PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur: SEMWAL, SUDHANSHU KUMAR	
Date of Birth — Date de naissance: MAY 11, 1960	Country of Birth — Lieu de naissance: INDIA
Permanent Address — Résidence fixe: C-844, MAHANAGAR EXTENSION, LUCKNOW-6, U.P. 226006, INDIA.	
Title of Thesis — Titre de la thèse: DATA STRUCTURES FOR SPATIAL INFORMATION	
University — Université: OF ALBERTA, EDMONTON.	
Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée: M.Sc.	
Year this degree conferred — Année d'obtention de ce grade: 1984.	Name of Supervisor — Nom du directeur de thèse: DR. W. A. DAVIS.

Date: MARCH 16, 1984	Signature: [illegible]

THE UNIVERSITY OF ALBERTA

Data Structures for Spatial Information

by

(C) Sudhanshu Kumar Semwal

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE
OF Master of Science

Department of Computing Science

EDMONTON, ALBERTA

FALL, 1984

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR	Sudhanshu Kumar Semwal
TITLE OF THESIS	Data Structures for Spatial Information
DEGREE FOR WHICH THESIS WAS PRESENTED	Master of Science
YEAR THIS DEGREE GRANTED	FALL, 1984

(SIGNED) [illegible]

PERMANENT ADDRESS:

C-844, MAHANAGAR EXTENSION,
LUCKNOW-6, U.P.
226006, INDIA

DATED March 17, 1984

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled Data Structures for Spatial Information submitted by Sudhanshu Kumar Semwal in partial fulfilment of the requirements for the degree of Master of Science.

W. A. Davis
....
Supervisor

[illegible]
....

[illegible]
....

[illegible]
....

Date 18 June 84

ABSTRACT

Data formats, for representing spatial information, are surveyed and evaluated. The emphasis is on the time and space complexity, ease of encoding, displaying, browsing, scaling and set-theoretic operations.

A new data format - Octagonal Tree Data Structure (OTDS) - is presented. In this data structure, spatial information is organized in the form of a tree so that updates, addition and deletion of a point or line can be efficiently performed. Particularly, search for a point is logarithmic in the number of points in the image.

Implementation details are also summarized for programs to obtain a rectangular portion of two data banks, namely - Edmonton Map Data Bank and World Data Bank II.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude towards Dr. W. A. Davis, my thesis supervisor, for his guidance, assistance and patience throughout the course of this research.

I am also grateful to the members of my examining committee, Dr. S. Cabay, Dr. M. Green and Dr. J. C. Muller, for their helpful comments.

I also wish to thank the National Science and Engineering Research Council of Canada for their support of this research through Grant G0883 and the Department of Computing Science for their financial assistance.

Finally, I would like to thank my parents, family members and friends for the various form of assistance that they have given me.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

This thesis is concerned with effective and efficient ways of representing spatial data. There has been wide interest [Same83, Tamu82, Nagy79, Dutt78] in the efficient design of spatial databases in recent years. This is due primarily to the increased use of digital data in applications such as medicine, geography, remote sensing, television, etc. In these applications, the two-dimensional nature of the data influences the organization of the data and the design of the algorithms for processing the data. As a result, the choice of a suitable format is a very important feature of the data. Indeed, when the systems being investigated are distributed, interactive and involve large image files [Nagy79 and Davi82], the need is even more urgent.

Effective distribution of information in a distributed system involves issues such as the need for a high speed communication link between devices. This can be illustrated by considering a typical query in a distributed Geographical Information Processing System (GIPS) [Nagy79 and Dutt78].

Suppose a user in some location wants to display from the database those regions having a population of more than 100 persons per square mile. The query originates at the user site and since the data is probably distributed among various sites, there is an increase in the data traffic in the system as large files are transferred from one location to another. The fact that the system is interactive makes it important that the information transfer be fast. This can be assured not only by having high speed communication but also by an appropriate format. Other important issues include: data security, error detection, correction and recovery. These issues have not been dealt with in any detail in this thesis, which is not intended as a reflection on their importance. The major emphasis of this thesis is to discuss the data organization and retrieval techniques for representing two dimensional regions, point and curvilinear data and to consider their relative merits. For more details on the earlier results and related topics, the interested reader may consult the surveys by Nagy and Wagle [Nagy79], Bentley and Friedman [Bent79] and the technical report by Samet [Same80].

1.1 Fundamentals

Spatial information can be broadly classified into three data-types: *point, line and region*. Two regions that share a common boundary segment are called *neighbors*. Data formats for representing spatial information have been called *spatial data structures*. In this thesis, the terms data format, data representation and data organization will also refer to spatial data structures. A good data representation scheme should:

1. be compact for efficient data storage,
2. permit easy and precise encoding,
3. facilitate effective data extraction for analysis, manipulation and display,
4. be suitable for efficient data transmission,
5. be flexible to accommodate future developments and breakthroughs,
6. allow browsing.

In general, there is a trade off between the display time (including access time) and the amount of memory required to store the information. Usually, if the data organization is very compact (i.e., requires less memory) then there is an increase in access time and processing overheads for encoding, manipulating and displaying the information [Free82].

A data organization scheme should be capable of performing operations such as: set-theoretic operations (e.g., intersection and/or union of regions), deletion, addition and insertion of any of the data-types, rotation about a point or a line and neighborhood location. These operations are henceforth referred to as *basic operations*.

There are two conventional methods for storing the spatial information namely *a cellular organization* and *a* *linked organization* [Nagy79]. In a cellular organization, the extent under consideration is divided into a matrix of cells of uniform size and computer storage is reserved for each cell. In the reserved memory either the information associated with each cell or a pointer to that information is stored. On the other hand, in a linked organization, a point is represented by a coordinate, a line by a sequence of points and a region by a closed sequence of lines [Nagy79].

Spatial data structures are usually obtained by either encoding the information in terms of points, lines and regions or scanning the image line by line and detecting the intensity of each pixel value in each line. The resultant spatial data is termed as *vector and raster data* respectively. Data formats for representing vector (raster) data are termed as *vector (raster) data formats*.

Sometimes it is required to maintain the inter-relationship between two data-types to facilitate inclusion/exclusion queries such as: 'Is the object (a point) inside the polygon (a region)?', 'Does the road (a line) cross the forest (a region)?' or 'Is the city (a region) inside the state (a region)?'. For this purpose spatial data is organized in a hierarchical, or tree form. In the tree, each node representing a data-type is the combination of the data-types corresponding to the immediate son-nodes. The root of the tree signifies the whole image under consideration. These data organizations are termed *hierarchical data structures* (HDS) [Gibs82] or *tree data structures* (TDS) [Dyer82]. In a tree data structure, a region is recursively subdivided into smaller regions. The recursion continues till regions with a pixel or equal intensity pixels are obtained. These regions, containing equal intensity regions are called *terminating regions* (TR).

Sometimes, in case of vector data, the points are arranged in a tree format by recursively subdividing the extent under consideration into sub-regions with an equal number of points. The recursion continues till regions with only one point are obtained, which are called *terminating partitions* in this thesis. The line-segments in the image are maintained by defining a set for every point (P) in a terminating partition. This set contains an array of points indicating that each point of the array is connected by a line-segment to the point (P). These data organizations have been defined as *hybrid data structures* (HDS) or *hybrids*.

Surprisingly, the time and space efficiency of all four classifications is totally different. In general, raster data structures offer time and memory performance of order $\Theta(n*m)$, where n is the number of pixels in the scan line and m is the number of horizontal scans required. On the other hand, vector data formats are linear in terms of the number of points and lines and tree-data organizations are logarithmic in access time. In case of hybrid data structures, points in the vector data are arranged as leaf-nodes of a balanced tree and can be accessed logarithmically in the number of points. The memory required is linear in terms of number of points and lines.

1.2 Thesis organization

In Chapter 2, there is a discussion of the essential characteristics and data organization of the existing data formats. The survey is intended to provide a complete overview of current research in this area.

In the next chapter, a data structure has been proposed and algorithms are presented for performing the basic operations. The memory required is bounded by the number of points and lines in the image and the search time for a data-type is logarithmic in the number of points.

In Chapter 4, all the data structures have been analyzed by considering an image consisting of a triangle and an isolated point. The time and space complexity and ease of encoding and displaying the spatial information are briefly compared and summarized.

7

Finally, in the concluding chapter, the results are summarized and further areas of research have been identified. [illegible]tanding problems and areas that require further deta[illegible] solutions are also indicated.

CHAPTER 2

SPATIAL DATA STRUCTURES: A SURVEY

In this chapter, a review of existing data formats is presented. The discussion of existing data formats has been divided into following four sections:

1. *POINTS,*
2. *LINES,*
3. *REGIONS, and*
4. *PICTURE (IMAGE).*

The above classification groups together the available data formats for representing point, line, region and raster data. This is an attempt to provide the necessary options depending upon the nature of the spatial data. It should be noted that the discussion of time and space performance, along with other factors effecting the choice of a suitable data format will be briefly covered in Chapter four.

2.1 POINTS

Point information is usually represented in absolute or relative coordinates. A point in k-dimensions can be uniquely defined by storing absolute coordinate of the point in k dimensions. This results in a data structure consisting of an array of points in which every element of the array has k entries.

Sometimes a reference point is chosen and all the other points are defined by specifying the difference in the coordinate information with respect to the reference point. Again, an array of points results such that the first entry in that array specifies the coordinates of the reference point and the following entries contain the difference in the coordinate information of the other points with respect to the reference point. In some cases, a point is specified by storing the difference between the coordinates of the point referred last and the point being specified. Both the above techniques are basic to all the data structures as will be shown in Sections 2.2 and 2.3.

2.2 LINES

A line can be defined as sequence of vectors. These vectors in turn can be stored by an array of consecutive points such that every consecutive pair of points are connected. Thus, 'n' points would be needed to denote a line with (n-1) line-segments. This section describes the data organizations which can be used to store the linear information. The data structures belonging to this category are: chain codes and its variations, run length coding and strip trees.

2.2.1 Chain Codes

When a rectangular grid is used to encode a picture, each point or node on the rectangular grid is surrounded by eight near neighbors or points [Free82, Free74]. If the neighboring eight points are numbered from 0 to 7 in a counter-clockwise direction starting from the positive x axis then a line may be represented by a sequence of octal numbers, see Figure 2.1. This unique representation is termed a Chain Code (CC). The digits 0 to 7 are called links and 'links' representing the structure are called a chain [Free74]. Several codes may be used to specify the end of a chain and repetitions of identical links to save memory.

By utilizing the property that successive elements in a chain are not statistically independent a more compact representation is possible. A change in direction would normally not exceed 45 degrees and rarely be 90 degrees. This suggests a variable length code, i.e., bits, for different ranges, e.g., 2 bits for 45 degrees. This scheme is called Chain Difference Coding (CDC) and results in a more compact data representation. Again this representation scheme is suitable for contour maps.

A natural extension of chain codes is a 3 dimensional linear representation obtained by quantizing the cubic lattice with 26 possible directions instead of 8 in 2 dimensions. Since there is no ambiguity in representing all possible directions as links, this produces a unique and useful scheme. Chain codes depend largely on the faithful

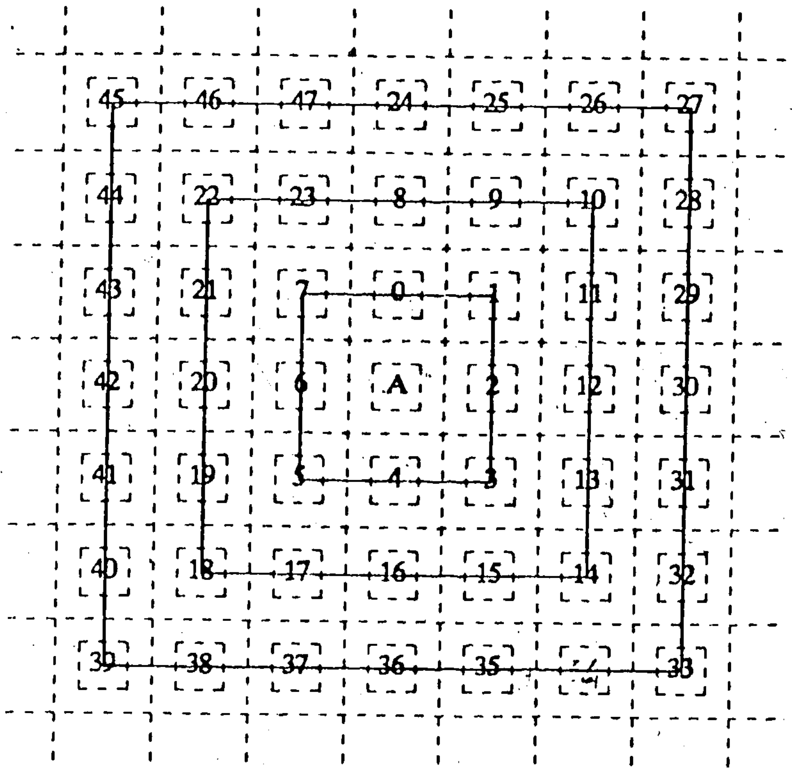


Figure 2.1:

Chain Coding Scheme showing the concentric rectangles. In this example, line segments of 48 different lengths are available [Case 3 of section 2.2.2].

coding of the data and hence the consideration of noise in quantizing devices, noise filtering and related problems deserve attention.

2.2.2 Other variations of chain codes

Depending upon the manner in which the sequence of nodes is encoded, the following five possibilities will result [Free82]:

1. Connecting lattice nodes with short straight line segments: this results in the basic chain coding scheme. Each line segment is of length T or 1.4 T, where T is the grid spacing and there are 8 directions, as discussed earlier.
2. Use of the longest possible straight line segments between lattice nodes: The scheme is referred to ~~as~~ *polygonal approximation* [Free82]. The line segments tend to be larger for straight lines and smaller for sharp curves. Rarely occurring line segments must be avoided in order to produce a compact scheme. This is shown in case 3 below.
3. The Generalized Chain Coding Scheme (CGGS) is a compromise between case 1 and 2, where the scheme provides many but not all permissible line segment lengths. If we visualize the present node being surrounded by concentric squares of nodes then case 1 results when only nodes in square 1 (0 to 7) are considered. As larger concentric squares are

permitted, line segments of longer lengths are available. Better angular resolution results, which facilitates more accurate representation of curves.

4. Curves may be utilized instead of line segments, and successive nodes may be farther apart when connected by a curve than by a line [Free82]. This results in fewer but more complex approximations.

5. For each node, a set of concentric squares and a set of predetermined curves are available to connect present nodes to the nodes of the concentric squares. The longest curve segment is always chosen to advance from one node to another. This scheme is a compromise between case 3 and 4 and retains all the advantages of GCCS with less processing and an encoding complexity of case 4.

With GCCS, a 40% reduction in memory was obtained. By taking 2 bytes to represent the link number, so that link numbers greater than 255 may also be used, a 91% savings was achieved. For the same data, the Chain Difference Coding Scheme(CDCS) resulted in a reduction by a factor of 13 in storage. The coding used was link numbers 0-127 in one byte, 127-16383 in two byte and greater than 16383 in 3 bytes [Refer Free82]. It was thus confirmed that both GCCS and CDCS result in a substantial savings in storage. At the same time, only a little precision, processing speed, encoding and decoding is sacrificed. Moreover the natural sequential format of the source data is preserved.

2.2.3 Strip Trees

In this representation, a curve (C) is first approximated by an open polygonal line given by an ordered list of discrete points P_0, $P_1, \ldots,$ P_y for some resolution (w_i) [Nort82, Ball81]. The curve is then enclosed by rectangular *strips*, each defined as a six tuple x_i, y_i, x_j, y_j, w_k, w_n where (x_i, y_i) is the start (P_0), (x_j, y_j) is the end (P_y) and (w_k, w_n) denote the left and right extent of the curve from the directed line segment (L) joining the start (P_0) and the end (P_y) [Ball81]. The width (w) of the strip is defined as

$w=(w_k+w_n)$ [Figure 2.2a].

A smallest rectangle, with sides parallel to the line segment (L), passing through P_0 and P_y, and covering all the points in that curve is obtained. This is defined as the root of the tree. Now, a point P_n is chosen which touches one of the sides of the rectangle that is parallel to L. Two line-segments,

$$(P_0, P_1, \ldots, P_n) \text{ and } (P_n, \ldots, P_y);$$

where y > z > 1, are obtained. The process is repeated for the two line segments and terminates when all strips have width $w \leq w_i$ [Ball81]. Figure 2.2b shows two levels of recursion for the strip tree representation of a curve.

A Strip tree is a convenient data format to represent a two dimensional curve. A curve may be represented at different resolution levels. Operations such as:

1. find the length of a curve,

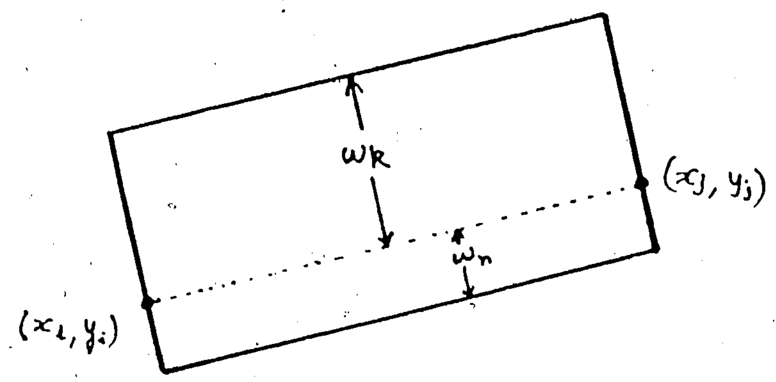


Figure 2.2.a:

Strip-tree defined by six tuple x_i, y_i, x_j, y_j, w_k, w_n.

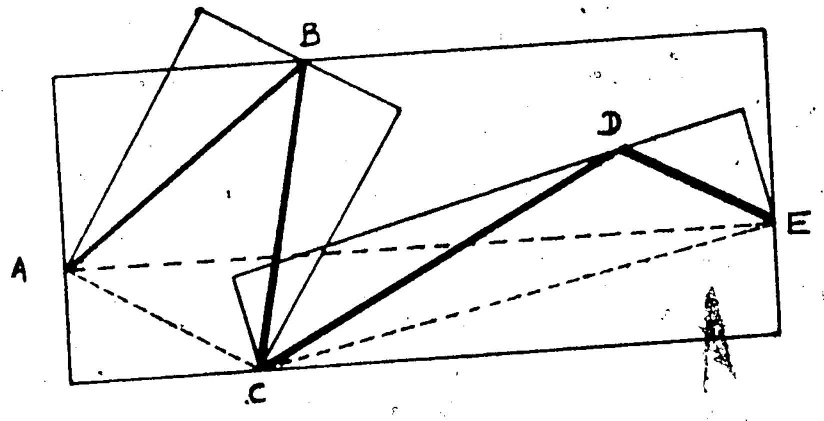


Figure 2.2.b:

A curve (A-B-C-D-E), containing five points has been coded into strip-tree data format.

Figure 2.2:

Strip tree and two level of recursion shown in Figrure 2.2.a and 2.2.b.

2. display a curve at different resolutions,

3. determine whether the point is inside the curve,

can be performed efficiently. If n is the number of points, then the memory required to represent the curve by strip trees is roughly 4n. Search for a point can be of Θ(log n) for a balanced strip tree and is linear in the worst case.

2.3 REGIONS

A region may be represented by a set of consecutive points defining a closed curve. This section deals with the discussion of data organizations which has been suggested for representing 'region' information of an image. It should be noted that these data organizations are capable of defining a point or a line as a special case of a *region*. Such representations are not efficient and are therefore not considered to be viable, except in extreme cases. In following sections, the discussion is mainly restricted to regions having area.

