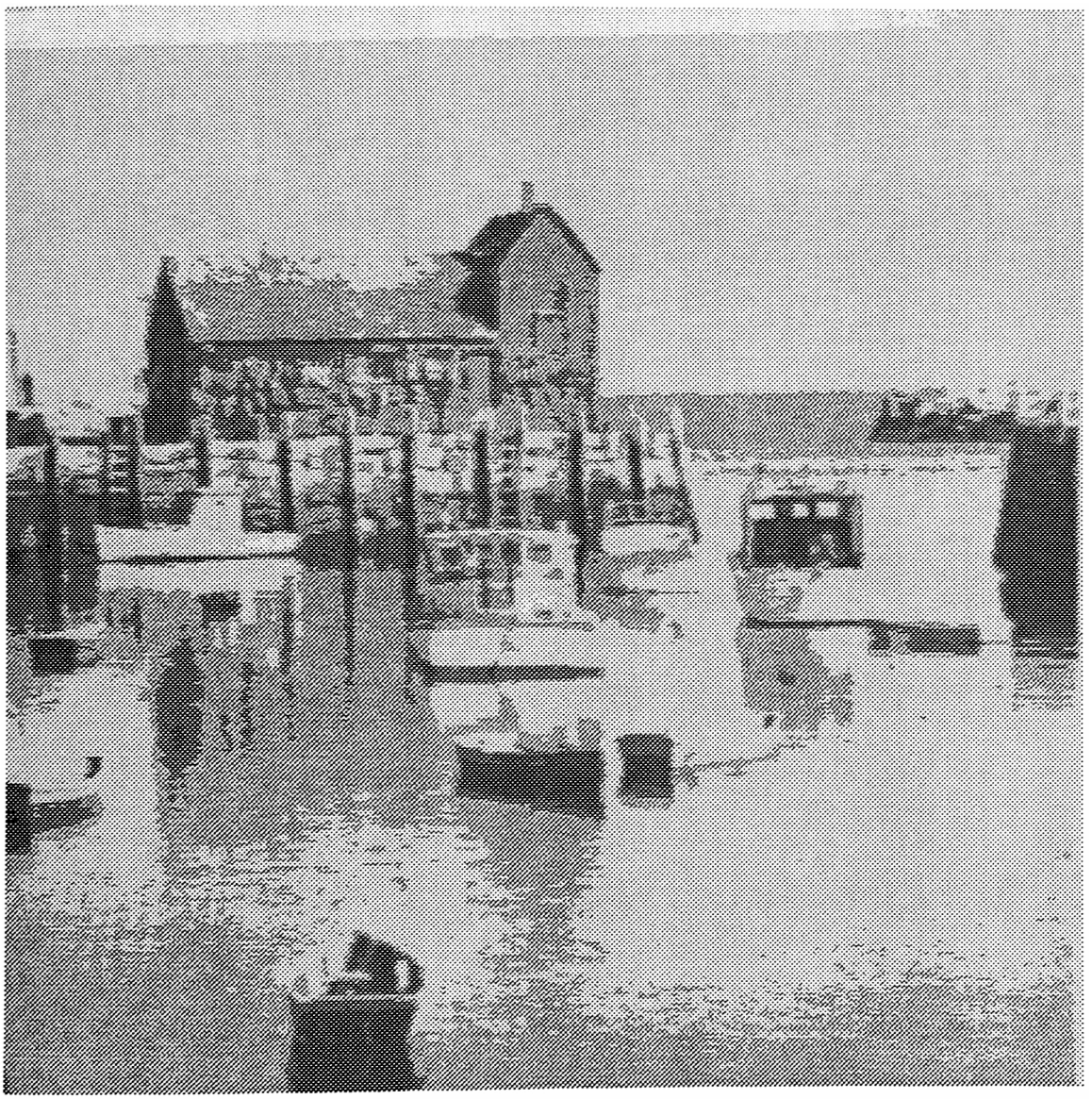


FIG. 4.9. Threshold Boat.img from 256 to 8 gray levels.

number of lines: 437
number of pixels: 430
start line: 5
start pixel: 35
date: 02/22/91

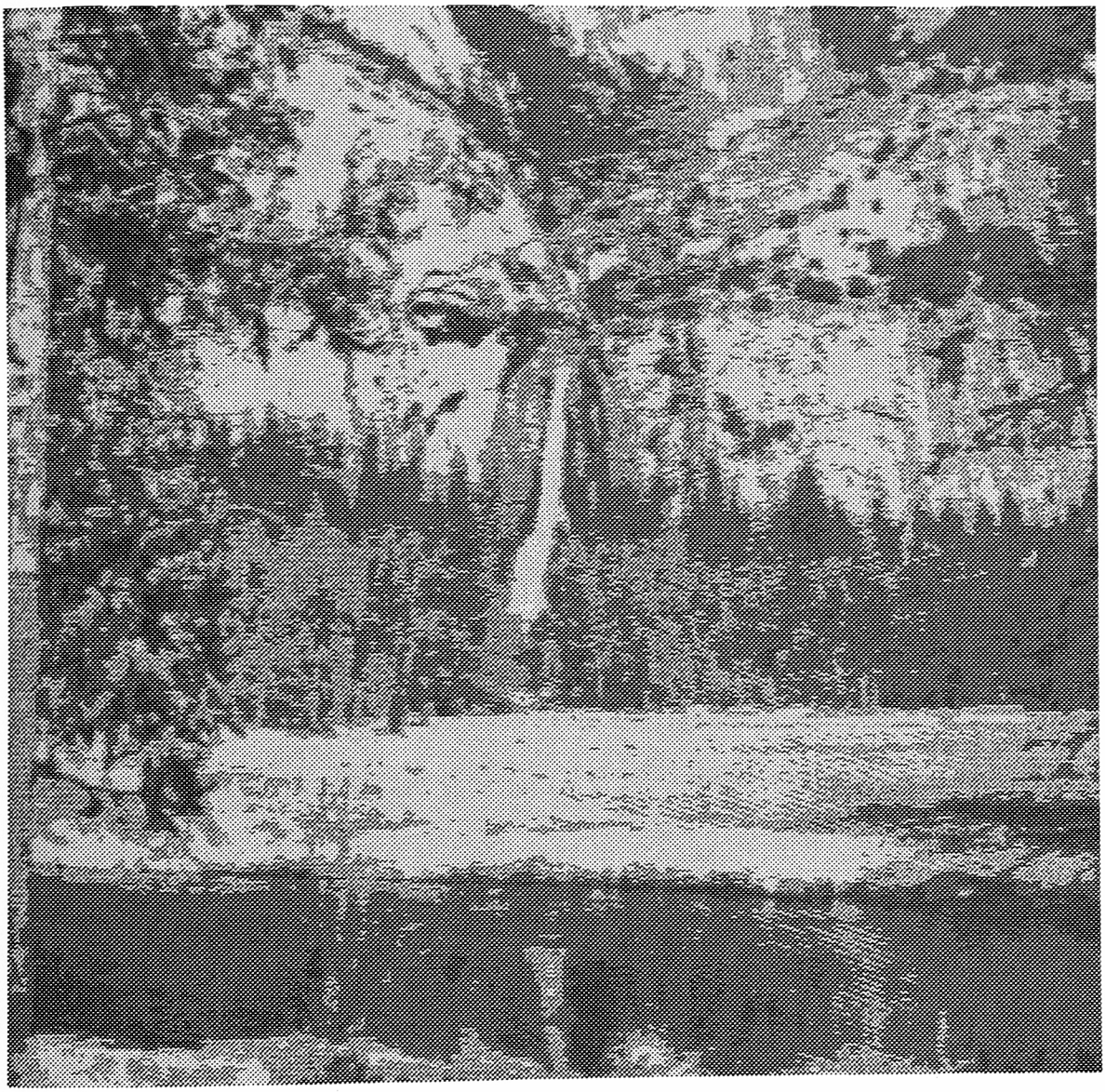


FIG. 4.10. Threshold Forest.img from 256 to 8 gray levels.

number of lines: *441*
number of pixels: *444*
start line: *0*
start pixel: *12*
date: *02/21/91*

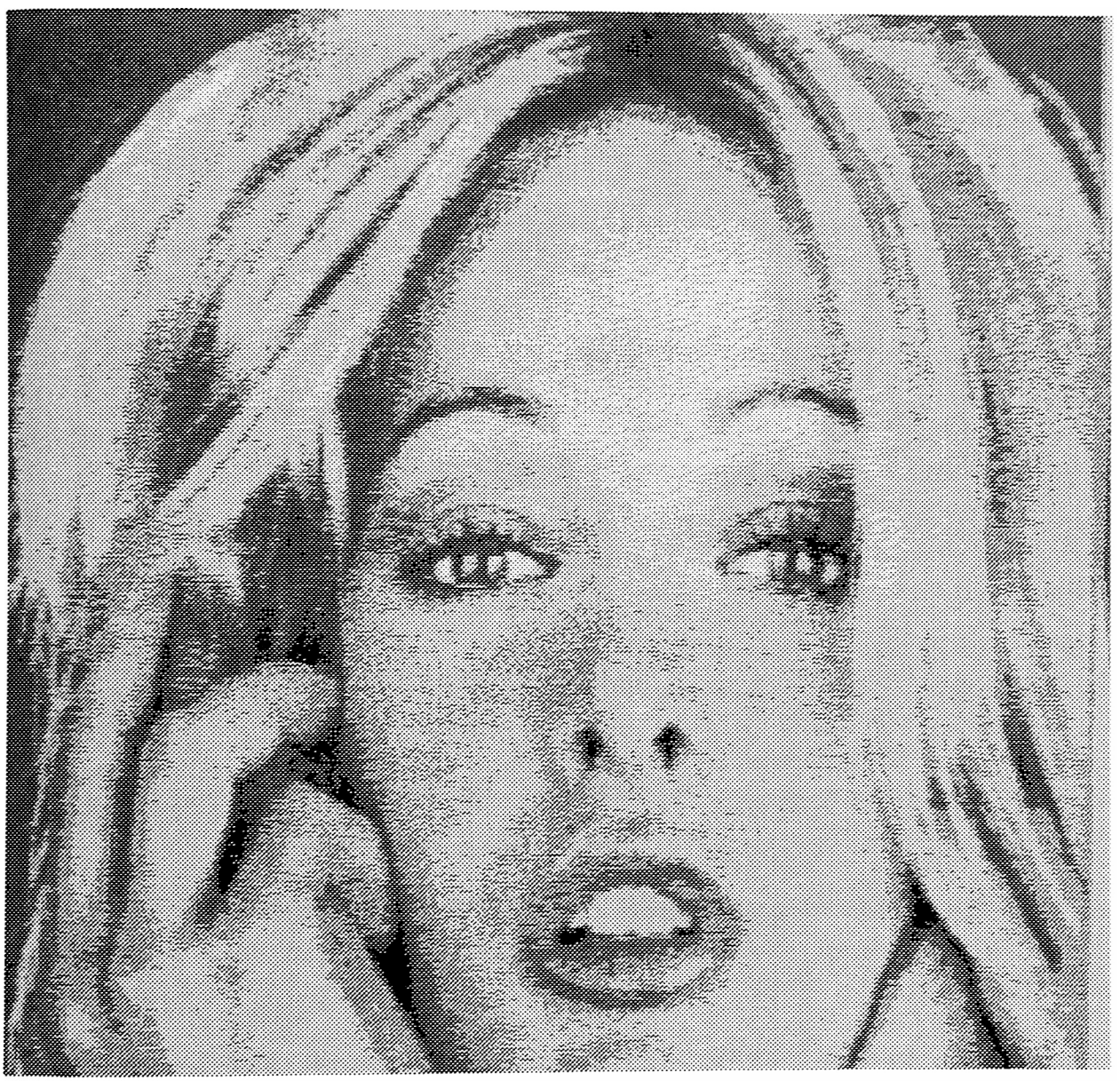


FIG. 4.11. Threshold Girl.img from 256 to 8 gray levels.

number of lines: *456*
number of pixels: *471*
start line: *0*
start pixel: *20*
date: *02/21/91*

Table 1. Image parameter analysis.

Image (.img)	Dimension $2^n \times 2^n$	Maximum tree depth (n)	$\sum_{i=0}^{n} 4^i$ Max = # of nodes (bytes)	Number of Nodes (bytes)	Nodes Ratio: No. of nodes / Max # of nodes
chapel	256 x 256	8	87381	31455	.360
boat	128 x 128	7	21845	7547	.345
Forest	128 x 128	7	21845	11043	.505
Girl	64 x 64	6	5461	3551	.650
Text	256 x 256	8	87381	19955	.228

Table 1 is a representation of each image, dimensions *($2^n x 2^n$)*, maximum tree depth, maximum number of nodes, number of nodes, and Nodes Ratio.

Table 2. Image compaction ratio analysis.

Image	Dimension $2^n \times 2^n$	Number of Nodes (bytes)	Total pixels (bytes)	Compaction Ratio: # of nodes / Total pix. (bytes)
chapel	256 x 256	31455	65536	.479
boat	128 x 128	7547	16384	.460
Forest	128 x 128	11043	16384	.674
Girl	64 x 64	3551	4096	.866
Text	256 x 256	19955	65536	.304

Table 2, Image compaction ratio analysis. The total number of pixels is $2^n x 2^n$. This ratio is obtained by the number of bytes in the Q-tree (encoded) image file, divided by the total number of bytes in the original (unencoded) image file. The best case is a compaction ration of 0.0 the worst case is a ratio greater than or equal to 1.0.

Table 3. Execution time analysis for rotate algorithm.

Image	rotate rotate = 90o	Maximum tree depth (n)	Number of Nodes (bytes)	Execution time encoded (sec)	Execution time unencoded (sec)
chapel	90o	8	31455	54	73
boat	90o	7	7547	15	18
Forest	90o	7	11043	18	18
Girl	90o	6	3551	7	5

Table 3, Execution Time analysis illustrates the time it takes to perform rotation , on all images. The execution times are observed with encoded and unencoded images. See **Fig. 4.12 - 4.15** for results of rotation algorithm on images.

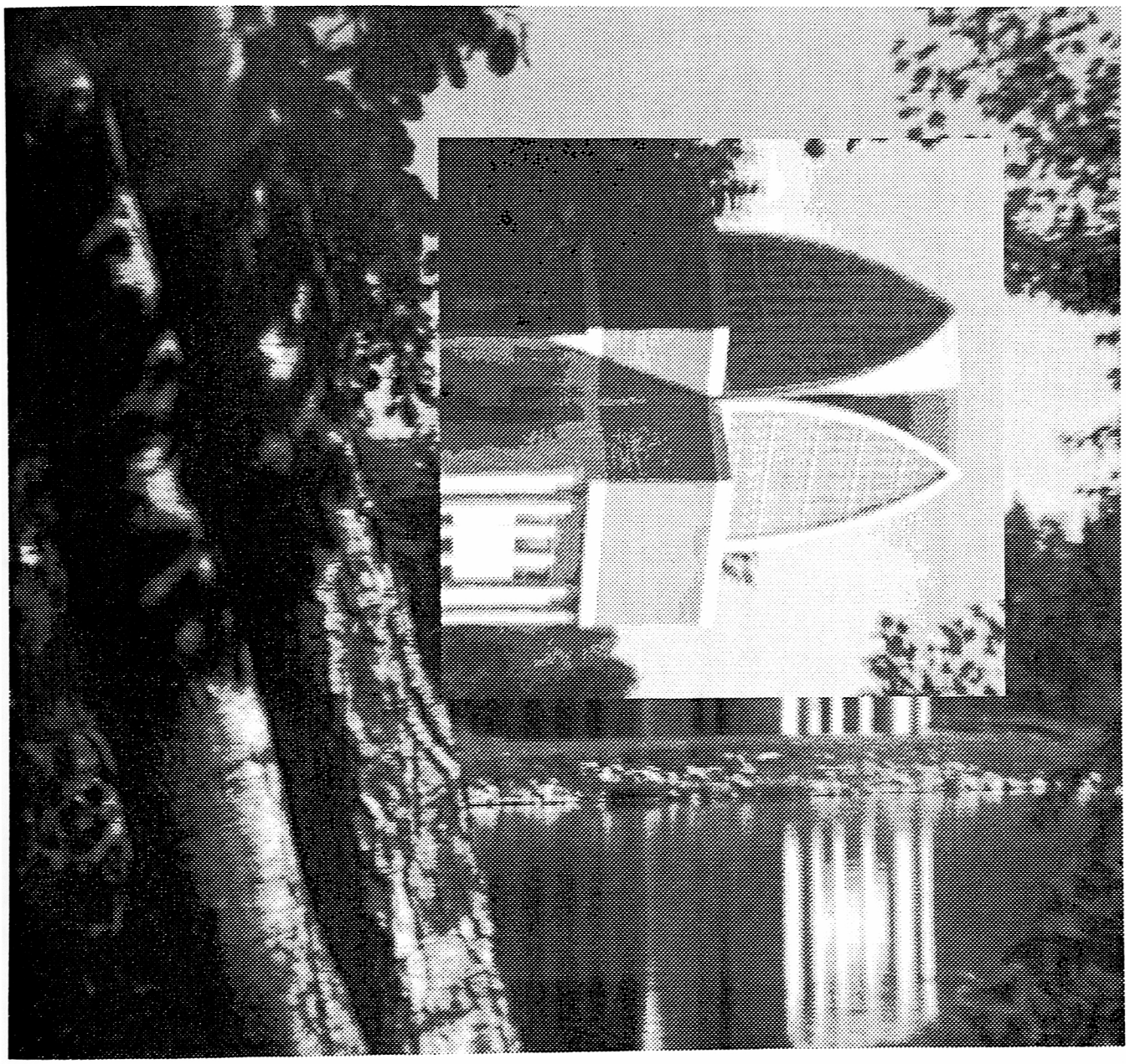


FIG. 4.12. Threshold Chapel.img from 256 to 8 gray levels; rot-90.

