CHAPTER 2

2D and 3D object representation schemes: A survey

Several data structures and representation schemes exist for the storage of 2D and 3D objects. Selecting the proper representation scheme for input data, intermediate structures and output data is a crucial factor for the construction of a good algorithm. A data structure may be chosen to represent a particular structural relationship, to save space or to allow for fast access of data. Different data structures have been developed to cater to the above requirements by many researchers. But no single data structure has all the desired properties. Hence, as a result, some more representation schemes have been developed. Depending on the need, one chooses the representation scheme that suits best. In view of the plethora of these representation schemes, to make the selection easy, a review of the important schemes of representation of 2D and 3D objects in the context of their construction, various operations that can be performed and their space and time complexities, is presented in this chapter. In section 2.1, the representation schemes for 2D objects is reviewed and in section 2.2, for 3D objects.

A few of the queries that could be addressed to any representation scheme are

(i) Point-membership: given an object representation and a point, to determine if the point lies inside the object.

(ii) Computation of the area (volume), perimeter (surface area)
for 2D (3D) objects.

(iii) The primitive set operations of intersection and union of two objects, A and B, complement of an object and the set difference of A and B. Using these primitive operations, many more complex image processing tasks can be performed.

(iv) Geometric transformations, namely, translation, rotation and scaling. Any geometric transformation can be obtained by a composition of these three operations.

(v) Connected component labeling: A connected component may be defined as a region in which any pair of points may be connected by a curve lying entirely inside the region. It is also desirable to count the connected components.

(vi) Euler number: Also called Genus, it may be defined as the number of holes subtracted from the number of connected components for a region. For a planar graph, it is defined as

\[ G = V - E + F, \]

where V, E and F are the number of vertices, edges and faces, respectively. But, for a binary image these can be interpreted as [Minsky et al 69]

\[
V = \text{number of black pixels} \\
E = \text{number of horizontal or vertical adjacent pairs of black pixels.} \\
F = \text{number of } 2 \times 2 \text{ blocks of black pixels.}
\]

Now let us look into various data structures for the storage of objects.
2.1 2D OBJECT REPRESENTATION

In this section, by an object, we mean a 2D object. The classification of various data structures, paradigms and representation schemes used for object representation is not simple because the dividing lines are not well drawn. The classification can be made depending on the object we would like to store or on the framework of the data structures used. In the second mode if the classification is based on the underlying approaches, this may be done depending on tree and non-tree structures or on those that are translation invariant and those that are not. But we follow the first mode, i.e., the classification based on the object because in a natural sequence the object comes first followed by the representation. A 2D-object can be visualized as having components of smaller dimensions. We classify the representation schemes based on the dimensionality of these components.

The classification is based on representations that store:

(i) point data, (0-dimension)
(ii) curvilinear data, (1-dimension) and
(iii) region data, (2-dimension)

2.1.0 POINT DATA

The storage, retrieval and ability to alter point data in a 2D region is very frequently used in geographic information systems (GIS). In a map city headquarters, administrative centres and important places are represented by points with appropriate coordinates. A sample query related to such a point data might be
"Find all the cities which are within 250 km of New Delhi and south of Dehradun".

Storing only the coordinates of cities is not desirable because this does not allow for an efficient query answering.

In the following subsections some of the data structures that store point data in a 2D image are presented. In this subsection the different methods of representation of point data are discussed.

2.1.0.1 POINT QUADTREES

The term point quadtree is employed to denote the quadtree as defined by Finkel and Bentley [74] and to avoid confusion with other types of quadtrees, which are presented in following sections (the history of quadtrees is also deferred).

The Point Quadtree, a tree with outdegree 4, is a divide-and-conquer paradigm that divides the image into four quadrants. In the point quadtree, the location of records with 2D keys is stored in the nodes. Each node stores one record and may have at most four sons. The root of the tree corresponds to the entire universe and its four sons divide the universe into four quadrants, viz. NE, NW, SW, and SE with the obvious meaning. Here it is assumed that the NE and SW sons are closed and the NW and SE are open (in the sense of point-set topology).