2.3.1 Tightly Closed Boundaries

Tightly Closed Boundaries or TCB were suggested by [Merr73] for representing basic pictorial features of contour maps, region coverage and line structures. For a region whose resolution is defined by a rectangular grid, a tightly closed boundary must first satisfy the following conditions:

1. The absolute distance between successive nodes on

the grid is not greater than the diagonal distance between nodes.

2. Sometimes, horizontal or vertical lines of the grid intersect the boundary tangentially. The points of intersection are called local maxima or minima. Every such point is augmented so that the test line always passes through an even number of boundary points at each extremum [Merr73].
3. Lastly, closed boundaries should not loop back on itself.

The data format is capable of defining an arbitrary region, the resolution of which depends on the grid size used while digitizing. Further, all the points obtained after digitization are sorted into ordered sets of points with the same y coordinate. In these ordered sets, the x value of all the points increases monotonically. The ordered sets are defined as y-partitions of TCB. Now, TCB (T_i) of region R_i can be defined as follows [Merr73]:

$$T_i = y_j,\ y_k,\ S_1,\ S_2, \ldots,\ S_n;$$

where y_j and y_k are the smallest and the largest y coordinate respectively and the number of partitions (n) has the value $n=(y_j-y_k+1)$. The Y-partitions (S_y) of the TCB are denoted as follows:

$$S_y = r_y,\ x_1,\ x_2, \ldots,\ x_z.$$

where r_y is the number of x coordinates and is always an even number and x_i, where $1 \le i \le z$, denotes the x value of the points with the y_y coordinate of the image. Apart from storing the above points, the extreme x values, i.e., x_j and x_k of the TCB, are determined while the above mentioned preprocessing is being performed. These x values, the y_j and y_k values define a rectangular area termed as the *Geometric Index* of the TCB. Also included is a header containing the displacement of each y-partition set stored. The header contains pointers to the starting location of each of the sets. A region with 'n' y-partitions will have n such entries. Figure 2.2.c shows an m by n rectangular area containing a closed curve. The points of intersection of the curve and each of the n horizontal scan results in 'n' Y-partitions. In this case, some of the Y-partitions contains no point or 2 points of intersection. In each Y-partition, only x-coordinates of the points of intersection are stored. Minimum and maximum value of the x-coordinate is followed by remaining x-coordinates of the points of intersection in that scan. As earlier explained, all points of minima and maxima are considered twice. As shown in Figure 2.2.c, n_1 Y-partitions contain 4 points of intersection.

It has been shown [Merr73] that this data structure is suitable for handling queries like:

1. Is the point inside the polygon?
2. Compute the area enclosed by the polygon.
3. Find maximum distance of a point from the boundary.

The representation provides a rapid way of searching a large file and is suitable for contour maps, line structure and

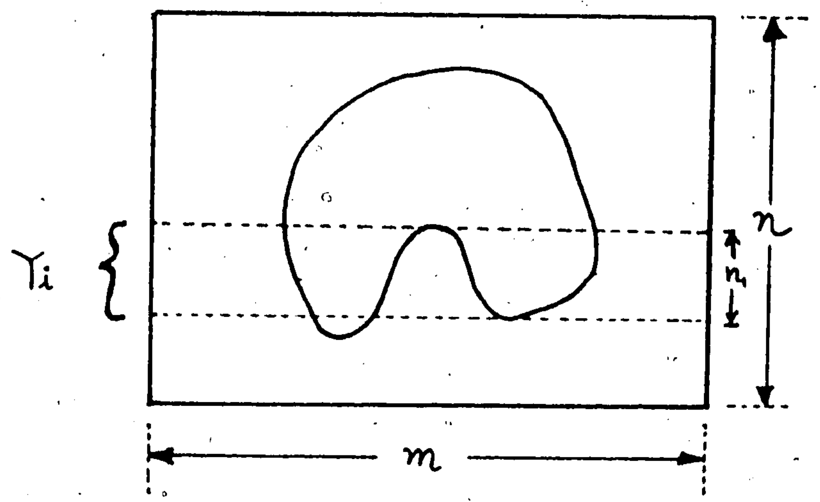


Figure 2.2.c:

Shows the concept of TCB representation for a closed curve enclosed by a m by n rectangle.

region or area coverage. The data structure also possesses very useful algebraic and geometric properties which lead to efficient representation of the relationship and attribute values of pictorial features. It must be noted that this representation is less efficient than CCS in terms of the storage requirement, but is more convenient to area oriented problems. Thus, CCS would be preferred in applications where the shape of the line structure is of interest instead of the area enclosed by it. A terrain Information Handling System has been implemented containing contour maps, line and curves using this data representation [Merr73].

2.3.2 Skeletons

This data format provides a compact representation of an image. The concept is based on the *principle of maximal neighborhood* [Rose67]. Let $P_1(i_1, j_1)$ and $P_2(i_2, j_2)$ be two points and $d(P_1, P_2) = |i_1 - i_2| + |j_1 - j_2|$. It is easy to verify that following properties will hold true:

1 $d(P_1, P_2) > 0$ iff $P_1 \neq P_2$,

2 $d(P_1, P_2) = 0$ iff $P_1 = P_2$,

2 $d(P_1, P_2) = d(P_2, P_1)$, and

3 $d(P_1, P_2) \leq d(P_1, P_3) + d(P_2, P_3)$;

for all the points P_1, P_2, P_3 [Rose67].

If r is a non-negative integer then a point P_0 with radius r as the diagonal distance of a square region would contain a few other points of the image. Let this set be defined as S_0. All the points P in this set satisfy the

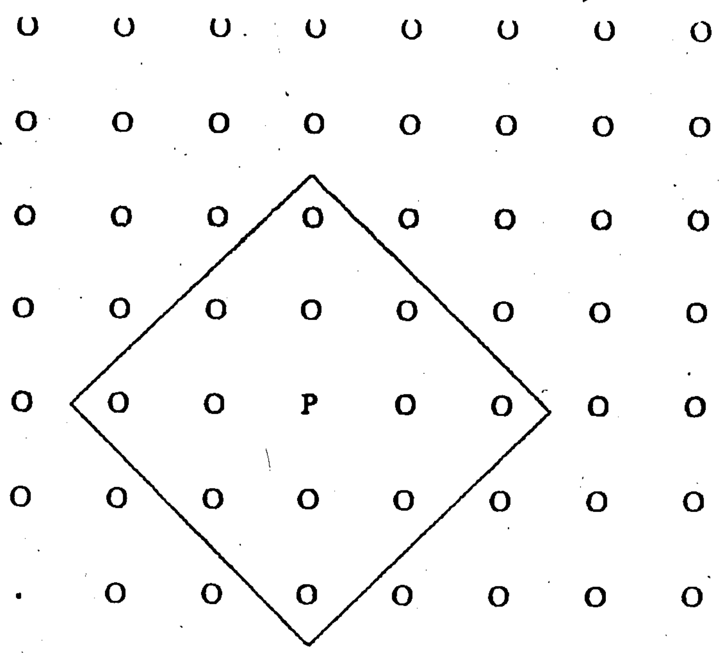


Figure 2.3:

Neighbourhood of point P with radius 2.

[refer section 2.3.2].

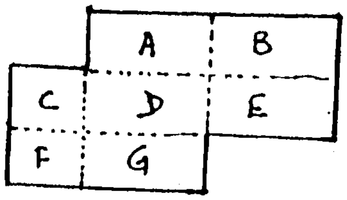
property that $d(P_0, P) \le r$ [Figure 2.3]. Clearly then set S_0 is a set of points centered on the point P_0 encompassed by a square with sides $(r+1)$ units. If $r = 0$ then set S_0 is a null set and defines the point P_0 itself.

Now, let R be a region, a subset of the image A. Then the *neighborhood* of a point P_0 with radius r is defined as set S_0. This concept of neighborhood is extended to define the whole region occupied by the picture in two dimensions. It is easy to see that some neighborhood of any point P in the picture would always be contained in some R. Let Z_i be the set of neighborhoods of all the points in R. All these neighborhoods are contained in R. Now, the region R can be defined in terms of union of the neighborhoods, i.e.,

$$R = \text{union } i, \text{ where } i \in Z_i.$$

Maximal neighborhood is defined as a neighborhood in Z_i which is not properly contained in any other neighborhood. For example, if N_1 and N_2 are neighborhoods and all points in N_1 are also present in N_2 then N_1 is not a maximal neighborhood. Now, let MZ_i denote the set of maximal neighborhood for the region R, then by definition it follows that $R = \text{union } mr$, where $mr \in MZ_i$.

A neighborhood may be defined by the radius and the center. Thus a region R may be completely defined by its centers and radii of maximal neighborhoods. This method of representation of an image has been defined as *the Skeleton*. This is because set MZ_i may be visualized as a skeleton of region R. It has been shown that skeletons are comparable to



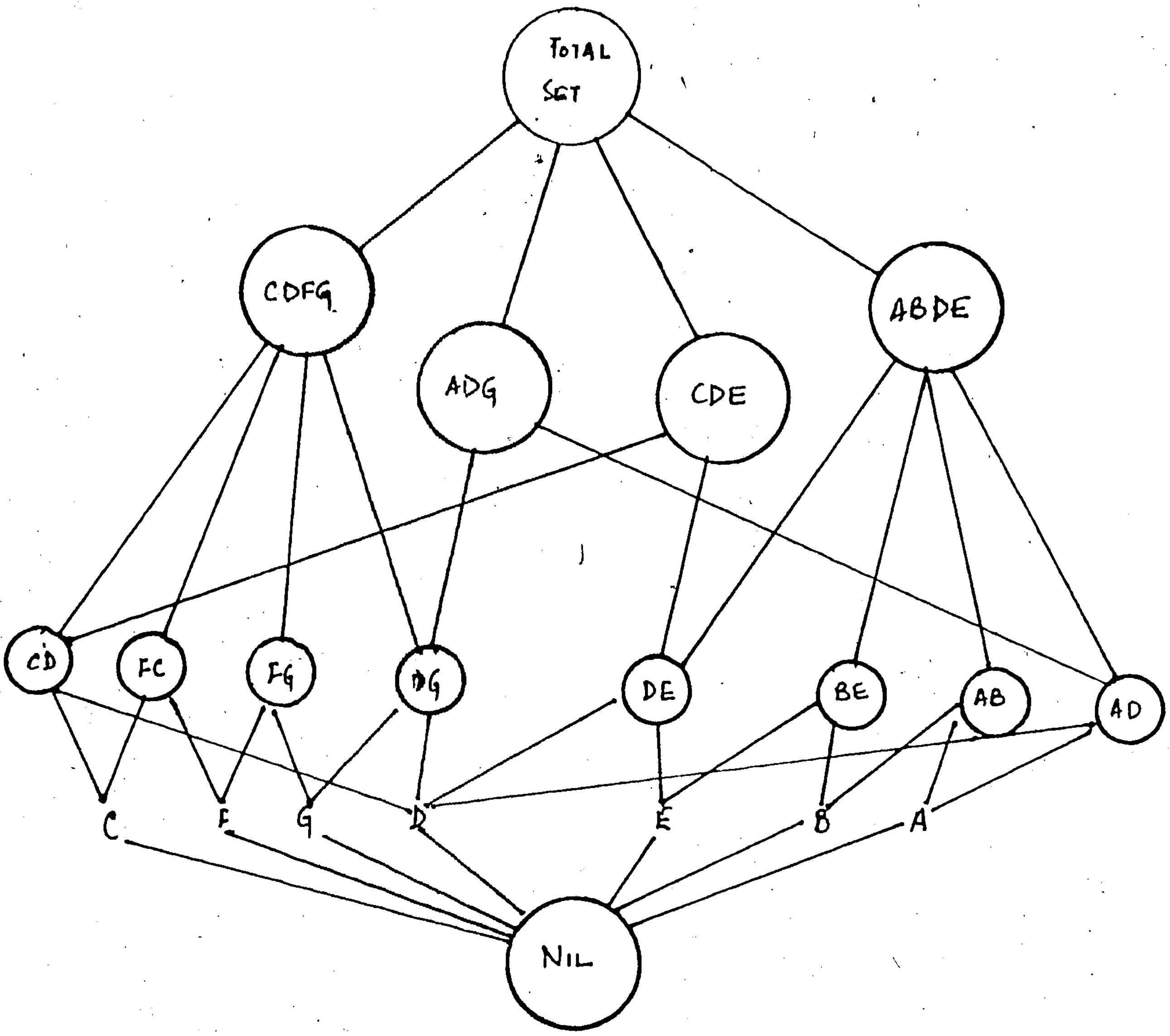


Figure 2.4:

Shows adjancency structure for the domains A, B, C, D, E, F and G as shown in the top figure.

chain codes in terms of memory requirements [Rose67]. They are particularly suitable for certain region queries such as those ascertaining if a given point is inside the region or not. Set-theoretic operations, like finding the intersection and union of two regions, can also be performed easily.

2.3.3 Adjacency structures

This structure uses the idea of regular decomposition of the picture for the first time [East70]. In this data structure, the picture is subdivided into rectangular domains, each domain's south, north, west and east near neighbors are also maintained. The decomposition continues till the near neighbors become nil or the smallest grid size is obtained. Figure 2.4 shows two rectangular overlapping regions and its adjacency representation.

It must be pointed out that four search pointers are maintained, so there is an increase in the storage requirement. Since an array is used to represent the adjacency structure so the term *variable array* is sometimes used. In comparison to a binary array this representation is efficient.

2.3.4 CIAD data format

The Communication and Display for Interactive Access to Distributed Spatial Databases or CIAD project [Davi82], being developed in University of Alberta, uses a linear data format for representing points, lines and regions. The data representation is divided into four subsection as follows:

1. The header contains information about the creation date, the maximum and minimum latitude and longitude (defining an 'enclosing rectangle') and pointers to the starting location of the point, line and region files [Davi82].
2. The point file consists of a set of unique points in latitude and longitude,
3. The line file consist of lines or curves defined by a set of connected points.
4. The region file that defines closed polygons by specifying the lines that define the polygon.

Because of the linear format insertion, deletion and search for a point, line or polygon is $\Theta(N)$ where N is the number of points and lines. The storage required is dependent on the number of points, lines and polygons in the image.

2.3.5 Boundary Network

A closed curve represents a region and therefore a region can be represented by a set of starting points, end points and an ordered set of intermediate points [Loom65]. The region enclosed by the closed curve can be defined as its domain. Thus an image may be covered by a set of regions which are disjoint except for the common boundary points. The inclusion property results in a tree data structure. Each node represents a boundary set in the tree and points to other boundary set(s) o a null set. Thus the son nodes are enclosed by the father nodes in the tree. This allows the following queries to be efficiently answered:

1. Does the straight line connecting any two points lie inside a region? If not then what are the intersection points?
2. Is the point inside the region?

This data structure allows easy insertions and deletions to reflect the boundary changes. The representation has been used to numerically control a robot's arm or a machine tool by computer generated paths. It should be observed that an open curve is approximated by extrapolation and may result in erroneous results.

2.3.6 Binary Searchable Polygonal Representation

In this case, a polyline is said to be a curve defined by a set of consecutive points joined by straight lines, and a polygon is considered to be a polygonal line with identical end points [Burt77]. A *section* is defined as set of consecutive sides and a *particular section* is defined as a set containing only a single side. Further, a *simple section* is a set of consecutive points with monotonic : and y values and a *significant point* of a polygonal line is defined as a local extreme in either x or y coordinate values or both. These points are said to be partitioning the line into *basic sections*. The Binary Searchable Polygonal Representation or BSPR has the following properties:

1. Any *section* is the union of two other sections which

are disjoint except for the end points. Only a *particular section* do not obey the above property [Burt77].

2. The location of the starting or end points of all the section are known.
3. The maximum and minimum of the x and y coordinate of a section are known. These four values define a rectangle called *section rectangle*.

The use of binary search in this data format enables a rapid search for finding point(s) or region(s) of interest.

2.3.7 The RSE structure

This scheme is the outcome of the problem of representation and control in the construction of visual modes. Suggested by Hanson and Riseman [Shap79], it contains minimum information that must be present in any symbolic representation of an image.

The data format contains three layers namely the region plane, the segment plane and the end-point plane. The region plane consists of the nodes containing region information which are defined by set of line segments. The segment plane consists of line segments forming the boundary of the region. The end-point plane contains nodes representing end points of the segment. The name RSE is the abbreviation for these three planes.

The relationship is represented by a directed path and facilitates rapid searching and also results in a compact data representation scheme.

2.3.8 Multi-dimensional binary search trees

Queries involving associative retrieval by a secondary key are frequent in database applications. To facilitate retrieval by a secondary key, multi-dimensional trees (or k-d trees) were introduced [Bent75]. Since then they have been found useful in many of the database applications because of their efficiency in storage and time requirements. Above all, the data structure is capable of handling many types of queries quite efficiently, for example, $\Theta(n)$ insertion, $\Theta(n**(k-1/k))$ deletion and $\Theta(\log n)$ search is guaranteed.

A file consists of several records. Each record may be represented as a set of tuples:

$$V_0, V_1, \ldots, V_y;$$

where $y = k-1$, and V_i denotes the key or the attributes of the records in the file. In the k-d tree representation, where k is the number of attributes of the records in the file, each record is considered as a node. In addition to the k keys comprising the record, each node contains two pointers, which are either null or pointing to another record in the tree. A discriminator (i), stored with the record, is an integer and has the value between 1 and (k-1). For a node (P), the pointers to two sontrees are defined by low-son (P) and high-son (P) for the left and right subtree respectively and $V_i(P)$ defines the value of the i-th

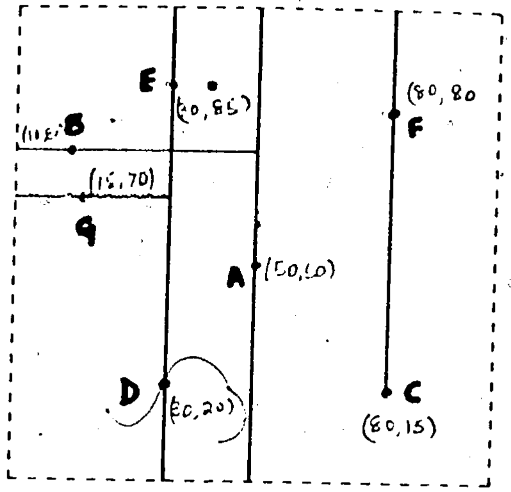


Figure 2.5:

Shows the concept of k-d tree representation. Points A, B, C, D, E, F and G has been arranged in 2-d representation as shown in figure 2.6.