number of lines: 476
number of pixels: 504
start line: 0
start pixel: 5
date: 02/22/91

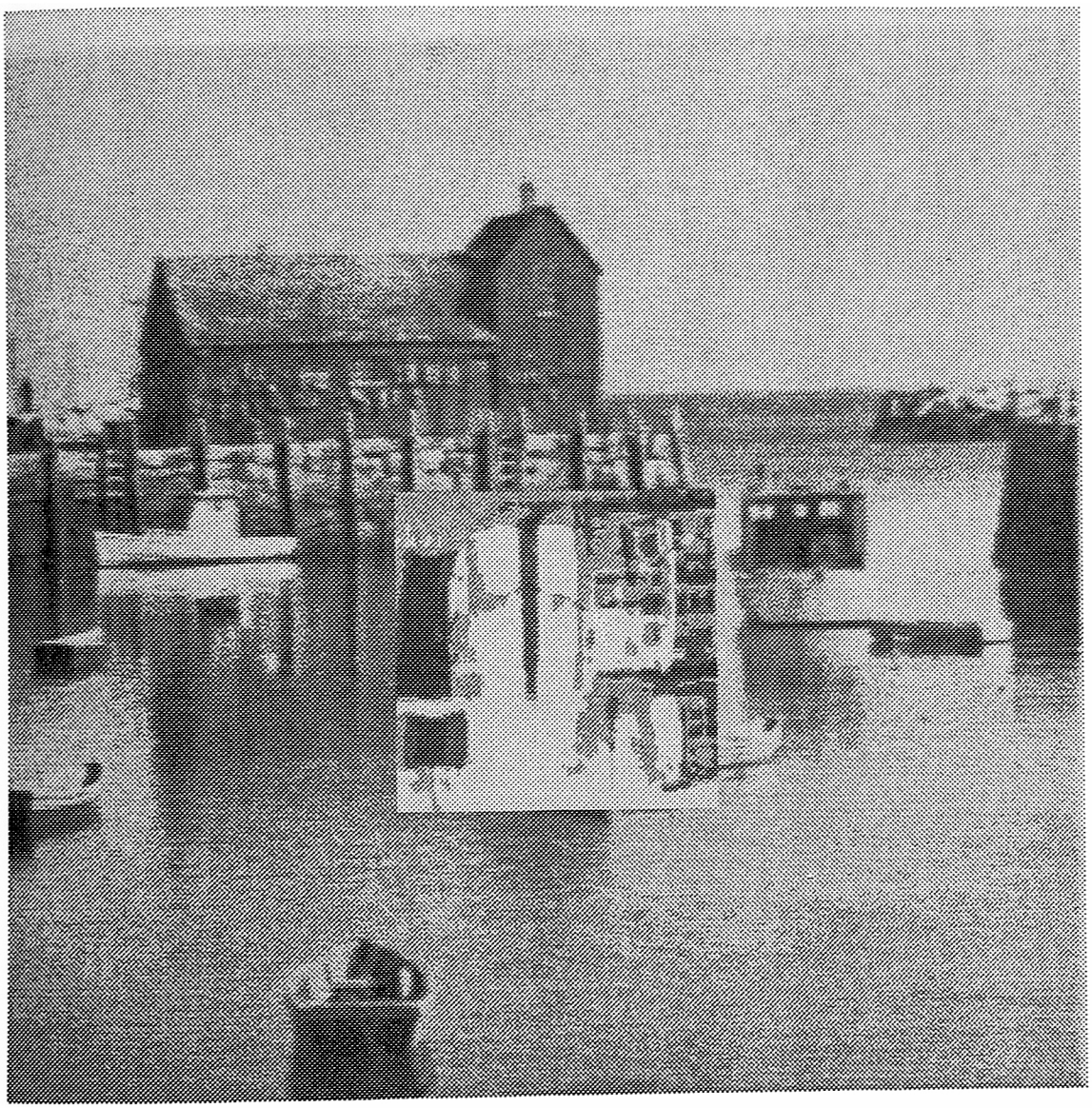


FIG. 4.13. Threshold Boat.img from 256 to 8 gray levels; rot-90.

number of lines: 437
number of pixels: 430
start line: 5
start pixel: 35
date: 02/22/91

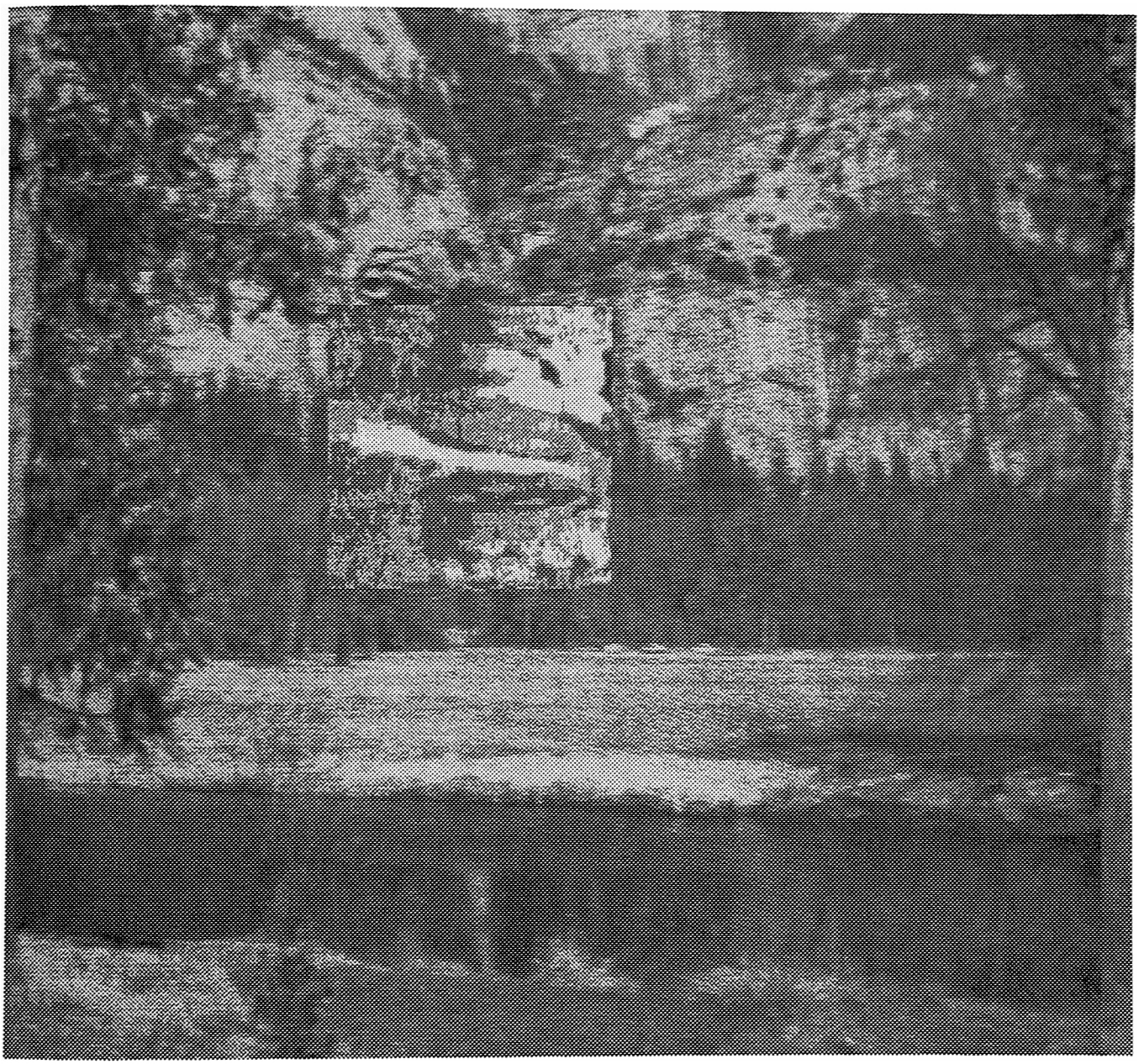


FIG. 4.14. Threshold Forest.img from 256 to 8 gray levels; rot-90.

number of lines: 476
number of pixels: 509
start line: 0
start pixel: 0
date: 02/21/91

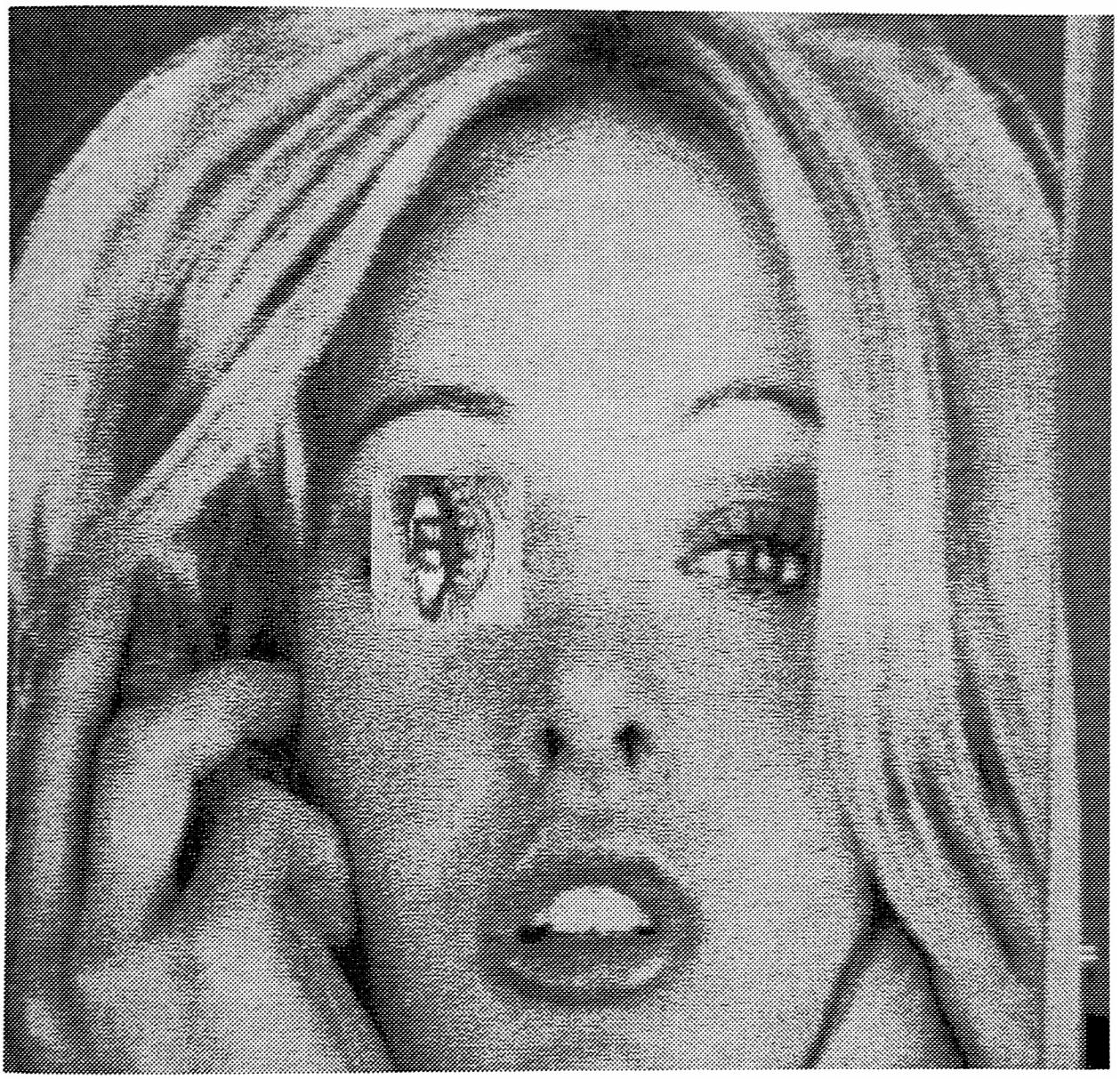


FIG. 4.15. Threshold Girl.img from 256 to 8 gray levels; rot-90.

number of lines: 456
number of pixels: 471
start line: 0
start pixel: 20
date: 02/21/91

Table 4. Execution time analysis for scale algorithm.

Image	scale scale factor(sf) = 2	Maximum tree depth (n)	Number of Nodes (bytes)	Execution time encoded (sec)	Execution time unencoded (sec)
chapel	2	8	31455	106	306
boat	2	7	7547	27	75
Forest	2	7	11043	32	76
Girl	2	6	3551	10	18

Table 4, Execution Time analysis illustrates the time it takes to perform 2x scaling , on all images. The execution times are observed with encoded and unencoded images. See **Fig. 4.16 - 4.20** for results of scale algorithm on images.

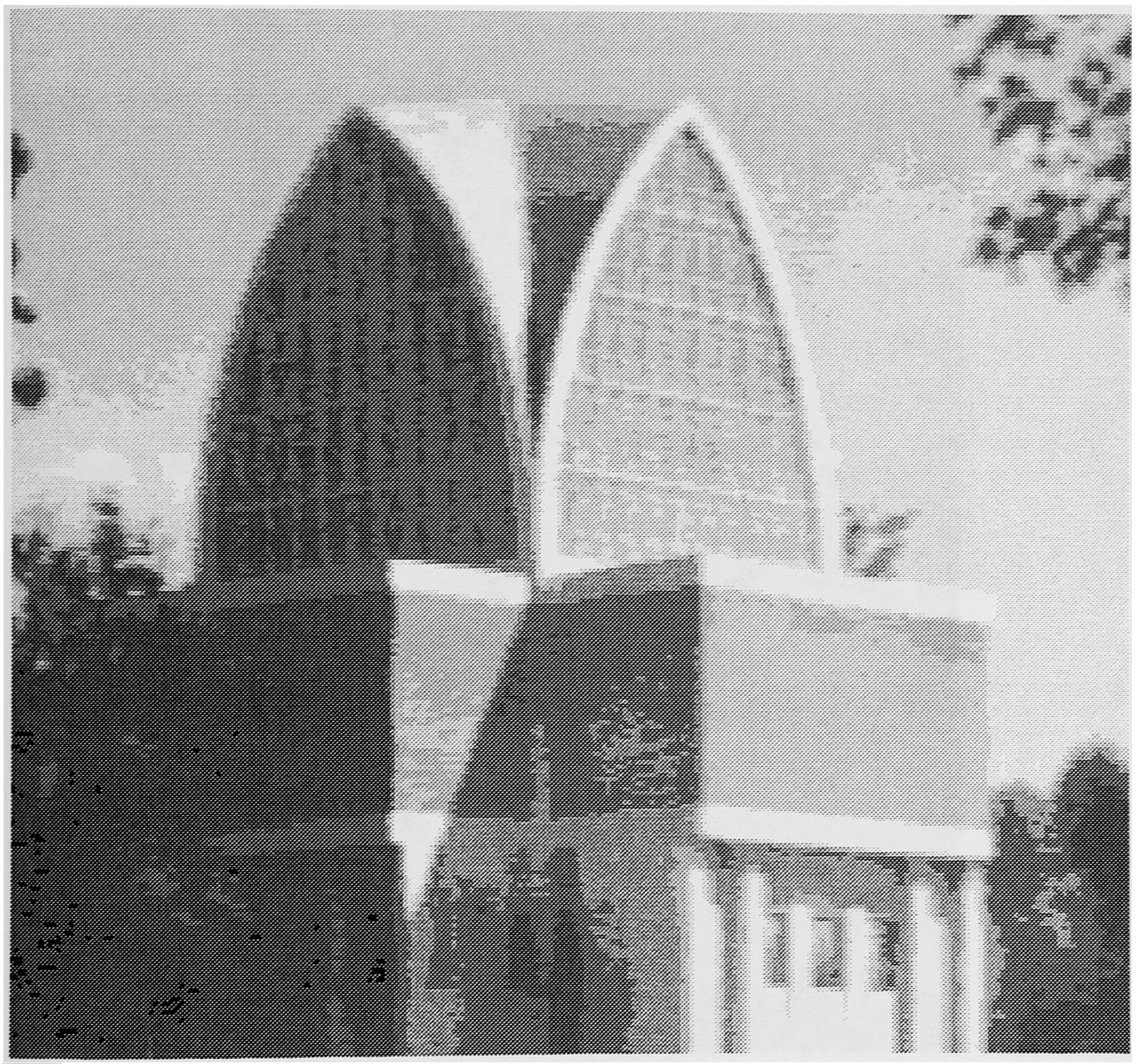


FIG. 4.16. Threshold Chapel.img from 256 to 8 gray levels; sf = 2;

number of lines: 476
number of pixels: 509
start line: 0
start pixel: 0
date: 02/21/91

