Quad trees in image analysis.

A. Chandrasekhar

University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.
NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ MICROFILMÉE TELLE QUE NOUS L'AVONS RÉCEUE.
Quad Trees in Image Analysis

by

A. Chandrasekhar

A Thesis
presented to the University of Windsor
in partial fulfillment of the
requirements for the degree of
Master of Applied Science
in
Electrical Engineering

© University of Windsor
Windsor, Ontario, 1984
To My Parents
ABSTRACT

Representation of extracted regions is an important aspect in image analysis. A method of representing the extracted regions based on a hierarchical data structure known as Quad-trees is studied.

Several methods of constructing such tree representations are discussed and the most efficient method has been used for the images considered. A method of finding neighbour nodes in a tree structure is discussed. Further processing of such representations is possible directly, without any need for switching back to the array representation. Methods of performig operations such as connected component labelling, boundary following, template matching, etc., directly on the tree are discussed. The complexity of the algorithms used are also derived. The amount of compaction that could be achieved with quad trees is also studied for a few images. The various advantages and limitations are also discussed. The results of the images represented by such structural representations, and processed directly on the tree are also shown.
ACKNOWLEDGEMENTS

I wish to express my sincere thanks and gratitude to my supervisor, Dr. M. Shridhar, for his invaluable guidance, encouragement and support throughout the course of my study. Thanks are also due to Dr. G. Julien and Dr. J. Soltis for their valuable suggestions and help. My appreciations are extended to Mr. Raj Patil, Dr. Si Tawng, Mr. Bill Irwin and Mrs. Ann L. Snaec of the Computer Centre, University of Windsor, for their help in transferring image data to the central computer. The invaluable assistance rendered by Mr. K. Suryanarayana, Department of Biology, University of Windsor, in the preparation of this thesis is very much appreciated. Thanks are also due to Dr. N. Mohankrishnan, Mr. N. Rajendran, Mr. M. K. Swaminathan, Mr. V. L. Narasimhan, Mr. Uday Aladanqady and Mr. Vijay Venkatagachalam for their help at various stages of my study. I would like to thank my teachers, Dr. C. Gopalachar and Prof. K. Ramamurthy, of the National Institute of Engineering, Mysore, for inspiring me to take up graduate study. Special thanks are due to my friends, without whom my stay at Windsor would not have been memorable.

Finally, I extend my deep appreciation to my parents and family members for their affection, sacrifice and support.
# TABLE OF CONTENTS

**ABSTRACT** ........................ iv

**ACKNOWLEDGEMENTS** ........................ v

**TABLE OF CONTENTS** ........................ vi

**LIST OF FIGURES** ........................ ix

**LIST OF TABLES** ........................ xi

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>LEVELS OF IMAGE REPRESENTATION</td>
<td>2</td>
</tr>
<tr>
<td>REVIEW OF RELEVANT LITERATURE</td>
<td>4</td>
</tr>
<tr>
<td>PROPOSED STUDY</td>
<td>7</td>
</tr>
<tr>
<td>II. TREE STRUCTURES</td>
<td>8</td>
</tr>
<tr>
<td>QUAD TREES IN IMAGE PROCESSING</td>
<td>9</td>
</tr>
<tr>
<td>PROPERTIES OF QUAD TREES</td>
<td>9</td>
</tr>
<tr>
<td>SEGMENTATION</td>
<td>12</td>
</tr>
<tr>
<td>TYPES OF SEGMENTATION</td>
<td>13</td>
</tr>
<tr>
<td>Region Merging</td>
<td>15</td>
</tr>
<tr>
<td>Region Splitting</td>
<td>15</td>
</tr>
<tr>
<td>Region Merging and Splitting</td>
<td>15</td>
</tr>
<tr>
<td>REGION REPRESENTATION</td>
<td>16</td>
</tr>
<tr>
<td>Quad trees</td>
<td>16</td>
</tr>
<tr>
<td>Homogeneity Criterion</td>
<td>17</td>
</tr>
<tr>
<td>CUT-SET IN A TREE</td>
<td>20</td>
</tr>
<tr>
<td>Definition</td>
<td>20</td>
</tr>
<tr>
<td>CONSTRUCTION OF QUAD TREES</td>
<td>20</td>
</tr>
<tr>
<td>Split-Only method</td>
<td>22</td>
</tr>
<tr>
<td>Merge-Only Method</td>
<td>25</td>
</tr>
<tr>
<td>Split-and-Merge Method</td>
<td>23</td>
</tr>
<tr>
<td>Choice of the Initial Level</td>
<td>29</td>
</tr>
<tr>
<td>DATA COLLECTION</td>
<td>32</td>
</tr>
<tr>
<td>III. SEGMENTATION ON QUAD TREES</td>
<td>34</td>
</tr>
<tr>
<td>CONNECTED COMPONENT LABELLING</td>
<td>38</td>
</tr>
<tr>
<td>NEIGHBOUR FINDING IN A QUAD TREE</td>
<td>38</td>
</tr>
<tr>
<td>The Problem</td>
<td>39</td>
</tr>
<tr>
<td>The Solution</td>
<td>41</td>
</tr>
<tr>
<td>Heading</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>OPERATIONS ON TREES</td>
<td>108</td>
</tr>
<tr>
<td>Traversal</td>
<td>108</td>
</tr>
<tr>
<td>Merging</td>
<td>109</td>
</tr>
<tr>
<td>Splitting</td>
<td>109</td>
</tr>
<tr>
<td>SOME THEOREMS ON TREES</td>
<td>111</td>
</tr>
<tr>
<td>B. DATA TRANSFER INFORMATION</td>
<td>117</td>
</tr>
<tr>
<td>TRANSFER TO IBM FROM NOVA</td>
<td>118</td>
</tr>
<tr>
<td>TRANSFER TO NOVA FROM IBM</td>
<td>119</td>
</tr>
<tr>
<td>G. COMPUTATION OF MOMENTS</td>
<td>126</td>
</tr>
<tr>
<td>Illustration</td>
<td>128</td>
</tr>
<tr>
<td>Object 1</td>
<td>128</td>
</tr>
<tr>
<td>D. OTHER APPLICATIONS OF QUAD TREES</td>
<td>130</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>132</td>
</tr>
<tr>
<td>VITA AUCTORIS</td>
<td>135</td>
</tr>
<tr>
<td>Figure no.</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>Levels of image representation</td>
</tr>
<tr>
<td>2</td>
<td>Non-position invariance of quad-tree structures</td>
</tr>
<tr>
<td>3</td>
<td>Varying resolution of quad-tree structures</td>
</tr>
<tr>
<td>4</td>
<td>Results of object extraction</td>
</tr>
<tr>
<td>5</td>
<td>An example of a quad-tree</td>
</tr>
<tr>
<td>6</td>
<td>Cut-set in a quad-tree</td>
</tr>
<tr>
<td>7</td>
<td>Split-only method of construction</td>
</tr>
<tr>
<td>8</td>
<td>Sequence of construction by split-only method</td>
</tr>
<tr>
<td>9</td>
<td>Merge-only method of construction</td>
</tr>
<tr>
<td>10</td>
<td>Sequence of construction by merge-only method</td>
</tr>
<tr>
<td>11</td>
<td>Sequence of construction by split-and-merge method</td>
</tr>
<tr>
<td>12</td>
<td>Computational gain of split-and-merge method</td>
</tr>
<tr>
<td>13</td>
<td>Segmentation scheme</td>
</tr>
<tr>
<td>14</td>
<td>Problem of neighbours in a quad-tree</td>
</tr>
<tr>
<td>15</td>
<td>Connected component labelling with uni-directional label propagation</td>
</tr>
</tbody>
</table>
Connected component labelling with bi-directional label propagation

Connected component labelling with uni-directional label propagation

Connected component labelling with bi-directional label propagation

Sequence of boundary following

Boundary code description

Adjacent node configurations

Aligned and non-aligned nodes

System block diagram

Conventional representation of a quad-tree

Types of trees

A typical node

Operations on quad-trees

Labelled image with three components
# List of Tables

<table>
<thead>
<tr>
<th>Table no.</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Conditions for common ancestor</td>
<td>43</td>
</tr>
<tr>
<td>2</td>
<td>Relationship for finding the descending path</td>
<td>43</td>
</tr>
<tr>
<td>3</td>
<td>Space savings for quad-trees</td>
<td>68</td>
</tr>
<tr>
<td>4</td>
<td>Node distribution of strain gauge</td>
<td>81</td>
</tr>
<tr>
<td>5</td>
<td>Computational break-up of merge-only method</td>
<td>83</td>
</tr>
<tr>
<td>6</td>
<td>Computational break-up of split-only method</td>
<td>84</td>
</tr>
<tr>
<td>7</td>
<td>Computational break-up of split-and-merge method (l=5)</td>
<td>86</td>
</tr>
<tr>
<td>8</td>
<td>Computational break-up of split-and-merge method (l=6)</td>
<td>87</td>
</tr>
</tbody>
</table>
Chapter I

INTRODUCTION

An image is represented digitally as a two-dimensional array of integers representing the brightness function sampled at discrete points in space and quantised to discrete levels. Such a digitised image represents (approximately) a real world scene of objects. The sampling interval and the levels of quantisation are so chosen that a reasonably good representation of the scene results. Thus a considerable number of integers are needed to represent an image.

Image processing deals with the processing of such digitised images with the objective of enhancing certain features in them. Thus the output of such processing will normally be another image of the same size as that of the input image. Image analysis encompasses methods of extracting objects from the scene with a view of classifying or describing them. Thus the output of such a system will be a description of the input scene. There are a number of stages involved in image analysis, after each of which the image undergoes certain transformation. The transformed image is represented in various forms during different stages of analysis.
1.1 LEVELS OF IMAGE REPRESENTATION

The input image is represented as a two-dimensional array of integers at the first level of representation. This representation is known as the 'array representation', where the information is in the form of pixel values. This pixel-wise information can be discarded if the image can be segmented into subsets or regions which still retain the information about the objects in the scene. The representation of such a segmented image is known as 'region representation', which forms the second level of image representation (Fig. 1). This representation is usually compact and convenient to process, than the array representation. Geometrical properties like area, perimeter, etc., can be extracted from this representation which could be used to construct the third level of representation known as the 'relational representation'. Such a representation discards all the geometric details and retains the descriptive details. A region adjacency graph is an example of such a representation in which the nodes represent the objects in the scene, and the connecting links, their relationships. Each node in such a graph contains information about the object area, perimeter, centroid, etc. This relational structure is much more compact than the other two representations. Hence there is a progressive amount of compaction from level to level during processing.
Fig. 1. Levels of Representation.
1.2 REVIEW OF RELEVANT LITERATURE

One of the major steps in image analysis is the extraction of the objects in the scene from the background. Several methods of object extraction have been proposed which have their own limitations. Another problem that is commonly encountered is the representation of the extracted objects. Such representation has the twin requirements of compactness and ease in handling. Following is the brief review of relevant literature on the various methods of object extraction\(^1\) and their representation.

A widely applicable procedure for object extraction is thresholding. Thresholding attempts to cluster picture points which have similar pixel values independent of their positions. Though thresholding is, in principle an appropriate technique for object isolation, simple fixed thresholding is rarely adequate. Also, the effectiveness of such schemes depend upon the chosen values of thresholds. There are a number of techniques of choosing the threshold which are not universally effective. An elegant procedure of choosing a threshold by histogram analysis was proposed by Chow & Kaneko (1972). The histogram of images with different object and background gray levels shows two dominant peaks. By choosing thresholds at the bottom of the valley between the two peaks, thresholding was quite effective, but locating the valley bottom accurately was difficult.