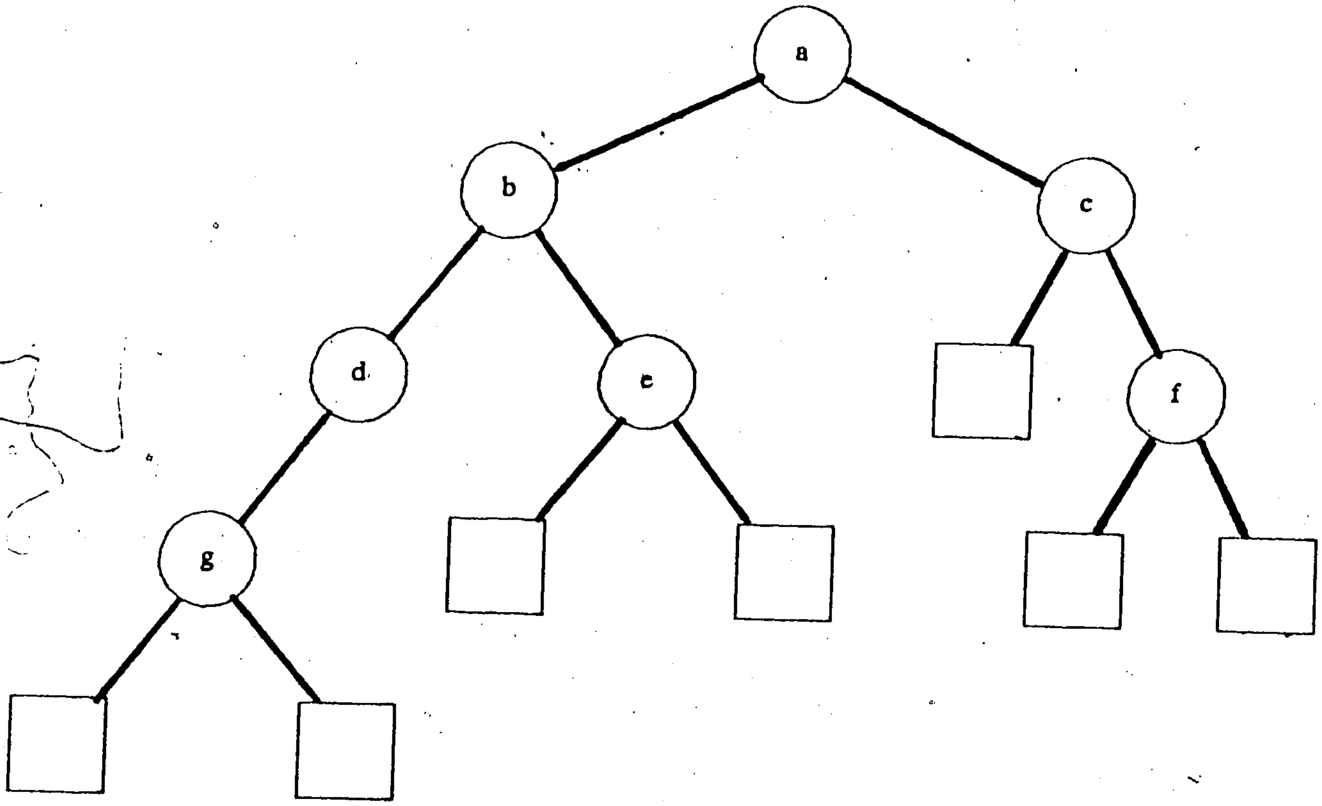


Figure 2.6:

Shows the 2-d tree for the points in the cartesian coordinates for the figure 2.5.

attribute of node P. Thus for any node P in the tree, if i is the discriminator of P then for any node Q in the low-son(P) it is true that $V_i(Q) \le V_i(P)$ and all the nodes R in the high-son(P) subtree satisfy $V_i(R) > V_i(P)$. All the node at any given level has the same discriminator. Further discriminator i is mod(k) function, i.e., if the discriminator of the present level has the value (k-1) then the next level would have the discriminator value of zero. If the two keys are equal then this case is resolved as described in [Bent75].

Figure 2.6 illustrates the concept of 2-d tree and corresponds to the Figure 2.5. The k-d data structure, developed to store k-dimensional data, has been popular because of their flexibility in allowing insertions and deletions. Some work has also been done in developing adaptable k-d data structures. They have gained popularity in various applications involving image databases, geographical information processing systems and speech recognition.

2.3.9 Hierarchical array

Stanford Research Institute came up with an idea of decomposing a rectangular region into four equal and disjoint rectangles [East70]. The recursion was only three levels down, i.e., a rectangular region was subdivided up to maximum of 64 smaller rectangles. Homogeneous rectangular regions were not subdivided. In comparison to the string representation [Section 2.4.2], this representation is efficient in storage requirement.

2.3.10 Quad-trees

Partitioning a picture into four subareas with the same shape and equal areas is called quadratization. This results in a new way of representing a digital picture by subareas instead of individual pixels. The resulting tree is called a quad-tree [Same83, Kling80, Hunt79, Rose79], where the leaves represent the regions of the picture. Each leaf is labeled with the color of the picture which it represents. Every node is associated with a square region of the picture. A root is associated with the entire picture.

Quadrant partitioning establishes a father-son [Figures 2.7 and 2.8] relationship among the parent and the successor quadrants. No parent node has the same color for all its descendents, because, if this is the case then all four children may be removed and the parent colored with the color of the children. This results in a more compact representation scheme, since each node in the tree can be named by its relative geographical location. The representation offers the following advantages:

1. It is easy to access small portions of a data base,
2. Elimination of the picture areas of no interest guarantees that only the useful picture segments are present.
3. No time is wasted in processing the waste data.

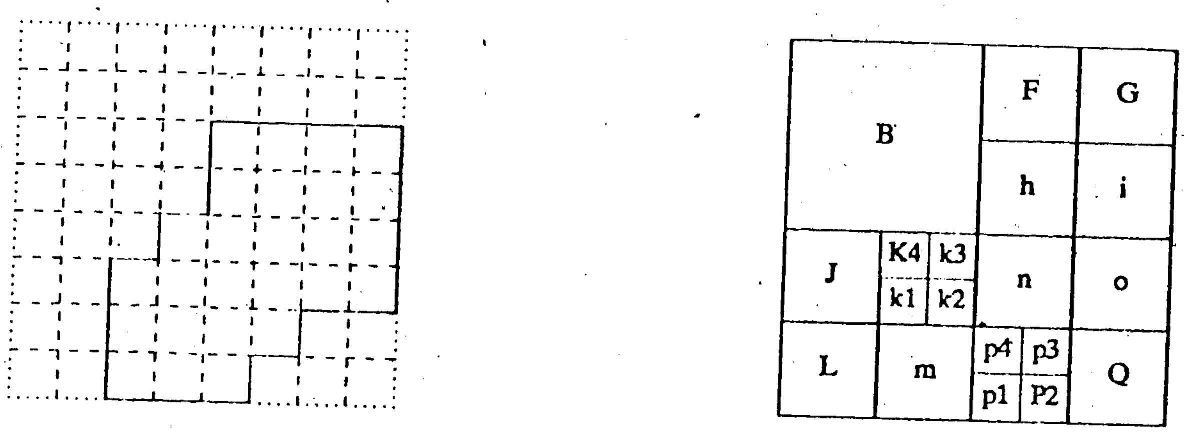


Figure 2.7:

Shows the quadratization of the contour in the left. Capital (small) alphabets has been used to denote regions outside (inside) the contour. the quad-tree has been shown in the figure 2.8.

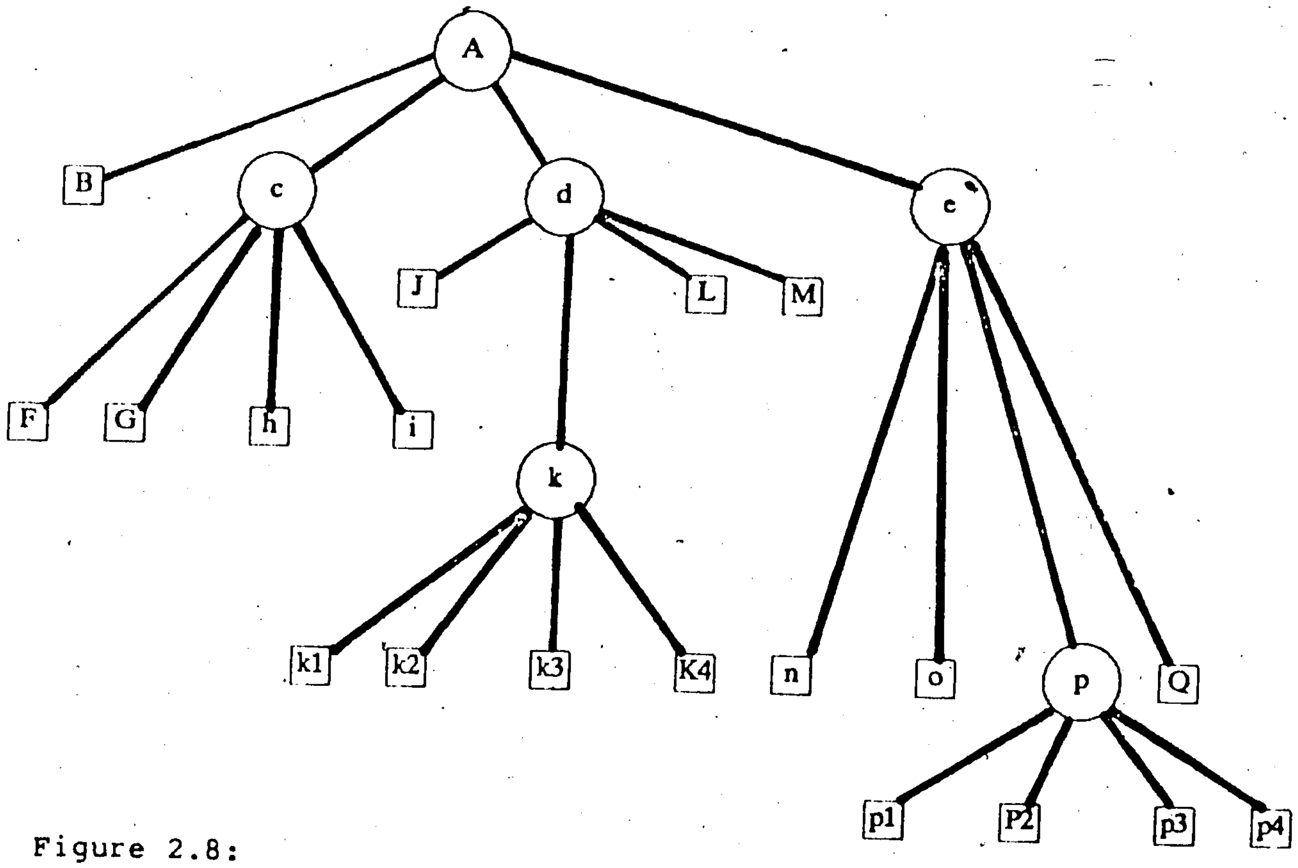


Figure 2.8:

Shows the quad-tree for the contour region in the the figure 2.7.

4. Access time is less in reaching portions of a picture to be processed.

The structure is particularly suitable for applications where fast access of information is necessary. Segmentation stops when empty nodes are reached. In the worst case it would mean that every 1 by 1 area is different from one-another, this results in an increase in the storage requirement. Because of the considerable storage requirement, the scheme would also be difficult to extend to representations colored maps.

In another consideration, picture data generated through raster scanning devices resulted in large sequential files. One of the techniques of organizing the data for faster access and less storage on magnetic tapes and disks employs conversion of the data to a quad-tree representation [Klin79].

2.3.10.1 Oct-tree data structure

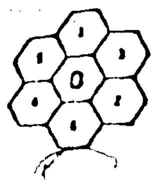
A natural extension of quad-trees for 3-D data representation is an Oct-tree. Here each node has 8 children representing disjoint cubes or subregions. Because the single primitive shape is a cube, every arbitrary object can be represented to the precision of the smallest cube. Moreover, a set of algorithms are sufficient for manipulation of the stored data.

An oct-tree representation has the severe disadvantage of requiring large amounts of storage. This would be useful in those applications where memory is not the only constraint. Because the algorithms do not require floating point operations, multiplications or divisions, they may be implemented in relatively low cost hardware. Each node generates up to 8 independent sub-calculations which suggests that large numbers of processors may be used to perform the operations in parallel, enhancing the speed in general.

2.3.11 Generalized Balanced Ternary representation

Hexagonal cells are used to define a region in the Generalized Balanced Ternary (GBT) scheme of data representation [Gibsa]. GBT is a tree data structure in which pixels are stored and processed using a hexagonal based tree data structure. At each level, cells are constructed by aggregating the cells of the previous lower level. The covering results in a uniform adjacency property, i.e., each cell, surrounded by six other cells, share exactly one-sixth of its boundary with each neighbor. The first level of aggregation is formed by taking a single hexagon and six near neighbors [Figures 2.9 and 2.10]. In general an aggregate at level n is obtained by taking a cell and it's six near neighbors. It has been shown that a planar covering of uniform adjacency is possible at each level [Gibsb] in the ternary tree.

The GBT addressing scheme has a '0' for the center cell and the six neighbors are defined to be '1' to '6'. The whole aggregate is defined by a number '7'. Thus, a ternary



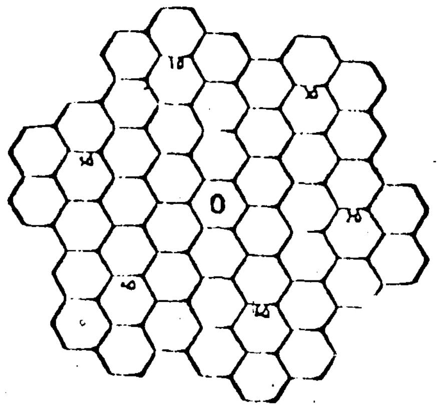


Figure 2.9: Shows the first and the second level aggregate for GBT representation.

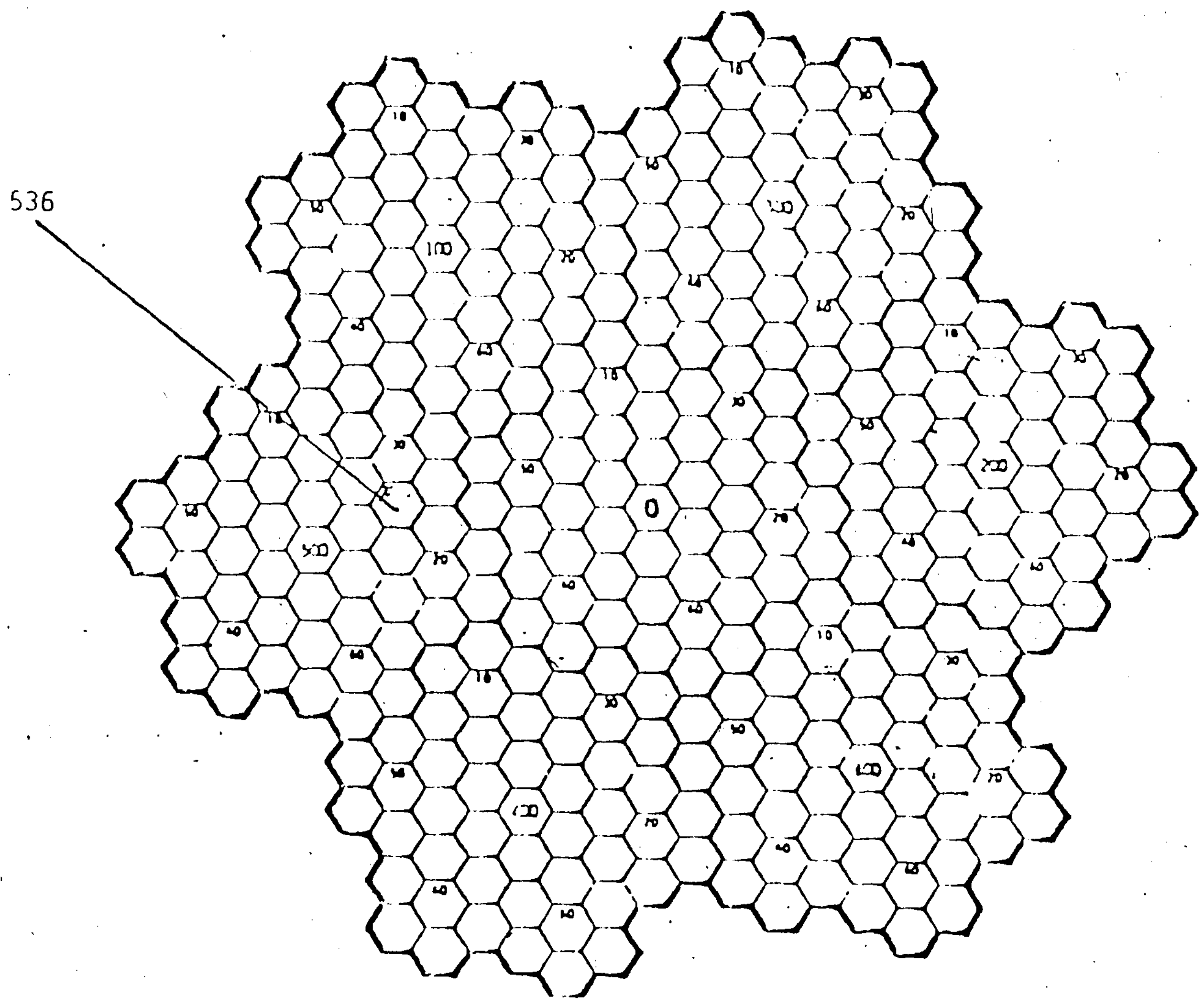


Figure 2.10:

Shows the GBT representation illustrating the first, second and third level of aggregate structure.

tree with k-levels would be addressed with a k digit number as follows:

$$d_n, \ldots, d_2, d_1;$$

where $0 \le d_i \le 7$, $1 \le i \le n$ and $n=(k-1)$. Each digit d_i corresponds to a digit in the i-th level. Figure 2.9b shows the cell addressed by the address 536.

The data structure was developed while performing raster-to-vector conversion. This data format was introduced by Dean Lucas and John Gibson and has been used in COMRAC information processing systems.

2.3.12 Pyramids

This data format is resolution based. To explain it more clearly let us assume A(0) defines a single pixel and A(n) to be a 2**nX2**n image array [Same83, Shap79]. Now, a pyramid can be defined as an exponentially tapering sequence such that A(i-1) is the representation of A(i) at half the resolution.

Pyramids are used for segmentation, feature detection and extraction since they can be used to limit the scope of the search. First coarse levels can be searched for an approximate region and then exhaustive search can be performed at lower levels to obtain finer details.

Reduction of the resolution means that at coarser levels the smaller objects are visually affected and the edges may tend to get smeared and the region separation may disappear. This seems to be a problem with pyramids.

2.4 IMAGE

This section deals with data formats for raster data. Some of the data structures belonging to this section have already been discussed [Sections 2.3.9, 2.3.10, 2.3.11 and 2.3.12]. The two data formats discussed in this section are *binary-array representation*, *run length encoding* and *string representation*.

2.4.1 Binary-array representation

The simplest type of raster organization is the square grid where individual raster cells or pixels are simply stored in sequences of ze s or ones signifying the absence or presence of an intersection of the grid and the contour [Peuq79, East70]. A binary array or sequential list may store the individual cells. The address of a pixel is implicit and can be determined by the location of the cell in the binary array or sequential list.

Because of the simple nature of the data representation, calculation of the perimeter of a polygon or area enclosed by it are simple counting problems in which the number of pixels defining the boundary or inside the region are to be determined. Windowing and the clipping algorithms involve the calculation of the locations corresponding to the diagonally opposite corners (i.e., top right and bottom left corners for a left-to-right horizontal scan of the image) of the rectangular window in the binary array and then selecting only the pixels in between the two locations.

The nature of the data format is grid size dependent and a larger grid size may leave out certain details of the curve. Since both the presence and absence of the intersection point is to be stored, the data format requires a large memory or secondary storage.

2.4.2 Run length encoding

While horizontally scanning an image, consecutive pixels with equal intensity values may be obtained [Pavl82]. For a horizontal scan of n pixels, let 'i' consecutive pixels have the intensity 'd'. In this case, instead of storing the intensity of every pixel, the value of 'i' and 'd' is stored to denote that a sub-portion of the horizontal scan contains 'i' pixels with intensity 'd'. This technique is called *Run Length Coding.* In this representation, a horizontal scan of n pixels may be broken into sequence of vectors:

$$\ldots, P_i, d_i, P_k, \ldots;$$

where P_i and P_k denote the coordinates of the two end points for some portion of the horizontal scan with pixels of intensity d_i. Alternatively, this information can also be stored as:

$$\ldots, n_i,\ d_i,\ n_k,\ d_k, \ldots;$$

In this case n_i, d_i represents a portion of the horizontal scan with n_i consecutive pixels of intensity d_i.