In a point quadtree, inserting a new record \((x,y)\) is trivial, and can be done in \(O(\log n)\) time, where \(n\) is the number of records [Finkel and Bentley 74]. At each node, a comparison is made and the correct subtree is chosen for the next test; upon falling out of the tree, the record is inserted at the proper
Fig. 2.1: A point quadtree and the records it represents (a) Numbering for the sons, (b) cities (not to scale) and (c) the representation.
place. For instance, if the record \((x,y)\) is to be inserted and coordinates of the record stored in the leaf node are \((p,q)\), then the record with \((x,y)\) will be

\[
\begin{align*}
\text{NE} & \text{ son if } x \geq p \text{ and } y \geq q \\
\text{NW} & \text{ son if } x < p \text{ and } y > q \\
\text{SW} & \text{ son if } x \leq p \text{ and } y \leq q \\
\text{SE} & \text{ son if } x > p \text{ and } y < q 
\end{align*}
\]

Search queries are of two types - point search and region search. A point search could be like "Determine the address of the node denoting the point (94,61), if it exists". The second, a region search, is invoked by "What are all the cities, within a circle of radius 270 km, centered at (77,62) ?". The point search basically is a tree traversal and can be accomplished in \(O(\log n)\) time, if there are \(n\) records. However range search takes \(O(n \log n)\) time.

The deletion of records from point quadtrees turns out to be very difficult to perform. The difficulty lies in deciding what is to be done with the subtrees attached to the deleted node. They are to be merged with the rest of the tree, which is not an easy process. The only way out, is to reinsert all of the stranded nodes, one by one, into the new tree. This operation of deleting a node is of \(O(n \log n)\), where \(n\) is the total number of nodes in the two trees to be merged.

The point quadtree is an efficient means of storage for 2D data, when the data is not very dynamic and searches are high. The basic concepts involved can easily be generalized to an arbitrary number of dimensions. The k-d tree is a step in this regard.
2.1.0.2 k-d TREES

The k-dimensional binary search tree [Bentley 75], k-d tree for short (this terminology is due to Donald.E.Knuth), is devised to organize a set of points in multidimensional space. In a 2D space, it becomes a 2-d tree. For the construction of a 2-d tree, the map space is recursively partitioned into rectangular blocks by a set of straight lines parallel to the axes. The partitioning process for the generation of 2-d tree is given below.

(a) The lengths of the horizontal and vertical sides of the rectangle to be partitioned are computed.

(b) Divide the rectangle into two by a straight line parallel to the shorter side. The position of the dividing line is determined so that each of the resultant smaller rectangles contain the same amount of data.

(c) To each resultant rectangle, (a) and (b) is recursively applied until the amount of data in any rectangle becomes "manageable".

The manageable amount of data depends on the application. Matsuyama et al [84] used the 2-d tree for storing data in a geographic information system (GIS), where they recursively divide the rectangles till the amount of data in any rectangle becomes less than the capacity of a memory page. An example of partitioning by 2-d tree where the capacity of a page is 2 is given in Fig.2.2

Here, a node in the 2-d tree represents a rectangular region generated by the partitioning process and contains the
position and the direction of the dividing line of the region. The root node denotes the whole map space, and, the leaves, the ultimate blocks. At the beginning, the tree is balanced, but as many records are inserted/deleted, the tree may become heavily unbalanced and may necessitate the reorganization. Friedman et al [77] have given an algorithm for finding best matches in $O(\log n)$ time, if there are $n$ records. Murphy and Selkow [86] used the k-d tree for efficiently retrieving from a file of fixed length binary key words the best match (in Hamming metric) to a given input word.

2.1.1 CURVILINEAR DATA

In this category, the boundary of the object is approximated by straight line segments and information relating to these segments is stored. Owing to inherent nature of the representational approach, computation of perimeter and performing geometric operations are done easily.
2.1.1.1 POLYLINES

A polygon is a closed polyline of a finite number of line segments. A polygon, \( P \), of \( N \) edges can be represented either by the equations of the straight lines or equivalently, by the coordinates of the vertices. Usually, the vertices in a cyclic order and the space requirement for storing a polygon of \( N \) sides by the vertices is of \( 2N \), whereas by the line equations it is \( 3N \). The vertices can be stored by means of an array or a circular linked list. This scheme of representation is also referred to as vector representation in the literature.