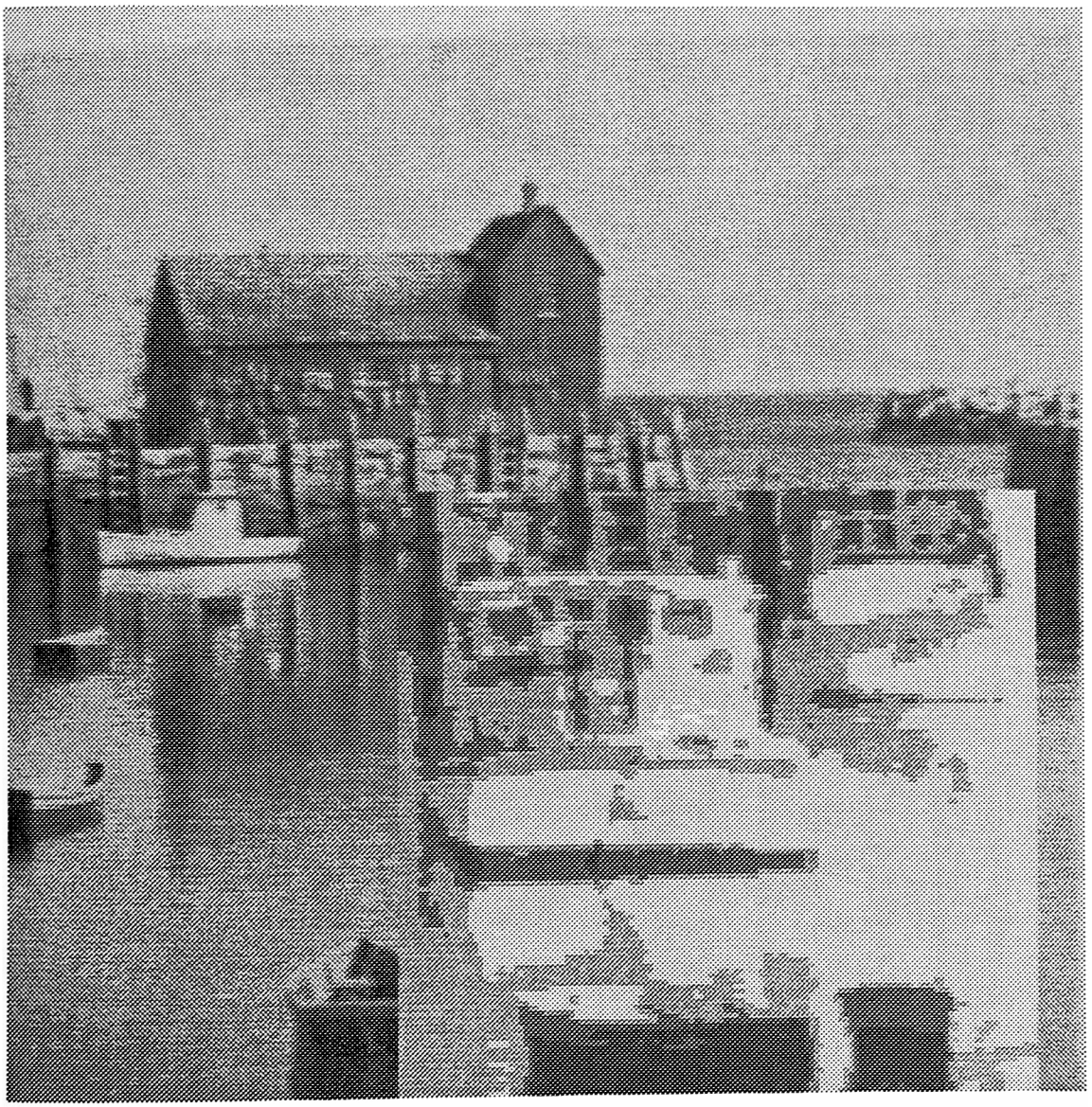


FIG. 4.17. Threshold Boat.img from 256 to 8 gray levels; sf = 2.

number of lines: 437
number of pixels: 430
start line: 5
start pixel: 35
date: 02/22/91

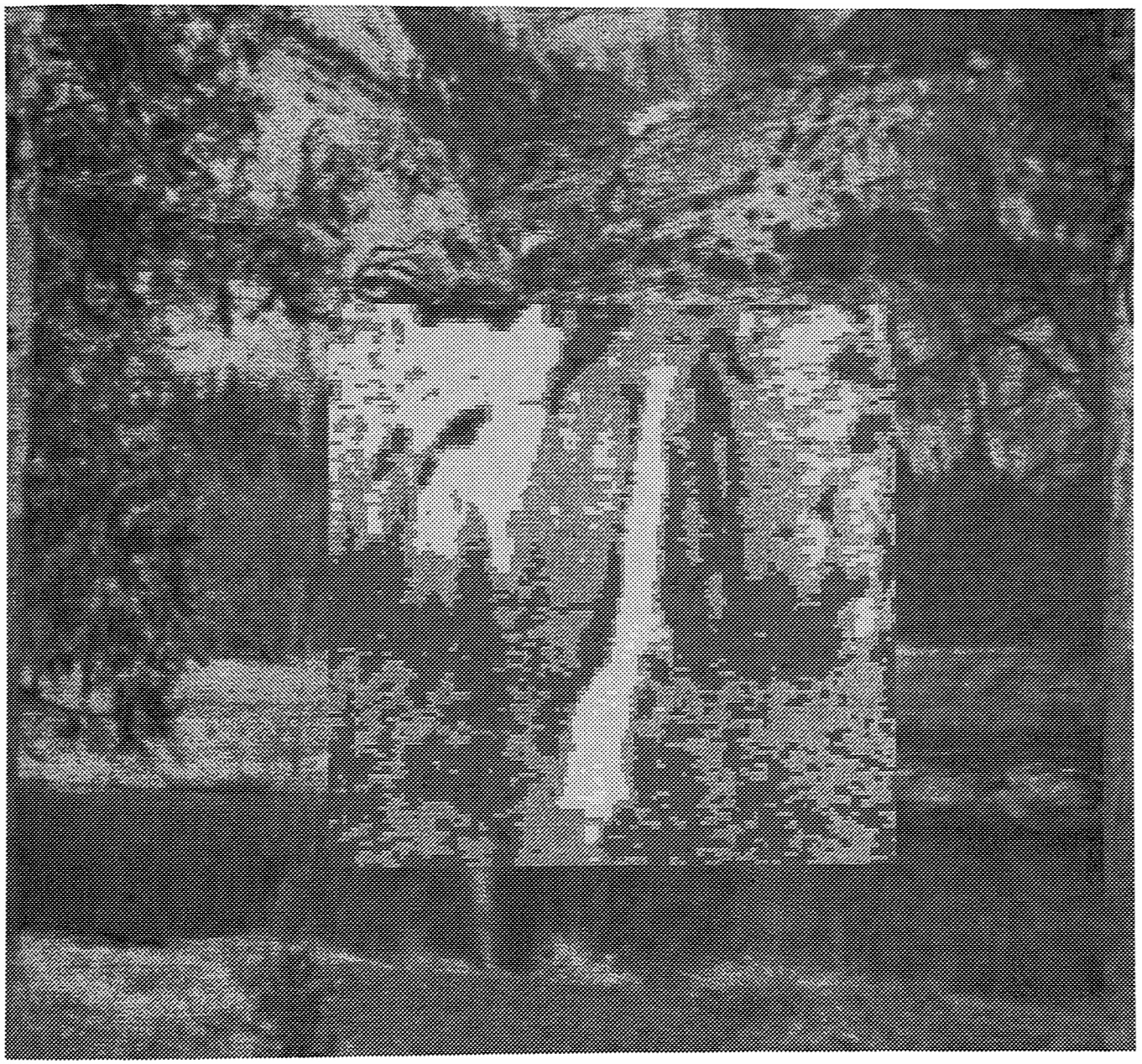


FIG. 4.18. Threshold Forest.img from 256 to 8 gray levels; sf = 2.

number of lines: 476
number of pixels: 509
start line: 0
start pixel: 0
date: 02/21/91

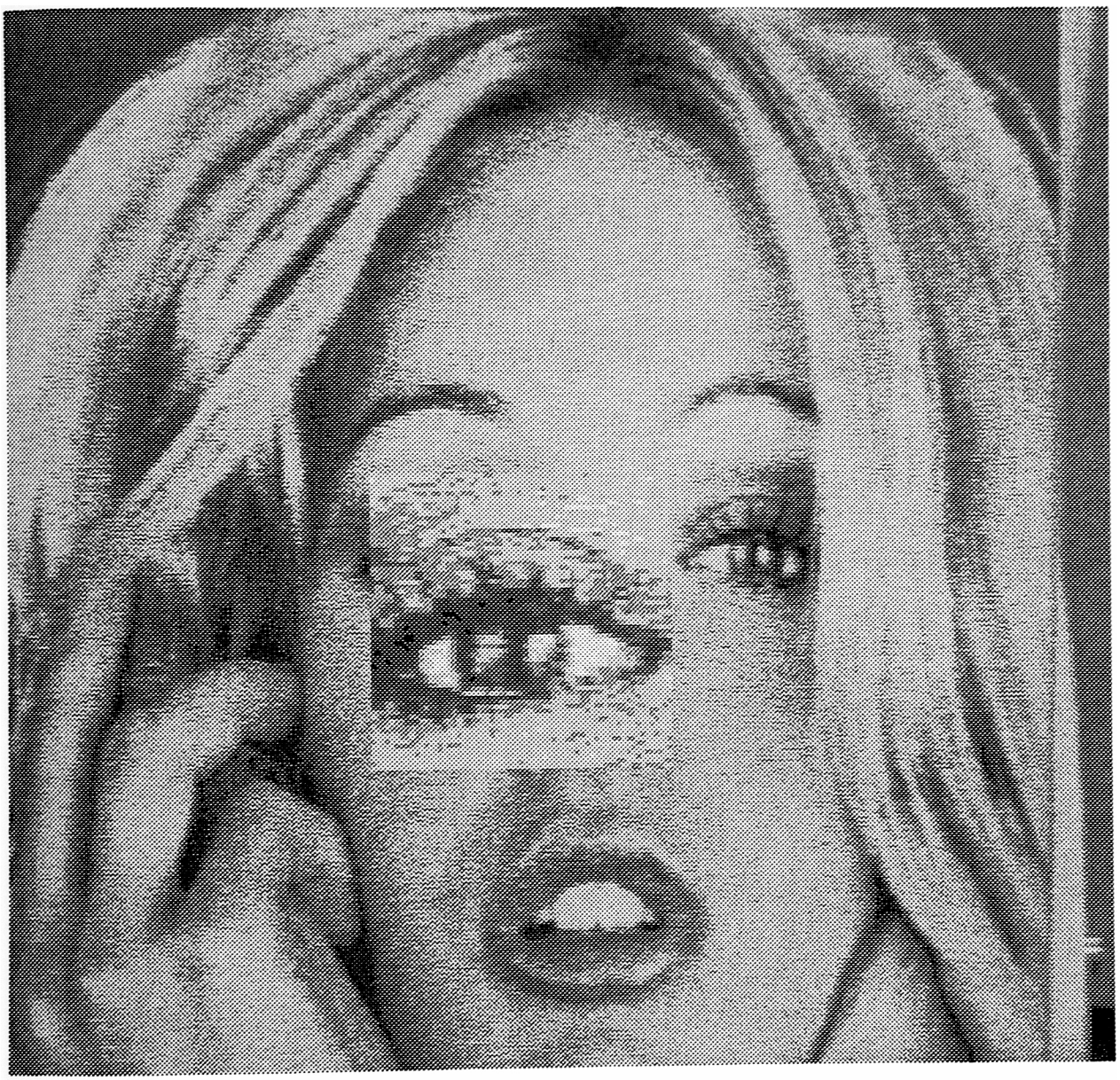


FIG. 4.19. Threshold Girl.img from 256 to 8 gray levels; sf = 2.

number of lines: 456
number of pixels: 471
start line: 0
start pixel: 20
date: 02/22/91

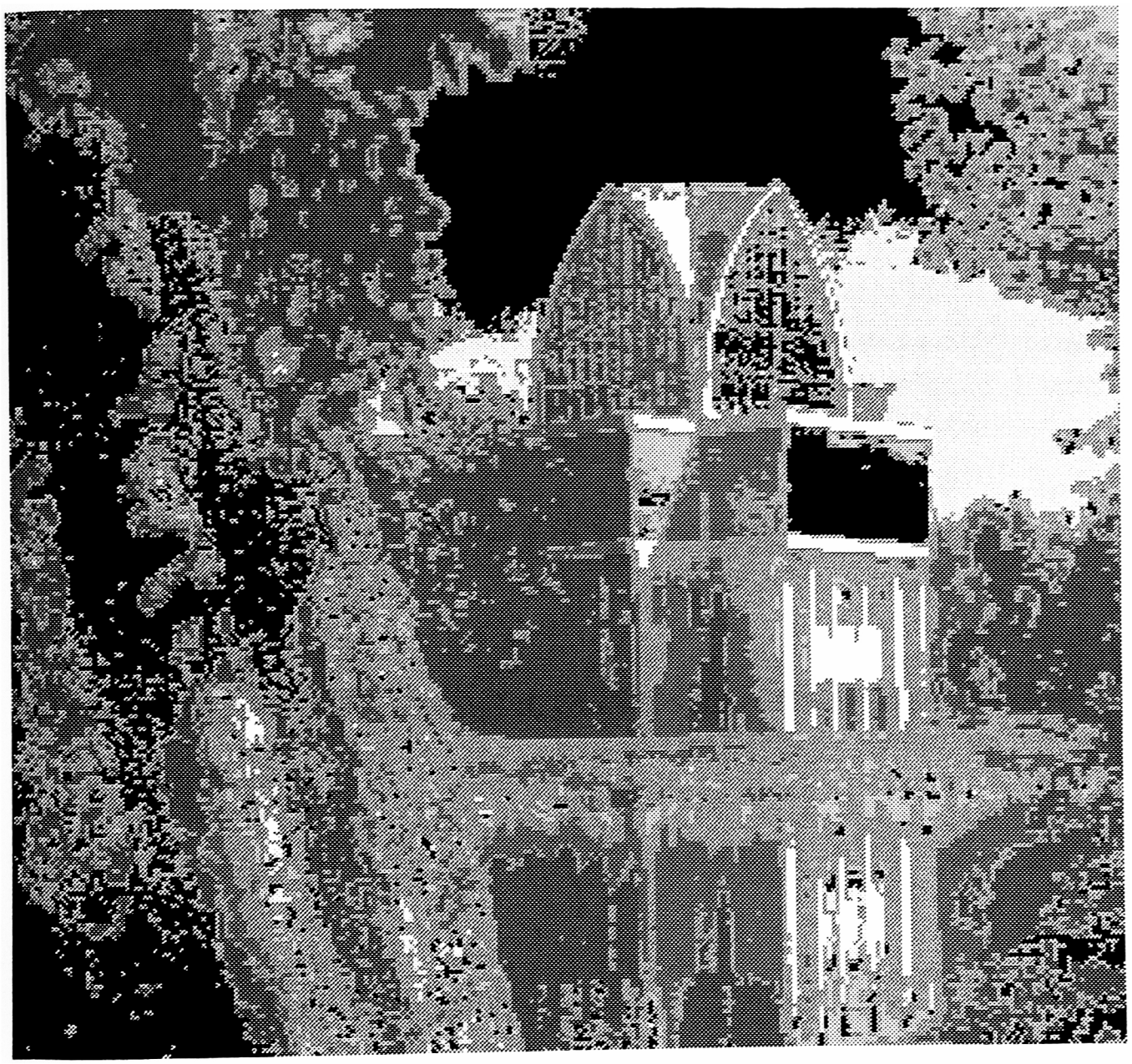


FIG. 4.20. Segmentation of Chapel.img gray5 (160-191).

number of lines: 476
number of pixels: 504
start line: 0
start pixel: 5
date: 02/22/91

5. CONCLUSION

In section 1 it was mentioned that the goal of this Thesis was to investigate the use of the Q-Tree data structure for image processing primitives, compaction, segmentation, progressive transmission, browsing and storage applications. The results described in section 4, illustrate that the goal was accomplished and several advantages were identified. With the encoded image the user can utilize the browsing feature of a Q-tree encoded image. This browsing feature allows the user to view images from level 0 (entire image) to level n (border pixels). Thus the user may terminate this browsing at any satisfactory level. A common list of advantages of Q-tree encoding are as follows:

1. Q-Tree encoding can address the issues of storage by recursive subdivision of images. Images which are physically too big to store in fast memory at one time can be processed as a sequence of subpictures extracted during Q-tree encoding.