In the case of binary images, it is suff··ient to indicate the value of the first pixel (0 or 1) and then the number of occurrences of each value, since they must alternate:

$$0, n_1, n_2, n_3, \ldots$$

2.4.3 String representation

As indicated above, binary arrays have a uniform grid size and can be visualized as representing a set of uniform domains [East70]. It is possible to vary the grid size while traversing the image horizontally. This can be done as follows: each object, say O_i, may have some dimension 'd_i' and can be represented as $O_i d_i$. Thus a horizontal scan can be defined by a string of patterns which look like:

$$(O_1 d_1,\ O_2 d_2,\ \ldots,\ O_i d_i,\ \ldots,\ O_k d_k);$$

where open and close brackets signify the start and end of the scan and k is the number of objects which were found on that horizontal scan. There may be a few horizontal scans which will not have any objects resulting in a blank scan. These blank scans may be skipped. In the string representation, this is done by specifying the vertical dimension of the i-th scan, say v_i, before the open bracket. Thus a pattern of characters such as:

$$\ldots v_i(\ \ldots,\ O_j d_j,\ \ldots)\ v_k(\ \ldots\ ;$$

would result. In this representation the address is not implicit and therefore search for a point is complicated. The representation gives a compact encoding of the picture with rectangular features, e.g., a floor plan of a building. Because of the large search required to answer a query such as 'draw only object O_k', this representation is highly application dependent [East70].

2.5 Summary

Since early seventies, definite progress has been made towards finding efficient data formats for representing spatial data. Earlier, emphasis was more on representations facilitating hard copy output [Nagy79] and data structures were simpler in nature [Sections 2.2.3, 2.3.9, 2.4.1, 2.4.2]. Since then the emphasis has shifted towards data formats providing better access time and faster basic operations [Section 1.1]. For this reason, preprocessing of the spatial data is invariably done [Sections 2.3.2, 2.3.2, 2.3.6, 2.3.8].

In recent years, tree data structures [Sections 2.3.10, 2.3.10.1, 2.3.11, 2.3.12, 2.3.13] and vector data organizations [Sections 2.2.1, 2.2.2, 2.2.3, 2.3.7] have gained popularity. Various data formats have been developed. As time passes, alternative representations will undoubtedly be developed. Recent work is towards developing hybrid data structures, e.g., point quad-trees and edge quad-trees [Same83]. One such data structure is suggested in the following chapter.

CHAPTER 3

A NEW DATA STRUCTURE

In this chapter, two fundamental properties of hierarchical and hybrid data structures have been highlighted. Later a new method of subdividing a region, along with the description of the basic operations such as deleting, inserting and updating a point or a line, are discussed. This data format is a tree data structure, which can be obtained from the image files containing points and lines. In other words, a vector organization can be used to generate the new tree data structure.

3.1 Recursive partitioning of a region

As suggested in [Same83], a data structure based on the principle of recursive decomposition, should satisfy the following properties:

1. The partition of a region should be infinitely [illegible]ive, and,
2. [illegible]tition should be such that the regions are infinitely decomposable into increasingly finer patterns.

It has been shown that only hexagonal, triangular and rectangular tessellations satisfy the above mentioned property 1. Moreover, only triangular and rectangular subdivisions satisfy both the properties. The triangular, rectangular and hexagonal regions have been considered as terminating partitions [Nort82, Gibsa, Same83]. In the following section, with the help of rectangular and hexagonal terminating partitions, a new method of subdividing an octagonal region has been described. The data structure satisfies both of the above mentioned properties.

3.2 Octagonal Tree Data Structures (OTDSs)

A region R, consisting of a set (N) of n points and a set (L) of q lines, can be covered by an octagon. The octagonal region can be recursively divided into octagons with the help of rectangular and hexagonal regions [Figure 3.1]. The resulting data format is termed as an Octagonal Tree Data Structure (or OTDS). The rectangular and hexagonal regions need not be divided recursively into set of four quadrants or regular hexagons respectively. Instead the emphasis is to find the terminating nodes. This is done by recursively subdividing the rectangular region into two rectangular partitions with equal points until the terminating partitions are found [Section 3.4]. Hexagonal regions are subdivided into two rectangular regions and two hexagonal regions such that no region contains points more than half the points of that hexagonal region [Figures 3.3

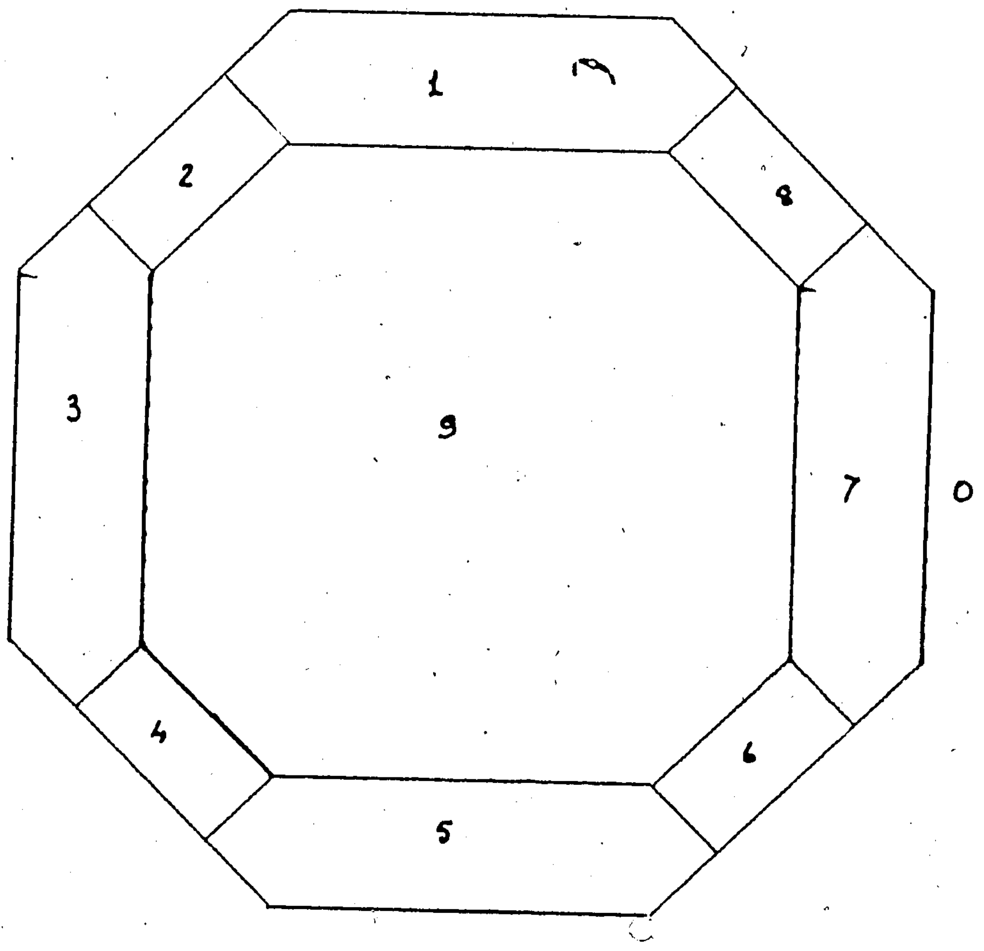


Figure 3.1:

Shows the basic sub-regions of an octagonal region into four hexagons, four rectangles and one octagon.

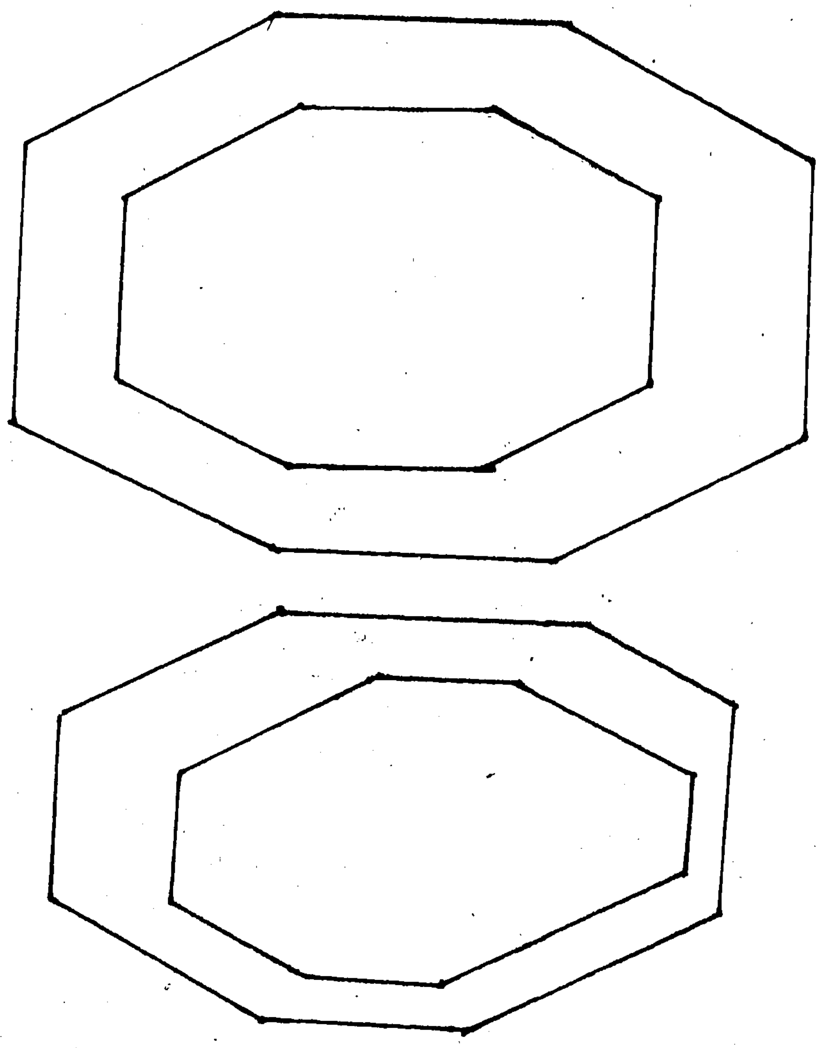


Figure 3.2:

Shows case 1 and 2 of Section 3.4. For case 1, all the sides and angles, of any octagon, are of same dimension. The side-lengths vary in case 2. Whereas, in case 3 (see Figure 3.7.a) the angles may also be different.

and 3.4]. The rectangles, resulting from the recursive hexagonal subdivision, are further subdivided as already explained.

Let a subdivision of an octagonal region be such that 9 sub-regions are obtained:

1. Four rectangular regions, termed as Lateral Rectangular Regions (LRRs),
2. Four hexagonal regions called as Lateral Hexagonal Regions (LHRs), and
3. One Vertical Octagonal Region (VOR).

Both LRR and LHR are further subdivided (as explained above) till terminating regions containing at most one point are obtained. A VOR terminates when only one point is left after recursive subdivision. These terminating LRRs, LHRs and VOR are termed as Minimum Region (MR) nodes or terminating partitions in the tree. The line-segments, corresponding to the lines (q) in the line file (L), are stored in a linked list (called line-list), as will be explained in Section 3.5.

By definition it follows that any region consisting of lines and/or points may be coded into octagonal tree data structure [Section 3.4]. As in the case of binary tree, the height (H) of the tree would be at most $\lceil \log n \rceil$. This is because each terminating region (i.e., a hexagonal, a rectangular or an octagonal region) is obtained by dividing any region into sub-regions such that each sub-region contains at most half the points.

3.3 Basic configurations of OTDSs

The subdivision of an octagon may result in following three possibilities:

1. Case 1: Symmetric Division: Octagonal regins can be such that all the sides and angles are equal [Figure 3.2]. In this case, a side length and a reference point can uniquely define an octagon.
2. Case 2: Symmetric Division with unequal sides: In this case, all the angles of the octagon are equal but not the side lengths. At most 7 side lengths and a reference point would be needed to define an octagon [Figure 3.2].
3. Case 3: Asymmetric Division with unequal sides: All the angles and the side lengths of the octagon vary. In this case, 7 sides and angles along with a reference point should be stored for defining an octagon [Figure 3.8].

It should be noted that, for all the 3 cases above, an octagon can be defined by specifying the coordinates of the 7 points of the octagon.

The rectangular region in the tree can be defined by specifying the coordinates of the two diagonally opposite points of the rectangle. For a hexagonal region, as shown in Figure 3.4, two point (M and N) would be needed to define the subdivision of the hexagonal region.

3.4 Conversion from vector format

In this section, an algorithm is presented to obtain octagonal data format from the vector representation. Let N denote a set of n points and L be a set of q lines in a two dimensional plane. The following is the algorithm:

1. Define an octagon around the whole image such that all the given points are inclosed.
2. Sort the points in x and y direction.
3. Select the point P which is the center of all the points.
4. Define an octagon (P) such that 4 rectangles, 4 hexagons and 1 octagon are obtained such that none of the sub-regions has more than half the points in P.
5. The other rectangular and hexagonal regions, obtained with the help of the already defined octagons, can be recursively divided into rectangular and hexagonal regions as follows:

   a. Divide the po. ts in the rectangular region into two subsections with equal points. Recursively continue till resulting rectangular regions contain only one point.

   b. Divide a hexagonal region into two hexagonal and two rectangular region [Figure 3.4] such that no sub-region has more than half the points. Divide the rectangular region as done in the above step. Recursively subdivide the resulting hexagonal regions till terminating nodes with only one point result.

c. If a line from one terminating region to another terminating region exists then add a pointer to the linked-list of both the terminating regions. The pointer points to an entry in the linked list for the q lines (called line-list) [Section 3.5, Figures 3.6 and 3.7]. The line-list also has two pointers, pointing back to the two terminating regions.

6. Recursively subdivide the octagonal region (repeat steps 4 and 5 above with half the points) till the center point P is the only point included in the octagon.
7. stop.

The algorithm terminates with at least n terminating regions. Each terminating node in the tree contains linked list pointing to the line-list entries [Section 3.7]. In the worst case, since n^2 pointers and lines are to be stored, hence memory required for implementation of the data-structure would be $\Theta(n^2)$ [Section 4.1.4].

3.5 Data organization for performing basic operations

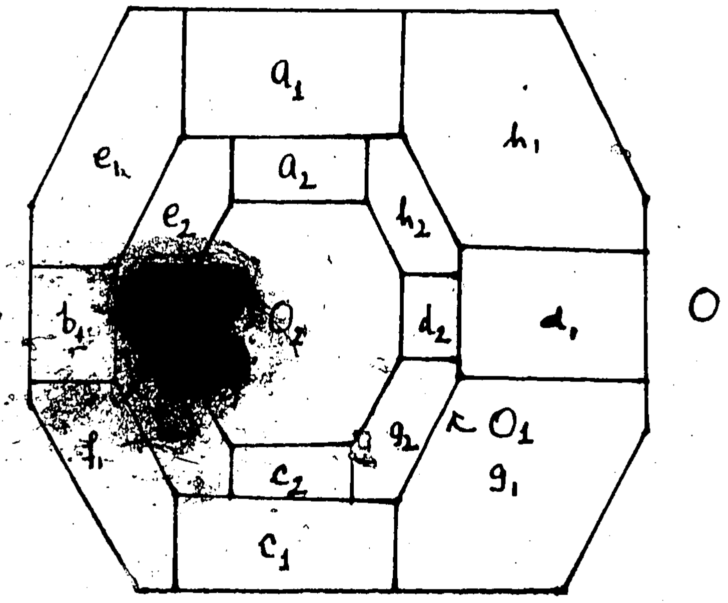
It should be noted that the line and point files of CIAD line data format directly corresponds to the set N and L and therefore this representation can be developed from the CIAD data format[1]. A point may be attached to (n-1) other points. This information can be maintained by keeping doubly linked list for every point in the terminating partition. Each element of this link list may point to the element(s) of another doubly linked list, which is the list of (q) lines in the line file (L). Let this link list be called as line-list [Figures 3.6 and 3.7]. Each element of the line-list has two pointers pointing to the terminating partitions in the octagonal tree, denoting the existence of a line between the points defined by those terminating node.

3.5.1 Neighbor finding technique

Often many applications require the examination of some nodes which are adjacent to the node being processed. In this representation, this reduces to a tree traversal first to the immediate common ancestor for the common side and than traversing the tree down to locate the neighboring region. For example, for the tree resulted by an octagonal subdivision of the region as shown in Figure 3.5, the near neighbor along the side CD (for node A) can be found by traversing up the tree to node O through node O_1 and then traverse down the tree to find the desired node B.

---

[1] The CIAD line and point files can be obtained from World Data Bank II and Edmonton MAP Data Bank [Appendix I and II].



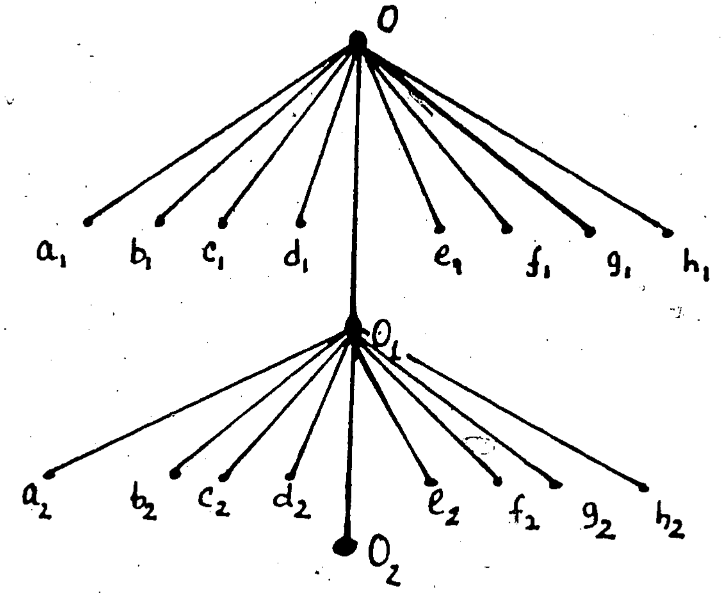
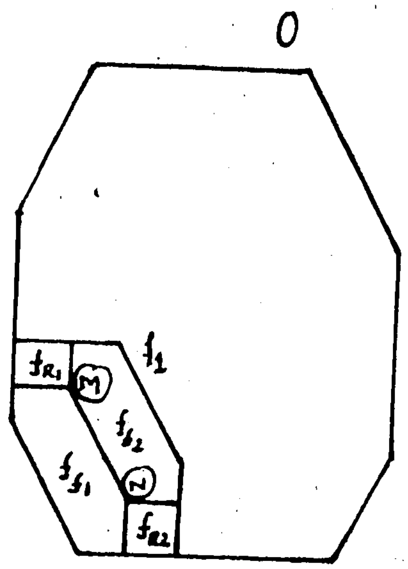


Figure 3.3:

Shows the two levels of an octagonal subdivision of a region and the corresponding tree structure.



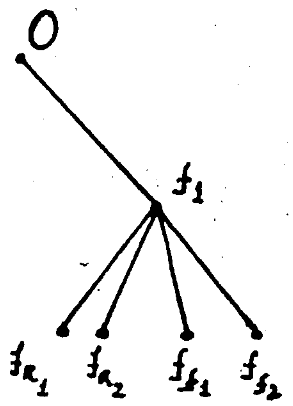


Figure 3.4:

Fragmenting the hexagonal region into two rectangles and two hexagonals and the corresponding tree (also refer Figure 3.3 and section 3.3).