\(^1\) also known as segmentation.
Rosenfeld & Davis (1977) proposed iterative histogram modification techniques to overcome this. However for images whose histogram exhibits a multi-modal behaviour, the single threshold method was not effective as some objects were bound to be submerged in the background. The technique of multi-level slicing by placing thresholds at all significant minima of the histogram was used by Prewitt & Mendelsohn (1966). A similar approach was used by Ohlander (1975) to process colour photographs with nine one-dimensional histograms. A Gaussian fitting technique for choosing thresholds was proposed by Rajendran and Sid-Ahmed (1983). Brenner et al. (1977) used geometric information to improve the effectiveness of object extraction.

An alternative approach to find uniform regions is to start with a partitioning of the image into a very large number of small uniform regions, and to grow them by merging similar regions. The first major attempt to region growing was the early work of Muerle & Allen (1968), who used a statistical test for merging neighbouring regions. Considerable improvement to this technique was suggested by Brice & Pennema (1970) who used a pair of heuristics which control the region merging. These heuristics evaluate parameters that depend on more than one region. A major problem with these region-growing methods is that they are computationally expensive for high resolution pictures. The other alternative was to start with the complete picture and
form regions by repeated sub-divisions. Robertson et al. (1973) used a mean vector of gray levels of multi-spectral image to perform region dividing. Klinger et al. (1974) proposed to use regular decomposition for image segmentation. These splitting techniques were found to be even more expensive than the region-merging techniques.

A natural compromise between the two methods, the one which starts with an intermediate partition, and can perform both region-merge and region-split was proposed by Horowitz & Pavlidis (1976). Regions were described in terms of an approximating function, and adjacent regions having similar approximations were merged while regions with large approximating errors were split. This method, known as the Split-and-Merge method is not only computationally more efficient, but also an aesthetically more appealing approach to region formation.

Once an interesting region in a picture has been defined, it can be represented and described in a variety of ways. Some representations preserve all of the information about the region, while others intentionally lose information, preserving only what is of interest for a particular task. Since the ultimate goal of image analysis is always the extraction of just the desired information, the main reason for considering any alternative representation is that it may simplify subsequent analysis.
1.3 PROPOSED STUDY

The present study is aimed at a detailed study of quad trees as a form of region representation. Different methods used in constructing quad trees are studied and their computational efficiency evaluated. With the view of studying an image analysis system with quad trees, the following operations are to be performed:

(i) derivation of a quad tree from an image array and vice-versa,
(ii) connected-component labelling directly on the quad tree,
(iv) boundary following on the quad tree and
(v) feature extraction from the quad tree
(vi) construction of a RAG
(vii) evaluation of the computational requirements of the algorithms used.
Chapter II

TREE STRUCTURES

Trees have been in existence since the third day of creation, and through the ages tree structures have been in common use. The concept of tree as a formally defined mathematical entity is said to have appeared first in the work of Kirchoff connected with his work in electrical network theory. The term 'tree' was first used by Arthur Cayley in 1857 when he studied these structures in connection with polymer chemistry. Since then trees have diffused into the various branches of science and have been recognised as a very important data structure in computer science. The concept of trees was first applied to images by Klinger (1976), and studied in detail by Hunter and Steiglitz (1976). Dyer, Rosenfeld and Samet (1980) have extended the concept to greater depths by developing efficient algorithms. This resulted in the growth of research on trees as applied to image processing by many researchers. These structures have lately been used for three-dimensional image processing by Meagher (1982) and Srihari (1983). Thus trees have rooted themselves firmly as an important entity in the field of image processing. For a detailed information on trees, refer to Appendix A.
2.1. **QUAD TREES IN IMAGE PROCESSING**

Quad trees have been found to be useful to represent the successive sub-division of images into its quadrants. This is studied in detail in the subsequent sections. However, some of the important features of quad trees are discussed here.

2.2. **PROPERTIES OF QUAD TREES**

Quad trees have some important properties that are worth mentioning. They are:

(i) Non-position invariance: The structure of the tree completely changes if the object in the image undergoes positional translation. This property is illustrated with an example in Fig. 2.

(ii) Variable resolution: The resolution varies from level to level. Each node in the tree corresponds to a certain image area; nodes at higher levels of the tree correspond to larger areas than the nodes at the lower levels (Fig. 3).

(iii) The higher levels of the quad tree represent approximations of objects while the lower levels represent their boundaries.

(iv) The maximum number of levels in a tree depends upon the image size.

(v) The maximum number of nodes at any level of the tree is given by $4^{(i-1)}$ for $i = 1, 2, \ldots, n$. 
Fig. 3. Varying resolution of Quad-tree structures
After having come to understand tree structures, segmentation methods are studied next before associating trees with segmentation.

2.3 SEGMENTATION

Image analysis almost always involves describing or classifying objects in a scene. A major step in this direction is to separate the objects of interest from the background before describing or classifying. This operation of separating the objects from the scene is known as 'segmentation'. The objects in the scene are described/classified only by operating on the segmented image rather than the input image. Thus the accuracy of the description/classification very much depends upon the accuracy of segmentation. Hence scene segmentation is one of the fundamental operations and one of the most important problems in image analysis.

This separation is essential for various reasons such as data compression, descriptive economy or as the first stage in understanding the image. Segmentation of an image is based on two basic assumptions:

(i) the area representing an object is more or less uniform in its local properties like brightness, colour, texture, etc., and

(ii) there is a detectable discontinuity in local properties between the objects, and between
the objects and the background.

These assumptions signify that the 'regions' are characterised by homogeneity and 'edges' are characterised by discontinuity in local properties, and they can advantageously be used to extract the objects from the scene. Segmentation schemes that use homogeneity of regions as a criterion result in the extraction of regions pertaining to the objects in the scene (region extraction), while schemes using the discontinuity result in extraction of their boundaries (edge detection). Fig. 4 represents the results of both these criteria on an image. However, it should be noted here that knowledge about the region stipulates the boundaries and vice-versa. Thus they are 'duals' of each other. This study is centered about the segmentation by region extraction.

2.4 TYPES OF SEGMENTATION

Regions can be extracted from the image in a number of ways, and can be classified into three categories as follows, viz

(i) Region Merging,

(ii) Region Splitting and

(iii) Region Merging and Splitting.
Fig. 4. Results of object extraction.
2.4.1 Region Merging

The input image is divided into a large number of small 'atomic' regions (possibly coinciding with single pixels). These small regions are merged to form larger regions based on a merging criterion. The 'phagocyte' algorithm of Brice & Pennema (1970) is typical of this kind. The criterion that is used to merge two adjacent regions is important as it affects the final segmentation.

2.4.2 Region Splitting

The input image is considered initially to be a single region and is successively broken down into smaller regions until all the regions are homogeneous, thus arriving at the final segmentation. As in the case of region merging, the final segmentation depends on the splitting criterion. The regular decomposition algorithm of Klinger (1976) is a good example of this type.

2.4.3 Region Merging and Splitting

In this method of region extraction, the above two methods are combined with certain advantages. Instead of starting at the atomic levels or at the single region level, an initial segmentation with a number of regions is assumed. The process of merging and splitting starts from this level. After all the regions satisfying the merging criterion are merged together, splitting of regions that do not satisfy
the criterion takes place. The final segmentation is arrived at after all possible splits and merges are complete. This method was first proposed by Horowitz & Pavlidis (1976), and is found to be more computationally efficient than the first two methods of region extraction.

Once the region extraction is complete, all the information pertaining to the objects of interest have been extracted discarding all unwanted information. This implies that there must be a reduction in the amount of information after segmentation.

2.5 Region Representation

The way of representing the extracted regions is known as 'region representation' (in contrast with the 'boundary representation' for extracted edges). A good representation should not only be compact, but also be easy to perform operations on it.

2.5.1 Quad Trees

A quad tree is a form of region representation based on the successive sub-division of the image array (Klinger, 1976). If the input array is not completely covered by a single region, it is divided into four equal quadrants along the principal axes and this process is repeated for each of the quadrants, sub-quadrants, etc., as long as necessary, until blocks (possibly single pixels) that are entirely contained
in the region or entirely disjoint from it are obtained. This process can be represented by a tree with nodes having an out-degree of four. Such a tree is known as a 'quaternary tree' or a 'quad tree' for short. The quad tree obtained this way is shown in Fig. 5. The root node of such a tree corresponds to the entire image and its four quadrants correspond to the four son nodes, and the leaf nodes in the tree correspond to the blocks for which no further sub-division is necessary. Thus the quad tree will have leaves that represent blocks which completely belong to a region or completely disjoint from it. In other words, the leaves will not represent blocks that have both the object and the background. The division of a block into its quadrants depend on the homogeneity with respect to a local property within the block, and the measure of this homogeneity is by a criterion called the homogeneity criterion.

2.5.2 Homogeneity Criterion
A block of pixels in an image is said to be homogeneous with respect to a certain local property, if its variation over that block is less than a specified tolerance. A criterion that measures the variation of a local property over a picture block, in effect measures the homogeneity of the block with respect to the same local property.
Fig. 5. An example of a Quad-tree
Since the most commonly used local property is the brightness level, we say that a block is homogeneous if and only if the brightness variation is less than the tolerance. A measure of variation of the brightness level over a block is the absolute difference between the maximum and the minimum values over that block. This is expressed as

\[
\text{IF} \quad |P_{\text{max}} - P_{\text{min}}| < \text{Epsilon}
\]

\[
\text{THEN} \quad \text{The Block is Homogeneous}
\]

\[
\text{ELSE} \quad \text{The Block is Non-homogeneous}
\]

where \(P_{\text{max}} (P_{\text{min}})\) is the maximum (minimum) pixel value over the domain of the node \(P\).

Thus the extremal pixel values have to be determined over each block before applying the above homogeneity criterion. The criterion based on the extremal values measure the variation much better than a criterion based on the average values. Hence such a criterion will result in a good separation of objects from the background. But such a criterion is more susceptible to noise. Since the splitting process depends on this criterion, the final tree obtained also depends on this criterion. The leaves in this final tree must necessarily represent regions that are homogeneous.
2.6 **CUT-SET IN A TREE**

The set of nodes representing the leaves in a tree represent all the regions in the input image. This set of nodes is known as the cut-set.⁲

### 2.6.1 Definition

The cut-set of the tree is defined (Pavlidis, 1976), as a subset of its nodes in which

(i) no two nodes belong to the same path from the root to the leaf, and

(ii) no more nodes can be added to it without it losing the first property.

The cut-set of a tree is shown in the Fig. 6. Alternatively, the cut-set is a set of nodes that are completely homogeneous. The cut of the tree represents the complete image in the tree. It is this set of nodes that is of interest in the tree.

2.7 **CONSTRUCTION OF QUAD TREES**

Quad trees can be constructed in three different ways corresponding to the types of region extraction, viz.

(i) Split-Only method,

(ii) Merge-Only method and

(iii) Split-and-Merge method.

² also referred to as the cut of the tree.
Fig. 6. Cut-set (---) in a Quad-tree
As the construction process needs the application of the homogeneity criterion, the extremal pixel values have to be evaluated at every stage of construction. The construction process is said to be complete only when all the leaves correspond to homogeneous blocks, and no four homogeneous children are left without being merged into their parent. Thus, the construction of the tree is aimed at reaching the cut-set.