The computation of perimeter is by summing over the lengths of the line segments, joining the vertices, \((x_i, y_i), i = 0, 1, \ldots, N-1\).

\[
\sum_{i=0}^{N-1} \sqrt{(x_i-x_{i+1})^2 + (y_i-y_{i+1})^2}
\]

Here subscript calculations are modulo \( n \).

The area \( \Delta \) is given by

\[
\Delta = \frac{1}{2} \sum (x_{i+1}y_i - x_iy_{i+1}).
\]

Since the three basic geometric transformations are homomorph, i.e., for a transformation,

\[
T : \mathbb{R}^2 \longrightarrow \mathbb{R}^2
\]

\[
T(X+Y) = T(X) + T(Y), \; X, Y \in \mathbb{R}^2,
\]

the transformations can be applied on the vertices and the transformed vertices can be joined to obtain the transformed polygon. The homogeneous coordinates [Newman et al 84] are used to represent transformations by matrices. For this, the dummy variable, \( w \), is made use of. In the new coordinate system \((x,y)\) becomes \((xw, yw, w)\). For our purposes, \( w \) is assigned \( 1 \). Hence a
translation by \((T_x, T_y)\), in the new system would be
\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
T_x & T_y & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
1
\end{pmatrix}
= \begin{pmatrix}
x+T_x \\
y+T_y \\
1
\end{pmatrix}.
\]

Rotation of \(\theta\) degrees about the origin,
\[
(x, y) \rightarrow (x\cos\theta + y\sin\theta, -x\sin\theta + y\cos\theta)
\]
is given by
\[
\begin{pmatrix}
\cos\theta & -\sin\theta & 0 \\
\sin\theta & \cos\theta & 0 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
1
\end{pmatrix}
= \begin{pmatrix}
x\cos\theta + y\sin\theta, \\
-x\sin\theta + y\cos\theta, \\
1
\end{pmatrix}.
\]

Scaling, equivalently called zooming, is the change in size and proportion of the polygon. For the scaling by factor \(s\),
\[
(x, y) \quad \rightarrow \quad (x S_x, y S_y),
\]
the matrix form is
\[
\begin{pmatrix}
S_x & 0 & 0 \\
0 & S_y & 0 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
1
\end{pmatrix}
= \begin{pmatrix}
x S_x, \\
y S_y, \\
1
\end{pmatrix}.
\]
When \(S_x = S_y\), the polygon is enlarged or reduced depending on whether \(S_x\) is greater or less than 1. If \(S_x\) and \(S_y\) are different, the scaling has the effect of distorting polygons by elongating or shrinking them along directions parallel to the coordinate axes. The mirror images can be obtained by assigning negative values to \(S_x\) or \(S_y\).

The membership problem of a point \(p\) in a simple \(n\)-gon, \(P\), can be answered in \(O(n)\) time, without any preprocessing [Preperata et al. 85]. The algorithm is based on the Jordan Curve theorem. Consider the horizontal line \(l\) passing through \(p\). Assuming that \(l\) does not pass through any vertices of \(P\), if \(L\), the number of intersection of \(l\) with the boundary of \(P\) to the left of \(P\), is odd, the point is internal. When \(l\) passes through
one of the vertices of P, a small rotation of \( l \) around \( p \) removes the degeneracy.

\[
\begin{align*}
\text{function \textsc{member}(p, P)} & \quad \text{begin} \\
& \quad \quad L = 0 \\
& \quad \quad \text{for } i = 1 \text{ to } N \text{ do} \\
& \quad \quad \quad \text{begin} \\
& \quad \quad \quad \quad \text{if (edge \( (i) \) is not horizontal) then} \\
& \quad \quad \quad \quad \quad \text{if (lower extreme of edge \( (i) \) intersects to the left of \( p \))} \\
& \quad \quad \quad \quad \quad \quad \text{then } L = L + 1 \\
& \quad \quad \quad \quad \text{end} \\
& \quad \quad \quad \text{if (\( L \) is odd) then \( p \) is internal else \( p \) is external.} \\
& \quad \quad \text{end.}
\end{align*}
\]

Algorithm 2.1

Using the vector representation, one can reconstruct the polygon exactly, but the display on the monitor is effected by the pixel resolution of the graphic monitor. Besides, this representation is dynamic, i.e., vertices can be added, deleted or modified easily by adding, deleting or updating the corresponding nodes in the linked list.