2. Q-Tree encoding enables addressing for rapid access to any geographical part of the image.

3. Q-Tree encoding retains explicitly in the data structure a hierarchical description of the picture patterns, elements, and their relationships. Hence, this scheme may also be used in conjunction with image processing algorithms [**Firschein-1,2, Pavlidis-4,5**].

4. Q-Tree encoding permits recursive analysis of subpictures.

5. The decomposition algorithm contains major routines (traversal, tree-creation) which are independent of image class (binary, gray, etc.). Small changes can adapt regular decomposition to widely different types of images.

6. The Execution time for geometric applications (i.e. scaling, rotation), are decreased, because each primitive execution is based on block excution instead of pixel execution.

As mentioned previously there are some pathological cases; a) the encoding of the checkerboard pattern, b) encoding of an image containing many non-

homogeneous regions, c) encoding an image containing varying neighboring textures d) rotation of regions that are not multiples of 90°, or e) chosing scale factors that are not multiples of 2. Clearly these cases introduce disadvantages of the use of the Q-Tree data structure. However, the advantages stated above clearly show promising application use of the Q-Tree hierarchical data structure.

The run time complexity of the algorithm is dependent on the complexity of the the scene under analysis, rather than the image size. Because the algorithm is quite general, it can be used to obtain a convenient data structure for a wide range of different pattern recognition and scene analysis tasks and image types. A common area of exploration is region approximation using quadrant connectivity. This concept can be further developed into a split-and-merge algorithm , as mentioned in section 2 [**Klinger**]. Another potential area of investigation would be to apply the Q-Tree data structure algorithm in VLSI Hardware.

REFERENCES

[Arnemann] S. Arnemann and M. Tasto, "Generating halftone pictures on graphic computer terminals using run length coding", *Computer Graphics and Image Processing*, vol. 2, pp. 1 -11, 1973.

[Firschein-1] O. Firschein and M. A. Fischler, "Describing and abstracting pictorial structures", *Pattern Recognition*, vol. 3, No. 4, Nov. 1971.

[Firschein-2] O. Firschein and M. A. Fischler, "A study in descriptive representation of pictorial data", *Pattern Recognition*, vol. 4, No. 4, Dec. 1972.

[Gonzalez] Gonzalez and Wintz, "Digital Image Processing 2nd ed", *Addison-Wesley*, 1987.

[Gargantini-1] I. Gargantini, "Translation, rotation and superposition of linear quadtrees", *Communication of the ACM*, pp 253-263, 1982.

[Gargantini-2] I. Gargantini, "An effective way to represent quadtrees", *Com. of the ACM*, vol. 25, No. 12, pp 905 - 910, Dec. 1982.

[Hunter-1] G.M. Hunter and K. Steiglitz, "Operations on images using quad trees", IEEE *Transactions on Patterns Analysis and Machine Intelligence*, vol. 2, pp. 145 - 153, April 1979.

[Hunter-2] G.M. Hunter and K. Steiglitz, "Linear transformation of pictures represented by quad trees", *Computer Graphics and Image Processing*, vol. 10, No. 3, pp. 289 - 296, July 1979.

[Jarvis] J.F. Jarvis, C.N. Judice and W.H. Nike, "A survey of techniques for the image display of continuous tone pictures on bilevel display", *Computer Graphics and Image Processing*, vol. 5, pp. 13 - 40, Mar. 1976.

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

[Knuth] D.E. Knuth, "The Art of Computer Programming: Fundamentals of Algorithms".

[Limb] S.O. Limb and I.G. Sutherland, "Run-length coding of television signals", *Proc. IEEE 53*, pp. 169 - 170, Feb. 1965.

[Pavlidis-1] S.L. Horowitz and T. Pavlidis, "Picture segmentation by a tree traversal algorithm", *J. Ass. Computing Mach.*, vol. 23, pp. 368 - 388, April 1976.

[Pavlidis-2] T. Pavlidis, "Algorithms for Graphics and Image Processing", *Computer Science Press*, 1982.

[Pavlidis-3] T. Pavlidis and S.L. Horowitz, "The use of algorithms of piecewise approximations for picture processing applications", *ACM Transactions on Mathematical Software*, vol. 2, No. 4, pp. 305 - 321, Dec. 1976.

[Pavlidis-4] T. Pavlidis, "Analysis of set patterns", *Pattern Recognition*, vol. 1, Nov. 1968.

[Pavlidis-5] T. Pavlidis, "Structural pattern recognition:Primitives and juxtaposition relations, in Frontiers of Pattern Recognition (S. Watanabe, Ed.),", *Academic Press,* New York 1972.

[Rosenfeld-1] A. Rosenfeld, "Connectivity in Digital Pictures", *J. ACM*, vol 17., No. 1, Jan. 1970.

[Rosenfeld-2] A. Rosenfeld, "Figure extraction, in Automatic and Classification of Images (A. Grasselli, Ed.)", *Academic Press*, New York, 1969.

[Samet-1] H. Samet, "Applications of Spatial Data Structures", *Addison-Wesley*, 1990.

[Samet-2] H. Samet, "Design and Analysis of Data Structures", *Addison-Wesley*, 1989.

[Samet-3] H. Samet, and R. Webber, "Hierarchical Data Structures and Algorithms for Computer Graphics", *IEEE Computer Graphics & Application* , pp. 48 - 68, 1988.

[Samet-4] H. Samet, "Region representation: Quadtree from binary arrays", *Computer Graphics and Image Processing*, vol. 13, No. 1, pp. 88 - 93, May 1980.

APPENDIX

Programs:

1. **Videorw.c** - This program provides modules to enable read / write functions of each pixel in the frame grabber to the video display monitor.
2. **Xform.c** - This program provides homogeneous transformation (i.e. scale, rotate) algorithm on raw bit-map areas.
3. **Thesisiqt.c** - This program provides the following modules; user interface, encodequadtree (compression), decodequadtree (decompression), thresholding of image and utilities modules to enable display of image to the video monitor.
4. **Thmake** - This file is used with Microsoft "make" program to produce executable file (Thesisqt.exe).
5. **Itexpfg.h, Stdtyp.h, Initial.c** - These modules are not included but used, these modules are provided by the Rochester Institute of Technology, Department of Computer Engineering, Image Processing Lab.

```
0001 #include <graph.h>
0002 #define MAX_LINE_SIZE 512
0003 #define BLACK 0
0004 #define WHITE 255
0005
0006 /*-------------------------------------------------------------------------
0007         ATTENTION!!! - please define system in use
0008 -------------------------------------------------------------------------*/
0009 #define SYSPC
0010
0011 /*************************************************************************
0012 *       ATTENTION!!     UDC SYSTEM                                       *
0013 *                                                                       *
0014 *************************************************************************/
0015 #if defined (SYSUDC)
0016 extern int far *biu_page;
0017 extern unsigned char far *fb_address;
0018
0019 /*************************************************************************
0020 *                                                                       *
0021 * Description:  Write  pixel to UDC display                             *
0022 *                                                                       *
0023 * Input: x, y, datum                                                    *
0024 * Output: none                                                          *
0025 * Reference: none                                                       *
0026 * Used in: none                                                         *
0027 *                                                                       *
0028 *************************************************************************/
0029 Bwpixel( x, y, datum )
0030 int x, y;
0031 unsigned char datum;
0032
0033 {
0034   long offset;
0035
0036   *biu_page = y >> 5;
0037
0038   offset = y & 0x001F;
0039   *( fb_address + ( offset << 11 ) + x ) = datum;
0040
0041   return;
0042 }
0043
0044 /*************************************************************************
0045 *                                                                       *
0046 * Description:  Read  pixel from UDC display                            *
0047 *                                                                       *
0048 * Input: x, y                                                           *
0049 * Output: byte value                                                    *
0050 * Reference: none                                                       *
0051 * Used in: none                                                         *
0052 *                                                                       *
0053 *************************************************************************/
0054 unsigned char Brpixel( x, y )
0055 int x, y;
0056
0057 {
0058   long offset;
0059
0060   *biu_page = y >> 5;
0061
0062   offset     = y & 0x001F;
0063   return ( *( fb_address + ( offset << 11 ) + x ) );
0064 }
0065
0066 /*************************************************************************
0067 *                                                                       *
0068 * Description:  Write a line of data to UDC display memory-A            *
0069 *                                                                       *
0070 * Input: x, y, n, pixarray                                              *
```

```
0071 * Output: none                                                          *
0072 * Reference: none                                                      *
0073 * Used in: Thesisqt.c - Blacksect(), ThresholdImage(),                 *
0074 *          DisplayQuadValue(), LoadImage().                            *
0075 *                                                                      *
0076 ************************************************************************/
0077 Bwhlinea( x, y, n, pixarray )
0078 int x, y, n;
0079 unsigned char *pixarray;
0080
0081 {
0082   int i;
0083   long offset;
0084
0085   *biu_page = ( y >> 5 );
0086
0087   offset     = y & 0x001F;
0088   offset = offset << 11;
0089   for( i = x; i < x + n; i++ )
0090     *( fb_address + offset + i ) = *( pixarray + i - x );
0091
0092   return;
0093 }
0094
0095 /************************************************************************
0096 *                                                                      *
0097 * Description:  Read a line of data from UDC display memory-A          *
0098 *                                                                      *
0099 * Input: x, y, n, pixarray                                             *
0100 * Output: stuff pixarray                                               *
0101 * Reference: none                                                      *
0102 * Used in: Thesisqt.c - ThresholdImage(), GetQuadValue().              *
0103 *                                                                      *
0104 ************************************************************************/
0105 Brhlinea( x, y, n, pixarray )
0106 int x, y, n;
0107 unsigned char *pixarray;
0108
0109 {
0110   int i;
0111   long offset;
0112
0113   *biu_page = y >> 5;
0114
0115   offset = y & 0x001F;
0116   offset = offset << 11;
0117   for( i = x; i < x+n; i++ )
0118     *( pixarray + i - x ) = *( fb_address + offset + i );
0119
0120   return;
0121 }
0122
0123 /************************************************************************
0124 *       ATTENTION!!     PC-VISION SYSTEM 512 x 512                     *
0125 *                                                                      *
0126 ************************************************************************/
0127 #elif defined (SYSPC)
0128 #define fga 0x300
0129 unsigned char intemp, outtemp, pixeldata;
0130 extern int far *biu_page;
0131 extern unsigned char far *fb_address;
0132
0133
0134
0135 /************************************************************************
0136 *                                                                      *
0137 * Description:  Read pixel from PC-VISION display memory-A             *
0138 * Input: x, y                                                          *
0139 * Output: none                                                         *
0140 * Reference: none                                                      *
```

```
0141 * Used in: xform.c - Xform().                                              *
0142 *                                                                          *
0143 ***************************************************************************/
0144 Brpixela(int x, int y)
0145 {
0146 int temp;
0147  intemp = inp(fga);
0148  outtemp = intemp & 0x1F;
0149  temp = y/128;
0150  outtemp = outtemp + (temp << 5);
0151  outp(fga,outtemp);
0152  pixeldata = *(fb_address + ((y % 128) << 9) + x);
0153 }
0154
0155
0156 /***************************************************************************
0157 *                                                                          *
0158 * Description:  Write pixel to PC-VISION display memory-B                  *
0159 * Input: x, y                                                              *
0160 * Output: none                                                             *
0161 * Reference: none                                                          *
0162 * Used in: xform.c - Xform().                                              *
0163 *                                                                          *
0164 ***************************************************************************/
0165 Bwpixelb(int x, int y)
0166 {
0167 int temp;
0168  intemp = inp(fga);
0169  outtemp = intemp & 0x1F;
0170  temp = (y/128) + 4;
0171  outtemp = outtemp + (temp << 5);
0172  outp(fga,outtemp);
0173  *(fb_address + ((y % 128) << 9) + x) = pixeldata;
0174 }
0175
0176 /***************************************************************************
0177 *                                                                          *
0178 * Description:  Write a line of data to PC-VISION display memory-A         *
0179 *                                                                          *
0180 * Input: x, y, n, pixarray                                                 *
0181 * Output: none                                                             *
0182 * Reference: none                                                          *
0183 * Used in: Thesisgt.c - Blacksect(), ThresholdImage()                      *
0184 *          LoadImage().                                                    *
0185 *                                                                          *
0186 ***************************************************************************/
0187 Bwhlinea( x, y, n, pixarray )
0188 int x, y, n;
0189 unsigned char *pixarray;
0190
0191 {
0192   int i, bank, temp;
0193   unsigned char intemp, outtemp;
0194   long framepoint;
0195
0196   intemp = inp(fga);
0197   outtemp = intemp & 0x1F;
0198   bank = y >> 7;
0199   temp = bank << 5;
0200   outtemp = outtemp + temp;
0201   outp(fga, outtemp);
0202
0203   framepoint     = ((y % 128) & 0x007F);
0204   framepoint = framepoint << 9;
0205   for( i = x; i < x + n; i++ )
0206     *( fb_address + framepoint + i ) = *( pixarray + i - x );
0207
0208   return;
0209 }
0210
```

```
0211 /*************************************************************************
0212 *                                                                        *
0213 * Description:  Write a line of data to PC-VISION display memory-A       *
0214 *                                                                        *
0215 * Input: x, y, n, pixarray                                               *
0216 * Output: none                                                           *
0217 * Reference: none                                                        *
0218 * Used in: Thesisqt.c - Blacksect(), ThresholdImage(),                   *
0219 *          DisplayQuadValue(), LoadImage().                              *
0220 *                                                                        *
0221 *************************************************************************/
0222 Bwhlineb( x, y, n, pixarray )
0223 int x, y, n;
0224 unsigned char *pixarray;
0225
0226 {
0227   int i, bank, temp;
0228   unsigned char intemp, outtemp;
0229   long framepoint;
0230
0231   intemp = inp(fga);
0232   outtemp = (intemp & 0x1F) + 128;
0233   bank = y >> 7;
0234   temp = bank << 5;
0235   outtemp = outtemp + temp;
0236   outp(fga, outtemp);
0237
0238   framepoint     = ((y % 128) & 0x007F);
0239   framepoint = framepoint << 9;
0240   for( i = x; i < x + n; i++ )
0241     *( fb_address + framepoint + i ) = *( pixarray + i - x );
0242
0243   return;
0244 }
0245
0246 /*************************************************************************
0247 *                                                                        *
0248 * Description:  Read a line of data from PC-VISION display memory-A      *
0249 *                                                                        *
0250 * Input: x, y, n, pixarray                                               *
0251 * Output: stuff pixarray                                                 *
0252 * Reference: none                                                        *
0253 * Used in: Thesisqt.c - ThresholdImage(), GetQuadValue().                *
0254 *                                                                        *
0255 *************************************************************************/
0256 Brhlinea( x, y, n, pixarray )
0257 int x, y, n;
0258 unsigned char *pixarray;
0259
0260 {
0261   int i, bank, temp;
0262   unsigned char intemp, outtemp;
0263   long framepoint;
0264
0265   intemp = inp(fga);
0266   outtemp = intemp & 0x1F;
0267   bank = y >> 7;
0268   temp = bank << 5;
0269   outtemp = outtemp + temp;
0270   outp(fga, outtemp);
0271
0272   framepoint = (y % 128) & 0x007F;
0273   framepoint = framepoint << 9;
0274   for( i = x; i < x+n; i++ )
0275     *( pixarray + i - x ) = *( fb_address + framepoint + i );
0276
0277   return;
0278 }
0279
0280 #endif
```

```
0281
0282 /*****************************************************************************
0283 *                                                                           *
0284 * Description:  Output black screen to Display monitor                      *
0285 *               memory-A or memory-B                                        *
0286 * Input: none                                                               *
0287 * Output: none                                                              *
0288 * Reference: Bwhlinea(), Bwhlineb().                                        *
0289 * Used in: Thesisqt.c - MainMenu().                                         *
0290 *                                                                           *
0291 *****************************************************************************/
0292 Blackscr()
0293 {
0294 int i,j;
0295 unsigned char blackline[MAX_LINE_SIZE], c;
0296
0297  for (i=0; i < MAX_LINE_SIZE; i++)              /* init blankline array */
0298  {
0299   blackline[i] = BLACK;
0300  }
0301   clearscreen(0);
0302  printf("\n     (a) black out screen-A");
0303  printf("\n     (b) black out screen-B: ");
0304  c = getch();
0305  switch(c)
0306  {
0307   case 'a':
0308   case 'A':
0309             for (j=0; j < MAX_LINE_SIZE; j++)          /* inc rows */
0310             {
0311              Bwhlinea(0, j, MAX_LINE_SIZE, blackline);
0312             }
0313             break;
0314   case 'b':
0315   case 'B':
0316             for (j=0; j < MAX_LINE_SIZE; j++)          /* inc rows */
0317             {
0318              Bwhlineb(0, j, MAX_LINE_SIZE, blackline);
0319             }
0320             break;
0321   default:  break;
0322  }
0323 }/* end of Blackscr */
0324
0325 /*****************************************************************************
0326 *                                                                           *
0327 * Description:  Output white screen to Display monitor                      *
0328 *               memory-A or memory-B                                        *
0329 * Input: none                                                               *
0330 * Output: none                                                              *
0331 * Reference: Bwhlinea(), Bwhlineb().                                        *
0332 * Used in: Thesisqt.c - MainMenu().                                         *
0333 *                                                                           *
0334 *****************************************************************************/
0335 Whitescr()
0336 {
0337 int i,j;
0338 unsigned char whiteline[MAX_LINE_SIZE], c;
0339
0340  for (i=0; i < MAX_LINE_SIZE; i++)              /* init whiteline array */
0341  {
0342   whiteline[i] = WHITE;
0343  }
0344   clearscreen(0);
0345  printf("\n     (a) White out screen-A");
0346  printf("\n     (b) White out screen-B: ");
0347  c = getch();
0348  switch(c)
0349  {
0350   case 'a':
```

```
0351    case 'A':
0352                for (j=0; j < MAX_LINE_SIZE; j++)           /* inc rows */
0353                {
0354                 Bwhlinea(0, j, MAX_LINE_SIZE, whiteline);
0355                }
0356                break;
0357    case 'b':
0358    case 'B':
0359                for (j=0; j < MAX_LINE_SIZE; j++)           /* inc rows */
0360                {
0361                 Bwhlineb(0, j, MAX_LINE_SIZE, whiteline);
0362                }
0363                break;
0364    default:    break;
0365   }
0366 }/* end of Whitescr */
0367
0368 /* EOF */
```

```
0001 #include <graph.h>
0002 #include <math.h>
0003 #include <time.h>
0004 #include <dos.h>
0005 /*************************************************************************
0006
0007 *************************************************************************/
0008 #define _GCLEARSCREEN 0
0009 #define CENTER 1
0010 #define HITANYKEY printf("\nPress any key to continue");getch();
0011 extern unsigned char far *fb_address;
0012 float *xfrm,transfrm[9],*invxfrm,inv[9];
0013
0014 /*************************************************************************
0015 *                                                                       *
0016 * Description:  Determine scale factor, and rotation value.             *
0017 *               Perform scale and rotate using homogeneous xformation.  *
0018 *                                                                       *
0019 * Input: x, y, width                                                    *
0020 * Output: none                                                          *
0021 * Reference: FindInverse(), Brpixel(), Bwpixelb() and C-lib functions   *
0022 * Used in: Thesisqt.c - MainMenu().                                     *
0023 *                                                                       *
0024 *************************************************************************/
0025 Xform(int x, int y, int width)
0026 {
0027 int xold,yold,j,k,count,xspan,yspan,origin;
0028 int i,xleft,yleft,xright,yright;
0029 float scalex,scaley,rotate,transx,transy,oldx,oldy,*hold,hd[9];
0030 double radfn,cose,sine;
0031 int c;
0032 time_t start, finish;
0033
0034  origin = 1;
0035  xleft = x;
0036  yleft = y;
0037  xright = x + width - 1;
0038  yright = y + width - 1;
0039  xfrm = transfrm;
0040  invxfrm = inv;
0041  hold = hd;
0042  transx = transy = 0;
0043  scalex = scaley = 1;
0044  rotate = 90;
0045                                          /* Initialize transform matrix */
0046         for (i=0; i < 9; i++)
0047                 *(xfrm + i) = 0.0;
0048         *(xfrm) = 1.0;
0049         *(xfrm + 4) = 1.0;
0050         *(xfrm + 8) = 1.0;
0051
0052                                          /* Input transform info   */
0053
0054  count = 0;
0055  c = 1;
0056  while ((c == 1) || (c == 2) || (c == 3))
0057  {
0058    _clearscreen(_GCLEARSCREEN);
0059   printf("\n 1 - translation, 2 - scaling, 3 - rotation, 4 - to run\n");
0060   printf("\n Enter transformation number: ");
0061   scanf("%d",&c);
0062   if (c == 1)
0063   {
0064    printf("\nEnter translate x integer value, (now = %d: ) ", transx);
0065    scanf("%f",&transx);
0066    printf("\n\nEnter translate y integer value, (now = %d: ) ", transy);
0067    scanf("%f",&transy);
0068    count = count + 1;
0069
0070                                      /* Update the transform matrix  */
```

```
0071    for (i=0; i < 9; i+=3)
0072         *(xfrm + i) += (*(xfrm + i + 2) * transx);
0073    for (i=1; i < 9; i+=3)
0074         *(xfrm + i) += (*(xfrm + i + 1) * transy);
0075   }
0076   else if (c == 2)
0077   {
0078    printf("\nEnter scale x value, (now = %.1f: ) ", scalex);
0079    scanf("%f",&scalex);
0080    printf("\n\nEnter scale y value, (now = %.1f: ) ", scaley);
0081    scanf("%f",&scaley);
0082    count = count + 1;
0083
0084                                        /* Update the transform matrix */
0085    for (i=0; i < 9; i+=3)
0086         *(xfrm + i) = *(xfrm + i) * scalex;
0087    for (i=1; i < 9; i+=3)
0088         *(xfrm + i) = *(xfrm + i) * scaley;
0089   }
0090   else if (c == 3)
0091   {
0092    printf("\nEnter rotate x value; + cw and - ccw, (now = %.1f: ) ", rotate);
0093    scanf("%f",&rotate);
0094                                                  /* Update the transform matrix */
0095    radfn = (double) (rotate * (3.14159 / 180.0));
0096    cose = cos(radfn);
0097    sine = sin(radfn);
0098    *(hold) = *(xfrm);
0099    *(hold + 3) = *(xfrm + 3);
0100    *(hold + 6) = *(xfrm + 6);
0101    for (i=0; i < 9; i+=3)
0102      *(xfrm+i)=(*(xfrm+i) * cose)+(*(xfrm+i+1) * sine);
0103    for (i=1; i < 9; i+=3)
0104      *(xfrm+i)=(*(xfrm+i) * cose)-(*(hold + i - 1) * sine);
0105    count = count + 1;
0106   }
0107   else ;
0108  }
0109  if (count == 0) return;
0110  else ;
0111
0112  printf("\nRunning transformation please wait...\n");
0113
0114   time(&start);                                  /* start processor timer */
0115
0116   FindInverse();                                 /* Find inverse matrix */
0117
0118
0119  if (origin == CENTER)
0120  {
0121   yspan = (yright - yleft);
0122   xspan = (xright - xleft);
0123   for (i = (((yspan)/2)*scaley); i >= ((-(yspan)/2)*scaley);
0124           i--)
0125   {
0126    for (j = ((-(xspan)/2)*scalex); j <= ((xspan/2)*scalex);
0127            j++)
0128    {
0129     oldx = (i * (*(invxfrm + 1))) + (j * (*(invxfrm + 4)));