2.7.1 **Split-Only method**

This method of construction (Fig. 7) stems from the basic definition of the quad tree as 'a successive subdivision of the image array into its quadrants'. The image array is divided into its quadrants if it is found non-homogeneous. The extremal pixel values over each of these quadrants are evaluated for the application of the homogeneity criterion. This process is repeated on each of the quadrants, sub-quadrants, etc., until all the blocks are homogeneous, after which the final quad tree is reached. The evaluation of the extremal pixel values over the domain of each block after every division adds on to the computational burden in constructing the tree. This method of constructing the tree is known as the *Split-Only method* since only splitting is carried throughout the construction process.
Fig. 7. Split-only method of construction
Fig. 8. Sequence of construction by split-only method.
Sometimes it is also referred as 'Top-Down' construction signifying the order in which the tree is constructed. The sequence of construction by this method is represented in Fig. 8.

2.7.2 **Merge-Only Method**

This method (Fig. 9) is the inverse of the Split-Only method. Instead of starting at the whole image, the construction starts at the pixel level. Starting from this lowest level, the next higher level of the tree is constructed and are linked to the children at the lower level. The domain of these nodes at the higher level is the union of the domains of its children nodes. The entire tree is built upwards, level by level, until the root is reached. This type of tree construction is known as the 'Bottom-Up' construction. A point to be noted is that the 'children were born before their parent!'

This method of construction will not lead us to the final tree because any four adjacent blocks satisfying the homogeneity criterion would exist separately. Thus a merging process is needed which will merge all four adjacent blocks that satisfy the homogeneity criterion, to their parent and remove them from the tree structure. However, this process of merging can be combined with that of growing. After every new node is created, the homogeneity criterion is applied to the node.
Fig. 9. Merge-only method of construction.
If the new node is found to be homogeneous, then the descendents are immediately removed from the tree structure. Thus the tree arrived at by this method will be the final quad tree. This method of constructing the quad tree is known as the Merge-only method.

The evaluation of the extremal pixel values of blocks at higher levels of the tree is done in a much simpler fashion by computing the maximum of the maxima and the minimum of the minima. Thus this method needs less amount of computation as compared to the Split-Only method. The sequence of construction is shown in Fig. 10.

2.7.3 Split-and-Merge Method

This method of construction is based on the method of segmentation proposed by Horovitz and Pavlidis, and is a combination of the two methods described earlier. In this method the construction starts at some intermediate level between the root and the leaves. The nodes at this intermediate level are created and the extremal values over the domain of each of the nodes are evaluated. The tree build-up starts at this intermediate level and proceeds up to the root removing the nodes that could be merged in the process. Once this merging process comes to an end, the splitting process takes over. The nodes at the intermediate level that survived the merging process and that are non-homogeneous are split until they become homogeneous. Thus
the quad tree is constructed by both splitting and merging operations. It should be ensured that no nodes that are merged will be split. The fact that merging removes the nodes from the tree structure guarantees that the nodes at the intermediate level that survived the merging process cannot be merged, but possibly be split. Fig. 11 shows the sequence in construction of the tree for this method.

This method of tree construction is computationally more economical than the previous methods, but the choice of the intermediate level very much affects the computational economy. Hence this level must be carefully chosen.

2.7.4 Choice of the Initial Level

The computational efficiency of the split and merge method depends upon the initial level of the tree, and hence its choice is very important.

The goal of all the construction methods is to arrive at the cut of the tree. Since the cut of the tree will usually be spread over a number of levels in the tree, the computational effort needed to reach the cut varies. However, if one starts at an initial level at which lie the maximum number of nodes in the cut, then minimal effort would be spent in arriving at the cut of the tree. Fig. 12 illustrates the computational effort required for the different methods of construction.
Fig. 11. Sequence of construction by split-and-merge method.
Fig. 12. Computational gain of split-and-merge method (after Horowitz and Pavlidis, 1976)
The level at which the maximum number of nodes exist in the
cut can be determined only after studying the node
distribution of the cut. Hence a priori information about
the image class should be used in choosing the initial
level. Merge-only and Split-only are special cases of split-
and-merge method. Merge-only results when the initial level
is at the pixel level, and split-only when the initial level
is at the root level.

2.8 DATA COLLECTION
Images needed for testing the algorithm were obtained
through a Hamamatsu Vidicon Camera, and were digitised to
128 x 128 with 256 levels of gray. These digitised images
were stored in the Data General NOVA 840 minicomputer. As
PASCAL compiler was not available on the NOVA computer,
images were processed on the IBM computer. Therefore the
digital images had to be transferred to the IBM. No direct
link of data transfer between the two computers were
available, and hence a 'protocol' of data transfer had to be
established. Also, the processed images had to be brought
back to the NOVA 840 computer to be displayed on the video
monitor. Special routines were developed to transfer data
between the two machines using magnetic tapes. Many images
of manufactured parts, human faces and bio-medical images
were collected for testing the algorithm. Several images
were transferred to the IBM and an image data base has been
created for future use. Information on data transfer is provided in Appendix B.
Chapter III
SEGMENTATION ON QUAD TREES

In the conventional segmentation schemes, the output will be an image of the same size as that of the original image which represents the extracted information. Even though the process of segmentation has discarded unwanted information from the input image, the output segmented image is of the same size as that of the input image. Thus the array representation of the extracted information may not be a compact representation. Quad trees can be used to represent the extracted information in a compact manner. Thus a segmentation scheme that has the input in the array form and the output in the tree form is desired. In such a scheme, the input image is first converted to a tree and the operation of segmentation is performed directly on the tree. The block diagram of such a scheme is shown in Fig. 13.

Since PASCAL was found to be more suitable for tree structures with its advanced data structures and dynamic allocation capability, it was used in implementing the above scheme.
Fig. 13. Segmentation Schemes
(a) Conventional
(b) Desired
(c) Proposed
The nodes in the tree were stored as a record which has a number of fields that contain the information stored in them. As we need to move down the structure, pointers to the descendent nodes were maintained. In order to achieve mobility in both ways, pointers to the children nodes and the parent node were stored, making the tree a doubly linked structure. For the terminal nodes, pointers to the children are set to NIL signifying that there are no children nodes further down the hierarchy. Similarly, the pointer to the parent of the root node is also set to NIL. The nodes also contain information fields for storing the maximum pixel value and the minimum pixel value over the domain of the node. In addition, information about the level of the node and the region number is also stored in them. The nodes were declared as a packed record achieving compaction.

Split-and-merge method was used to construct the quad trees for the various images studied. The nodes of the intermediate level of the tree were created and initialised with the maximum and minimum pixel values. These nodes have to be temporarily stored in an array till their parents are created, upon which they can be linked to them. The size of this array depends upon the intermediate level chosen. This array is repeatedly used as higher and higher levels of the tree is being constructed. In addition, smaller portion of this array is used to construct higher levels of the tree.
As pointed out earlier, four nodes that satisfy the homogeneity criterion must be merged into their parent, and the individual nodes removed from the tree structure. In order to remove these nodes, an additional tree-traversal was needed after completion of construction. This clean-up operation is done while the tree is being constructed, thus saving one tree traversal which otherwise would have become necessary. Thus the Split-and-merge method of tree construction along with the simultaneous removal of merged nodes results in a very efficient tree construction. In addition, merged nodes are immediately de-allocated, releasing the memory space occupied by the merged nodes for subsequent use. This, in fact, prevents a very high dynamic memory requirement. A quad-tree that is represented by the minimum number of nodes results after the splitting is complete.

Once the optimal quad tree is constructed, there is no special operation that is needed to perform segmentation. The cut of the tree corresponds to the homogeneous blocks of the image which were obtained by separating the regions. Therefore, such a quad-tree represents the complete segmented image. In fact, the segmentation of the image array corresponds to the cut of the tree (Pavlidis, 1976). Thus segmentation is performed while constructing the tree. The final segmentation is influenced by the homogeneity criterion used for tree construction.
3.1 CONNECTED COMPONENT LABELLING

The connected-component labelling is done immediately after segmentation and it can be said that object separation is incomplete without labelling. The connected-component labelling can be considered as an operator that transforms a segmented image into a labelled image. This operation can be performed directly on the quad tree without the need for switching back to the array representation (Samet, 1980).

Exploring the neighbourhood for establishing the connectedness is the preliminary step in the connected-component labelling. Since the segmented image is in the form of a quad tree, determination of neighbouring regions in a quad tree is not straightforward as in case of an array representation due to the fact that the adjacent regions are scattered far apart in the tree structure. Thus the problem of finding neighbouring blocks in a quad tree had to be solved first before processing the quad tree.

3.2 NEIGHBOUR FINDING IN A QUAD TREE

An extensive survey of literature did not reveal a solution to the neighbour-finding problem. Hence a method of finding neighbours of a node in a tree structure was developed and is described in detail in the following sections.
3.2.1 The Problem

The neighbour-finding problem can best be illustrated by an example. Consider the image array (Fig. 14a) and its corresponding tree structure (Fig. 14b). The neighbours of node # 5 in the tree will be studied. The western neighbour (node # 4) and the southern neighbour (node # 10) both have the same parent and can be seen to lie in adjacent branches. But the northern neighbour (node # 2) and the southern neighbour (node # 16) do not have the same parent as the other neighbours and they lie in different sub-trees. This shows that there is no definite pattern with which the neighbours can be found directly in the tree. Since a pixel block in the image can have neighbouring blocks that are larger or smaller in size, nodes in a tree can have neighbours that exist at different levels of the tree. Nodes # 5 and # 16 are typical examples.
Fig. 14. Problem of neighbours in a Quad-tree
However it can be observed that two neighbouring nodes in the tree have a common ancestor. This common ancestor can lie at any level depending upon the relative position of the nodes and the direction of their neighbourhood. The neighbour can be reached in the tree structure only after reaching the common ancestor, thereafter descending down the tree till one arrives at the required neighbour. Thus the neighbour-finding problem in a tree structure can be split into two simpler problems, namely

(i) finding a common ancestor and

(ii) finding the descending path.

3.2.2 The Solution

By a careful study of the tree structure, it was found that the common ancestor does have a relationship in the subtree having the reference node that is unique for a given direction. These relationships are tabulated in table 1. and could be used to determine the common ancestor as one keeps ascending the tree. As the tree is being ascended, these conditions are checked at every level to determine if it is a common ancestor. If the conditions are met, then that node is a common ancestor for the neighbour in the given direction from which the neighbour can be reached. Thus the ascending phase comes to an end after reaching the common
ancestor. If the common ancestor is not reached even after reaching the root, then the node in question lies on the image frame and does not have a neighbour in that direction.

It was also observed that depending upon the direction of the neighbour there exists a definite node-relationship between the nodes in the ascending path to that of the descending path for a given direction (Table 2), which can be used in the determination of the descending path from the common ancestor. The node-relationship encountered while ascending the tree is stored on a stack (LIFO), which is used to find the descending path. After reaching the common ancestor, the descending path is chosen based on the relationship stored on stack. The descending relationships can be obtained from table-2. As a node can have neighbours at any level of the tree, care should be exercised to check the terminality of each node while descending. In case of a neighbour at lower levels, the tree has to be descended further and we do not have any corresponding relationship while ascending and hence we follow the convention that we will be reaching the node that is clockwise-first in the given direction. Thus we will reach the western-most of the northern neighbour when we refer to the neighbour in the northern direction. Thus the problem of finding neighbours in a tree is solved by a method of ascending and descending in the tree.
Table - 1.