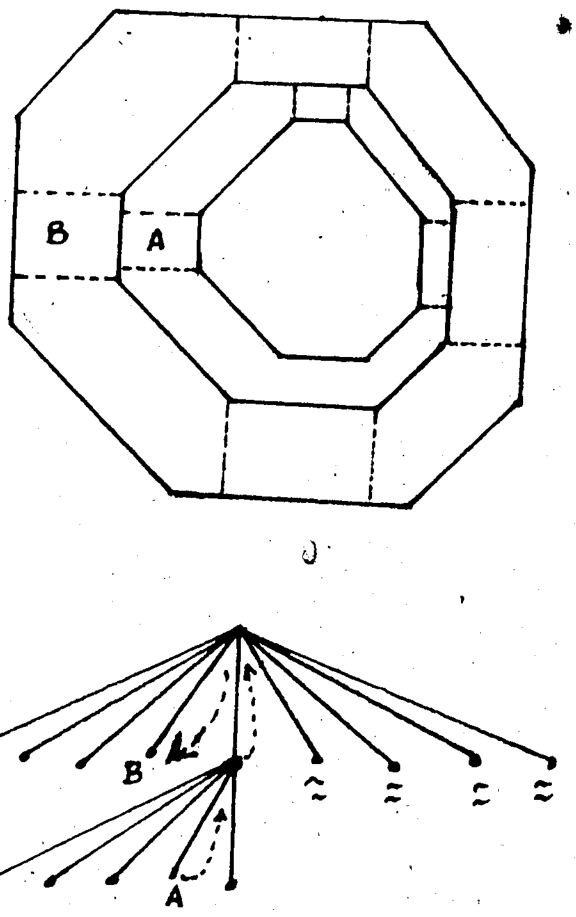


Figure 3.5:

Shows the neighbor finding technique (Section 3.5) in the octagonal representation. Region A's near neighbor 'B' is found out by traversing the common ancestor and then going down to find region 'B'.

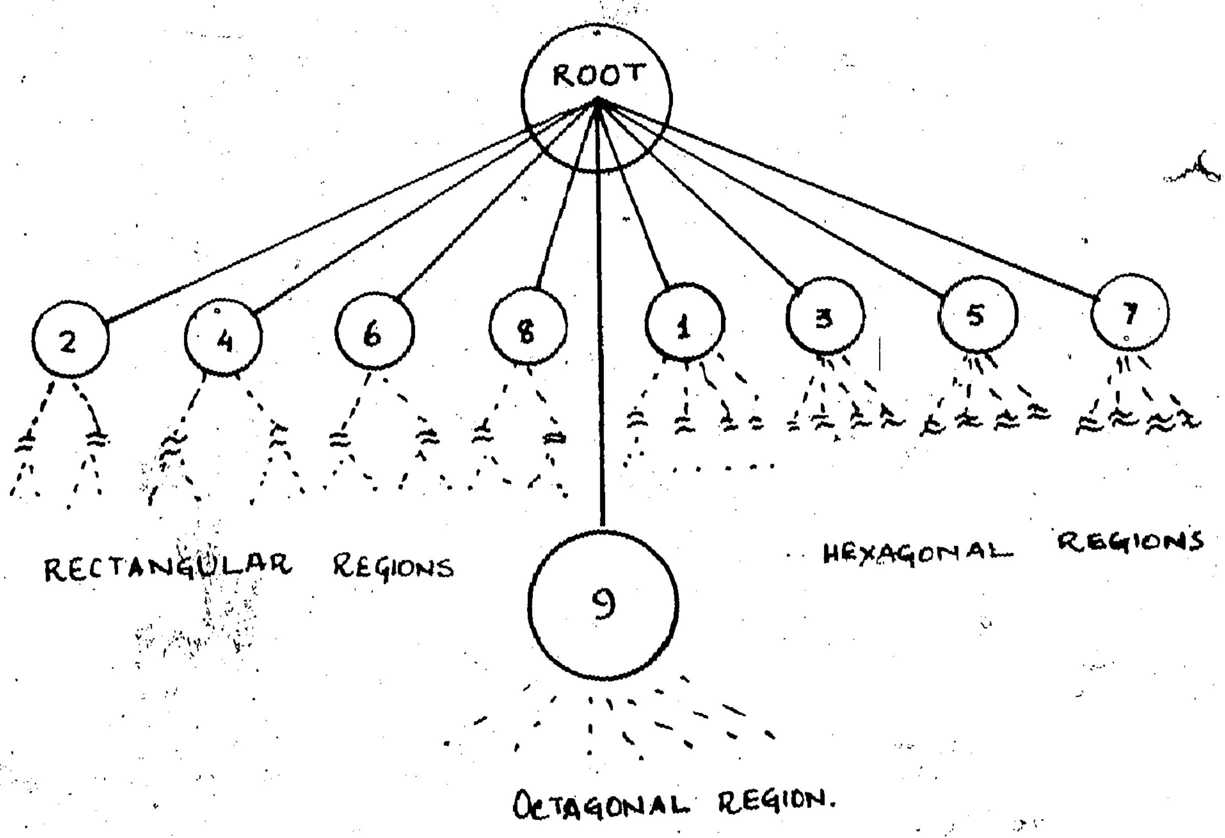


Figure 3.6:

The OTDS representation with the terminating partitions as the leaf-nodes. The leaf-nodes can be a link list which points to the line-list as shown in Figure 3.7.

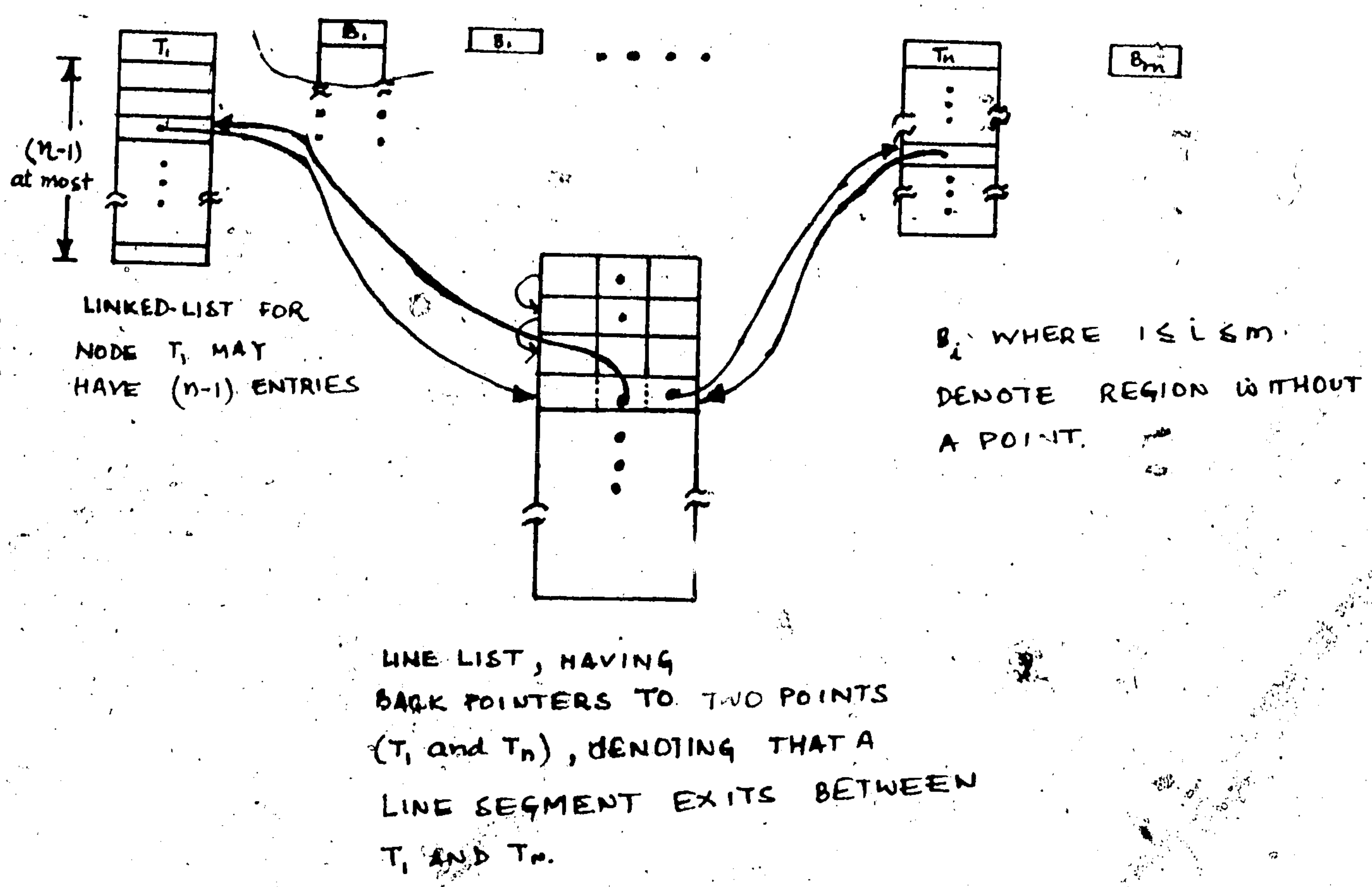

T_1
B_1
B_1
T_n
B_m
(n-1) at most
LINKED-LIST FOR NODE T_1 MAY HAVE (n-1) ENTRIES
B_i WHERE $1 \leq i \leq m$ DENOTE REGION WITHOUT A POINT.
LINE LIST, HAVING BACK POINTERS TO TWO POINTS (T_1 and T_n), DENOTING THAT A LINE SEGMENT EXITS BETWEEN T_1 AND T_n.

Figure 3.7:

Shows the memory organization for an OTDS, every terminating partition has a link-list pointing to the entries in the line-list, which in turn points to some other terminating partition indicating that the two terminating partitions are connected by a line-segment (refer Section 3.5).

3.5.2 Search for a point or a line

Search for a point is a simple case of tree traversal. This at most would take $\Theta(\log n)$ time. A line search can be performed by searching for the two end points of the line. Again it would be done in $\Theta(\log n)$ time [Figure 3.6].

3.5.3 Deletion of a point or a line

A point (P) can be deleted by finding the terminating partition (say T_1) corresponding to the point P. Let point P be connected to a point in some terminating partition T_n. One of the entry in the link list for T_1 would point to the some entry in the line-list. This entry in the line-list, in turn, points to an entry in the link list for the terminating partition T_n [Figure 3.7]. This entry in the link list for T_n points back to the same entry in the line-list. Now, a line segment between T_1 and T_n can be deleted in linear time. This is because deletion of a line-segment corresponds to deleting an entry in the line-list and link lists for the terminating partitions T_1 and T_n.

If the point (P) was connected to k other points then k line-segments would have to be deleted. Hence, deletion of a point can be done in $\Theta(k + \log n)$ time. The maximum value of k can be (n-1), corresponding to a intersection point. Therefore, in the worst case, the point can be deleted in $\Theta(n)$ time. For an isolated point the value of k is zero. Hence, the point can be deleted in $\Theta(\log n)$ time.

In case of deleting a line, terminating nodes for the two end points is found in Θ(log n) time and the respective entries (in the line-list) can be deleted in linear time. This facilitates deletion in Θ(n) time.

3.5.4 Addition of a point or a line

Addition of a point is a case of inserting one more node in the tree. This can be done by first finding a leaf-node which can be used to define the given point. The leaf-node may correspond to a terminating partition (i.e., a partition with one point) or a region without a point (i.e., a null region). In case of a region corresponding to a terminating partition, one more subdivision of the terminating partition is done and the tree height may, at most, increase by one. The time required to perform the addition is Θ(log n).

Addition of a line corresponds to inserting two nodes corresponding to the two end points of the line (if the nodes are absent) or updating the link list corresponding to the terminating partitions of both the end-points (if the terminating partitions corresponding to the two end-points already exist). In both the cases, the line-list has to be either updated or one more element is inserted in the line-list. This once again can be performed in Θ(log n) time.

3.5.5 Updating a point or a line

Updating a point can be thought of as a deletion and a addition. In this case there can occur following two cases:

1. All the lines connected to that point are to be deleted. Once this is done, point can be inserted in the tree as an isolated point. This can be done in $\Theta(k + \log n)$ time, where k is the number of lines going through that point. Because a point can be connected to all the other point, the operation will be of $\Theta(n)$ time complexity in the worst case.
2. In the second case, it may be desired that all the lines connected to that point are reinserted. This can be performed in $\Theta(n)$ time, in the worst case, since (n-1) line entries (in the line-list) will have to be updated to the pointer value of the terminating partition containing the new node.

Updating a line may involve some more work. Since both the end-points may be connected to at most all the other (n-2) points. At most (2*(n-1) + 1) entries in the line-list would be modified. Two points, corresponding to the end-points of the new line, can added to the tree in $\Theta(\log n)$ time and the new line is added to the line-list in constant time. Thus, the operation can be performed in linear time in the worst case.

3.5.6 Rotation

In this case rotation can be performed by rotating the lines of the desired sub-region (R) of the image with reference to a point or a line. Suppose rotation of $x°$ is to be obtained, then the desired region can be found out and all the points and lines inside that region can then be rotated about the given point or line to obtain the desired results. The time complexity depends on the number of points and lines in the image. At most, n and q lines will have to be investigated corresponding to the entire image.

3.5.7 Set-operations

Set theoretical operations can be performed by simply finding the regions R_1 and R_2 from the tree. Now, overlaying can be done by finding the intersection of lines and checking points and lines for overlapping in both the regions.

Union of two regions would mean that all the common points and lines are not to included twice in that region. The union of region R_1 and R_2 would encompass all the terminating regions belonging to the nodes in the tree for both the regions only once. Similarly intersection of the two regions will corresponds to all the common nodes of the terminating-nodes belonging to both the regions.

3.5.8 Space efficiency of OTDS

The height (H) of the resulting octagonal tree is $\lceil \log p \rceil$ (where, p is the number of points in the image). An octagon can be defined by 8 points [Section 3.3]. Let n bit be required to represent one of the coordinates of a point in two dimensional plane. Thus, 2n bits are needed to represent a point and an octagonal node (or a VOR) in the tree can be specified by 16*n (i.e., 8*2n) bits. Each of the hexagonal (LHR) and rectangular (LRR) nodes is specified with the help of two points [Section 3.3] and would need 4n (i.e., 2*2n) bits.

A double link-list (called line-list) is maintained for 'q' lines in the image. A link list, containing the pointers to the line-list, also exists [Section 3.5] for every terminating node in the tree. Each entry in the line-list can be specified with the help of 8 pointers (2 for each entry in the link-list for the two terminating nodes as end-points of the line and 4 for each entry in the line-list itself) [Section 3.5]. Let, s bit be sufficient for representing a pointer. Thus, 'q' lines can be specified by:

$$Z_1 = t*s*q \text{ bits},$$

where t= 8. The number of hexagons and rectangles can be obtained by observing that at most 2**(H) leaf-nodes are there in the tree [Section 3.2]. In the (H-1) th level, number of nodes will be 2**(H-1), and so on. Thus, total number of nodes (Z_2) in the tree would be:

$$Z_2 = \{2**H + 2**(H-1) + 2**(H-2) + \ldots + 1\},$$

or, $Z_2 = (2**(H+1) - 1)$.

Hence, the number of rectangular and hexagonal nodes would be:

$$Z_3 = (2**(H+1) - 1 - H),$$

for H number of octagons. Thus, total number of bits (Z) required for the OTDS representation would be as follows:

$$Z = \{4n*Z_3 + 16n*H + 8s*q)\}.$$

3.5.9 Pseudo-code for algorithms

In Appendix V, pseudo Pascal code has been given for OTDS data organization and algorithms in Sections 3.4, 3.5.3 and 3.5.4. This helps in analyzing the time complexity of these algorithms.

3.5.10 Point in polygon problem

A survey of the existing solutions for the 'point in polygon problem' has been done in [Nort82]. Similar to the triangular method described in [Gary78, Nort82], the OTDS can be used to solve the point-in-polygon problem. This can be done by storing the information 'whether the point (Z) is inside the polygon (G) or not' as an attribute in the record for every terminating partition. This results in $\Theta(\log n + K)$ time performance, where K is the number of attributes in the record corresponding to the terminating partition of the given point. This can be improved by incorporating binary search for the K attribute to give $\Theta(\log n + \log K)$ performance.

The search can also be made efficient by storing the pointer to the eldest non-leaf node in the OTDS tree for all the attributes in a link-list (L). In this way, all the points in the tree with given attribute can be obtained by only considering all the leaf-nodes, below the the non-leaf node(i.e. son nodes of the non-leaf node). The location of the non-leaf node in the tree can be obtained from the above mentioned link-listi (L).

3.5.11 Commenting on subdivision

There is a difference in subdividing a rectangle in between OTDS and quad-trees [Section 2.3.10]. The rectangle, in case of quad-trees, is subdivided into four sub-quadrants. Whereas, the rectangle is divided into wo quadrants for OTDS.

In contrast to GBT [Section 2.3.11], which expands from smallest hexagonal region, the OTDS converges to a rectangular, hexagonal and octagonal terminating partitions. The octagonal regions can be searched for approximate information. The binary search can be used for H octagonal nodes. This concept may be used to obtain $\Theta(\log H)$ quick search for approximate results, where $H = \lceil \log n \rceil$ [Section 3.2].

3.6 Summary

In this chapter, a new data structure has been suggested for storing vector data in which search for any point is logarithmic in number of points. This compares favorably with the Kirkpatrik's approach (Nort82). At this point, however, OTDS seems to be more practical approach for obtaining logarithmic search time for a point. Furthermore, basic operations such as deletions, insertions and updates can be efficiently performed as in a tree data structure. By definition [Section 1.1], the data structure belongs to the category of hybrids.

CHAPTER 4

ANALYSIS AND EVALUATION

In this chapter data formats, considered in Chapter two and three, are analyzed by considering a case of storing a triangle and an isolated point in a two dimensional plane. Other factors such as set-theoretic operations, ease of displaying and encoding are briefly reviewed. Later, the above mentioned factors are compared for various data formats in table 4.1 and 4.2.

4.1 Array representation

Let the image be covered by a m by n rectangle as shown in Figure 4.1. Now, n horizontal scans, each of m unit length, can be used to completely define the image with the help of m*n unit area rectangles. The intensity of each rectangle can be either one or zero depending upon whether any of the point or line segment lie inside the unit rectangle or not. The resulting array representation can be stored [illegible] n arrays with m elements. The elements of these array contains the intensity of the corresponding unit area of the image.

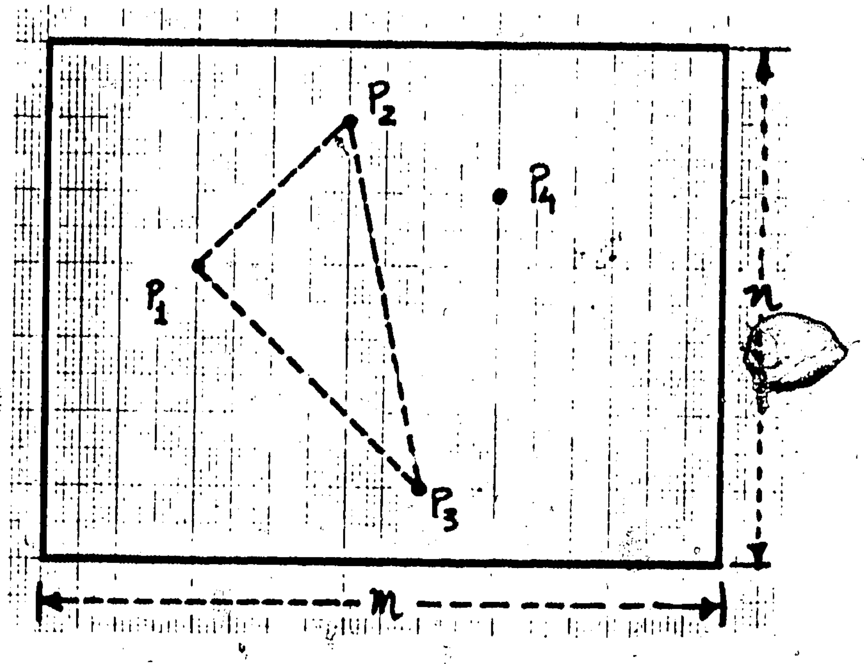


Figure 4.1: Shows the image enclosed in a rectangle of size m by n. The unit areas containing line segments of the triangle or the point (P4) have '1' intensity.