0130     oldx = oldx + *(invxfrm + 7);
0131     oldy = (i * (*(invxfrm))) + (j * (*(invxfrm + 3)));
0132     oldy = oldy + *(invxfrm + 6);
0133     xold = (int) oldx;
0134     if (((float)(oldx - xold)) >= 0.5)
0135         xold = xold + 1;
0136     if (((float)(oldx - xold)) <= -0.5)
0137         xold = xold - 1;
0138     xold = xold + ((xspan) / 2) + xleft;
0139     if ((xold >= xleft) && (xold <= xright))
0140     {
```

```
0141      yold = (int) oldy;
0142      if (((float)(oldy - yold)) >= 0.5)
0143         yold = yold + 1;
0144      if (((float)(oldy - yold)) <= -0.5)
0145         yold = yold - 1;
0146      yold = yold + ((yspan) / 2) + yleft;
0147      if ((yold >= yleft) && (yold <= yright))
0148      {
0149       Brpixela(xold,yold);
0150       Bwpixelb((j+xleft+((xspan)/2)), (yleft+i+((yspan)/2)));
0151      }
0152     }
0153    }
0154   }
0155  }
0156  else ;
0157  time(&finish);
0158  printf("\nDisplay took %f", difftime(finish,start));
0159  HITANYKEY
0160 }                                                    /* end of Xform */
0161
0162 /*****************************************************************************
0163 *                                                                            *
0164 * Description:  Swap row in transform                                        *
0165 * Input: row1, row2                                                          *
0166 * Output: none                                                               *
0167 * Reference: none                                                            *
0168 * Used in: FindInverse().                                                    *
0169 *                                                                            *
0170 *****************************************************************************/
0171 swaprow(int row1,int row2)
0172 {
0173 int i;
0174 float temp;
0175  for (i=0; i < 3; i++)
0176  {
0177   temp = *(xfrm + (row1 * 3) + i);
0178   *(xfrm + (row1 * 3) + i) = *(xfrm + (row2 * 3) + i);
0179   *(xfrm + (row2 * 3) + i) = temp;
0180   temp = *(invxfrm + (row1 * 3) + i);
0181   *(invxfrm + (row1 * 3) + i) = *(invxfrm + (row2 * 3) + i);
0182   *(invxfrm + (row2 * 3) + i) = temp;
0183  }
0184 }                                                    /* End swaprow */
0185
0186 /*****************************************************************************
0187 *                                                                            *
0188 * Description:  Find inverse transform of matrix using row operation         *
0189 * Input: none                                                                *
0190 * Output: none                                                               *
0191 * Reference: C-lib. functions                                                *
0192 * Used in: Xform().                                                          *
0193 *                                                                            *
0194 *****************************************************************************/
0195 FindInverse()
0196 {
0197 int i;
0198 float con;
0199
0200                                     /* Initialize inverse matrix */
0201 for (i=0; i < 9; i++)
0202         *(invxfrm + i) = 0.0;
0203 *(invxfrm) = 1.0;
0204 *(invxfrm + 4) = 1.0;
0205 *(invxfrm + 8) = 1.0;
0206
0207                                     /* Check to make sure an inverse exists */
0208  if ((*(xfrm) <= 0.001) && (*(xfrm) >= -0.001))
0209  {
0210   if ((*(xfrm + 1) != 0) && (*(xfrm + 3) != 0))
```

```
0211          swaprow(0,1);
0212    else if ((*(xfrm + 2) != 0) && (*(xfrm + 3) != 0))
0213          swaprow(0,2);
0214    else
0215    {
0216     printf("\nNo inverse exists for your\n");
0217     printf("transformation matrix.\n");
0218     exit();
0219    }
0220   }
0221   if ((*(xfrm + 4) <= 0.001) && (*(xfrm + 4) >= -0.001))
0222   {
0223    printf("\n made it\n");
0224    if ((*(xfrm + 3) != 0) && (*(xfrm + 1) != 0))
0225          swaprow(0,1);
0226    else if ((*(xfrm + 5) != 0) && (*(xfrm + 7) != 0))
0227          swaprow(1,2);
0228    else
0229    {
0230     printf("\nNo inverse exists for your\n");
0231     printf("transformation matrix.\n");
0232     exit();
0233    }
0234   }
0235
0236   if ((*(xfrm + 8) <= 0.001) && (*(xfrm + 8) >= -0.001))
0237   {
0238    if ((*(xfrm + 2) != 0) && (*(xfrm + 6) != 0))
0239          swaprow(0,2);
0240    else if ((*(xfrm + 5) != 0) && (*(xfrm + 7) != 0))
0241          swaprow(1,2);
0242    else
0243    {
0244     printf("\nNo inverse exists for your\n");
0245     printf("transformation matrix.\n");
0246     exit();
0247    }
0248   }
0249
0250                                        /* Use row operations to find inverse  */
0251   if (*(xfrm + 3) != 0)
0252   {
0253    con = -(*(xfrm + 3) / *(xfrm));
0254    for (i=3; i < 6; i++)
0255    {
0256     *(invxfrm + i) += (con * (*(invxfrm + i - 3)));
0257     *(xfrm + i) += (con *(*(xfrm + i - 3)));
0258    }
0259   }
0260   if (*(xfrm + 6) != 0)
0261   {
0262    con = -(*(xfrm + 6) / *(xfrm));
0263    for (i=6; i < 9; i++)
0264    {
0265     *(invxfrm + i) += (con * (*(invxfrm + i - 6)));
0266     *(xfrm + i) += (con * (*(xfrm + i - 6)));
0267    }
0268   }
0269   if (*(xfrm + 1) != 0)
0270   {
0271    con = -(*(xfrm + 1) / *(xfrm + 4));
0272    for (i = 0; i < 3; i++)
0273    {
0274     *(invxfrm + i) += (con * (*(invxfrm + i + 3)));
0275     *(xfrm + i) += (con * (*(xfrm + i + 3)));
0276    }
0277   }
0278   if (*(xfrm + 7) != 0)
0279   {
0280    con = -(*(xfrm + 7) / *(xfrm + 4));
```

```
0281    for (i = 6; i < 9; i++)
0282    {
0283     *(invxfrm + i) += (con * (*(invxfrm + i - 3)));
0284     *(xfrm + i) += (con * (*(xfrm + i - 3)));
0285    }
0286   }
0287   if (*(xfrm + 2) != 0)
0288   {
0289    con = -(*(xfrm + 2) / *(xfrm + 8));
0290    for (i = 0; i < 3; i++)
0291    {
0292     *(invxfrm + i) += (con * (*(invxfrm + i + 6)));
0293     *(xfrm + i) += (con * (*(xfrm + i + 6)));
0294    }
0295   }
0296   if (*(xfrm + 5) != 0)
0297   {
0298    con = -(*(xfrm + 5) / *(xfrm + 8));
0299    for (i=3; i < 6; i++)
0300    {
0301     *(invxfrm + i) += (con * (*(invxfrm + i + 3)));
0302     *(xfrm + i) += (con * (*(xfrm + i + 3)));
0303    }
0304   }
0305   for (i=0; i < 3; i++)
0306          *(invxfrm + i) *= (1 / *(xfrm));
0307   for (i=3; i < 6; i++)
0308          *(invxfrm + i) *= (1 / *(xfrm + 4));
0309   for (i=6; i < 9; i++)
0310          *(invxfrm + i) *= (1 / *(xfrm + 8));
0311                                                  /*      for(i=0;i<9;i++)
0312                                      printf("\n inv = %f\n",*(invxfrm + i));  */
0313 } /* end FindInverse  */
0314
0315
0316 /* EOF */
0317
0318
```

```
0001 /*****************************************************************************
0002 * Program listings for Quadtree Algorithms for Image Processing Thesis. *
0003 *                                                                           *
0004 * The following modules were supplied by RIT Image Processing Lab:         *
0005 *       "itexpfg.h"                                                         *
0006 *       "stdtyp.h"                                                          *
0007 *       Initial.c                                                           *
0008 *                                                                           *
0009 * Input: Program prompt commands from standard input/ouput (keyboard).     *
0010 * Output: Dialogue with user (UI).                                         *
0011 * References: modules in Xform.c, Videorw.c, Initial.c.                    *
0012 *                                                                           *
0013 * Development environment: MS-C & Quick-C compiler 5.1                     *
0014 * Hardware: IBM-PC clone.                                                  *
0015 * Files needed in current directory:    "itexpfg.h"                        *
0016 *                                       "stdtyp.h"                         *
0017 *                                       Initial.c                          *
0018 *                                       Videorw.c                          *
0019 *                                       Xform.c                            *
0020 *                                       Thesisqt.c                         *
0021 *                                       Thmake                             *
0022 * To Compile: c:\>make thmake                                              *
0023 * To run: c:\>Thesisqt                                                     *
0024 *                                                                           *
0025 * Developed by: Jim Isaac Benjamin                                         *
0026 * Name of program: Thesis.c                                                *
0027 * Dated: 2/22/91                                                           *
0028 * version: 1.0                                                             *
0029 * Updates description: released March'91                                   *
0030 *****************************************************************************/
0031
0032                                          /* INCLUDE FILES */
0033 #include "itexpfg.h"
0034 #include "stdtyp.h"
0035 #include <stdio.h>
0036 #include <stdlib.h>
0037 #include <string.h>
0038 #include <conio.h>
0039 #include <ctype.h>
0040 #include <math.h>
0041 #include <time.h>
0042 #include <dos.h>
0043 #include <graph.h>
0044
0045
0046 #define HITANYKEY printf("\nPress any key to continue");getch();
0047
0048                                          /* CONSTANTS */
0049 #define BYTE    unsigned char
0050 #define WORD    unsigned int
0051 #define LONG    long
0052 #define ULONG   unsigned long
0053 #define MAX_LINE_SIZE   512
0054 #define HEADER_SIZE     81
0055 #define MAX_GRAY_LEVELS 256
0056 #define BLACK_QUADRANT  3
0057 #define WHITE_QUADRANT  0
0058 #define GRAY1_QUADRANT  2
0059 #define GRAY2_QUADRANT  4
0060 #define GRAY3_QUADRANT  5
0061 #define GRAY4_QUADRANT  6
0062 #define GRAY5_QUADRANT  7
0063 #define GRAY6_QUADRANT  8
0064 #define MIXED_QUADRANT  1
0065 #define BLACK           0
0066 #define WHITE           255
0067 #define GRAY1           63
0068 #define GRAY2           95
0069 #define GRAY3           129
0070 #define GRAY4           159
```

```
0071 #define GRAY5              191
0072 #define GRAY6              223
0073 #define GRAY               128
0074 #define MAX_COUNT                  32767L
0075 #define MIN_COUNT                  32768L
0076 #define KB_C_N_ENTER               (int)13
0077 #define KB_C_N_ESC                 (int)27
0078 #define BEEP                       putch(7)
0079 #define NUL                        (BYTE)0
0080
0081 #define FALSE              0
0082 #define TRUE               1
0083
0084                                              /* FUNCTION RETURN VALUES */
0085 #define OK                 0
0086 #define ABORT              1
0087 #define FREAD_ERR          -1
0088 #define FWRITE_ERR         -2
0089 #define FSEEK_ERR          -3
0090 #define FOPEN_ERR          -4
0091
0092 UWORD powers_of_2[10] = {1, 2, 4, 8, 16, 32, 64, 128, 256, 512};
0093 ULONG powers_of_4[10] = {1, 4, 16, 64, 256, 1024, 4096, 16384, 65536, 262114};
0094
0095                                     /* KEY-FUNCTION DECLARATIONS */
0096 void main(int argc,unsigned char *argv[]);
0097 void MainMenu(int x, int y, int dx, int dy, int extval, int extract);
0098 void AreaOfIntrest(int *x, int *y, int *dx, int *dy);
0099 void ThresholdImage(int x, int y, int dx, int dy);
0100 void Extraction(int *extval, int *extract);
0101 int EncodeQuadTree(int x, int y, int width);
0102 int DecodeQuadTree(int *extval, int *extract);
0103 void Blackscr();
0104 void Whitescr();
0105 void Load();
0106                                     /* UTILITIES-FUNCTION DECLARATIONS */
0107 void SetupHardware(void);
0108 BYTE GetQuadrantValue(int x, int y, int width);
0109 void DisplayQuadrantValue(int x, int y, int width, BYTE val,
0110                           int *extval, int *extract);
0111 int LoadImage(unsigned char *fn);
0112 void ReadHeader(FILE *fp, int *dx, int *dy);
0113 int GetCount(int *data_p);
0114 int PowerOfTwo(int num);
0115 void MakePath(unsigned char *fn, unsigned char *ext);
0116 void SetupQuadrantCoordinates(ULONG offset, int level, int *x, int *y,
0117                                     int rotation);
0118 void Rotate(int *val);
0119 void Blanksect(int x, int y, int dx);
0120
0121                                     /* GLOBAL DECLARATIONS */
0122 int x = 0, y = 0, extval = 0, extract = 0;
0123 int dx = MAX_LINE_SIZE;
0124 int dy = MAX_LINE_SIZE;
0125
0126 /*****************************************************************************
0127 *                                                                           *
0128 * Description: Main(), provides user interface (UI).                        *
0129 *                                                                           *
0130 * Input: From standard input/output (keyboard).                             *
0131 * Output: User dialogue                                                     *
0132 * Reference: HardwareSetup(), LoadImage(), MainMenu().                      *
0133 *                                                                           *
0134 *****************************************************************************/
0135 void main(int argc,unsigned char *argv[])
0136 {
0137 int rsp;
0138
0139   _clearscreen(_GCLEARSCREEN);
0140                                     /* init variables for AOI and extraction */
```

```
0141   x = y = extval = extract = 0;
0142   dx = dy = 512;
0143
0144
0145   if (argc > 2)
0146   {
0147    printf("\nusage: qt [file]");
0148    exit(0);
0149   }
0150                                    /* do hardware initialization */
0151   SetupHardware();
0152   if (argc > 1)
0153   {
0154    if (LoadImage(argv[1]) != OK)
0155    {
0156     printf("\nError loading image %s", argv[1]);
0157     exit(0);
0158    }
0159   }
0160
0161   MainMenu(x, y, dx, dy, extval, extract);
0162
0163 }                                          /* END OF MAIN */
0164
0165
0166 /*****************************************************************************
0167 *                                                                           *
0168 * Description: MainMenu(), enables pop-up user menu.                        *
0169 *                                                                           *
0170 * Input: From standard input/output (keyboard).                             *
0171 * Output: Pop-up menu to monitor.                                           *
0172 * Reference: AreaOfInterest(), Blanksect(), DecodeQuadTree(),               *
0173 *            EncodeQuadTree(), Extraction(), ThresholdImage()               *
0174 *            Xform(), Load(), Blackscr(), Whitescr(), exit().               *
0175 * Used in: main().                                                          *
0176 *                                                                           *
0177 *****************************************************************************/
0178 void MainMenu(int x, int y, int dx, int dy, int extval, int extract)
0179 {
0180 int c;
0181
0182  for (;;)
0183  {
0184    _clearscreen(0);
0185   printf("\n\n");
0186   printf("\n                              Thesis-experiments");
0187   printf("\n                          Quad Tree Image Encoding");
0188   printf("\n                                    vs.");
0189   printf("\n                           Linear Transformations");
0190   printf("\n\n\n");
0191   printf("\n     A) Area-of-Interest");
0192   printf("\n     I) Isolate pixels inside A-O-I");
0193   printf("\n     D) Decode image from Q-Tree");
0194   printf("\n     E) Encode image with Q-Tree");
0195   printf("\n     S) Separate / Extract gray level");
0196   printf("\n     T) Threshold Image");
0197   printf("\n     X) Xformation");
0198   printf("\n     L) Load");
0199   printf("\n     B) Black out display");
0200   printf("\n     W) White out display");
0201   printf("\n\n  Q) Quit");
0202   printf("\n\n\nEnter letter of choice: ");
0203   c = getch();
0204   switch(c)
0205   {
0206     case 'q':
0207     case 'Q':   printf("\n\n");
0208                 printf("\n>> So long for now!! <<");
0209                 exit(0);
0210     case 't':
```

```
0211      case 'T':   ThresholdImage(x, y, dx, dy);
0212                   clearscreen(0);
0213                  break;
0214      case 'a':
0215      case 'A':   AreaOfIntrest(&x, &y, &dx, &dy);
0216                   clearscreen(0);
0217                  break;
0218      case 'i':
0219      case 'I':   printf("\n\nRunning please wait...");
0220                  Blanksect(x, y, dx);
0221                   clearscreen(0);
0222                  break;
0223      case 'e':
0224      case 'E':   EncodeQuadTree(x, y, dx);
0225                   clearscreen(0);
0226                  break;
0227      case 'd':
0228      case 'D':   DecodeQuadTree(&extval, &extract);
0229                   clearscreen(0);
0230                  break;
0231      case 'x':
0232      case 'X':   Xform(x, y, dx);
0233                   clearscreen(0);
0234                  break;
0235      case 's':
0236      case 'S':   Extraction(&extval, &extract);
0237                  DecodeQuadTree(&extval, &extract);
0238                   clearscreen(0);
0239                  break;
0240      case 'l':
0241      case 'L':   Load();
0242                   clearscreen(0);
0243                  break;
0244      case 'b':
0245      case 'B':   Blackscr();
0246                   clearscreen(0);
0247                  break;
0248      case 'w':
0249      case 'W':   Whitescr();
0250                   clearscreen(0);
0251                  break;
0252      default:    break;
0253    }                                        /* End of switch */
0254  }                                          /* END OF FOR */
0255 }                                           /* End of MainMenu() */
0256
0257 /*****************************************************************************
0258 *                                                                           *
0259 * Description: SetupHardware, initialize system's Frame Grabber             *
0260 *              Hardware, provided by RIT'S Image Processing Lab             *
0261 * Input: none                                                               *
0262 * Output: none                                                              *
0263 * Reference: sethdw(), setdim(), fgon(), these modules are located in       *
0264 *            Initial.c.                                                     *
0265 * Used in: main().                                                          *
0266 *                                                                           *
0267 *****************************************************************************/
0268 void SetupHardware(void)
0269 {
0270    sethdw( 0x300, 0xD0000L, DUAL );      /* Init hardware */
0271    setdim( 512, 512, 8);
0272    fgon();
0273 }                                           /* End of SetupHardware() */
0274
0275 /*****************************************************************************
0276 * Description:  Determine and mark user area of interest                    *
0277 * Input: x, y, dx, dy                                                       *
0278 * Output: Error message and changes x, y, dx, dy values                     *
0279 * Reference: scanf();                                                       *
0280 * Used in: MainMenu().                                                      *
```

```
0281 *                                                                      *
0282 ************************************************************************/
0283 void AreaOfIntrest(int *x, int *y, int *dx, int *dy)
0284 {
0285 int t1, t2, t3;
0286 _clearscreen(_GCLEARSCREEN);
0287  for (;;)
0288  {
0289   printf("\n\nEnter the horizontal image orgin, (now = 0): ");
0290   scanf("%d",&t1);
0291   break;
0292  }
0293  for (;;)
0294  {
0295   printf("\n\nEnter the vertical image orgin, (now = 0): ");
0296   scanf("%d",&t2);
0297   break;
0298  }
0299  for (;;)
0300  {
0301   printf("\n\nEnter the image width, (now = 512): ");
0302   scanf("%d",&t3);
0303   break;
0304  }
0305  *x = t1;
0306  *y = t2;
0307  *dx = t3;
0308  *dy = t3;
0309 }                                       /* End of AreaOfIntrest() */
0310
0311
0312 /************************************************************************
0313 *                                                                      *
0314 * Input: x, y, width                                                   *
0315 * Output: none                                                         *
0316 * Reference: Bwhlinea().                                               *
0317 * Used in: MainMenu().                                                 *
0318 *                                                                      *
0319 ************************************************************************/
0320 void Blanksect(int x, int y, int width)