Conditions for common ancestor

<table>
<thead>
<tr>
<th>Direction</th>
<th>Required relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>North</td>
<td>Third/Fourth</td>
</tr>
<tr>
<td>East</td>
<td>First/Third</td>
</tr>
<tr>
<td>South</td>
<td>First/Second</td>
</tr>
<tr>
<td>West</td>
<td>Second/Fourth</td>
</tr>
</tbody>
</table>

Table - 2.

Relationship for finding the Descending path

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Son1</th>
<th>Son2</th>
<th>Son3</th>
<th>Son4</th>
</tr>
</thead>
<tbody>
<tr>
<td>North/South</td>
<td>Son3</td>
<td>Son4</td>
<td>Son1</td>
<td>Son2</td>
</tr>
<tr>
<td>East/West</td>
<td>Son2</td>
<td>Son1</td>
<td>Son4</td>
<td>Son3</td>
</tr>
</tbody>
</table>
3.3 LABELLING SCHEME

The above neighbour-finding technique is made use of in this connected-component algorithm which is an equivalent tree-version of an ordinary connected-component algorithm. This algorithm, a variant of the algorithm proposed by Samet, is a two-phase algorithm. The first phase is known as the 'labelling phase', and the second phase, the 'relabelling phase'.

The tree is traversed starting from the root using the post-order tree-traversal searching for nodes that represent the 'regions' (hereafter referred as black nodes). Once a black node is reached, its neighbourhood needs to be explored to find all the connected regions. Since we are interested in four-connected neighbours, exploration in all four directions are needed; however exploration in the western and the northern directions are redundant as we have visited all nodes in the northern and western directions before arriving at the node in question. Hence it can be concluded that for the first black node encountered, there are no nodes that are black in the northern or the western directions. Thus exploration in only the eastern and southern directions are sufficient to explore the complete neighbourhood. Thus for every black node encountered in the tree two neighbour explorations are made.
3.3.1. **Labelling Phase**

The tree is traversed from the root in the post-order fashion. When a black node is encountered, its eastern and southern neighbours are explored. If these neighbours are also black, then they all form a connected group of nodes which should bear the same label. If any one of the neighbour nodes already have a label assigned to it, then that label will be assigned to the other unlabelled nodes in the group. Thus a single label gets propagated to the other nodes without generating a fresh label. It should be noted that the label gets propagated in both directions in this algorithm in contrast to the unidirectional label propagation in Samet's algorithm. This feature, in fact, results in less number of labels being generated during the labelling phase resulting in reduced computation during relabelling. Fig. 15 and Fig. 17 both show the labelling sequence of an image with unidirectional label propagation, and Fig. 16 and Fig. 18 show the corresponding labelling sequence with bidirectional label propagation. In fact, it can be seen that the number of labels generated during initial labelling is reduced.
Fig. 15. Connected Component Labelling with unidirectional label propagation.
Fig. 16. Connected Component Labelling with Bi-directional label propagation.
Fig. 17. Connected Component Labelling with uni-directional label propagation.
Fig. 18. Connected Component Labelling with bi-directional label propagation.
If however, two connected regions bear different labels, then these labels must be declared equivalent signifying they both represent the same region. These equivalent labels are settled during the relabelling phase. Thus at the end of the tree-traversal, all black nodes would have been assigned a label and all labels that must be equivalenced are entered into the table of equivalences.

3.3.2 Relabelling Phase

This phase is the important phase which settles all the equivalences among labels and assigns labels unique for a component. When a black node is encountered while traversing the tree, the equivalence class to which the node-label belongs is obtained from the table and that class-number is assigned as a fresh label to the node. If the label is not found in the table, then there are no equivalents for that label, but it is entered as a separate class in the equivalence table and the class number being assigned as the new label. Thus at the end of this traversal, all the components will have an unique label.

3.4 Feature Extraction

The objects in a scene are characterised by certain properties like area, perimeter, centroid, horizontal and vertical extent, etc. These are known as the features of the object. In the analysis of an image, these features play
an important role. Extraction of these features is done after the labelling process. This operation can also be performed directly on the quad tree.

3.4.1 Computation of area

The area of an object in the image can be computed by summing the areas of the blocks representing the object. Since each of the objects are identified by a unique label, the area of the nodes in the tree having the label of the object are computed. The sum total of the area of these individual nodes is the area of the object. Thus the area of all the objects can be computed in one tree-traversal.

3.4.2 Computation of perimeter

The perimeter of an object can be computed by searching for the nodes on the edges of objects and adding the length of these nodes that contribute to the periphery of the object. This needs the exploration of two adjacencies for every terminal node in the tree. The perimeter can also be computed during boundary following operation. The sum total of the length of the boundary codes gives the perimeter of an object.
3.4.3 Computation of the centroid

The position of objects in an image and its extent in the various directions can be obtained by computing moments. The zeroth-order moment and the first-order moments in both the $X$ and $Y$ directions are used to compute the centroid of objects. For a detailed discussion on computation of moments, refer to Appendix C.

Thus the various features of the object can be extracted from the segmented image. All these features can be extracted in a single pass of the tree. However, the computation of the perimeter is done during boundary following because of certain computational advantages.

3.5 Construction of RAG

When the objects in a scene and their features have been extracted, it becomes possible to construct a graphical description of the scene. Such descriptions include representation of the objects, their properties and the relations amongst them. The commonly used description is a Region Adjacency Graph or RAG for short. Graphs, in general, have a number of nodes or vertices that are connected by edges. These structures can be cyclic i.e., there can exist closed paths in them unlike in a tree structure. The nodes in the RAG correspond to the objects in the scene and they contain information about the object, like area, perimeter, centroid, etc. Also, the nodes do have fields of information
about the other connected objects. Thus the relational representation gives a good description of the scene and is more compact than the original scene. Thus analysing a scene amounts to analysing the RAG.

3.6 **BOUNDARY FOLLOWING**

Quad trees are basically a form of region representation and hence contain information about the regions of the objects in the scene. However the need to know about the boundaries of certain objects in a scene often arises. Boundary information can be obtained from a quad tree by a method of edge-tracking inside the tree. The edge of an object is followed from an initial starting point in a fixed direction till the same point is reached. A four-direction boundary code is output during the tracking process that describes the boundaries. These codes are stored in a file that can be used for further processing.

The boundary following algorithm is logically similar to any conventional algorithm except that it is executed directly on the tree. This boundary following algorithm is a single pass algorithm which performs two functions: scanning and tracking. The aim of scanning is to search the tree in an orderly manner for an edge between a labelled component and the background. Such an edge is characterised by two adjacent nodes that are differently coloured (black

---

3 The terms edge-tracking and boundary-following are used interchangeably.
and white). Hence the first occurrence of such nodes during the tree-traversal constitute an edge-point. Since these two adjacent nodes can have four possible positions relative to each other, the first occurrence of a black node which has a white node as a northern neighbour is taken as the correct edge-point. During scanning, the tree is traversed from the root in a specified order such that no node is missed during the traversal and no nodes are visited twice. Post-order tree-traversal is used in this algorithm for scanning. The northern neighbour of every black node in the tree is checked during the traversal. The tree-traversal proceeds from the root till an edge point is encountered, whereupon the traversal is suspended. Thus the process of scanning brings one directly to an edge point in the tree. The edge-point, by definition lies on the boundary of the object and hence is a suitable point to start the tracking process. The aim of the tracking process is to follow the boundary of the component.

An edge is always enclosed by a pair of nodes coloured differently. Thus to track an edge, nodes on either side of the edge have to be found all along the edge. Hence the process of tracking should continuously determine the node pairs that next enclose the edge. Tracking proceeds in the clockwise direction all along the edge, outputting the correct code depending upon the relationship between the nodes on either side of the edge, till the starting point is
reached. The tracking process is terminated at this point at which the boundary of an object is completely traced.

As many components can exist in an image, there exist many edge-points. Hence the label of the component is used to qualify the component edge. The tree-traversal is resumed from the point where it was suspended, scanning for an edge of a different component. The process of scanning and tracking proceed alternately so that the boundary information is obtained component after component. This sequence is shown in the Fig. 19. In order to prevent detection of an edge already tracked, a trace is left along all the tracked edges. After all the components have been tracked, the scanning process continues to ensure that no more components exist in the tree. Upon completion of the scanning process, the boundary following algorithm is terminated.

3.6.1 Code description

The boundary of any component can be described by horizontal and vertical line segments that can be represented by a generalised four-direction code (Freeman, 1974). As the edge is always enclosed between two nodes, the boundary segments depend upon the relative positions of these nodes. Corresponding to each of the four possible combinations of relationship between the two nodes, there are four unique codes that describe their relationships (Fig. 20). This code is used to describe the boundary of the various objects.
Fig. 19. Sequence of boundary following
(-- ---) scanning
(★) edge-point
Fig. 20. Boundary Code Description.

Fig. 21. Adjacent Node Configurations.
(a) large black node
(b) small black node
(c) equal size nodes
As neighbouring nodes can be of different sizes, nodes that enclose an edge need not be of the same size. This leads to three different cases of node pairs enclosing an edge between them. They are

(i) Black node being larger (Fig. 21a),
(ii) Black node being smaller (Fig. 21b) and
(iii) Black node being equal (Fig. 21c).

When nodes of unequal size enclose an edge, then the code describing the edge between them is only over a length equal to the minimum of the two nodes.

3.6.2 Types of node junctions

When two nodes enclosing an edge between them, extend up to a four-node junction as shown in Fig. 22a, then they are said to be 'aligned'. Fig. 22e shows the case of node pairs that are 'non-aligned'.

3.6.3 Determination of next node pair

In order to track an edge continuously in a quad tree, all the node pairs that enclose an edge between them have to be found in a sequence so that the tracking will proceed without any discontinuity. Hence from an arbitrary node pair, one must be able to find the next node pair that encloses the edge. The method of determination of the next node pairs differs in the aligned and non-aligned case.
Fig. 22. Aligned (a - d) and non-aligned (e - g) nodes.
3.6.4 The aligned case

Since the nodes are aligned, an edge will run into a four-node junction. There are only three possible directions that an edge can take from such a four-node junction: turn right, turn left, or proceed straight. These possible directions are shown in dotted lines (Figs. 22a). The direction taken depends on the colour of the nodes forming the junction, hence all the four nodes forming the junction have to be examined. Since two of them already constitute the current node-pair, only the other two nodes have to be determined. Hence two nodes have to be found in the tree for every determination of the next node-pair. The next node-pair is so chosen that all connected black coloured regions are enclosed i.e. they must lie to the right relative to the direction of tracking. The colour of the two nodes are examined starting from left to right. The three possible combinations are shown in Figs. 22b thru 22d. along with the correct next node-pairs. For this case, the tracking tip is shown at corner # 2, however, the principle of choosing the next node-pair is the same for any other corner.

3.6.5 The non-aligned case

In the case of non-aligned nodes, the nodes form a three node junction as shown in Fig. 22c. This implies that there are two possible directions that an edge can take, and it
depends upon the colour of the nodes forming the junction. As two of the nodes already constitute the current node pair, only one node in the tree has to be found. The colour of this node determines the next node-pair and consequently determines the direction of tracking. As in the previous case, the tracking must proceed in such a way as to enclose all the connected black nodes. The two possible combinations of the next node pair are shown in Fig. 22f thru 22g. alongwith their next node-pairs.