4.2 Run length encoding

Run length encoding may group the line segments having same intensity in a horizontal scan. As explained in Section 2.10, only the number of zeros or ones need to be stored for the given binary image. Thus the run-length encoding may look like as:

$$0,\ n1,\ n2,\ n3, \ldots$$

where n1, n2,... are the number of alternate zeros or ones corresponding to the entries in the array representation.

4.3 String representation

If triangle is defined as an object with name T and point P4 as object P, then corresponding to the binary image [as explained above] each horizontal scan would contain the information of the presence of each object. Thus some blank scans (all zeros) would also result. Other scans would contain two occurrences of object T. Only one scan contains one occurrence of object P.

4.4 Chain codes

The given image can be converted to the chain codes by considering the 8 near-neighbor chain coding as shown in Figure 4.2. The triangle P1 P2 P3 can be encoded by storing the coordinates of the point P1 followed by digits from 0 to 7. For example the next digit is 1 if starting from the point P1 and following the line P1 P2. Thus, the image can be stored by encoding the lines P1 P2, P2 P3 and P3 P1 as sequence of digits 0-7. The chain code of point P4 is the point itself.

4.5 Strip trees

A strip tree can be obtained by considering the curve P1 P2 P3 as shown in Figure 4.3. Points P2 and P3 are the starting and the end points and are connected. The resulting strip has P1 as the mid point and the line segments P1 P3 and P1 P2 are stored as lines. The point P4 is stored as a single point.

4.6 Tightly Closed Boundaries

A TCB representation can be obtained by observing that the image can be inclosed in a m by n rectangular region [Figure 4.1]. The horizontal scan intersect the triangle at two points. Hence all the horizontal lines would not have any intersecting point, or would have two intersecting points. One of the horizontal scan would have three points when the scan contains point P4 also. In this way number of Y- partitions are the difference of the Y coordinates of points P2 and P3.

4.7 Skeletons

The skeleton of the given image can be defined by string the three points defining the regular figure, i.e. the triangle, and the point P4. Figure 4.4 shows one other way of obtaining the skeleton of the given image. The dotted

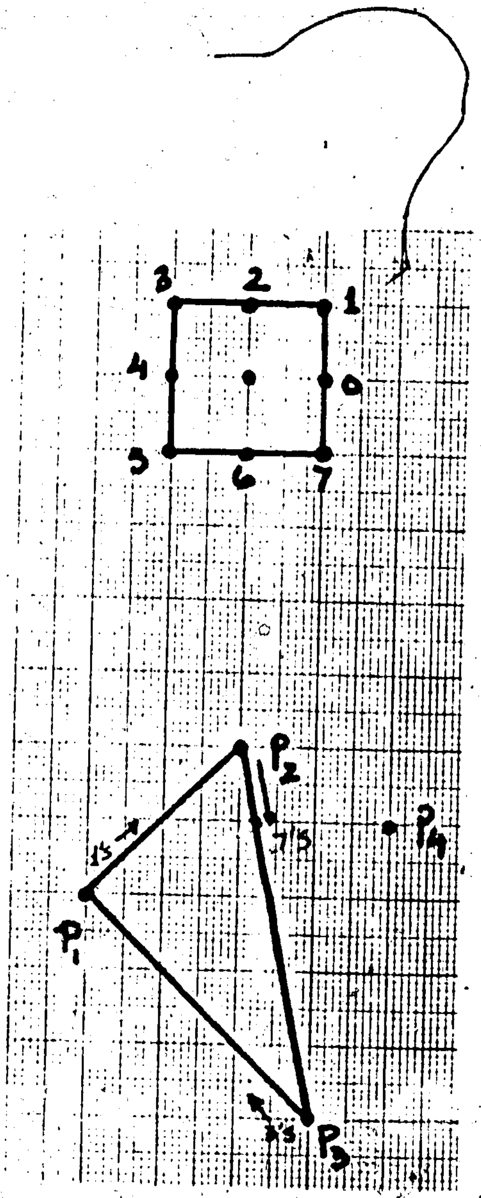


Figure 4.2: Shows the Chain Code with 8 near neighbors. Line P1 P2 can be approximated by sequences of 1's if starting from point P1.

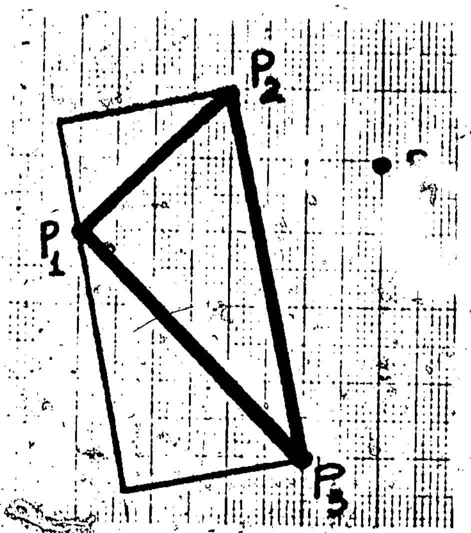


Figure 4.3: Shows the formation of Strip trees

lines in the Figure 4.4 results when only the middle point of the line segment (resulted from the intersection of the horizontal scan and triangle) inside the triangle and half the length of that line segment is stored. The triangle is defined by these points. The skeleton of the given image is, therefore, the dotted points and the coordinates of point P4.

4.8 Adjacency structures

The image is divided into variable size rectangular domains. The near-neighbors are also maintained. A particular case of subdividing into rectangular areas is quad tree and is discussed later.

4.9 CIAD Data format

The point file of CIAD data format consists of four points. The line file consists of three lines of the triangle of the image. Both files are preceded by a header [Section 2.3.1]. The polygon file consists of the pointers to the points in the point file, i.e., points P1, P2 and P3 followed by pointers to the lines P1 P2, P2 P3 and P3 P1. The point P4 is defined as isolated point and other points are intersection points. They are differentiated on the basis of the tag attached to each point.

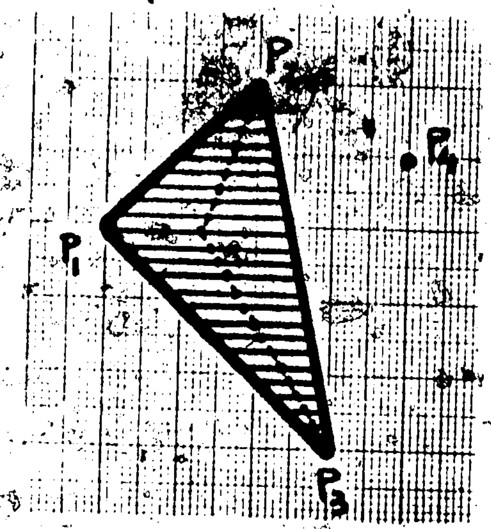


Figure 4.4: The Skeleton of the given image.

4.10 Boundary network

The given image is stored as a set of the starting ıt (P1), intermediate points (P2) and the end point (P3). ıe point P4 is stored as a boundary with only one point.

4.11 Binary Searchable Polygonal Representation

The curve P1 P2 P3 is stored as defining the root of a binary tree. The son-nodes of the tree are line-segments defined by the curve P1 P3 as left son node and line P3 P1 as the right son-node. The curve P1 P3 is divided into two line segments P1 P2 and P2 P3. The point P4 defines another binary tree without son-nodes.

4.12 The RSE Structure

In this representation, the given image is defined as a set of points P1, P2, P3 and P4 followed by lines P1 P2, P2 P3 and P3 P1 defined by pointers to the respective points. This is followed by a region information, which is the triangle in this case, defined by pointers to the three lines.

4.13 Multi-dimensional binary search trees

If point P3 is chosen as the root node of the 2-d tree then either of the points P1 or P2 can be chosen as the left son. Let P1 be the left son. The right son is point P4. The above decision has been taken on the basis of the value of X coordinates of all the four points. Now, P2 - the only point left, which is stored as the right son of point P1 on the basis of the Y coordinate of points P1 and P2 [Also refer Section 2.3.8]. The information that some points are connected can be stored as attributes at the respective nodes for the points in the 2-d tree.

4.14 Quad trees

Given image can be enclosed in a 2**p by 2**p area. As suggested in [Same83], an edge quad tree can be obtained by subdividing the rectangular area recursively into four equal areas till rectangular regions, with only one point or without a point, are obtained. The resulting subdivision of the given image has been given in Figure 4.5.

4.15 OTDS representation

In case of OTDS representation, the image is enclosed in an octagon. An enclosing octagon is found with the help of the maximum and minimum X and Y coordinate values of the points in the image. These values defines a enclosing rectangle. Let d1 and d2 be the difference in the maximum and minimum values in X and Y coordinates and C0 be the center point of the enclosing rectangle. As shown in Figure 4.6, the enclosing octagon (O) can be obtained by defining an octagon whose sides are of length d (larger of d1 and d2). The octagon O is centered around point C0 [Figure 4.6]. Subsequent octagons are found by selecting the sides along the dotted triangle as shown in Figure 4.6. These octagons

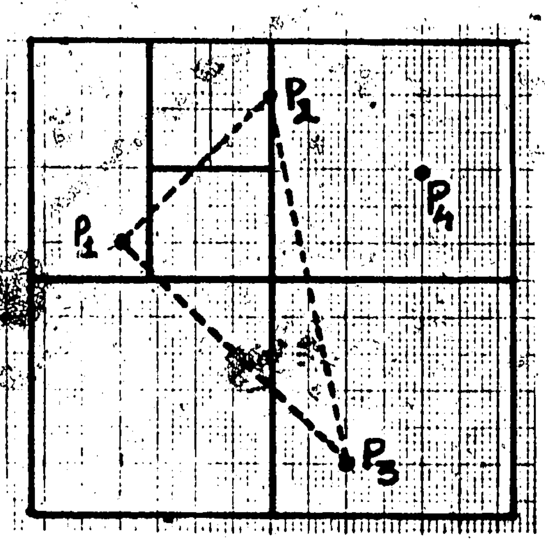


Figure 4.5: Edge Quad-tree for the given image.

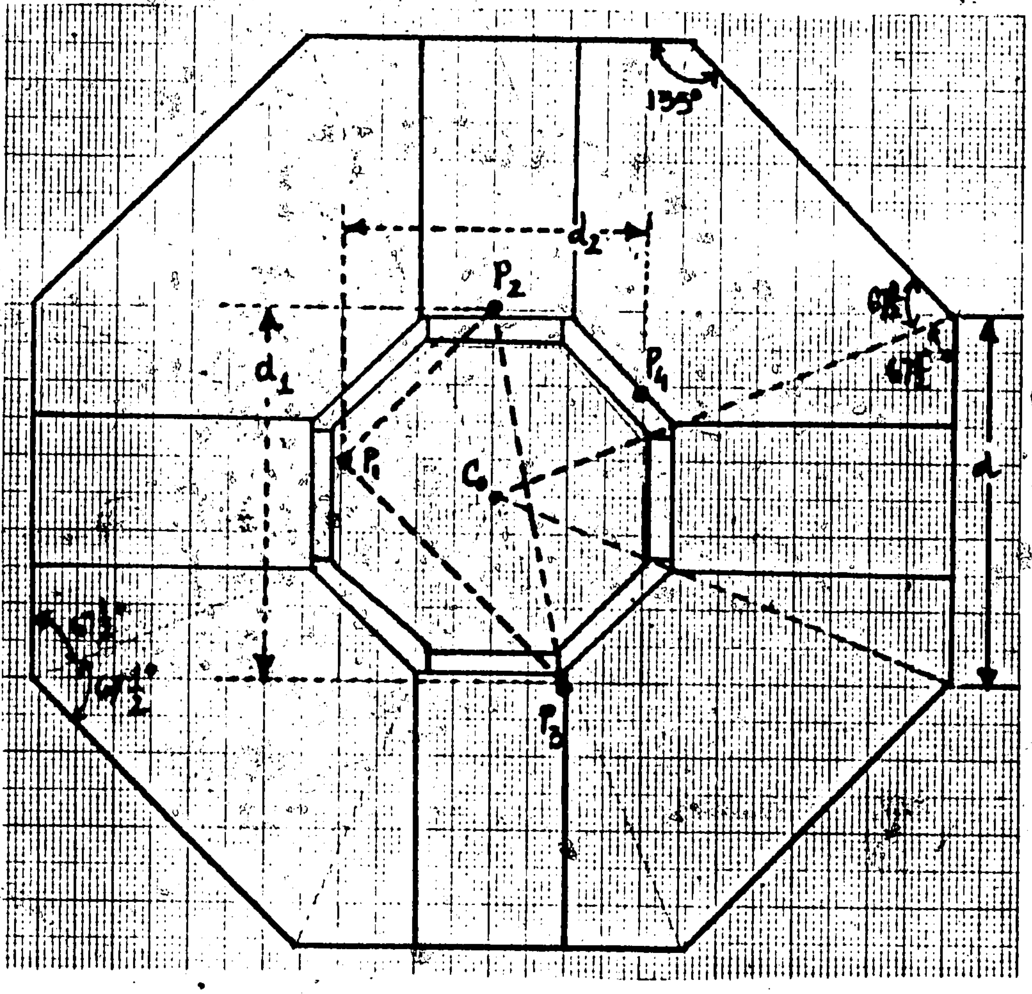


Figure 4.6: The OTDS representation of the given image.

have half the number of points then the preceding octagon, for example, octagon O1 and O2 contains two and one points respectively. The resulting hexagons and rectangles are not subdivided as each of them is a terminating partition [Section 3.5]. Sometimes octagons with half the points are obtained by changing the orientation, side lengths and angles as discussed in Appendix IV.

4.16 Pyramids

Pyramids can be obtained by defining the given image at different resolutions. Thus, at coarser levels the triangle may appear as a single point.

4.17 GBT representation

GBT representation is obtained by covering the given image by a hexagonal grid. As explained in Section 4.1, some of the hexagonal region will contain a portion of any of the three lines or four points defined as regions with intensity level one. The other hexagonal regions would have zero intensity.

4.18 Further considerations

The factors considered, in this section, are: scaling, browsing, ease of rotation about a point, set operations (e.g., intersection and union of two regions), ease of displaying the information, cost of digitization and preprocessing required. All these considerations are briefly discussed to compare the data formats considered in the previous section.

In general, scaling is difficult for image data in comparison to vector data. Hybrid (hierarchical) data structure facilitates easy browsing as neighboring terminating partitions (regions) can be efficiently obtained. For raster (vector) data formats browsing may be linear in terms of number of pixels (points) in the image.

In the case of quad-trees, rotation by multiple of 90° can be obtained by recursive rotation of sons at each level of the tree [Same83]. A transformation matrix is used for an array representation [Newm79]. In case of run length codes, rotation about a point may be obtained by rotating every run of 0's and 1's by the desired angle. In case of point-quad-trees and OTDS, the rotation can be performed by finding the region of interest and then rotating all the points and lines about a point. For 8 directional chain codes a rotation of (45*x)° [where, 0≤ x≤8] is erformed by adding 'x' to the chain code of every line-segment in the desired sub-section of the image [Same83].

Intersection and union operations are difficult to perform in case of chain codes. Other formats, such as array representation and run length coding, checks each pixel of the image irrespective of the region being a small part of the whole image. This means that for all such operations, these data formats would end up checking every pixel in the image. In the case of quad-trees, the problem is overcome by isolating the regions of interest. In the worst case, it still amounts to checking every pixel individually. In case of OTDS, the search for the desired region is performed similar to quad-trees. Once search has been done, each line and point is checked for any overlapping. This amounts to solving simultaneous equations in two dimensions. Note that in worst case checking for all the points and lines will have to be performed.

Displaying, i.e., interfacing with the output devices, is also a major criteria. Generally the output devices are raster in nature. Array representation, quad-trees and run length coding is directly mapped to the screen (frame buffer approach [Pavl82]). In case of chain codes, point-quad-trees and OTDS, a point is displayed by positioning the beam at the point and setting the beam intensity to desired intensity level. A line-segment (or a link) is displayed as a set of points [Pavl82].

Some of the data formats require heavy preprocessing [Sections 2.3.2, 2.3.6, 2.3.8] or overhead of maintaining the pointers [Sections 2.3.10, 2.3.11, 3.3]. This results in preferring simpler data formats [Sections 2.2.1, 2.2.3, 2.3.4] as is evident in most of the existing data banks [Appendices I and II] and geographical information processing systems [Nagy79].

4.19 Summary

A simple example has been considered for data formats considered in Chapter 2 and 3. The hierarchical data structures have higher encoding complexity. The characteristics of different data structures has been summarized on the basis of the time and space complexity, set-theoretic operations, encoding and decoding complexity in Table 4.1 and 4.2.

TABLE 4.1: Comparison of time and storage efficiency

Data Formats	Search	Deletion	Addition	Storage
Chain codes	F	F	F	ES
Strip trees	F	F	F	ES
TCB	F	F	F	FS
Skeletons	F	+	+	ES
Adjacency structures	F	+	+	ES
CIAD	F	F	F	FS
Boundary Network	F	F	F	FS
BSPR	E	E	+	IS
RSE	F	F	F	FS
k-d trees	E	E	E	IS
Quadtrees	E	E	F	IS
GBT	E	+	+	IS
Pyramids	F	F	E	IS
Binary Array	I	I	I	IS
Run length	I	I	I	IS
String representation	I	I	I	IS
OTDS	E	E	E	FS

• Following criterion has been followed:
Time performance:
+ Reorganization may be required.
I Linear in terms of number of pixels in an image.
F Linear in terms of number of points or (and) lines in the image.
E Logarithmic in number of pixels or points in the image.

Storage efficiency:

ES Only points and lines to be stored.
FS Overhead of maintaining the pointers for points and lines.
IS Overhead of maintaining pointers for all the pixels.

Table 4.1 Time and memory performance comparisons

TABLE 4.2: Comparison of other factors.

Data Formats	Display	Encoding	Browsing	Scaling	Rotation	Set operations
Chain Codes	F	F	D	E	E	D
Strip trees	F	F	D	E	D	E
TCB	F	D	D	D	D	D
Skeletons	F	D	D	F	D	E
Adjacency Structures	F	F	D	E	D	D
CIAD	F	E	D	E	D	F
Boundary network	F	F	D	F	D	D
BSPR	F	F	D	F	D	D
RSE	F	E	D	E	D	F
K-d trees	F	D	D	D	D	D
Quadtrees	E	D	E	F	F	E
GBT	E	D	E	F	D	E
Pyramids	E	D	D	E	D	D
Binary array	E	E	D	D	D	D
Run length encoding	E	E	D	D	D	D
String representation	E	F	D	D	D	D
OTDS	F	D	E	E	F	E

* Following criterion has been followed:
D Difficult.
F Fair.
E Easy.

Table 4.2 Other considerations

CHAPTER 5

CONCLUSIONS

The design considerations for an image data base system are guided by factors such as: response time, amount of data involved, frequency of insertions, updates and deletions of a data-type, ease of display and encoding of information. Raster and vector formats have been used in the existing systems to store the image files. Raster data format are convenient to use because display devices are raster in nature and scanners produce large amount of raster information. In case of vector formats, an image is a set of points and lines and conversion from vector-to-raster (and raster-to-vector) can be readily performed. The above mentioned properties make vector and raster formats a convenient choice for image applications [Nagy79].