2.1.1.2 CHAIN CODES

The chain code representation [Freeman 74] is a very commonly used scheme in cartographic applications. This is also known as the boundary or border code.

The Chain codes consist of line segments, one unit long, that lie on a fixed grid with a fixed set of possible orientations. Only the starting point is represented by its location. The coordinates of other boundary points are computed relative to this point. The chain code is a sequence of unit vectors (i.e., one pixel wide) in the four principal directions. The directions are represented by numbers: for example, let the
integer \( i, 0 \leq i \leq 3 \), represent the direction of 90 degrees. By traversing along the boundary from starting point in clockwise direction, the chain code is generated.

![Diagram of a grid and chain code](image)

111100110122222323333330000
\quad (c)

1000103013300053103000005000
\quad (d)

Fig. 2.3 i An object with the grid (a), the directional codes (b), the chain code (c), and its derivative (d).

This is generally written as \( 1^4 0^2 1^2 0^1 2^5 3^2 2^3 6^0 4 \). A power of \( i \) to the directional code indicates that directional code is repeated \( i \) times. For a better approximation of the curve instead of four directions only, one may choose six or eight or even more number of directions.

The Chain codes facilitate trivial computation of the perimeter by adding up all the powers. Using \((x,y)\) as the starting point on the boundary, the area can be computed using the function \(\text{AREA}(\ )\).
Algorithm 2.2

The chain codes allow for representation of planar curves by a simple scheme with minimal storage requirements. Besides, the chain codes are often used in place of original data. The operations of rotation, computation of width and height, testing for symmetry about an axis and the determination of area enclosed or under a chain code can be performed on them [Shapiro 79]. The derivative of a chain code is defined as the sequence of numbers indicating the relative direction of chain code segments; the number of left hand turns of $\pi/4$ needed to reach the next segment. The chain code and its derivative are given in Fig. 2.3. The derivative is useful because of its invariance to boundary rotation.

They also simplify the detection of features of a region boundary, such as corners [Freeman and Davis 77] or concavities. On the other hand, they do not facilitate the determination of properties like elongatedness and process of set operations of intersection and union [Samet 84a]. A chain correlation function can be defined to give a measure of similarity between two curves. Since the coordinates of the boundary points are stored relative to the starting point, translation is trivial. It involves shifting of the starting point. Scaling by powers of 2
and rotations by multiples of 90 degrees are easy to perform.

2.1.1.3 STRIP TREES

The strip tree [Ballard 81] is a hierarchical representation of curves. It is constructed by successively approximating segments of the curve by the smallest enclosing rectangles. The root of the strip tree, a binary tree, represents the entire curve. The node structure consists of eight fields, wherein the first six fields define the rectangle and the remaining two are pointers for the two sons. The original curve is broken up into strips and each strip is defined by a six-tuple \((x_1, x_2, y_1, y_2, w_1, w_r)\) as shown Fig.2.4

![Diagram of strip tree](image)

**Fig. 2.4**: The six-tuple \((x_1, x_2, y_1, y_2, w_1, w_r)\) that defines the rectangle to store a strip.

![Diagram of curve and strip tree](image)

**Fig. 2.5**: A curve with the enclosing rectangles (a) and its strip tree (b).
Here $X=(x_1,x_2)$, $Y=(y_1,y_2)$ are the two end points of the curve and $w_1$ and $w_r$ help in defining the rectangle. The algorithm for constructing a strip tree from a curve is given in Alg. 2.3.