3.6.6 The scheme

The tree is traversed looking for the boundary edges of the components in the image. If the boundary information of all the objects in the scene are needed, then the scanning and the tracking must proceed alternately until all the boundaries have been tracked. If the boundary information of a particular object is needed, then the tree is scanned looking for an edge qualified with the label of the component, which when found, is tracked completely.

Object boundaries can be classified into two categories as internal boundaries, and external boundaries. The boundary of an object that pertains to its periphery is called external boundary. If an object has a 'hole', then the boundary of such an internal hole is called internal boundary.
Internal boundary following is similar to that of the external boundary tracking except that the tracking proceeds in such a way as not to enclose any connected black coloured region i.e., the black coloured must should lie to the left of the direction of tracking.

When a boundary of an object hits the image frame, then the determination of the next-node pair is different from the normal case. The nodes are determined along the boundary so that the tracking proceeds along the frame boundary in the clockwise direction till the boundary emerges out of the frame.

3.7 Template Matching

Template matching is an operation of finding some known structural feature in an image that resembles the one in the reference template. Example of such templates being a point, a vertical line, a horizontal line, or a combination of these.

An object in a tree is represented by a group of nodes each of which correspond to a square block of the image. Hence edges in such a representation exist in horizontal and vertical directions only. An edge always exist between two differently coloured nodes in the tree structure. A horizontal/vertical edge exists between two differently coloured vertically/horizontally adjacent nodes. But nodes can be coloured only after performing segmentation. As only
information about the edges are needed, edges can be detected without segmenting the image.

The tree is traversed looking for an edge by applying the merging criterion (discussed in the earlier sections) to the two adjacent terminal nodes. If the criterion fails, then there exists an edge between the two nodes. The complete traversal of the tree detects all the edges in the given direction.

Any other template can similarly be used in the above procedure. It should be noted that the template must be specified as a group of nodes, and the matching in the tree at every terminal node must proceed according to the spatial configuration of the nodes in the template.
Chapter IV
RESULTS AND DISCUSSION

Various methods of tree construction were studied, and the best method of tree construction was used to derive quad trees for the images considered. The merged-node deletion process was combined with the construction process resulting in computational efficiency. A neighbour-finding method was developed for quad trees which was used in the processing stages. Connected component labelling, extraction of the features and boundary following were done on the quad tree using the same neighbour-finding procedures. The overall system block diagram using all the tree processing schemes described, is shown in Fig. 23.

The algorithms were tested on various images and the amount of compaction achieved, and the processing overheads were studied. The results of the various processing stages are analysed in the following sections.
Fig. 23. System Block Diagram

- Connected Component Labeling
- RAG Construction
- Segmentation
- Boundary Following
- TFE Construction
- Area, Perimeter, Centroid Computation
4.1 **COMPACTNESS OF QUAD TREES**

As one of the requirements of a good region representation is compactness, a number of images were considered for studying the compactness achieved. The comparison is made relative to the space requirement by an equivalent array representation. Before proceeding further with the comparison, the space needed for a tree has to be calculated.

The space needed to represent a tree in memory is proportional to the number of nodes in the tree, the constant of proportionality being equal to the memory space needed for each node in the tree. The memory space needed for each node can be estimated by considering all the information stored in the node. As information about the maximum pixel value, minimum pixel value, region number, x-coordinate, y-coordinate, and size are stored in the nodes, five integers are needed for these fields of information. In addition, links to the four sons and the parent are stored in each node, which needs five pointers. Assuming the pointer needs two bytes, and an integer one byte of memory space, a total of fifteen bytes are needed for each node. Also, the total memory space needed for a tree depends upon the number of nodes in it. Lesser the number of nodes in the tree, higher is the compaction achieved. If the image has more similar regions, clustered together, then they can be represented by a lesser number of nodes. Such regions are
called as 'homogeneous regions', and the compaction achieved depends upon the homogeneity of the image. The space needed for the tree representation and the array representation are presented in table - 3. It can be seen from the table that the strain gauge can be represented compactly, because of its large homogeneous regions. Thus the quad trees have the feature of compactness unlike an array representation which needs a fixed amount of space for a given image. The amount of compaction achieved depends on the homogeneity of the image. Thus quad trees do have this limitation for being a compact representation.
Table 3: Space savings for Quad-trees

<table>
<thead>
<tr>
<th>Image</th>
<th>Size</th>
<th>ARRAY</th>
<th>TREE</th>
<th>Percent Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Representation</td>
<td>Overheads</td>
<td>Total</td>
</tr>
<tr>
<td>Three Chips</td>
<td>128</td>
<td>16,384</td>
<td>24,576</td>
<td>40,960</td>
</tr>
<tr>
<td>Piston-Head</td>
<td>128</td>
<td>16,384</td>
<td>24,576</td>
<td>40,960</td>
</tr>
<tr>
<td>Strain-Gauge</td>
<td>128</td>
<td>16,384</td>
<td>24,576</td>
<td>40,960</td>
</tr>
</tbody>
</table>
4.2 **PROCESSING OVERHEADS**

In processing any image by any method, some additional arrays are needed. These are called the processing overheads, and these vary with the method of processing. The processing overheads associated with this method are as follows:

(i) An array of pointers for construction (32 x 32)

(ii) A table for equivalence processing (30 x 30)

(iii) Ten arrays for extracting features (1 x 10)

(iv) One array for the neighbour-finding (1 x 10)

These are in addition to the memory space needed for the original image and for the final tree. The memory needed to represent the final tree depends upon the number of nodes in the tree.

4.3 **GAIN IN PROCESSING OVERHEADS**

The processing overheads associated in processing an image represented by a tree is much less when compared to that of the array. Hence to compare the two, we consider both the space needed for representation and the associated overheads for each method. Even though the quad tree representation is not very compact, the processing overheads associated are much less. Thus a comparison shows that processing by the
tree needs less memory on the whole than the array method.

The final results are presented in table - 3.

4.4 EFFECTIVENESS OF SEGMENTATION

4.4.1 Predicates

The effectiveness of this object extraction algorithm depends not only on the criterion for tree construction, but also on the predicate chosen for the labelling phase. These are propositions which are evaluated for each terminal node to determine if they can be classified as a black node, and if they can bear the same label as that of the neighbour nodes. The predicates used can be classified into two categories: simple difference predicates and relative difference predicates. The simple difference predicate (P1) use the absolute simple difference between the gray levels of the neighbouring nodes, while the relative difference predicate (P2) use the absolute difference of gray levels relative to the gray level of the reference node in question. The two predicates that were used are as follows:

\[ P1: \text{IF } |P(\text{avg}) - Q(\text{avg})| < \Omega \text{ THEN assigns the same label to the neighbouring nodes } P \text{ and } Q. \]

\[ P2: \frac{|P(\text{avg}) - Q(\text{avg})|}{\max[P(\text{avg}), Q(\text{avg})]} < \Omega \]
THEN assign the same label to the neighbouring nodes P and Q.

where \( P(\text{avg}) = \frac{P(\text{max}) + P(\text{min})}{2} \)

and \( Q(\text{avg}) = \frac{Q(\text{max}) + Q(\text{min})}{2} \)

The results of the segmentation of the various images with these predicates are shown along with the original images in the following photographs.
Original image of three chips

Segmented image of three chips with relative predicate
Segmented image of three chips with simple predicate

Boundaries of three chips
Original image of the strain-gauge

Segmented image of the strain-gauge with relative predicate
Segmented image of strain-gauge with simple predicate

Boundaries of strain-gauge
Original image of the piston-head

Segmented image of the piston-head with relative predicate
Segmented image of piston-head with simple predicate

Boundaries of piston-head
Original image of the transmission gear

Segmented image of the transmission gear with relative predicate
Segmented image of the transmission gear with simple predicate

Boundaries of transmission gear
Chapter V

COMPUTATIONAL ANALYSIS

The computations that are involved in the various stages of tree construction and tree processing are estimated in this chapter.

5.1 COST ANALYSIS OF CONSTRUCTION METHODS

During the earlier discussion of the methods of tree construction, it was theoretically established that the split-and-merge method is an efficient method. The following analysis is aimed at supplementing this with the image of the strain gauge as an example. The node distribution of the final quad tree for the strain gauge is shown in table 4.

The cost of construction of the tree is directly proportional to the number of computations involved in extremal value evaluation and homogeneity criterion evaluation. The computations involved in evaluating the homogeneity of a block is much less compared to the extremal value evaluation. These are calculated individually for the various methods of construction.
Table - 4

Node distribution of strain gauge

<table>
<thead>
<tr>
<th>Level</th>
<th>Maximum</th>
<th>Existing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>256</td>
<td>64</td>
</tr>
<tr>
<td>6</td>
<td>1024</td>
<td>108</td>
</tr>
<tr>
<td>7</td>
<td>4096</td>
<td>80</td>
</tr>
<tr>
<td>8</td>
<td>16384</td>
<td>60</td>
</tr>
</tbody>
</table>

5.1.1 Case 1: Merge-only method

The construction scheme starts at the pixel value and moves up towards the root. Thus the number of comparisons for the evaluation of the extremal value decreases with decrease in level. As \((n-1)\) comparisons are essential to find the
maximum/minimum value of n elements, we need three comparisons to find one extremal value for every merge involving four elements. As two-extremal values are computed for every merge, six comparisons are needed. The total number of comparisons for the extremal value evaluation is shown in table - 5. The number of times the homogeneity criterion is applied depends upon the number of nodes. Hence the total number of comparisons needed for the application of homogeneity criterion is shown in the last column of table - 5. The total number of comparisons needed for the tree construction by this method is 38,227.

5.1.2 Case 2: Split-only method

When a node is split into its children, the extremal values have to be evaluated for each of the children nodes created over their respective domain. Thus the number of comparisons needed for extremal value evaluation depends upon the number of elements in the domain of the node; a quantity that varies with the level of the node. The number of comparisons needed for extremal value evaluation is shown level-wise in column 4 (table - 6). The total number of comparisons for this case is 130,759.
Table 5.

Computational break-up of merge-only method

<table>
<thead>
<tr>
<th>Level</th>
<th>Nodes</th>
<th># of Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Maximum</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>256</td>
<td>64</td>
</tr>
<tr>
<td>6</td>
<td>1,024</td>
<td>108</td>
</tr>
<tr>
<td>7</td>
<td>4,096</td>
<td>80</td>
</tr>
<tr>
<td>8</td>
<td>16,384</td>
<td>60</td>
</tr>
</tbody>
</table>

Total: 32,766 5,461

Grand Total: 38,227
Table - 6

- Computational break-up of split-only method

<table>
<thead>
<tr>
<th>Level</th>
<th>Nodes</th>
<th># of Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Maximum</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>256</td>
<td>64</td>
</tr>
<tr>
<td>6</td>
<td>1,024</td>
<td>108</td>
</tr>
<tr>
<td>7</td>
<td>4,096</td>
<td>80</td>
</tr>
<tr>
<td>8</td>
<td>16,384</td>
<td>60</td>
</tr>
</tbody>
</table>

Total: 130,446  313

Grand Total: 130,759.
5.1.3 Case 3: Split-and-merge method (level 5)
This method of construction needs an intermediate level above which merging and below which splitting can proceed. The level chosen for this case is level 5. The number of merges above the intermediate level and the number of splits below the intermediate level is shown in Table 7. The number of comparisons needed are computed and shown in the table. From the table it can be seen that the number of comparisons for initialising a level is 32,256, for merging is 510, and for splitting is 3720, contributing to a total number of comparisons of 36,486. The total number of comparisons for the application of the homogeneity criterion is 337 making the final total 36,823.