In recent years, the development of distributed image data bases has stressed the need for designing systems with good response time. Apart from the communication delay in the network, the time taken for searching a data type at a remote center is critical for user satisfaction. In case of stand alone systems, which do not require any file transfer, vector or raster representations may be used. But, in systems requiring frequent information transfer, hierarchical or hybrid formats should be used to minimize access time. The overhead of maintaining the pointers for these data formats can be justified considering the facility of archiving [Davi82] the information in distributed systems.

Hybrid data formats are useful for distributed systems using vector data [Sections 2.3.4, 2.3.5 and appendices I and II]. Among the hybrid data formats, strip-trees, point-quad-trees and GBT are useful. But, search for a point may be linear for strip-trees [Ball81]. The quad-tree is sensitive to orientation of the image [Same83] and GBT [Gibsa] seems to have complex addressing scheme.

In this thesis, another hybrid data format (OTDS) has been suggested. The data structure compares favorably with all the existing data formats in deleting, inserting and updating a point or a line. A conversion algorithm [Section 3.5] has been discussed which may use point and line files obtained in CIAD file format [Appendices I and II] to produce an OTDS. In comparison to all the other data structures considered in this thesis, search for a point is more efficiently performed for an OTDS.

5.1 Further Research

Though hybrid data structures provide substantial theoretical claim, there is a dearth of empirical results. This can be partly attributed to the fact that there is no practical system using a hybrid data format. In case of OTDS, much work has to be done. It would be beneficial to study the best, worst and average case behavior and establish empirical guidelines for designing a system based on this data format.

The conversion packages [Appendices I and II] can provide test data for developing software for CIAD project. This can be useful in designing a prototype for simulation tests. The OTDS can be used to provide a level of abstraction over the data in CIAD file format and define a topological tree for accessing the information. Finally, this data structure should be visualized as an attempt in providing an alternate data format for image applications.

REFERENCES

Ball81. D.H. Ballard, "Strip trees: A Hierarchical Representation for Curves", *CACM*, Vol. 24(5) pp. 310-321 (MAY 1981).

Bent75a. J.L. Bentley, "Multidimensional Binary search trees used for Associative searching", *CACM*, Vol. 18(9), pp. 509-517 (SEP 1975).

Bent75b. J.L. Bentley, "Analysis of Range search in Quad trees", *IPL*, Vol. 3(6), pp. 170-173 (JUL 1975).

Bent76. J.L. Bentley and W.A. Burkhard, "Heuristics for Partial match retrieval Database Design", *IPL*, Vol. 4(2), pp. 132-135 (FEB 1976).

Bent77. J.L. Bentley, D.F. Stanat, and E.H. Williams, "The Complexity of Fixed radius near neighbor searching", *IPL*, Vol. 6(6), pp. 209-212 (DEC 1977).

Bent78a. J.L. Bentley, "Algorithms for Multivariate Non-parametrics", CMU-CS-78, Department of Computer Science, CMU (1978).

Bent78b. J.L. Bentley and M.I. Shamos, "A Problem in Multivariate statistics: Algorithm, Data structures and Applications", CMU-CS-78, Department of Computer Science, CMU (1978).

Bent79a. J.L. Bentley, "Multidimensional Binary search trees in Database Applications", *IEEETSE*, Vol. 5(4), pp. 333-340 (1979).

Bent79b. J.L. Bentley, "Decomposable Searching problems", *IPL*, Vol. 8(5), pp. 244-151 (JUN 1979).

Brin82. K. Brinkmann, R. Buhr, H. Eisermann, and H Guenther et. al, "Integration of a Picture-oriented Data base system in an Image processing System", *Proceedings of IEEE 6th International Conference on Pattern Recognition*, Munich, Germany, pp. 798-801 (OCT 1982).

Burt77. W. Burton, "Representation of Many sided Polygons and Polygonal lines for Rapid Searching", *CACM*, Vol. 29(3), pp. 166-171 (MAR 1977).

Burt80. P.J. Burt, "Tree and Pyramid structures for Encoding Hexagonally Sampled Binary images", *Computer Graphics and Image Processing*, Vol. 14, pp. 166-171 (1980).

Choc81. A. Chock, A.F. Cardenas, and A. Klinger, "Manipulating Data structures in Pictorial Information Systems", *Computer*, Vol. **14**(11), pp. 43-53 (1981).

Davi83. W. A. Davis, *CIAD Design Group Report: File System Architecture*, Department of Computing Science, U of A, Edmonton, (MAR 1983).

Deo77. N. Deo, J. Nievergelt, and E. Reingold, *Combinatorial Algorithms*, Prentice-Hall, Inc, Englewood Cliffs, NJ, USA (1977).

Dutt79. G.H. Dutton and W.G. Nisen, "Expanding Realm of Computer Cartography", *Datamation*, pp. 134-142 (JUN 1979).

Dyer82. R.C. Dyer, "Space Efficiency of Quad trees", *Computer Graphics and Image Processing*, Vol. **19**, pp. 335-348 (1982).

East70. C.E. Eastman, "Representation for Space Pl nning", *CACM*, Vol. **13**(4), pp. 242-250 (APR 1970).

Edso77. D.T. Edson and G.Y.G. Lee, "Ways of Structuring Data within a Digital Cartographic Data Base", *Computer Graphics quarterly report*, Vol. **11**(2), pp. 148-157 (Summer 1977).

Fink74. R.A. Finkel and J.L. Bentley, "Quad trees: a Data structure for Retrieval on Composite keys", *ACTA*, Vol. **4**(1), pp. 1-9 (1974).

Free74. H. Freeman, "Computer Processing of Line drawing Images", *COMP*, Vol. **6**(1), pp. 57-97 (MAR 1974).

Free80. H. Freeman and H. Samet, "Region Representation: Quad-trees from Boundary codes", *CACM*, Vol. **23**(3), pp. 163-170, Paper presented in IFIP congress 1980, Tokyo, Japan and Melbourne; 6-7 October 1980 (MAR 1980).

Free82. H. Freeman and P.J. Frauenknecht, "An Efficient Computer Representation Scheme for Linear Data", *IEEE*, pp. 315-320 (1982).

Garg82a. I. Gargantine and Z. Tabakman, "Linear Quad and Oct Trees: Their use in Generating Simple algorithms for Image Processing", *Graphics Interface*, pp. 123-127, 17-21 May, Toronto, Ontario (1982).

Garg82b. I. Gargantine, "Linear Oct-trees for Fast Computer Processing for Three dimensional Objects", *Computer graphics and image processing* , Vol. **20**(4), pp. 365-374 (DEC 1982).

Gibsa. L. Gibson and D. Lucas, *Line and Edge Extraction Using Hierarchical Methods*, Interactive System Corporation, 5574 South Prince Street, Littleton, Colorado.

Gibsb. L. Gibson and D. Lucas, *Introduction to GBT*, Interactive System Corporation, 5574 South Prince Street, Littleton, Colorado.

Gros83. W.I. Grosky and R. Jain, "Optimal Quad-trees for Image Segments", *IEEE transactions on Pattern Analysis and Machine Intelligence*, Vol. 5(1), pp. 77-83 (JAN 1983).

Hero80. C. F. Herot, "Spatial Management of Data", *ACM transactions on Database Systems*, Vol.5(4), pp. 493-514 (1980).

Klin76. A. Klinger and R. C. Dyer, "Experiments in Picture Representation using Regular Decomposition", *Computer Graphics and Image Processing*, Vol. 5, pp. 68-105 (1976).

Klin79. A. Klinger and M.L. Rhodes, "Organization and Access of Image data by Areas", *IEEE transactions on Pattern Analysis and Machine Intelligence*, Vol. 1, pp. 51-60 (JAN 1979).

Lee78. D.T. Lee and C.K. Wong, "Worst case Analysis for Region and Partial region Searches for Multidimensional Binary Search trees and Quad trees", *ACTA*, Vol. 9(1), pp. 23-29 (1978).

Li82. M. Li, W.I. Grosky, and R. Jain, "Normalized Quad-trees with respect to Translations", *Computer Graphics and Image Processing*, Vol. 20, pp. 72-81 (1982).

Loom65. R.G. Loomis, "Boundary networks", *CACM*, Vol. 8(1), pp. 44-48 (JAN 1965).

Luca. D. Lucas and L. Gibson, "Vectorization of Raster Images Using Hierarchical Methods", *Computer Graphics and Image Processing*, Vol. 20, pp. 82-89 (1982).

Mart82. J.J. Martin, "Organization of Geographic Data with Quad trees and Least Square Approximations", *Pattern Recognition and Image Processing*, pp. 458-463 (1982).

Mats. T. Matsyama, "A File Organization for Geographic information Processing Systems based on Spatial proximity", *Pattern Recognition and Image Processing*, Proceedings Sixth International Conference, Munich, Germany (OCT 1982).

McKe77. D.M. McKeown and D.R. Reddy, "A Hierarchical Symbolic Representation for an Image database", *Proceedings of IEEE workshop on Picture Data Description and Management*, pp. 40-44 (1977).

Meag82. D. Meagher, "Geometric modeling using Oct-trees", *Computer Graphics; June 1982*, Vol. 19(2), pp. 458-463 (JUN 1982).

Merr73. R.D. Merril, "Representation of Contours and Regions for Efficient search", *CACM*, Vol.16(2), pp. 69-81 (FEB 1973).

Mont70. U. Montanari, "A note on Minimum length Polygon Approximation", *CACM*, Vol. 13, p. 41 (1970).

Nagy80. G. Nagy and S. Wagle, "Computational Geometry and Geography", *The Professional Geographer*, Vol. 32(3), pp. 343-254 (AUG 1980).

Nagy82. G. Nagy, *Optical Character Recognition: Theory and Practice*, North-Holland Publishing Company, Amsterdam-New York-Oxford (1982).

Nagy79. S. Nagy and S. Wagle, "Geographic data processing", *COMPSUR*, Vol. 11(3), pp. 139-181 (1979).

Newm79. W.M. Newman and R.F. Sproull, "Chapter 5: Clipping and Windowing", *Principles of interactive Graphics*, McGraw-Hill, Inc. pp. 63-74 (1979).

Nort82. M.S. Nortey, "Point-in-Polygon Algorithms for Geographic Information Systems", *M.Sc. thesis, University of Alberta*, Comp. Sc. Dept., Edmonton, Canada, pp. 6-88 (1982).

Pavl82. T. Pavlidis, *Algorithms for Graphics and Image Processing*, Computer Science Press, Rockville, MD, USA, pp. 1-126 (1982).

Peuq79. D.J. Peuquet, "Raster Processing: An alternative Approach to Automated Cartographic Data Handling", *The American Cartographer*, Vol. 6(2), pp. 129-139 (1979).

Peuq81. D.J. Peuquet and D.J. Peuquet, *An examination of Techniques for Reformatting Digital Cartogrphic Data PART22 The Vector-to-Raster Process*, Resource Planning Analysis Office, United States Department of Interior, Geology Survey, Reston, Virginia (1981).

Peuq83. D.J. Peuquet, "An hybrid Data structure for the Storage and Manipulation of Large Spatial Data sets", *Computer Graphics and Image Processing*, Vol. **24**(1), pp. 14-27 (OCT 1983).

publ. COMRAC Design systems publications, *An efficient Low cost System for handling Forest resource inventories*, The Agriculture Building Embarcadero at Mission San Franscisco, California.

Rose67. A. Rosenfeld and J. L. Pfaltz, "Computer Representation of Planar Regions by their Skeletons", *CACM*, Vol. **10**(2), pp. 119-122 (FEB 1967).

Rose76. A. Rosenfeld and J. L. Pfaltz, "Sequential operations in Digital Picture Processing", *Journal of the Association for Computing Machinery*, Vol. **13**(4), pp. 471-494 (OCT 1976).

Rose82. A. Rosenfeld, Hong, and Wu, "Threshold selection using Quad trees", *IEEE transactions on Pattern Analysis and Machine Intelligence*, Vol. **4**(1), p. 90 (JAN 1982).

Same80. H. Samet, "Region Representation: Boundary Codes from Quad-trees", *CACM*, Vol. **23**(3), pp. 171-179 (MAR 1980).

Same81a. H. Samet, "Computing Perimeters of Regions in Image Represented by Quad-trees", *IEEE transactions on Pattern Analysis and Machine Intelligence* Vol. **3**(6), pp. 683-687 (NOV 1981).

Same81b. H. Samet, "An algorithm for Converting Rasters to Quad-trees", *IEEE transactions on Pattern Analysis and Machine Intelligence*, Vol. **3**(1), pp. 93-95 (JAN 1981).

Same82. H. Samet, "Neighbor finding techniques for Image represented by Quad-trees", *Computer Graphics and Image Processing*, Vol. **18**, pp. 37-57 (1982).

Same83. H. Samet, "Using Quad-trees to represent Spatial data", TR-1287, University of Maryland, Maryland (MAY 1983).

Shap79. L.G. Shapiro, "Data structures for Picture processing: A survey", *Computer Graphics and Image Processing*, Vol. **11**, pp. 162-184, (1979).

Shne81. M. Shneier, "Calculations of Geometric properties using Quad-trees", *Computer Graphics and Image Processing*, Vol. 16, pp. 296-302 (1981).

Stei79. K. Steiglitz and G.M. Hunter, "Operation on Images using Quad-tress", *IEEE transactions on Pattern Analysis and Machine Intelligence*, Vol. 1, pp. 145-153 (APR 1979).

Tani82. R. Taniguchi and M. Yokota, et. al, T. Henderson, and E. Triendl, "The K-D tree representation of Edge descriptions", *Proceedings of IEEE 6thInternational Conference on Pattern Recognition, Munich*, Germany, pp. 806-807 (Oct 1982).

Tani76. S. Tanimoto, "Pictorial Feature distribution in a Pyramid", *Computer Graphics and Image Processing*, Vol. 5, pp. 333-352 (1976).

Tani75. S.L. Tanimoto and T. Pavlidis, "A Hierarchical Data structure for Picture processing", *Computer Graphics and Image Processing*, Vol. 4(2), pp. 104-119 (1975).

Zobr81. A.L. Zobrist and G. Nagy, "Pictorial Information processing of Landsat data for Geographic analysis", *COMP*, Vol. 14(11), pp. 34-41 (1981).

APPENDIX I.

A definite line file format has been proposed in the CIAD[2] Design Group Report[3]. At that time, it was felt necessary to create CIAD point and line files. The digitized map of Edmonton was used to obtain these files. The details of the format of the digitized map has been explained in the thesis by Mailloux[4].

In this section, the implementation detail of a conversion package has been discussed. This conversion package may be used to obtain CIAD line and point files and list of polygons from the digitized map of city of Edmonton or Edmonton Map Data Bank (EMDB).

Interpreting various fields of EMDB

In Edmonton Map Data Bank (EMDB), whole city has been digitized along main thoroughfares. The data bank contains the (X, Y) coordinates of each point on the thoroughfare

----

[2] Communication and Display for Interactive Access to Distributed Spatial Databases.
[3] 'CIAD Design Group Report: File System Architecture' by W. A. Davis, et al., Department of Computing Science, University of Alberta.
[4] 'Computerized Map: City of Edmonton' by Mailloux I.S. in Department of Geography, University of Alberta, spring, 1973.

with respect to the origin (101 street and 101 avenue). This along with few other fields (in all seven) constitute a record in the data bank. Let, first field be defined as f_1, second field as f_2 and so on. The master and detail records differ in the value of f_1 which is unique for a record. The master record is followed by one or more detail records. These detail records are points on the thoroughfares. The value of the field f_3 (or f_4) signifies the distance of the point from the origin in feet. The positive value of f_3 (f_4) signifies the distance of the point from the origin in east (north) direction, whereas, negative value of x (y) represents the distance in west (south) direction.

The detail records (in between two master records) may be visualized as forming a set of points (set S_1). Four types of points have been defined - namely intersection point, end point, bending and pseudo point. There is no difference between a pseudo and a bending point. Both the points have been included to account for natural curves and abrupt changes in the thoroughfares. The points can be differentiated on the basis of the value of f_2. Two consecutive points in set S_1 are connected to each other except when both of them are end points. The first and the last point, in set S_1, are the starting and the end points of the line. An intersection point appears in all S_1 sets corresponding to the thoroughfares (lines) going through that point. So, a point may be present more than once in the EMDB.

It follows that: if V denote a set of vertices, where each vertex correspond to a unique point on the Edmonton map, then all the set S_2 are subset of set V and may not be disjoint.

Implementing the 'Window'

The user specifies bottom left and top right coordinates of a window. The data is extracted on the basis of this window. The algorithm has been explained in [Newm79].

Obtaining CIAD point file

The nature of the input data is such that the points can be easily obtained, as f_3 and f_4 are the values of x and y coordinates. The problem is to define the nature of the point, i.e., whether the point is an end point (EP), isolated point (IS), edge point (ED) or intersection point (IN). This is done by checking the value of f_2 in the detail record of fileg1. Following interpretations have been followed:

1. If f_2 is '. ' then the point is an intersection point (IN).
2. If f_2 is '&' then it is an end point (EP).
3. If f_2 is '#' then this is a pseudo point or a bending point and is entered as an intersection point (IN).

4. If only one detail record exist in the set S_1 then this point is termed as isolated point (IS).
5. If the point lies on any of the edges of the window then point is an edge point (ED).

In this way a point file ('filep') is obtained.

Obtaining a line file

A line file is obtained with the help of the value of f_2 field in fileg1. The two detail records will form a valid CIAD line entry if following interpretation holds true:

1. The two consecutive detail records are either intersection, pseudo or bending points, or
2. One of the point is an end point, or
3. Both the points are not end points.

The line file 'filel' is obtained on the basis of the above interpretation.

Obtaining a list of polygons

Finding a polygon was a case of generating all the cycles in a acyclic graph G(V, E) where V is set of vertices and E is set of edges. In this case, all the unique points in the point file are the elements of the set V and the set of edges E is the set of lines. The algorithm implemented is as described in [Deo77]. After the above algorithm is executed, the polygon file (fileg6) is obtained. For this case, test-files (SEMW:t) containing only one triangle, was only considered.

Run command

The CIAD point and line files can be obtained from Edmonton Map Data Bank (EMDB) by using the following command on MTS:

```
$r *pascalW scards=program sprint=-a par= filem=-v,
filel=-1, filep=-p, filepoly=-poly, file1=-1, file2=-2,
fileg1=-g1, fileg2=-g2, fileg3=-g3, fileg4=-g4, fileg5=-g5,
fileg6=-g6, output=*msource*, fdata=data, filed=*msource*,
filevalue=-v2.
```

where, files 'program' and 'data' are 'SEMW:CIAD20BACK' and RLTH:edmonton.(db1 + db2 + db3). The point and line files are 'filep' and 'filel', respectively [See 'program' listing].

Analysis and complexity

Let |N| denote number of points in the point file and |E| be number of edges in line file. Also, let 'D' and 'C' denote number of records and cycles, respectively, in the sub-section of EMDB. Now, the time and complexity of the above implementat[illegible] can be analyzed as follows:

1. The Clipping algorithm has to consider all the detail records in the EMDB, hence is $\Theta(|D|)$.
2. Both, point and line file would be created in $\Theta(|D|)$ time.
3. Obtaining polygons is to generate all the cycles. This would require $\Theta((|N| + |E|)*(C + 1))$ basic operations [Deo77].

Most of the memory is required for finding the polygons and would be $\Theta(|V| + |E|)$ [Deo77].