0321 {
0322 int i, j;
0323 BYTE blackline[MAX_LINE_SIZE];
0324
0325  for (i=0; i < MAX_LINE_SIZE; i++)              /* init blankline array */
0326  {
0327   blackline[i] = BLACK;
0328  }
0329  for (j=0; j < y; j++)                                          /* rows */
0330  {
0331   Bwhlinea(0, j, MAX_LINE_SIZE, blackline);
0332  }
0333  for (j=y; j < y + width; j++)                                  /* rows */
0334  {
0335   Bwhlinea(0, j, x, blackline);
0336  }
0337  for (j=y; j < y + width; j++)                                  /* rows */
0338  {
0339   Bwhlinea((x + width), j, (MAX_LINE_SIZE - (x + width)), blackline);
0340  }
0341  for (j=y + width; j < MAX_LINE_SIZE; j++)                      /* rows */
0342  {
0343   Bwhlinea(0, j, MAX_LINE_SIZE, blackline);
0344  }
0345 }                                               /* End Blanksect() */
0346
0347
0348 /************************************************************************
0349 *                                                                      *
0350 * Description: ThresholdImage(), reduce image gray levels              *
```

```
0351 * Input: x, y, dx, dy and user define values                              *
0352 * Output: user prompt                                                     *
0353 * Reference: Brhlinea(), Bwhlinea() and C-library functions               *
0354 * Used in: MainMenu().                                                    *
0355 *                                                                         *
0356 **************************************************************************/
0357 void ThresholdImage(int x, int y, int dx, int dy)
0358 {
0359 int thres_val = 128;
0360 int i, src_byte;
0361 BYTE scanline[MAX_LINE_SIZE], c;
0362 _clearscreen(_GCLEARSCREEN);
0363
0364  printf("\n(a) - Threshold value 128");
0365  printf("\n(b) - Threshold 256 to 8: ");
0366  c = getch();
0367
0368  if (c == 'a' || c == 'A')
0369  {
0370   for (;;)
0371   {
0372    printf("\n\nEnter the threshold value, (now = 128)");
0373    scanf("%d",&thres_val);
0374    break;
0375   }                                        /* end of for ;; loop */
0376   for (i =y; i < y + dy; i++)
0377   {
0378    Brhlinea(x, i, dx, &scanline[x]);
0379                                            /* move across the line, left to
0380                                            right, setting each byte to white
0381                                            or black */
0382    for (src_byte = x; src_byte < x + dx; src_byte++)
0383    {
0384     if((int)scanline[src_byte] >= thres_val)
0385         scanline[src_byte] = WHITE;
0386     else  scanline[src_byte] = BLACK;
0387    }
0388     Bwhlinea(x, i, dx, &scanline[x]);
0389   }                                        /* end of for i < y + dy */
0390  }                                         /* end of if */
0391  else
0392  {
0393   for (i =y; i < y + dy; i++)      /* move down each row in image */
0394   {
0395    Brhlinea(x, i, dx, &scanline[x]);
0396                                    /* move across each column in image,
0397                                       setting each byte to black,
0398                                       gray1, gray2, gray3, gray4, gray5,
0399                                       gray6 or white */
0400    for (src_byte = x; src_byte < x + dx; src_byte++)
0401    {
0402     if(((int)scanline[src_byte] >= 0) && ((int)scanline[src_byte] <= 31))
0403         scanline[src_byte] = BLACK;
0404     else if(((int)scanline[src_byte] >= 32) &&
0405                 ((int)scanline[src_byte] <= 63))
0406         scanline[src_byte] = GRAY1;
0407     else if(((int)scanline[src_byte] >= 64) &&
0408                 ((int)scanline[src_byte] <= 95))
0409         scanline[src_byte] = GRAY2;
0410     else if(((int)scanline[src_byte] >= 96) &&
0411                 ((int)scanline[src_byte] <= 129))
0412         scanline[src_byte] = GRAY3;
0413     else if(((int)scanline[src_byte] >= 130) &&
0414                 ((int)scanline[src_byte] <= 159))
0415         scanline[src_byte] = GRAY4;
0416     else if(((int)scanline[src_byte] >= 160) &&
0417                 ((int)scanline[src_byte] <= 191))
0418         scanline[src_byte] = GRAY5;
0419     else if(((int)scanline[src_byte] >= 192) &&
0420                 ((int)scanline[src_byte] <= 223))
```

```
0421          scanline[src_byte] = GRAY6;
0422       else         scanline[src_byte] = WHITE;
0423
0424       }
0425     Bwhlinea(x, i, dx, &scanline[x]);
0426    }                                          /* end of for */
0427   }                                           /* end of else */
0428  }                                            /* End of ThresholdImage() */
0429
0430 /*****************************************************************************
0431  *                                                                           *
0432  * Description: Construct  a quad tree and output a file to disk name        *
0433                 encodeqt.out.                                               *
0434  *              The breadth-first approach is implemented, all nodes         *
0435  *              (i.e. father, sons) are stored as a byte of data, and        *
0436  *              beneath homogeneous  regions are not stored.  This           *
0437  *              approach facilitates the amount of space required            *
0438  *              for a given image.                                           *
0439  * Input: x, y, width                                                        *
0440  * Output: message to screen                                                 *
0441  * Reference: GetQuadrantValue() and C-library functions                     *
0442  * Used in: MainMenu().                                                      *
0443  *                                                                           *
0444  *****************************************************************************/
0445 int EncodeQuadTree(int x, int y, int width)
0446 {
0447
0448 BYTE curr_scanline[4];
0449 BYTE prev_line;
0450
0451 int mixed_level;
0452 int block_width;
0453 int level;
0454 int max_level;
0455 int i, tmp1, tmp2;
0456
0457 LONG prev_ctr, elements_in_prev;
0458
0459 FILE *in_fp, *out_fp, *qt_fp;
0460 unsigned char in_fname[20];
0461 unsigned char out_fname[20];
0462
0463 _clearscreen(_GCLEARSCREEN);
0464
0465   if (NULL == (qt_fp = fopen("encodeqt.out","wb")))
0466   {
0467    printf("\nError opening file encodeqt.out for write");
0468    printf("\n>>hit any key to continue<<");
0469    getch();
0470    return(FOPEN_ERR);
0471   }
0472
0473   max_level = PowerOfTwo(width);
0474
0475                                      /* write out header information */
0476   fwrite(&x, sizeof(int), 1, qt_fp);
0477   fwrite(&y, sizeof(int), 1, qt_fp);
0478   fwrite(&width, sizeof(int), 1, qt_fp);
0479
0480   for (level = 0; level <= max_level; level++)
0481   {
0482    printf("\rEncoding image into quad tree, level now = %d", level);
0483
0484    mixed_level = FALSE;
0485
0486    if (level == 0)
0487    {
0488     in_fname[0] = NUL;
0489    }
0490    else
```

```
0491    {
0492     strcpy(in_fname, out_fname);
0493     if (NULL == (in_fp = fopen(in_fname, "rb")))
0494     {
0495      printf("\nError opening file in_fname for read");
0496      printf("\n>>hit any key to continue<<");
0497      getch();
0498      return(FOPEN_ERR);
0499     }
0500    }
0501
0502                            /* create a filename for output image of form
0503                               "qtmpXX", where XX is the current level. */
0504    sprintf(out_fname, "qtmp%02d.out", level);
0505
0506    if (NULL == (out_fp = fopen(out_fname, "wb")))
0507    {
0508      printf("\nError opening file out_fname for write");
0509      printf("\n>>hit any key to continue<<");
0510      getch();
0511      return(FOPEN_ERR);
0512    }
0513
0514                    /* If we are at the first level, we need to
0515                       generate a quadrant value for the whole image.  */
0516    if (level == 0)
0517    {
0518     curr_scanline[0] = GetQuadrantValue(x, y, width);
0519     fwrite(curr_scanline, sizeof(BYTE), 1, qt_fp);
0520     fwrite(curr_scanline, sizeof(BYTE), 1, out_fp);
0521     if (curr_scanline[0] == MIXED_QUADRANT) mixed_level = TRUE;
0522    }
0523    else
0524    {
0525     block_width = width >> level;        /* shift right is to divide */
0526     elements_in_prev = 1 << ((level - 1) * 2);
0527     for (prev_ctr = 0; prev_ctr < elements_in_prev; prev_ctr++)
0528     {
0529                     /* if we are under an all black, all white or all grayi
0530                        quadrant, the record this internally but don't
0531                        write it to our quad tree file */
0532      fread(&prev_line, sizeof(BYTE), 1, in_fp);
0533      if (prev_line != MIXED_QUADRANT)
0534      {
0535       for (i=0; i < 4; i++)
0536        curr_scanline[i] = prev_line;
0537       fwrite(curr_scanline, sizeof(BYTE), 4, out_fp);
0538      }
0539                     /* if we are not under an all black, all white or all
0540                        all grayi quadrant.
0541                        Calculate the quadrant values and
0542                        record it internally and in out file. */
0543      else
0544      {
0545       SetupQuadrantCoordinates(prev_ctr, level-1, &tmp1, &tmp2, 0);
0546       tmp1 *= block_width;
0547       tmp2 *= block_width;
0548       tmp1 *= 2;
0549       tmp2 *= 2;
0550
0551       curr_scanline[0] = GetQuadrantValue(x + tmp1, y + tmp2, block_width);
0552       curr_scanline[1] = GetQuadrantValue(x + tmp1 + block_width,
0553                                           y + tmp2, block_width);
0554       curr_scanline[2] = GetQuadrantValue(x + tmp1, y + tmp2 + block_width,
0555                                           block_width);
0556       curr_scanline[3] = GetQuadrantValue(x + tmp1 + block_width,
0557                                           y + tmp2 + block_width, block_width);
0558       fwrite(curr_scanline, sizeof(BYTE), 4, qt_fp);
0559       fwrite(curr_scanline, sizeof(BYTE), 4, out_fp);
0560       for (i=0; i < 4; i++)
```

```
0561        {
0562         if (curr_scanline[i] == MIXED_QUADRANT) mixed_level = TRUE;
0563        }
0564       }          /* END ELSE NOT SOLID QUADRANT */
0565     }/* END FOR (prev_ctr) */
0566    }/* END ELSE (not first level) */
0567
0568                                    /* if all elements in current
0569                                       level are white, black or grayi, then
0570                                       we're done. */
0571    if (!level)
0572     ;
0573    else if (in_fp != NULL) fclose(in_fp);
0574    if (out_fp != NULL) fclose(out_fp);
0575    if (level != 0) remove(in_fname);
0576    if (!mixed_level) break;
0577    }
0578
0579   if (qt_fp != NULL) fclose(qt_fp);
0580   remove(out_fname);
0581   printf("\nHit any key to continue ......");
0582   getch();
0583   return(OK);
0584  }                                        /* End of EncodeQuadTree() */
0585
0586  /*************************************************************************
0587  *                                                                        *
0588  * Description: Determine present quadrant value                          *
0589  * Input: x, y, and width                                                 *
0590  * Output: unsigned char quadrant value                                   *
0591  * Reference: Brhlinea(), and C-library functions                         *
0592  * Used in: EncodeQuadTree().                                             *
0593  *                                                                        *
0594  *************************************************************************/
0595  BYTE GetQuadrantValue(int x, int y, int width)
0596  {
0597  BYTE scanline[MAX_LINE_SIZE];
0598  ULONG white, gray1, gray2, gray3, gray4, gray5, gray6, black;
0599
0600  int i, j;
0601                                   /* check pixels in the sub-image
0602                                      until we have searched the whole
0603                                      area or have determined intensity
0604                                      value of pixels present */
0605   black = white = 0;
0606   gray1 = gray2 = gray3 = gray4 = gray5 = gray6 = 0;
0607   for (i=0; i < width; i++)
0608   {
0609    Brhlinea(x, y+i, width, scanline);
0610    for (j=0; j < width; j++)
0611    {
0612     if (scanline[j] == 0)
0613         black++;
0614     else if(scanline[j] == 63)
0615         gray1++;
0616     else if(scanline[j] == 95)
0617         gray2++;
0618     else if(scanline[j] == 129)
0619         gray3++;
0620     else if(scanline[j] == 159)
0621         gray4++;
0622     else if(scanline[j] == 191)
0623         gray5++;
0624     else if(scanline[j] == 223)
0625         gray6++;
0626     else white++;
0627
0628     if (black && white)
0629         return(MIXED_QUADRANT);
0630     else if ((black && gray1) || (black && gray2) || (black && gray3)
```

```
0631              || (black && gray4) || (black && gray5) || (black && gray6))
0632            return(MIXED_QUADRANT);
0633     else if ((white && gray1) || (white && gray2) || (white && gray3)
0634              || (white && gray4) || (white && gray5) || (white && gray6))
0635            return(MIXED_QUADRANT);
0636     else if ((gray1 && gray2) || (gray1 && gray3)
0637              || (gray1 && gray4) || (gray1 && gray5) || (gray1 && gray6))
0638            return(MIXED_QUADRANT);
0639     else if ((gray2 && gray3) || (gray2 && gray4)
0640              || (gray2 && gray5) || (gray2 && gray6))
0641            return(MIXED_QUADRANT);
0642     else if ((gray3 && gray4) || (gray3 && gray5) || (gray3 && gray6))
0643            return(MIXED_QUADRANT);
0644     else if ((gray4 && gray5) || (gray4 && gray6))
0645            return(MIXED_QUADRANT);
0646     else if ((gray5 && gray6))
0647            return(MIXED_QUADRANT);
0648     else ;
0649    }
0650   }
0651   if (black && white)
0652         return(MIXED_QUADRANT);
0653   else if ((black && gray1) || (black && gray2) || (black && gray3)
0654              || (black && gray4) || (black && gray5) || (black && gray6))
0655            return(MIXED_QUADRANT);
0656   else if ((white && gray1) || (white && gray2) || (white && gray3)
0657              || (white && gray4) || (white && gray5) || (white && gray6))
0658            return(MIXED_QUADRANT);
0659   else if ((gray1 && gray2) || (gray1 && gray3)
0660              || (gray1 && gray4) || (gray1 && gray5) || (gray1 && gray6))
0661            return(MIXED_QUADRANT);
0662   else if ((gray2 && gray3) || (gray2 && gray4)
0663              || (gray2 && gray5) || (gray2 && gray6))
0664            return(MIXED_QUADRANT);
0665   else if ((gray3 && gray4) || (gray3 && gray5) || (gray3 && gray6))
0666            return(MIXED_QUADRANT);
0667   else if ((gray4 && gray5) || (gray4 && gray6))
0668            return(MIXED_QUADRANT);
0669   else if ((gray5 && gray6))
0670            return(MIXED_QUADRANT);
0671
0672   if (black) return(BLACK_QUADRANT);
0673   else if (gray1) return(GRAY1_QUADRANT);
0674   else if (gray2) return(GRAY2_QUADRANT);
0675   else if (gray3) return(GRAY3_QUADRANT);
0676   else if (gray4) return(GRAY4_QUADRANT);
0677   else if (gray5) return(GRAY5_QUADRANT);
0678   else if (gray6) return(GRAY6_QUADRANT);
0679   else return(WHITE_QUADRANT);
0680  }                                        /* End of GetQuadrantValue() */
0681
0682  /*************************************************************************
0683   *                                                                       *
0684   * Description:  Generate a display to video monitor of image file       *
0685   *               that was stored in Q-Tree format named encodeqt.out     *
0686   *               The image will be viewed from level-0 to level-n.       *
0687   *               The user is prompt for scale factor by power 2,         *
0688   *               rotation by increments of 90, and how to view border    *
0689   *               pixels.                                                 *
0690   * Input: extval, extract                                                *
0691   * Output: none                                                          *
0692   * Reference: PowerOfTwo() and C-library functions                       *
0693   * Used in: MainMenu().                                                  *
0694   *                                                                       *
0695   *************************************************************************/
0696  int     DecodeQuadTree(int *extval, int *extract)
0697  {
0698                          /* the program will be implementing a 'rotating stack'
0699                             containing the current and previous
0700                             levels of the quad tree.
```

```
0701                                We will monitor all nodes including terminal nodes
0702                                */
0703 BYTE curr_scanline[4], prev_line;
0704
0705 int mixed_level, c;
0706 int x, y, width;
0707 int block_width;
0708 int level;
0709 int max_level;
0710 int i, tmp1, tmp2;
0711 int rotation, zoom;
0712 int sort_order[4];
0713 int idborder = 0,
0714 transx = 0,
0715 transy = 0;
0716 char borderval = BLACK_QUADRANT;
0717
0718 int mode = 1;                                   /* set default display to auto */
0719
0720 LONG prev_ctr, elements_in_prev;
0721
0722 FILE *in_fp, *out_fp, *qt_fp;
0723 unsigned char in_fname[20];
0724 unsigned char out_fname[20];
0725
0726 time_t start, finish;
0727
0728 if (NULL == (qt_fp = fopen("encodeqt.out","rb")))
0729 {
0730  printf("\nError opening previously generated quad tree file");
0731  return(FOPEN_ERR);
0732 }
0733  _clearscreen(_GCLEARSCREEN);
0734 for (;;)
0735 {
0736  printf("\nEnter scaling factor, (now = 1) ");
0737  scanf("%d",&zoom);
0738  break;
0739 }
0740 for (;;)
0741 {
0742  printf("\nEnter number of quadrants through which to rotate, (now = 0) ");
0743  scanf("%d",&rotation);
0744  break;
0745 }
0746  printf("\nEnter an integer number representing the \n");
0747  printf("no. of pixels you want to move in the x  \n");
0748  printf("direction.\n");
0749  scanf("%d",&transx);
0750  printf("\nEnter an integer number representing the \n");
0751  printf("no. of pixels you want to move in the y \n");
0752  printf("direction.\n");
0753  scanf("%d",&transy);
0754
0755 printf("\nIdentify border pixels y/n: "); c = getch();
0756 if ((c == 'Y') || (c == 'y')) idborder = TRUE; else ;
0757 printf("\nDecoding quad tree please wait...");
0758 time(&start);
0759 if (zoom == 0) zoom = 1;
0760
0761                                    /* Whitescr(); use for SYSUDC */
0762
0763                                    /* read in out header information */
0764 fread(&x, sizeof(int), 1, qt_fp);
0765 fread(&y, sizeof(int), 1, qt_fp);
0766 fread(&width, sizeof(int), 1, qt_fp);
0767
0768 max_level = PowerOfTwo(width);
0769
0770                                    /* determine the display order of
```

```
0771                                    quadrants from the rotation */
0772 for (i=0; i < 4; i++)
0773 {
0774  tmp1 = 4 - rotation;
0775  if (tmp1 == 4) tmp1 = 0;
0776  tmp2 = i;
0777  while (tmp1)
0778  {
0779    Rotate(&tmp2);
0780    tmp1--;
0781  }
0782  sort_order[i] = tmp2;
0783 }
0784
0785 for (level=0; level <= max_level; level++)
0786 {
0787   mixed_level = FALSE;
0788
0789   if (level == 0)
0790     in_fname[0] = NUL;
0791   else
0792   {
0793     strcpy(in_fname, out_fname);
0794     if (NULL == (in_fp = fopen(in_fname, "rb"))) return(FOPEN_ERR);
0795   }
0796
0797                                  /* create a filename for output image
0798                                     of the form "qtmpXX", where XX is the
0799                                     current level.  */
0800  sprintf(out_fname, "qtmp%02d.out", level);
0801  if (NULL == (out_fp = fopen(out_fname, "wb"))) return(FOPEN_ERR);
0802
0803                                  /* if we are at the first level, we
0804                                     need to generate a quadrant value of the
0805                                     image.  */
0806  if (level == 0)
0807  {
0808    fread (curr_scanline, sizeof(BYTE), 1, qt_fp);
0809    fwrite(curr_scanline, sizeof(BYTE), 1, out_fp);
0810    DisplayQuadrantValue(0+transx, 0+transy, width,
0811                         curr_scanline[0], extval, extract);
0812    if (curr_scanline[0] == MIXED_QUADRANT) mixed_level = TRUE;
0813  }
0814  else
0815  {
0816    block_width = width >> level;
0817    block_width *= zoom;
0818
0819    elements_in_prev = 1 << ((level - 1) * 2);
0820
0821    for (prev_ctr = 0; prev_ctr < elements_in_prev; prev_ctr++)
0822    {
0823                                  /* if we are under all black, all
0824                                     white or all grayi
0825                                     quadrant, then record this
0826                                     internally. */
0827      fread(&prev_line, sizeof(BYTE), 1, in_fp);
0828      if (prev_line != MIXED_QUADRANT)
0829      {
0830        for (i=0; i < 4; i++) curr_scanline[i] = prev_line;