5.1.4 Case 4: Split-and-merge method (level 6)
The number of comparisons needed for the split-and-merge method with a different level at level 6 is computed. The results are tabulated in Table 8.
Table 7: Computational break-up of split-and-merge method (l=5)

<table>
<thead>
<tr>
<th>Level</th>
<th>Nodes</th>
<th># of Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Maximum</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>5</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>6</td>
<td>1,024</td>
<td>108</td>
</tr>
<tr>
<td>7</td>
<td>4,096</td>
<td>80</td>
</tr>
<tr>
<td>8</td>
<td>16,384</td>
<td>460</td>
</tr>
</tbody>
</table>

Total: 36,486, 337

Grand Total: 36,823
Table 8.

Computational break-up of split-and-merge method (l=6)

<table>
<thead>
<tr>
<th>Level</th>
<th>Nodes</th>
<th># of Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Maximum</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>96</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>384</td>
</tr>
<tr>
<td>5</td>
<td>256</td>
<td>1,536</td>
</tr>
<tr>
<td>6</td>
<td>1,024</td>
<td>30,720</td>
</tr>
<tr>
<td>7</td>
<td>4,096</td>
<td>3,240</td>
</tr>
<tr>
<td>8</td>
<td>16,384</td>
<td>60</td>
</tr>
</tbody>
</table>

Total: 36,006 529

Grand Total: 36,535
5.2 DISCUSSION OF RESULTS

Summarising the results of the above four cases, we see that the number of comparisons needed is maximum for the split-only method. Thus split-only method is relatively a costly method of construction. The merge-only method, even though not as costly as the split-only method, is costlier than the split-and-merge method. As discussed earlier, the computational gain by the split-and-merge method varies with the chosen initial level. The last two cases of split-and-merge method in fact, proves this point. Hence the initial level must be properly chosen for optimum construction cost. The number of comparisons for level 6 is found to be less than that for level 5. It can be observed from the node distribution, that the maximum number of nodes in the final tree exist at level 6. This confirms the fact that the initial level chosen must correspond to the level with the maximum nodes for computational efficiency.

But the level at which the maximum number of nodes exist in the tree can be found only after constructing the tree. Thus a priori information about the image must be used in choosing the initial level. In the absence of any such information about the image, it is safe to start at a level close to the lowest level in the tree.
5.3 TIME-COMPLEXITY OF THE CONNECTED COMPONENT ALGORITHM

The connected-component algorithm can functionally be split into three segments: labelling, relabelling, and equivalence processing. Hence the total time taken by the algorithm is the sum of each of these segments. This can be expressed as follows:

\[ \text{Total time} = \text{Time(label)} + \text{Time(relabel)} + \text{Time(equiv)} \]

The time taken to access a node in a tree is not a constant quantity as in case of an array representation, where any element can be accessed in constant time, but varies with the level of the node in the tree structure. Also, the time taken is proportional to the number of nodes that must be visited before reaching the required node. As any operation performed on the tree structure involves access of many nodes, the time needed for the operation can be estimated by counting the total number of nodes so visited. The operations of labelling and relabelling are performed directly on the tree and the number of nodes visited are used to obtain the time-complexity of the algorithm.

5.3.1 Time-Complexity of Labelling Phase

In order to assign a label to each of the black nodes, the whole tree has to be searched. Hence the total number of nodes visited is bounded by the number of nodes in the tree.
(Theorem 1, Appendix A). After locating a black node, two adjacencies are explored as discussed earlier. The number of nodes visited in exploring each of the adjacencies is bounded by five on the average (Theorem 4, Appendix A). Hence, for every black node in the tree, a maximum of ten nodes are visited in exploring the adjacencies. Summarising,

The maximum number of nodes that can be visited in looking for black nodes,

\[ \frac{4(B + W)}{3} \]

The maximum number of nodes that can be visited while exploring the adjacencies,

\[ 2 \times B \times 5 \]

Total maximum number of nodes that can be visited for the labelling phase,

\[ \frac{(34B + 4W)}{3} \]

Time (labelling) = \( k1 \times \left[ \frac{(34B + 4W)}{3} \right] \)

where \( k1 \) is a constant of proportionality.

5.3.2 Time-Complexity of Relabelling Phase

The relabelling phase traverses the tree once assigning a new label to each of the black node. Hence, the number of nodes visited during this phase is again bounded by the number of nodes in the tree. Thus,
The maximum number of nodes that can be visited in looking for labelled black nodes
Total maximum number of nodes that can be visited for the relabelling phase

\[ \text{Time (relabel)} = k_2 \times \left\lceil \frac{4(B + W)}{3} \right\rceil \]

where \(k_2\) is a constant of proportionality.

5.3.3 Time-Complexity of Equivalence Processing

This operation is done external to the quad-tree and hence does not depend on the number of nodes visited in the tree as the two previous operations. The time taken to settle the equivalences depend on the method adopted to process them.

The equivalence processing algorithm used also makes use of trees: binary trees. The nodes in these trees carry information about the labels. When an equivalence class is encountered, a binary tree is created in which the nodes have labels that represent the equivalents. Thus, for all the labels declared equivalent, their corresponding nodes lie in the same tree. The problem of equivalence processing amounts to searching the binary trees for labels, and if not found attach them. The effort needed to search a binary tree is proportional to the number of levels in the tree.
The maximum number of levels in a binary tree constructed with a set of $B$ labels is

$$\log B$$

The maximum number of equivalent labels (worst case) is $2 \times B$.

The maximum number of levels searched in the binary tree for processing equivalences is

$$2 \times B \times \log B$$

Time (equal) $= k_3 \times [2 \times B \times \log B]$,

where $k_3$ is a constant of proportionality.

Hence the total time taken for the algorithm is

$$= O(B + B \times \log B).$$

### 5.4 Time-Complexity of the Boundary Following Algorithm

The total time taken for the boundary following algorithm is the sum of the time taken for scanning and tracking. Thus

Time Taken $= \text{Time (scanning)} + \text{Time (tracking)}$
5.4.1 Time-Complexity of Scanning Phase

During scanning, all the nodes in the tree are visited once. In order to detect an edge-point, one neighbourhood is explored for every black node in the tree. Thus

The maximum number of nodes that can be visited while traversing the tree $= \frac{4(B + W)}{3}$

The maximum number of nodes that can be visited while exploring the neighbourhood of every black node $= 0.5 \times B$

Total maximum number of nodes that can be visited while scanning the tree $= \frac{(19B + 4W)}{3}$

Time (scanning) $= k1 \times \left[ \frac{(19B + 4W)}{3} \right]$ where $k1$ is a constant of proportionality.

5.4.2 Time-Complexity of Tracking Phase

As the next node-pair determination forms part of the tracking, the number of nodes visited for this determination has to be estimated. As discussed earlier, the number of neighbours found for the next node-pair determination depends upon the node alignment. However, for the worst-case
when all nodes along the boundary are aligned, two neighbours are found for every next-node pair determination-node-pair. The number of nodes visited in finding a neighbour is bounded by five (Theorem 4, Appendix A). Hence an average of ten nodes are visited for next node-pair determination. Summarising,

\[
\text{The maximum number of nodes visited while tracking objects} = 10 \times P \\
\text{with the total perimeter } P
\]

\[
\text{Total maximum number of nodes that can be visited while tracking} = 10 \times P
\]

\[
\text{Time (tracking)} = k_2 \times \lceil 10 \times P \rceil
\]

where \( k_2 \) is a constant of proportionality.

Hence the total time taken for the boundary following algorithm is

\[
= k_1 \times \lceil (19B + W)/3 \rceil + k_2 \times \lceil 10xP \rceil
\]

\[
= O (B + W + P).
\]

Hence the boundaries of objects in an image represented by a tree can be found in time of the order of the sum of the
black and white nodes and the total perimeter of the objects.

5.5 **Time-Complexity of the Feature Extraction Algorithm**

The various features in the quad tree can be extracted in just one tree-traversal. Thus,

- The maximum number of nodes visited while extracting the features
  
  \[ \frac{4(B' + W)}{3} \]

- Total maximum number of nodes that can be visited during feature extraction
  
  \[ \frac{4(B + W)}{3} \]

\[
\text{Time (tracking) } = k_4 \times \left\lceil \frac{4(B + W)}{3} \right\rceil
\]

where \( k_4 \) is a constant of proportionality.

\( = O(B + W) \).

It should be noted that the Big 'O' notation does not give the exact information, but gives an approximate idea of how complex the algorithm becomes with increase in the input size.

Summarising, the split-and-merge method is the best method of tree construction when the initial level
corresponds to the level at which the maximum number of nodes exist in the tree. The time-complexity of the various algorithms for processing trees were found to depend on the number of terminal black and white nodes.
Chapter VI

CONCLUSIONS AND CONTRIBUTIONS

The study of the quad trees resulted in drawing the following conclusions.

(i) Quad trees were found to be a compact form of representation for a class of images that had large homogeneous regions (like the strain gauge). The amount of compaction achieved depends on the homogeneity of the images.

(ii) The processing overheads associated with the quad trees were much less compared to the array processing.

(iii) Split-and-merge method of construction with the simultaneous removal of merged nodes from the tree is an efficient method of construction.

(iv) Image processing operations like segmentation, connected component labelling, boundary following, template matching, etc., can be performed directly on the quad tree without the necessity of switching back to the array representation.
The following are the contributions as part of the study:

(i) Software for the conversion of images into its quad tree representation and vice-versa was developed.

(ii) A neighbour-finding technique for quad trees was developed.

(iii) Software for processing images represented by quad trees was developed. This encompasses the following:

(a) implementation of the connected-component labelling algorithm (Samet)
(b) adaptation of the boundary following algorithm to quad-trees
(c) extraction of features like area, perimeter, etc. on the quad-tree
(d) development of the template-matching algorithm for quad-trees

(iv) A variation is suggested to Samet's connected component algorithm that ensures minimum number of labels being generated.

(v) The upper-bound theorem has been generalised for a k-ary tree

(vi) A method of data transfer between the IBM and the NOVA computer was established.
(vii) An image data base has been created on the IBM for future users.
Appendix A

INTRODUCTION TO TREE STRUCTURES

A tree is a hierarchical structure that describes the relationship between its various members. The members of such a structure being a root, node, leaf, and sub-tree. The tree has a number of nodes in them and these nodes being part of a hierarchy have parents and children. Each tree has a unique node from which the hierarchy grows. Such a node will not have a parent and is known as the 'root' of the tree. Nodes in a tree are of two types: terminal and non-terminal. A node which has no children is called as a terminal node or as a 'leaf'. A node having children is called a non-terminal node or a branch node. These branch nodes exist at the intermediate levels of the tree structure. Even though a root has children, it is excluded from the class of non-terminal nodes. A part of the tree is sometimes called as a sub-tree or a 'branch'. The information in the tree is stored in the nodes. Nodes in a tree can have any number of children, but trees in which all the non-terminal nodes have the same number of children is known as uniform trees. In this study, only uniform trees are being considered.
Many methods have been used to represent trees graphically. Fig. 24 shows a conventional representation of a general tree with an illustration of the notation used. In this conventional representation of a tree, the root is shown at the top of the tree and the leaves at the bottom (contrary to the way trees grow in nature). Trees can be classified based on the degree of the nodes as binary, quad, oct, etc. A binary/quad/oct tree is a tree in which the non-terminal nodes are of degree two/four/eight. Fig. 25 shows examples of these tree structures. For any tree the maximum number of nodes that can exist in the structure is proportional to the number of 'leaves'. The upper bound on the total number of nodes in a general tree is established by the following theorem. This theorem is a generalization of the upper bound theorem for a binary tree.
Fig. 24. Conventional representation of a Quad-tree.