Comment

The package has been implemented in 'PascalW' and runs on Amdahl 5860 under MTS operating system.

A rectangular area defined by (-5000, -5000) and (5000, 50000) was used to obtain the point and line files (files EMDB.p18 and EMDB.l18 under $stuart/edm/semwal on PDP-11/45). These files were then transferred to the PDP-11/45 and an image file (MAP18) was obtained with the help of the CIAD file editor (/usr1/image/stuart/edx). The generation of polygon file was tested by using the file SEMW:t as the data file for this program.

APPENDIX II.

World Data Bank II consist of coordinates of coastlines, islands, lakes, rivers, international boundaries of the world. The data made available to the University of Alberta is a subset of the original data, as some details such as intermittent features, reefs, salt pans, canals, ice shelves etc., are absent. The data can be obtained by running the typical session as given in tutorial by Computing services, University of Alberta[5]. The implementation detail of such a package has been discussed along with time and space complexity analysis. The package can be used to obtain CIAD line (file 'linefi') and point (file 'pointfi') files from World Data Bank II.

World Data Bank II

The world data bank is at the scale of 1:1 million to 1:4 million. The coordinates are measured in units of latitude and longitude to the nearest second. The data is organized as consisting of set of points. A master and a detail record can be differentiated on the basis of the

[5] Tutorial T128.0382, World Data Bank II: Instruction for use by Computing Services, University of Alberta.

seventh character, which is either an integer (for a point or detail record) or a character (for master record).

The user specifies the top right and bottom left corners (in latitude and longitude) of a rectangular window. All the points and line sections, lying inside this rectangular window, are obtained in files (fileg2, fileA and fileB). These files are used to obtain CIAD point and line files. The clipping algorithm has been described in [Newm79].

Finding intersection points

In World Data Bank II, intersection points are not listed. The intersection points can be obtained with the help of identical files- fileA and fileB. Let, S_1 denote the set of detail records in between two master records and 'p' be the number of sets in fileA (or fileB). Two consecutive detail records define a line-section in any set S_1. Each line section (L_1) in set S_1 is tested with all line sections (L_2) except the lines belonging to same set S_1. An intersection point is a valid point if it lies on both lines L_1 and L_2. A valid intersection point is inserted between two end-points of both the lines L_1 and L_2. For avoiding consideration of a pair of lines L_1 and L_2 more than once, fileA and fileB are used for processing. Let i th set (S_i), in fileA, be the set being considered for finding intersections points. All sets S_j in fileB are considered for processing such that $(i+1) \leq j \leq p$. In this way fileg3, containing all the points including intersection points, is obtained. Now, fileg2 and fileg3 are equivalent to the set of unique points (V) and edges (E), respectively.

Point and line files

The point file can be obtained by listing points in World Data Bank II. There is one to one correspondence in CIAD file format and World Data Bank. For every point in World Data Bank II, a point in CIAD file format is obtained according to the following interpretation:

1. if the point is on the edge of the rectangular window then it is an edge (ED) point.
2. If not more than one detail record is present in between two master records, then the point is an isolated (IS) point.
3. Intersection (IN) points are obtained as defined above.
4. All the other points are defined as end (EP) points.

The line file (filel) is obtained by considering line sections in detail record sets. Two consecutive points in a set of detail records form a line in CIAD file format. Further, with the help of set of points (V) and edges (E), a list of polygon can be obtained as described in [Appendix I].

Analysis and complexity

Let |N| denote the number of points in the point file and |E| be the number of edges in the fileA or fileB (after windowing). Also, let 'D' denote the number of records in the fileA (or fileB) and the number of cycles in the graph be 'C', then the time complexity of the above implementation can be analyzed as follows:

1. The Windowing algorithm has to consider all the detail records in the fileA, hence is $\Theta(|D|)$.
2. Finding Intersection is basically a search and would take $\Theta(D^2)$.
3. Both the point and the line file would be created in $\Theta(|D|)$.
4 Generating polygons is to generate all the cycles. This would require $\Theta((|N| + |E|)*(C + 1))$ basic operations [Deo77].

Most of the memory required is for finding the polygons. It would be $\Theta(|V|+|E|)$ [Deo77].

Run command

The CIAD line and point files can be obtained from the World Bank Data II, by using the following command on MTS:

```
$r *pascalW scards=program sprint=-t par= fileA=-a,
fileB=-b, fileC=-c, infile=*msource*, output=*msource*,
fileD=data, filetd=*msource*, fileg1=-g1, fileg2=-g2,
fileg3=-g3, fileg4=-g4, fileg5=-g5, fileg6=-g6 filel=-l,
filep=-p, filevalue=-v2.
```

where, 'program' and 'data' files are 'SEMW:WORLD.BACK' and the sub-section of the World Data Bank II.

Comment

The package runs under MTS on Amdahl 5860 and is written in pascalW. It can be used for obtaining point and line files for project CIAD. The window on sub-sections [NA.RIV.01 + NA.BDY.01 + NA.PBY'] of the Data Bank was defined as (48° latitude, -120° longitude) and (60° latitude, -110° longitude) to obtain map of Alberta. The point (AB.PP) and line (AB.LL) files were transferred to PDP-11/45 (under $semwal/ciaduseful) from MTS. These files can be used as input to the CIAD file editor (under $stuart/edm/edx and /mkedm) to obtain a CIAD file AB.1 (under $semwal/ciaduseful). Now, the file can be displayed using 'edx'. It must be noted, that because of the high resolution of the World Data Bank II, many points on the VDP-screen overlap. For this reason, this file has not been used for comparison [Appendix III]. The generation of list of polygons [illegible] done by using file 'SEMW:w.dt2' as 'data' file. This da[illegible]e contained only two triangular polygons.

' World Data Bank II: In[illegible]ructio[illegible]s for use T128.0382, Computing Services, U. Of A., Edmonton.

APPENDIX III.

All the following programs have been written in 'C' and are under $semwal/ciaduseful on PDP-11/45.

Program 'RasterToRun'

The program can be used to obtain the run-length code of a binary image displayed on the VDP screen. This is useful in obtaining a run-length of CIAD files (e.g., $stuart/edm/semwal/MAP18).

The file MAP18 can be displayed on the VDP-screen using CIAD file editor 'edx' (under $ stuart/edm/semwal). Now, the run command:

```
$RasterToQuad
```

is used and the program prompts for the file name. In this way, file 'R.MAP18' (under $semwal/ciaduseful) is obtained. This file is the run length of area of size 510 pixels by 480 pixels (excluding the menu-area).

Program 'ShowRun'

This program can be used to display the run length encoded file on the VDP screen. The run command:

```
$ShowRun
```

prompts for the input file name (file 'R.MAP18' can be specified). The '0' ('1') information is displayed in color 'blue' ('red').

Program 'RasterToQuad'

The program can be used to obtain a quad-tree of a binary-image on the VDP screen. The run command:

```
$RastertoQuad
```

is used and the program prompts for the file name. In this file, the coordinates of diagonally opposite corners of homogeneous square regions are obtained. The intensity value (zero or one) is also specified. The file 'Q.MAP18' (under $semwal/ciaduseful) is such a file for the binary image obtained using 'edx' to display the file MAP18 [as in RasterToRun]. The quad-tree corresponds to an area of size 512 pixels by 512 pixels. The menu-area starts at (509, 1) and the width of the VDP is 480 pixels so the area (509, 1) to (512, 512) and (1, 480) to (512, 512) is considered to be of uniform (0) intensity. Now, recursive partitioning of the region is done to obtain a quad-tree [Section 2.3.10].

Program 'ShowQuad'

The program can be used to display the quad-tree from the file which is obtained after running 'RasterToQuad' program. This can be used to verify the correctness of the conversion from raster to quad-tree representation. The run command is:

```
$ShowQuad
```

and the program prompts for the file name ('Q.MAP18'). The squares with zero (one) intensity are displayed in color red (blue).

Observation

The space required to represent an image of size (640 pixels by 480 pixels), in raster format, would take .3M bytes approximately (considering 1 byte per pixel). The MAP18 is in CIAD file format and take .2M bytes, approximately. The file Q.MAP18 requires almost .9M bytes for the quad-tree and run-length coded file for the same image takes .14M bytes approximately. The results are as expected [Sections 4.1]

Other CIAD files PIC5 (.19 M bytes) and PIC6 (.11 M bytes) under $semwal/ciaduseful on PDP-11/45 were run length encoded resulting in R.PIC5 (.145 M bytes) and R.PIC6 (.083 M bytes). These files were also encoded into quadtrees, as explained above, to obtain files Q.PIC5 (.837 M bytes) and Q.PIC6 (.6 M bytes). All these files are under $semwal/ciaduseful on PDP-11/45.

APPENDIX IV.

Let O be the octagon with n points and O1 and O2 be two octagons such that the number of points in the hexagons and rectangles between O1 and O2 be H1, H2, H3, H4 and R1, R2, R3, R4 respectively.

Let the points inside O2 be P, and, Q the points between the area O and O1. The desired octagon (O3) should be such that the points P and the points in the hexagons and rectangles between the octagons O and O3 (shown by dotted lines in the Figure A.1) is not more than n/2 points. One of the ways to obtain the desired octagon O3 is to decrease/ increase the side lengths of O2. The desired distribution of the points can also be achieved by varying the side lengths, angles or orientation of the octagons in the range of O1 and O2. In Figure A.1, one of the side of octagon is decreased as shown by dotted line.

It should be noted that changing the orientation and angles of the octagon may result in the formation of trapezoids and/ or parallelograms instead of rectangles.

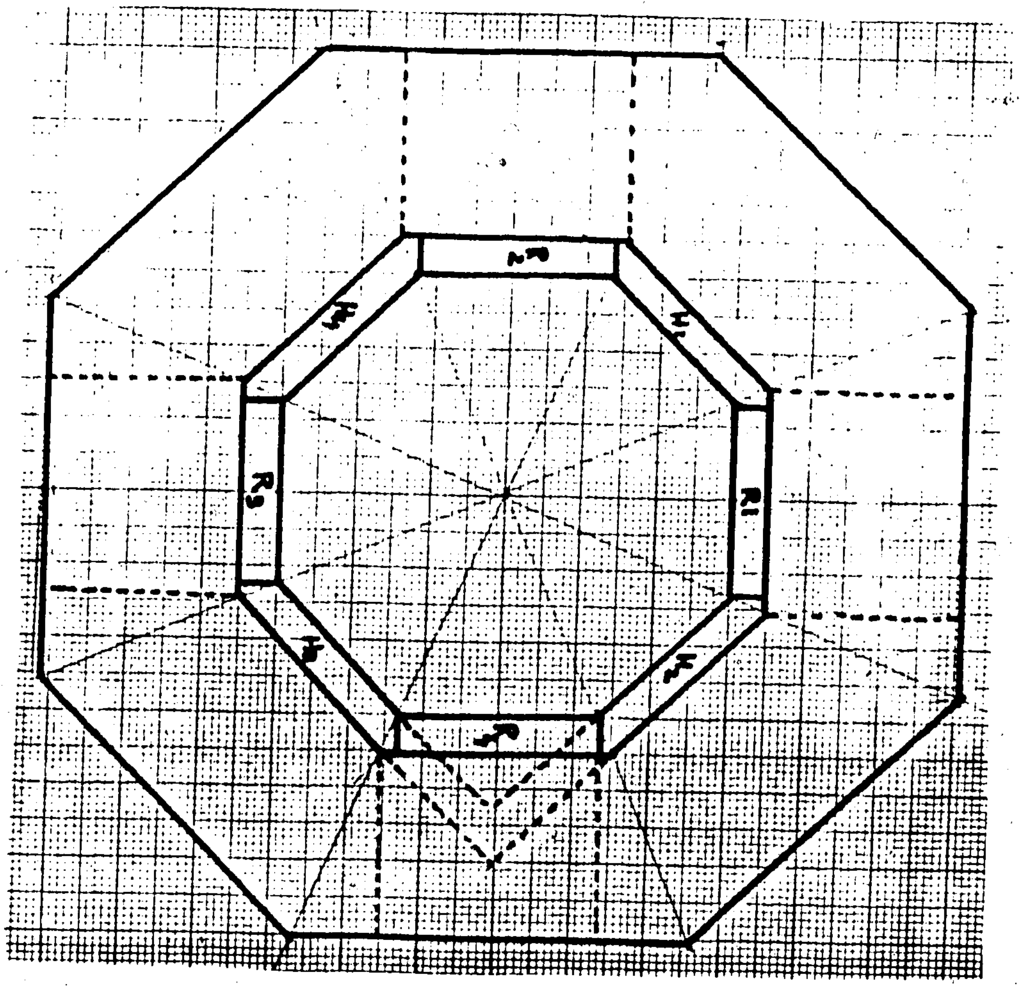


Figure A.1: Shows decrease in side length of octagonal O2 for obtaining the desired octagon O3.

APPENDIX V.

Let an image consist of 'n' points and 'q' lines. In this section, pseudo Pascal code for algorithms, discussed in Chapter 3, has been given. For each terminating partition a link list, containing the pointers to the entries in the line-list, is maintained. In the line list, back pointers to the entry in the link-list for two terminating partitions are stored as shown in Figure 3.7. Number of entries in the line-list and link-list is bounded by 'q' and 'n-1', respectively.

```
/************************************
* FOLLOWING IS THE PSEUDO CODE     *
* FOR ALGORITHMS IN CHAPTER 3      *
************************************/

    ptr_link = @link_list ;
     link_list = record ;
           p1_ptr : ptr_link ;
           n1_ptr : ptr_link ;
           v1_ptr : ptr_line
          end;                        /* link list record */
    ptr_line = @line_list ;
     line_list = record ;
           p2_ptr   : ptr_line ;
           n2_ptr   : ptr_line ;
           v2_ptr_f : ptr_link ;
           v2_ptr_l : ptr_link
```

```
      end;                          /* line list record */
   ptr_hex = @hex ;
      hex  = record ;
         RH_1 : ptr_rect ;
         RH_2 : ptr_rect ;
         HH_1 : ptr_hex  ;
         HH_2 : ptr_hex
         D1   : array of   /* storing the dimension */
                  numbers ; /* of hexagon            */
      end ;                /* hexagonal region record */
   ptr_rect = @rect ;
      rect = record ;
         R_son : ptr_rect ;
         L_son : ptr_rect ;
         D2  : array of  /* dimension of rectangle */
                  numbers  ;
      end ;                  /* rectangle record */
   ptr_term_R = @term_R ;  /* terminating rectangle */
      term_rect = record ;
         D3   : array of numbers ;  /* dimension */
         list_1 : link_list
      end ;
   ptr_term_H = @term_H ;      /* terminating hexagon */
      term_H  = record ;
         D4  : array of numbers ;  /* dimension */
         list_2 : link_list
      end ;
```

```
ptr_Oct = @Oct ; /* basic octagonal configuration */
     Oct = record ;
       R1 : ptr_rect ;
       R2 : ptr_rect ;
       R3 : ptr_rect ;
       R4 : ptr_rect ;
       H1 : ptr_hex  ;
       H2 : ptr_hex  ;
       H3 : ptr_hex  ;
       H4 : ptr_hex  ;
       Son_Oct : ptr_oct ;
       D5 : array of numbers  /* dimension */
     end;
ptr_term_oct: @term_oct; /* terminating octagon */
       D6 : array of numbers  /* dimension */
       list_3 : link_list
     end ;

var   Q1, Q2 end_list, first_list : ptr_link ;
      Q3, Q4, end_line, first_line: ptr_line ;

/* start of the line-list and link list and
   end of the line list and link-list are
   above declared.
 */

      truth  : boolean
```

```
/***********************************************
*  Algorithm Make_OTDS:                          *
*   Given an image of n points and q lines       *
*   this algorithm produces OTDS representation.*
************************************************/


Make_OTDS()


begin

  1. Find enclosing octagon [Appendix V].


  2. Do for non_terminating octagon (S)
         subdivide  S into 9 sub-regions
         such that no sub-region contains
         more than half the points in S.
         This results in
     a.  4 rectangles,
     b.  4 hexagons and
     c.  1 octagon (S1).


 3. subdivide each rectangle (R) in (a)
         such that two rectangular sub-regions
         contains half the points in R.
 4. subdivide each hexagon (H) in (b)
         such that two rectangular and two
```

```
        hexagonal sub-regions contains no
        more than half the points in H.
  5. repeat step 2, 3, 4 for S1.

 end;

/*******************************************
* Algorithm Delete_Point(P)               *
*      deletes a point in the OTDS. It also *
*      deletes the corresponding entries in *
*      the line-list for all the connected  *
*      points and link-list of connected    *
*      terminating partition.               *
*******************************************/

 Delete_point(P)
   begin
1. Find terminating partition for point P.
  /* terminating partition may be any of
     the hexagonal, rectangular or octagonal
     region.
  */
2. if (1) true then
     do  begin
       for all k pointers (Q1) in the link-list for T
           do begin
3.            Q3 := Q1@v1_ptr ;
```

```
   4.            Q2 := Q3@v2_ptr_1 ;
   5.           delete the entries pointed by Q1 and Q2
                 in the respective link-lists and Q3
                 in the line-list.
              end;
          end;
       end;
/* Time Complexity: Step 1 is logarithmic and step 2
    is linear in number of points. Hence, Delete_Point
    is linear in number of points as step 3, 4, 5
    takes constant time.
*/


/*****************************************************
* Algorithm Delete_Line(P1, P2)                       *
*    Deletes the line between two points P1 and P2    *
*****************************************************/

Delete_Line(P1, P2)
 Begin
     1. Find terminating partitions T1 and T2 for P1
        and P2 respectively.
     2. Truth := false ;
     3. For each pointer (Q1) in T1
           while not truth do
             begin
     4.        Q3 := Q1@v1_ptr ;
    5.     a. Q2 := Q3@v2_ptr_1 ;
           b. Q4 := Q2@v1_ptr ;
    6.       if (Q3 = Q4 ) then Truth := true
           end ;
    7. Step 5 of Delete_point(P) algorithm.
  end ;
/* Time Complexity: Step 1 is logarithmic and
        Step 3 may be linear in number of points
        in the worst case. Hence, the algorithm
        is linear in number of points in the image.
 */


/*****************************************
 *  Algorithm Add_Point(P)               *
 *       Adds a point P in the tree.     *
 *****************************************/


  Add_Point(P)
   Begin
    1. Find terminating partition (T) which may
       contain P.
    2. If T contains a point then subdivide as in
       step 2, 3, 4 of Make_OTDS.
    3. if P connected to k other existing points
        then for each of the k point
     do begin
        4. Find terminating partition T1
        5. Get new pointers :
              a. Q1 := new(ptr_link) for T ;
              b. Q2 := new(ptr_link) for T1 ;
              c. Q3 := new(ptr_line)
        6. Store the suitable pointer values in T and T1
              a. Q1@v1_link := Q3 ;
              b. Q2@v1_link := Q3 ;
              c. Q3@v2_ptr_f := Q1 ;
              d. Q3@v2_ptr_l := Q2 ;
           modify the next pointers of the link-list for
              T1 and T and line list to value nil.
        end;
    end;
/* Step 4 is logarithmic and may be repeated 'n-1' times
    in the worst case. Hence, the above algorithm is
    O(n log n).
 */


/************************************************
*  Add_Line(P1, P2)                             *
*       adds a line between P1 and P2.          *
************************************************/


 Add_line(P1, P2)
  Begin
     1. Do step 1 and 2 of Add_Point(P) for P1 and P2.
     2. For the terminating partitions T1 and T2
```

```
          for points P1 and P2  do step 5 and 6
          of Add_point(P) algorithm above.
    end;
/* The algorithm Add_Line is logarithmic as step 1
    is logarithmic in number of points and step 2
    takes constant time.
 */
```

END

27 06 86

FIN