0831        fwrite(curr_scanline, sizeof(BYTE), 4, out_fp);
0832      }
0833
0834                                  /* we are not under an all black,
0835                                     all white  or all grayi quadrant.  Read in
0836                                     the quadrants values. */
0837      else
0838      {
0839        fread(curr_scanline, sizeof(BYTE) , 4, qt_fp);
0840        fwrite(curr_scanline, sizeof(BYTE), 4, out_fp);
```

```
0841
0842        SetupQuadrantCoordinates(prev_ctr, level-1, &tmp1, &tmp2, rotation);
0843        tmp1 *= (block_width * 2);
0844        tmp2 *= (block_width * 2);
0845
0846        if ((idborder) && (level == max_level))
0847         DisplayQuadrantValue(tmp1 + transx, tmp2 + transy, block_width,
0848                 borderval, extract, extval);
0849        else DisplayQuadrantValue(tmp1 + transx, tmp2 + transy, block_width,
0850                         curr_scanline[sort_order[0]], extract, extval);
0851        if ((idborder) && (level == max_level))
0852         DisplayQuadrantValue(tmp1 + block_width + transx,
0853                 tmp2 + transy, block_width,
0854                 borderval, extract, extval);
0855        else DisplayQuadrantValue(tmp1 + block_width + transx,
0856                 tmp2 + transy, block_width,
0857                 curr_scanline[sort_order[1]], extract, extval);
0858        if ((idborder) && (level == max_level))
0859         DisplayQuadrantValue(tmp1 + transx, tmp2 + block_width + transy,
0860                 block_width,
0861                 borderval, extract, extval);
0862        else DisplayQuadrantValue(tmp1 + transx, tmp2 + block_width + transy,
0863                 block_width, curr_scanline[sort_order[2]], extract, extval);
0864        if ((idborder) && (level == max_level))
0865         DisplayQuadrantValue(tmp1 + block_width + transx,
0866                 tmp2 + block_width + transy,
0867                 block_width, borderval, extract, extval);
0868        else DisplayQuadrantValue(tmp1 + block_width + transx,
0869                 tmp2 + block_width + transy,
0870                 block_width, curr_scanline[sort_order[3]], extract, extval);
0871        for (i=0; i < 4; i++)
0872        {
0873          if (curr_scanline[i] == MIXED_QUADRANT) mixed_level = TRUE;
0874        }
0875      }
0876    }             /* END FOR (prev_ctr) */
0877  }      /* END ELSE (not first level) */
0878
0879   if (!mode)
0880   {
0881    printf("\nPress <Esc> to return to main menu, any key to continue");
0882    if (getch() == KB_C_N_ESC) break;
0883   }
0884   else ;
0885                                          /* if all elements in current
0886                                             level are white, black or grayi,
0887                                             then we're done.  */
0888   if (!level)
0889    ;
0890  else if (in_fp != NULL) fclose(in_fp);
0891  if (out_fp !=NULL) fclose(out_fp);
0892  if (level != 0) remove(in_fname);
0893  if (!mixed_level) break;
0894 }
0895 if (qt_fp != NULL) fclose(qt_fp);
0896 remove(out_fname);
0897 time(&finish);
0898 *extract = 0;
0899 printf("\nDisplay took %f", difftime(finish,start));
0900 c = getch();
0901 return(OK);
0902 }                                        /* End of DecodeQuadTree() */
0903
0904 /*************************************************************************
0905 *                                                                       *
0906 * Description: Display quadrant value                                   *
0907 * Input: x,y,,width,val,extract,extval                                  *
0908 * Output: none                                                          *
0909 * Reference: Bwhlineb()                                                 *
0910 * Used in: DecodeQuadTree().                                            *
```

```
0911 *                                                                      *
0912 ************************************************************************/
0913 void DisplayQuadrantValue(int x, int y, int width, BYTE val, int *extract,
0914                           int *extval)
0915 {
0916 BYTE scanline[MAX_LINE_SIZE];
0917 BYTE out_val;
0918 int i, j;
0919
0920   if (x > 512 || y > 512) return;
0921
0922   if (!*extract)
0923   {
0924    if (val == WHITE_QUADRANT) out_val = WHITE;
0925    else if (val == GRAY1_QUADRANT) out_val = GRAY1;
0926    else if (val == GRAY2_QUADRANT) out_val = GRAY2;
0927    else if (val == GRAY3_QUADRANT) out_val = GRAY3;
0928    else if (val == GRAY4_QUADRANT) out_val = GRAY4;
0929    else if (val == GRAY5_QUADRANT) out_val = GRAY5;
0930    else if (val == GRAY6_QUADRANT) out_val = GRAY6;
0931    else if (val == BLACK_QUADRANT) out_val = BLACK;
0932    else out_val = GRAY;
0933   }
0934   else
0935   {
0936    if ((val == WHITE_QUADRANT) && (val == *extval)) out_val = BLACK;
0937    else if (val == WHITE_QUADRANT) out_val = WHITE;
0938    else if ((val == GRAY1_QUADRANT) && (val == *extval)) out_val = BLACK;
0939    else if (val == GRAY1_QUADRANT) out_val = GRAY1;
0940    else if ((val == GRAY2_QUADRANT) && (val == *extval)) out_val = BLACK;
0941    else if (val == GRAY2_QUADRANT) out_val = GRAY2;
0942    else if ((val == GRAY3_QUADRANT) && (val == *extval)) out_val = BLACK;
0943    else if (val == GRAY3_QUADRANT) out_val = GRAY3;
0944    else if ((val == GRAY4_QUADRANT) && (val == *extval)) out_val = BLACK;
0945    else if (val == GRAY4_QUADRANT) out_val = GRAY4;
0946    else if ((val == GRAY5_QUADRANT) && (val == *extval)) out_val = BLACK;
0947    else if (val == GRAY5_QUADRANT) out_val = GRAY5;
0948    else if ((val == GRAY6_QUADRANT) && (val == *extval)) out_val = BLACK;
0949    else if (val == GRAY6_QUADRANT) out_val = GRAY6;
0950    else if ((val == BLACK_QUADRANT) && (val == *extval)) out_val = BLACK;
0951    else if (val == BLACK_QUADRANT) out_val = BLACK;
0952    else out_val = GRAY;
0953   }
0954                                  /* set all pixels in the sub-image to
0955                                     the output value and write it out.  */
0956   for (i=0; i < width && y + i < MAX_LINE_SIZE; i++)
0957   {
0958     for (j=0; j < width && j + x < MAX_LINE_SIZE; j++)
0959     {
0960       scanline[j] = out_val;
0961     }
0962     Bwhlineb(x, y+i, j, scanline);
0963   }
0964 }                                        /* End of DisplayQuadrantValue() */
0965
0966 /************************************************************************
0967 *                                                                       *
0968 * Description: Determine quadrant boundaries                            *
0969 * Input: offset, level, x, y, rotation                                  *
0970 * Output: none                                                          *
0971 * Reference: Rotate()                                                   *
0972 * Used in: DecodeQuadTree().                                            *
0973 *                                                                       *
0974 ************************************************************************/
0975 void SetupQuadrantCoordinates(ULONG offset, int level, int *x, int *y,
0976                               int rotation)
0977 {
0978 int i;
0979 int off_p4, x_p2;
0980 int tmp_rot;
```

```
0981
0982    *x = *y = 0;
0983
0984    for (i=level-1; i >= 0; i--)
0985    {
0986      x_p2 =0;
0987      off_p4 = 0;
0988      tmp_rot = rotation;
0989      while (offset >= powers_of_4[i])
0990      {
0991        off_p4++;
0992        offset -= powers_of_4[i];
0993      }
0994
0995      while (tmp_rot)
0996      {
0997        Rotate(&off_p4);
0998        tmp_rot--;
0999      }
1000
1001      x_p2 = off_p4;
1002      while (x_p2 > 1) x_p2 -= 2;
1003      if (x_p2) *x += powers_of_2[i];
1004      if ((off_p4 - x_p2) >> 1) *y += powers_of_2[i];
1005    }
1006  }                                       /* End of SetupQuadrantCoordinates */
1007
1008  /*************************************************************************
1009  *                                                                        *
1010  * Description: Determine rotation of 90 degrees increments in CW dir.    *
1011  * Input: val                                                             *
1012  * Output: none                                                           *
1013  * Reference: none                                                        *
1014  * Used in: DecodeQuadTree().                                             *
1015  *                                                                        *
1016  *                                                                        *
1017  *************************************************************************/
1018  void Rotate(int *val)
1019  {
1020    if      (*val == 0) *val = 1;
1021    else if (*val == 1) *val = 3;
1022    else if (*val == 2) *val = 0;
1023    else if (*val == 3) *val = 2;
1024  }                                       /* End of Rotate() */
1025
1026
1027  /*************************************************************************
1028  *                                                                        *
1029  * Description: Determine power of number                                 *
1030  *                                                                        *
1031  * Input: number                                                          *
1032  * Output: power                                                          *
1033  * Reference: C-library functions                                         *
1034  * Used in: EncodeQuadTree(), DecodeQuadTree().                           *
1035  *                                                                        *
1036  *************************************************************************/
1037  int PowerOfTwo(int number)
1038  {
1039  int temp, power;
1040
1041                                    /* non-positive integers are not allowd */
1042    if (number <= 0) return(-1);
1043
1044                                    /* at each iteration, divide by 2 and */
1045    power = 0;
1046    while (number > 1)
1047    {
1048      temp = number;
1049      number >>= 1;
1050      if (number << 1 != temp) return(-1);
```

```
1051      power++;
1052    }
1053    return(power);
1054  }                                        /* End of PowerOfTwo() */
1055
1056  /*****************************************************************************
1057  *                                                                           *
1058  * Description:  LoadImage will input image into grame grabber.              *
1059  *               Determine path if none is specified.                        *
1060  * Input: fn                                                                 *
1061  * Output: none                                                              *
1062  * Reference: MakePath(), ReadHeader(), Bwhlinea() and C-library             *
1063  *            functions.                                                     *
1064  * Used in: Load().                                                          *
1065  *                                                                           *
1066  *****************************************************************************/
1067  int LoadImage(unsigned char *fn)
1068  {
1069  int dx, dy, i, bytes_read;
1070  unsigned char full_fpath[_MAX_PATH];
1071  unsigned char buff[512];
1072  FILE *fp;
1073
1074    strcpy(full_fpath, fn);
1075    MakePath(full_fpath, "IMG");
1076
1077    if (NULL == (fp = fopen(full_fpath, "rb"))) return(FOPEN_ERR);
1078
1079    ReadHeader(fp, &dx, &dy);
1080    fseek(fp, (long)(HEADER_SIZE), SEEK_SET);
1081
1082    printf("\nLoading image please wait .........");
1083    for (i=0; i < dy; i++)
1084    {
1085      if ((bytes_read = fread(buff, sizeof(BYTE), dx, fp)) != dx)
1086      {
1087        if (ferror(fp))
1088        {
1089          fclose(fp);
1090          return(FREAD_ERR);
1091        }
1092        Bwhlinea(0, i, bytes_read, buff);
1093        break;
1094      }
1095      Bwhlinea(0, i, dx, buff);
1096    }
1097
1098    fclose(fp);
1099
1100    return(OK);
1101  }                                        /* End of LoadImage() */
1102
1103  /*****************************************************************************
1104  *                                                                           *
1105  * Description:  Read header information from *.img file                     *
1106  * Input: fp, dx, dy                                                         *
1107  * Output: none                                                              *
1108  * Reference: C-library functions                                            *
1109  * Used in: LoadImage().                                                     *
1110  *                                                                           *
1111  *****************************************************************************/
1112  void ReadHeader(FILE *fp, int *dx, int *dy)
1113  {
1114  int header[32];
1115
1116    rewind(fp);
1117    fread(header, sizeof(int), 32, fp);
1118    *dx = header[2];
1119    *dy = header[3];
1120  }                                        /* End of ReadHeader() */
```

```
1121
1122 /*************************************************************************
1123 *                                                                       *
1124 * Description:  Determine path, from filename specified.                *
1125 *               Use default extension .img if none is none is specified *
1126 *               Reference imgpath if no path is specified.              *
1127 * Input: fn, ext                                                        *
1128 * Output: none                                                          *
1129 * Reference: C-library functions                                        *
1130 * Used in: LoadImage().                                                 *
1131 *                                                                       *
1132 *************************************************************************/
1133 void MakePath(unsigned char *fn, unsigned char *ext)
1134 {
1135 unsigned char fdrive[_MAX_DRIVE];
1136 unsigned char fpath[_MAX_DIR];
1137 unsigned char fname[_MAX_FNAME];
1138 unsigned char fext[_MAX_EXT];
1139 unsigned char full_path[_MAX_PATH];
1140 unsigned char *ps;
1141
1142   _splitpath( strupr(fn), fdrive, fpath, fname, fext );
1143
1144                                   /* if we have a pth, use it */
1145   if ( (fdrive[0] != NULL) || (fpath[0] != NULL) )
1146   {
1147     strcpy( full_path, fdrive );
1148     strcat( full_path, fpath );
1149   }
1150
1151                                  /* if no path, try to use path
1152                                     specified in 'imgpath'. */
1153
1154   else if ( (ps = getenv( "IMGPATH" )) != NULL)
1155     strcpy( full_path, ps);
1156
1157   strcat( full_path, fname );
1158
1159                                  /* if no extension, used the passed
1160                                     default.  */
1161   if ( (fext[0] == NULL) || (strcmp( fext, ".") == 0) )
1162   {
1163     strcat( full_path, ".");
1164     strcat( full_path, ext);
1165   }
1166   else
1167     strcat( full_path, fext );
1168
1169                                  /* return the path in same memory as
1170                                     the original filename.  */
1171   strcpy(fn, full_path);
1172 }                                      /* End of MakePath() */
1173
1174 /*************************************************************************
1175 *                                                                       *
1176 * Description: Load image from menu                                     *
1177 * Input:none                                                            *
1178 * Output: none                                                          *
1179 * Reference: LoadImage()                                                *
1180 * Used in: MainMenu().                                                  *
1181 *                                                                       *
1182 *************************************************************************/
1183 void Load()
1184 {
1185  char loadinfo[30];
1186   _clearscreen(0);
1187  _printf("\nPlease type image name:");
1188  LoadImage(gets(loadinfo));
1189 }                                      /* End of Load() */
1190
```

```
1191 /*****************************************************************************
1192 *                                                                           *
1193 * Description: Setup pixel segmentation.                                    *
1194 *                                                                           *
1195 * Input: extval, extract                                                    *
1196 * Output: none                                                              *
1197 * Reference: none                                                           *
1198 * Used in: MainMenu().                                                      *
1199 *                                                                           *
1200 *****************************************************************************/
1201 void Extraction(int *extval, int *extract)
1202 {
1203  char c;
1204  int tmp;
1205
1206  *extract = 1;
1207  _clearscreen(0);
1208  printf("\nDetermine gray level to extract");
1209  printf("\n(a) black   0 - 31");
1210  printf("\n(b) gray1  32 - 63");
1211  printf("\n(c) gray2  64 - 95");
1212  printf("\n(d) gray3  96 - 129");
1213  printf("\n(e) gray4 130 - 159");
1214  printf("\n(f) gray5 160 - 191");
1215  printf("\n(g) gray6 192 - 223");
1216  printf("\n(h) white 224 - 255: ");
1217  c = getch();
1218  switch(c)
1219  {
1220   case 'a':
1221   case 'A':      tmp = 3;
1222                  *extval = tmp;
1223                  break;
1224   case 'b':
1225   case 'B':      tmp = 2;
1226                  *extval = tmp;
1227                  break;
1228   case 'c':
1229   case 'C':      tmp = 4;
1230                  *extval = tmp;
1231                  break;
1232   case 'd':
1233   case 'D':      tmp = 5;
1234                  *extval = tmp;
1235                  break;
1236   case 'e':
1237   case 'E':      *extval = 6;
1238                  break;
1239   case 'f':
1240   case 'F':      *extval = 7;
1241                  break;
1242   case 'g':
1243   case 'G':      *extval = 8;
1244                  break;
1245   case 'h':
1246   case 'H':      *extval = 0;
1247                  break;
1248   default:       break;
1249  }
1250 }                                          /* End of Extraction() */
1251
1252                                            /* EOF */
```

```
0001 initial.obj:    initial.c
0002         QCL /c initial.c
0003
0004 videorw.obj:    videorw.c
0005         QCL /c videorw.c
0006
0007 thesisqt.obj:           thesisqt.c
0008         QCL /c thesisqt.c
0009
0010 xform.obj:              xform.c
0011         QCL /c xform.c
0012
0013 thesisqt.exe: initial.obj videorw.obj thesisqt.obj xform.obj
0014         LINK initial+videorw+thesisqt+xform,thesisqt;
```

```
0001 /*****************************************************************************
0002
0003 ----
0004 * The purpose of this file is to give the user a description of
0005 * operation of the program thesisqt.exe.
0006
0007 * First to build the thesisqt.exe program on your PC perform the following:
0008         1 - copy these file into working directory
0009                 itexpfg.h
0010                 stdtyp.h
0011                 initial.c
0012                 videorw.c
0013                 xform.c
0014                 thesisqt.c
0015                 thmake
0016                 readme.now
0017         2 - at dos prompt type "make thmake" enter
0018                 output will be thesisqt.exe
0019
0020 * Second the program is currently setup to run on the PC-VISION systems,
0021 * thus both displays A and B are utilized.  To procede load the
0022 * original image into display A, next use display B as the working
0023 * display (these steps are done through ips program).
0024
0025 * Third the thesisqt.exe program can also work on the UDC
0026 * systems however this involves some
0027 * software modifications.  Using your editor of choice, in the
0028 * thesisqt.c program you will find the DisplayQuadrantValue() procedure
0029 * (line #913) inside this procedure on line #962 change
0030 *       Bwlineb() to Bwlinea()
0031 * and save changes. In the videorw.c program on line #9 change
0032 *       #define SYSPC to #define SYSUDC
0033 * and save changes. Next run
0034 * "make thmake" same as before, if all is successful you should now have
0035 * a new thesisqt.exe for the UDC systems.  ON THE UDC SYSTEM THE USER CAN
0036 * ONLY PERFORM QT COMPRESSION AND DECOMPRESSION, DO NOT ATTEMPT TO DO
0037 * TRANSFORMATION (Xformation).
0038
0039 * Have a happy journey!
0040
0041 >>So long for now!<<
0042 Jim
0043 4/12/91
0044 ----
0045
0046 *****************************************************************************/
```