( □ terminal node
  ○ non-terminal node ).
THEOREM 1: The total number of nodes in a K-ary tree with $N_n$ non-terminal nodes and $N_t$ terminal nodes is bounded by,

$$\left\lfloor \frac{K}{(K-1)} \right\rfloor \times N_t \qquad \text{for all } K > 1$$

PROOF: Any tree is an acyclic graph and hence the total number of edges is given by

$$(N_n + N_t - 1)$$

Also, the number of edges obtained by counting the number of sons is

$$K \times N_n$$

Equating the two, we get

$$K \times N_n = N_n + N_t - 1$$

i.e.,

$$K \times N_n - N_n = N_t - 1$$

$$(K - 1) \times N_n = N_t - 1$$

$$N_n = \left\lceil \frac{(N_t - 1)}{(K-1)} \right\rceil$$

Adding $N_t$ to both sides,

$$N_n + N_t = \left\lceil \frac{(N_t - 1)}{(K-1)} \right\rceil + N_t$$

$$= \left\lceil \frac{K}{(K-1)} \right\rceil \times N_t \cdot \left\lceil \frac{1}{(K-1)} \right\rceil$$

But $N_n + N_t$ is the total number of nodes in the tree. Hence
Total # of nodes = \[ \frac{K}{(K-1)} \times N_t \times N_{t'} - CON \]

where \( CON = \frac{1}{(K-1)} \), which is constant for a given value of \( K \).

Therefore,

\[ \text{Total # of nodes} \leq \left[ \frac{K}{(K-1)} \right] \times N_t \]

- Q.E.D

CASE 1. : For a binary tree, \( K = 2 \).

Hence the total number of nodes in the tree is

\[ \leq 2 \times N_t \]

The value of \( CON \) being \( 1 \).

CASE 2. : For a Quadtree, \( K = 4 \).

Hence the total number of nodes in the tree is

\[ \leq \left\lceil \frac{4 \times N_t}{3} \right\rceil \]

The value of \( CON \) being \( 1/3 \).

CASE 3. : For an Octree, \( K = 8 \).

Hence the total number of nodes in the tree is

\[ \leq \left\lceil \frac{8 \times N_t}{7} \right\rceil \]

The value of \( CON \) being \( 1/7 \).
A.1 REPRESENTATION OF TREES

The actual representation of a tree in a computer can easily be accomplished once we define the information to be represented by each node. As each node in the tree has associated 'relatives', they must have a mechanism of locating them. This is achieved by means of 'pointers' that establish the links between the related nodes. Thus if nodes B and C are children of node A (Fig. 26), then the address of node A is stored in the nodes B and C as their parent. Likewise, the address of nodes B and C are stored in A as its children. Thus every node in the tree structure will have a number of fields of information in them about its parent and children, which can easily be represented in memory. Hence by storing all the nodes in the tree structure and properly linking the related nodes, a tree can be represented in memory.

A.2 ACCESS OF NODES

Any node in the tree cannot be accessed directly as in case of an array. To access an intermediate node, one has to descend down the tree from the root by choosing suitable paths that lead to the desired node. The address of the root node is stored as a header, and is used to gain access to the tree. For terminal nodes, special pointers, called 'NIL' pointers are stored in the appropriate fields signifying the terminality of the hierarchy.
Fig. 26. A Typical node
A.3 OPERATIONS ON TREES

After having the complete tree structure, certain operations need to be performed to achieve desired results. They are

(i) Traversing,

(ii) Merging, and

(iii) Splitting.

A.3.1 Traversal

It is often necessary to examine each node in the tree in a fixed order. Working through the tree in such a way that each node is visited once is known as traversing the tree. Thus traversal is the method of examining the nodes of the tree systematically so that each node is visited exactly once without leaving out any node in the process. In addition, it should be done without getting lost inside a tree. There are various methods of traversing a tree and the one that is used here is known as the 'post-order' tree traversal. The order in which the tree is traversed is as follows:

(i) visit Son1,

(ii) visit the Parent,

(iii) visit Son2,

(iv) visit the Parent,

(v) visit Son3,

(vi) visit the Parent,
(vii) visit Son4, and
(viii) visit the Parent.

Repeated execution of the above steps will systematically lead us through the tree. The order in which the tree of Fig. 27a(ii) is traversed is shown by numbering the nodes. The corresponding scan of the image is also shown numbered in Fig. 27a(i).

A.3.2 Merging

The domain of any node in the tree is a union of the domains of its children. If the parent could completely represent what the children can, then the existence of the children in the structure becomes redundant. The process of removing the redundant children of a node is called 'merging' (Fig. 27b).

A.3.3 Splitting

The inverse operation of merging, is splitting. If the node cannot completely represent the content of its domain, then it gives 'birth' to children. This process of a node giving rise to its children is called as 'splitting' (Fig. 27c).
(a) Sequence of traversal

(b) Merging

(c) Splitting

Fig. 27: Operations on Quad-trees
A.4 SOME THEOREMS ON TREES

A few important theorems that were used in the computational analysis are given along with their proofs.

THEOREM 2: The maximum number of nodes visited in finding the neighbour of a node at level i with its common ancestor at level j is

\[ = 2(i-j-1) + 2^{j-1} \]

Proof: The number of nodes visited in reaching the common ancestor = \( i-j \).

The number of nodes descended from the common ancestor to an equal adjacent node = \( i-j-1 \).

Maximum number of nodes will be visited if the neighbour node happens to be at the lowest level in the tree.

The number of nodes visited in reaching the node at the lowest level from an equal adjacent node at level i

\[ = \sum_{k=0}^{n-i-1} 2^k \]

Therefore, the maximum number of nodes visited in finding the neighbour of a node at level i with a common ancestor at level j is

\[ = i - j + \sqrt{j-1} + \sum_{k=0}^{n-1} 2^k \]
\[ = 2 \binom{T - j}{i} + \sum_{k=1}^{n-1} 2^k \]

\( \{ \text{As} \sum_{k=1}^{r} 2^k = 2^{r+1} - 2 \} \)

\[ = 2 \cdot \binom{1 - j - 1}{i} + 2^{n-1} \]  

\[-Q.E.D-\]
THEOREM 3: The maximum number of neighbour node pairs in a tree at a level \( i \) for a given direction is:

\[
2^{i-1} (2^{i-1} - 1)
\]

Proof: The number of nodes in a tree at a level \( i \) is:

\[
2^{i-1} \times 2^{i-1} = 2^{2i-2}
\]

The number of nodes at level \( i \) that do not have a neighbour in the given direction is:

\[
2^{i-1}
\]

(These many number of nodes lie to the extreme side of the image in the given direction).

Hence the maximum number of neighbour node pairs at level \( i \) for the given direction is:

\[
2^{2i-2} - 2^{i-1} = 2^{i-1} (2^{i-1} - 1).
\]

Q.E.D.
THEOREM 4: The average of the maximum number of nodes visited in finding a neighbour in a quad tree is bounded by five.

Proof: The number of nodes at level \( i \) having a common ancestor at level \( j \) \((j > i)\) is

\[
2^{i-1} \times 2^{j-1}
\]

If the node is equally likely to exist at any of the levels in the tree, then the limits for \( i \) and \( j \) are

\[
2 \leq i \leq n \\
1 \leq j \leq i-1
\]

The total number of nodes at all levels in the tree having a common ancestor at all possible levels in the tree is

\[
\sum_{i=2}^{n} \sum_{j=1}^{i-1} 2^{i-1} \times 2^{j-1}
\]

The maximum number of nodes visited in finding a neighbour of a node at level \( i \) with the common ancestor at level \( j \)

\[
= 2 (i - j - 1) + 2^{n-i+1}
\]

\{ from Theorem 2 \}
If the neighbours are found for each of these nodes, then the maximum number of nodes that could possibly be visited

\[ n \sum_{i=2}^{n} \sum_{j=1}^{i-1} 2^{i-1} \times 2^{j-1} \times \left\{ 2^{(i-j)-1} + 2^{n-i+1} \right\}. \]

The maximum number of neighbour node pairs at level \( i \) is

\[ = 2^{i-1}, \quad (2^{i-1} - 1). \quad \{ \text{from Theorem 3} \} \]

The maximum number of neighbour node pairs in the complete tree is

\[ = \sum_{i=2}^{n} 2^{i-1}, \quad (2^{i-1} - 1) \]

The average of the maximum number of nodes visited in finding a neighbour

\[ = \frac{\text{Total maximum No. of nodes visited in finding all neighbours}}{\text{Total No. of neighbour-node pairs in the tree}} \]

\[ = \frac{\sum_{i=2}^{n} \sum_{j=1}^{i-1} 2^{i-1} \times 2^{j-1} \times \left\{ 2^{(i-j)-1} + 2^{n-i+1} \right\}}{\sum_{i=2}^{n} 2^{i-1}, \quad (2^{i-1} - 1)} \]
5 as \( n \) gets large.

Thus the average maximum number of nodes visited in finding a neighbour in a quad tree is bounded by five.

- Q.E.D
Appendix B

DATA TRANSFER INFORMATION

Transfer of data between the IBM and the NOVA computers cannot be done directly because of their internal differences. As magnetic tape drives are supported by both the computers, tape is the only efficient media for data transfer. The data to be transferred is written on the tape and read by the other computer. As the data written by one computer is not acceptable to the other, certain amount of processing is involved during the data transfer. This processing is essentially a type of data translation/interpretation so that the destination computer understands correctly the source computer. Thus, translation is needed in either direction of transfer.

Data is written on the tape as a number of blocks each having a fixed number of logical records. The blocksize and the record length can be chosen by the user on an IBM machine, while that on the NOVA is fixed at 514 and 4 respectively. This constraint has to be taken care while writing data on the tape. A method of transfer in both directions have been described here, and it should not be construed that this method is the only method or the best possible method of data transfer, but as the first method of data transfer to and from the IBM.

- 117 -
B.7 TRANSFER TO IBM FROM NOVA

The data is written on a magnetic tape (initialised on the NOVA) by using program 1. This program does not do any special task, but a dummy operation of addition and subtraction before writing the data on tape. This operation is essential because of the fact that any writing immediately after a reading corrupts the data on the NOVA.

The data is written in the standard NOVA format with a blocksize of 514 and a record length of 4.

The tape is read on the IBM using the FORT Loader by the program 2. This program reads the data from the tape and writes it on the disk temporarily. As a blocksize of 514 is inconvenient on the IBM, the data is stored on the disk with a blocksize of 512. Because of the earlier blocksize of 514, a record would have been split between two blocks alternately. The program takes care of this and writes the data in the correct format on the disk.

The temporary data on the disk is processed by the PASCAL program, program 3, which interprets the data on the disk correctly and transfers it to the magnetic tape. This data could be used on the IBM for subsequent processing.
B.2 Transfer to nova from IBM

The data to be transferred to the NOVA is written on the magnetic tape initialised on the IBM. No special program is needed to write the data on the tape, but the record length and the blocksize must be strictly followed. The data written on the IBM is interpreted on the NOVA by the program 4, which stores its output as a disk file. This data could be used for processing on the NOVA.
PROGRAM TO WRITE DATA ON TAPE FOR TRANSFERRING IMAGE TO IEN

INTEGER IX(128, 128), NAME(5)

TYPE "ENTER THE IMAGE FILENAME"
READ(1,1) NAME(I), I=1, 5

ACCEPT "ENTER THE SIZE OF INPUT IMAGE" N
OPEN 4, NAME
DO 20 I=1, N
READ(1) IX(I, J), J=1, N
20 CONTINUE
CLOSE 1

DO 101 I=1, N
DO 101 J=1, N
IX(I, J)=IX(I, J)+1
101 CONTINUE

DO 102 I=1, N
DO 102 J=1, N
IX(I, J)=IX(I, J)-1
102 CONTINUE

TYPE "ENTER TAPE FILENAME"
READ(1,1) NAME(1), I=1, 5

FORMAT(5R2)
NLEN=4*N
OPEN 0, NAME, LEN=NLEN
DO 3 I=1, N
WRITE(0, 2) IX(I, J), J=1, N
3 CONTINUE

FORMAT(128I4)
CLOSE 0
STOP
END
**PROGRAM 2.**

```plaintext
//ASCII JOB 'XXXXXXXXXXXXX', '#', CLASS=L, MSGLEVEL=(1,1)
// EXEC FORTRAN
//FORTY(SYSIN DD *)
DIMENSION P(514), S(512)
1 FORMAT(250A1, 250A1, 14A1)
20 FORMAT(250A1, 250A1, 12A1)
I = 1
3 READ (1,1, END=10) P
I = 1
4 S(I) = P(I)
I = I + 1
5 IF (I.EQ.513) GO TO 5
   GO TO 6
5 WRITE (2, 20) S
I = 1
6 IF (I.EQ.511) GO TO 3
   GO TO 4
10 CONTINUE
STOP
END
```

```
/* GO.FT01F001 DD DSN=FILE, UNIT=7800, VOL=SER=INTAPE, LABEL=(#, NL)
   DISP=OLD, PASS), DCB=(RECFM=F, LRECL=514, BLKSIZE=514, DEN=2,
   OPTCD=Q)
   GO.FT02F001 DD DSN=CHANDR, UNIT=3330-1, VOL=SER=DISK04,
   DISP=(NEW, KEEP), DCB=(RECFM=F, LRECL=512, BLKSIZE=512),
   SPACE=(TRK,(1,1))
*/
```
PROGRAM TEST(INFILE,OUTFILE);
(*$S .80000, X 1000000, P 50000 *).

TYPE
FILETYPE=FILE OF CHAR;
FILEVAR=FILE OF INTEGER;
ARRY=ARRAY [1..128,1..128] OF INTEGER;
CARRY=PACKED ARRAY [1..4] OF CHAR;
VAR
INFILE:FILETYPE;OUTFILE:FILEVAR;
I,J,K,MUL,NUM,COUNT:INTEGER;
IMA:ARRY;CH:CARRY;
PROCEDURE CONVERT(VAR NUM:INTEGER;CH:CARRY);
LABEL 11;
VAR
K:INTEGER;
BEGIN
COUNT:=0;NUM:=0;MUL:=1000;
FOR K:=1 TO 4 DO
BEGIN
MUL:=MUL DIV 10;
IF CHECKJ = ' ' THEN BEGIN COUNT:=0*MUL; GOTO 11; END;
IF CHECKJ = '0' THEN BEGIN COUNT:=0*MUL; GOTO 11; END;
IF CHECKJ = '1' THEN BEGIN COUNT:=1*MUL; GOTO 11; END;
IF CHECKJ = '2' THEN BEGIN COUNT:=2*MUL; GOTO 11; END;
IF CHECKJ = '3' THEN BEGIN COUNT:=3*MUL; GOTO 11; END;
IF CHECKJ = '4' THEN BEGIN COUNT:=4*MUL; GOTO 11; END;
IF CHECKJ = '5' THEN BEGIN COUNT:=5*MUL; GOTO 11; END;
IF CHECKJ = '6' THEN BEGIN COUNT:=6*MUL; GOTO 11; END;
IF CHECKJ = '7' THEN BEGIN COUNT:=7*MUL; GOTO 11; END;
IF CHECKJ = '8' THEN BEGIN COUNT:=8*MUL; GOTO 11; END;
IF CHECKJ = '9' THEN BEGIN COUNT:=9*MUL; GOTO 11; END;
11: NUM:=NUM+COUNT;
END;
END (* END OF CONVERT *);

PROGRAM 3.
BEGIN
RESET(INFILE);REWITE(OUTFILE);
FOR I:=1 TO 128 DO
BEGIN
FOR J:=1 TO 128 DO
BEGIN
READ(INFILE,CH[1],CH[2],CH[3],CH[4]);
CONVERT(NUM,CH);
IMA[I,J]:=NUM;
END;
READLN(INFILE);
END;
FOR I:=1 TO 128 DO
BEGIN
FOR J:=1 TO 128 DO
BEGIN
WRITE(OUTFILE,IMA[I,J]);
END;
END;
END.
$ENTRY
$IBSYS

PROGRAM 3.
PROGRAM TO READ TAPE BURIED ON 180

PROGRAMMED BY J. RAJENDRAN & A. CHANDRASEKHAR

ARRAY TO STORE IMAGE

IX ------- ARRAY TO STORE THE IMAGE DATA
IDUM ------- DUMMY ARRAY USED TO READ DATA FROM TAPE
NDATA ------- NUMBER OF DATA IN EACH BLOCK
NBD ------- NUMBER OF BYTES TO BE SKIPPED BETWEEN TWO DATA POINTS
NBL ------- NUMBER OF BYTES TO BE SKIPPED BETWEEN TWO BLOCKS
MASK ------- USED TO RETAIN THE M & N WHICH CONTAIN THE DATA
NMASK ------- USED TO RECTIFY THE SIGN EXTENSION DUE TO SHIFLING

INTEGER IX(128,128), NAME(5), IDUM(200)
TYPE "ENTER TAPE FILENAME"
READ(11,15)(NAME(I), I=1,5)
PERM(0,52)
ACCEPT "ENTER THE SIZE OF THE IMAGE" NSIZE
NDATA=64
NBD=5
NBL=257
NMASK=0.037777
MASK=0.171999
OPEN(0, NAME-
CALL RSECD, I, J, M, N, ICOUNT, IER)
IRON=1
KOUNT=1
ICOL=1

100 CALL RSECD, IX, IROH, ICOL, Y, IER
CALL CHECKER
ITEMF=1.0 * (IX, IROH, ICOL) * MASK
ITEMF=ITEMF/256
IF (ITEMF .GE. 0) GO TO 10
ITEMF=ITEMF * NMASK
10 IX=IROH, ICOL=ITEMF
ICOL=ICOL+1
IF (ICOL LE NSIZE) GO TO 121
ICOL=1
IRON=IRON+1
KOUNT=KOUNT+1
IF (KOUNT .GE. NDATH) GO TO 187

PROGRAM 4.
CALL RDSEC8(0, IDMC1, NSD, ICNT, IER)
CALL CHECK(IER)
GO TO 100

107 KOUNT+1
CALL RDSEC9(0, IDMC1, NSD, ICNT, IER)
IF (IER NE. 1) GO TO 102
GO TO 100

102 CLOSE 0
ACCEPT 'ENTER OUTPUT FILE NAME ='
READ (11, 15) NAME(11:1, 1:15)
OPEN 0: NAME
DO 105 I=1, NSICE
WRITE (0) (10(I), 1:1, J=1, NSICE)
CONTINUE

105 CLOSE 0
STOP
END

PROGRAM 4,
Appendix C

COMPUTATION OF MOMENTS

Moments about a point is defined as the product of mass and their distances from it. An object consists of a number of points clustered together. From the theory of moments, it is known that the sum of the moments of all the particles of an object is equal to the product of the sum of the masses of all particles and the distance of a 'fixed point' inside the object from the reference point. This point can be considered as the point where the entire mass of the object is concentrated. This point is referred to as the 'centre of mass' or centroid of the object. Thus if \( m_1, m_2, m_3, \ldots, m_n \) is the mass of each particle of an object at distances \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\) from the reference point, and if the coordinates of the centre of the mass is \((X, Y)\), then

\[
M \cdot X = m_1 \cdot x_1 + m_2 \cdot x_2 + \ldots + m_n \cdot x_n
\]

or

\[
X = \frac{(m_1 \cdot x_1 + m_2 \cdot x_2 + \ldots + m_n \cdot x_n)}{M}
\]

and

\[
M \cdot Y = m_1 \cdot y_1 + m_2 \cdot y_2 + \ldots + m_n \cdot y_n
\]

or

\[
Y = \frac{(m_1 \cdot y_1 + m_2 \cdot y_2 + \ldots + m_3 \cdot y_3)}{M}
\]

where \( M = \sum m_i \) is the total mass of the object.
Fig. 28. Labelled image with three components
In the segmented image, the mass of the particles belonging to an object can be considered as unity, and zero otherwise. Hence the coordinates of the centroid can be written as follows:

\[
X = \frac{(x_1 + x_2 + x_3 + \ldots + x_n)}{\text{Area}}
\]

and

\[
Y = \frac{(y_1 + y_2 + y_3 + \ldots + y_n)}{\text{Area}}
\]

C.0.1 Illustration

In order to illustrate the computation of centroid, a segmented image with three components as shown in Fig. 28 is considered. The first order moments in both the \(X\) and \(Y\) directions, and the area are computed as follows:

C.0.2 Object 1

The area of the object = 25

The first moment in the \(X\)-direction = 100

The first moment in the \(Y\)-direction = 125

Hence the coordinates of the centroid is:

\[
X = \frac{100}{25} = 4
\]

and

\[
Y = \frac{125}{25} = 5
\]
Thus the computed centroid for the object 1 is \((4, 5)\) which from the figure is seen to be correct.
Appendix D

OTHER APPLICATIONS OF QUAD TREES

Quad trees have been used in image processing not only for region representation, but also for certain other applications as well. They have been used in the display systems to display an image. The image is displayed at various resolutions progressively. In such progressive techniques, a crude representation of the image is transmitted first, and the details are added later. Such a presentation is desirable when a viewer wishes to recognize image content as soon as possible, and stop transmission of unwanted detail, or initiate actions simultaneous to transmission of the remaining detail. Progressive transmission is also desirable simply as a means to provide an aesthetically pleasing sequence which holds the viewers attention. For these purposes, a progressive display is more effective than a line-by-line raster scanning display. The image to be displayed is first converted to a tree and the levels of the tree are transferred at a time at varying resolutions. As the amount of information transmitted is less than that needed for the image array, it is also faster.
In addition, they have been used to selectively access image data in geographical image processing. In such applications, the nodes of the quad tree are made to represent certain geographical area. Depending upon the need, nodes are accessed in the tree which will represent a small portion of the area. Thus geographical data can be selectively accessed by using quad trees.
REFERENCES


VITA AUCTORIS

1956  Born March 4, Bangalore, India.

1971  Matriculation, Govt. Higher Sec. School,
      Channapatna, Karnataka, India.

1973  Pre-university Education,
      St. Joseph's College, Bangalore, India.

1978  Bachelor of Engineering
      (Electronics & Communications),
      University of Mysore, Mysore, India.

1979  Asst. Engr. (Pr. Tr.),
      Hindustan Aeronautics Limited, Bangalore,
      India.

1980  Systems Engineer (Numerical Control M/C),
      HMT Limited, Bangalore, India.

1984  Master of Applied Science
      (Electrical Engineering),
      University of Windsor, Windsor, Ontario,
      Canada.