Determine the smallest rectangle with a side parallel to the line segment $[X_0,X_n]$, that just covers all the points. This rectangle corresponds to the root of the tree. Pick a point $X_i$, that touches one of the sides of the rectangle. This point is termed 'splitting point'. If there are more than one, select the point that is at the maximum distance from the line joining $X_0$ and the endpoints of the curve. The splitting point $X_i$ divides the list into two, $[X_0,X_1,...,X_i)$ and $(X_{i+1},X_{i+2},...,X_n)$. Repeat the process for the above two, which will be the left and right sons of the root. The process is repeated till the desired approximation is reached.

Algorithm 2.3

The half-closed half-open interval facilitates the succeeding computations. A curve and its strip tree are shown in Fig. 2.5.

To determine if two curves intersect, first determine if the corresponding strips do. In case, they do, the procedure is applied recursively till the primitive level is reached.

![Diagram](image)

Fig. 2.6: A closed curve is represented by a pair of strip trees.
function STRIP-INT(T1,T2)
/* T1 and T2 are the strip trees representing the two curves */
begin
  case : intersection type of two strips T1 and T2 of
    primitive : return (true)
    null : return (false)
    possible : if T2 is the "fatter" strip then
                return (STRIP-INT(T1, lson(T2)) or
                        STRIP-INT(T1, rson(T2))
                else
                return (STRIP-INT(lson(T1), T2) or
                        STRIP-INT(rson(T1), T2))
  endcase.
end.

Algorithm 2.4

The union of two strip trees may be defined as a strip that encloses both the root strips. If the two curves are defined by \( [X_1, X_2, \ldots, X_n] \) and \( [Y_1, Y_2, \ldots, Y_m] \), then these two are concatenated and the strip tree is built for the new list.

A region may be represented by its boundary as shown in Fig 2.6, by dividing the closed curve into two and storing each of the curves by a strip tree.

2.1.1.4 EDGE QUADTREES

The technique of quadsecting the image space can also be used for storing curvilinear data. But when quadtrees are used for storing curvilinear information, the majority of the quadrants are required to be divided up to the pixel level along the boundaries. The edge quadtree of Shneier [81a] is a step to overcome such high division along the edges. This is an improvement for storing linear feature information for an image (binary or gray-scale) in a manner similar to that used for
storing region information. As in region quadtrees (Section 2.1.2.5), an image containing a linear feature or part thereof is subdivided into four quadrants recursively until quadrants are obtained that contain a single curve that is approximated by a single straight line. Each leaf node contains the following information regarding the edge passing through it:

- magnitude (1 in case of binary image and intensity level in gray-scale)
- direction
- intercept
- directional error term (error induced by a straight line)

![Fig. 2.7: A polygon and the partition created by the edge quadtree representation.](image)

In case an edge terminates in a node, a special flag is set and intercept denotes the end point of the edge. This method minimizes the storage requirement, although in the vicinity of the vertices, the density of small leaves could be high.

Other data structures which have slight variations to the edge quadtree are the PR quadtree [Orenstein 82], the MX quadtree [Hunter and Steiglitz 79a] and the PM quadtrees [Samet and Webber 85].
Another step in this regard was made by Ayala et al [85]. For the representation of polygons by quadtrees, the node structure of the region quadtree is modified. Besides the usual black, white and gray nodes, they introduced a new type, called edge, i.e., a node type could be one of black, white, gray or edge. Every edge node has a field for storing a pointer to the edge crossing that quadrant. The edges are represented by a triple of real numbers \((a,b,c)\) corresponding to the three coefficients in the line equation \(ax + by + c = 0\) of the edge. The triplet \((a,b,c)\) is determined so that the points \((x,y)\) inside the polygon satisfy \(ax+by+c > 0\).

Set operations like complementation, union and intersection of two regions, construction of quadtree and display of the polygon from its representation, and conversion from quadtree to edge quadtree are presented. This idea has also been extended to build nonminimal division octrees with node types of black, white, gray, face, edge and vertex. The edge quadtree is shorter than the corresponding quadtree for the same polygon and moreover, the boundary model of the object can be exactly reconstructed.

2.1.2 REGION REPRESENTATION

In this subsection, some of the representation schemes are presented that store the interior of an object, as compared to boundary. Owing to the inherent two dimensionality of region information, more schemes based on representing the interior are proposed by various workers.
2.1.2.1 SPATIAL OCCUPANCY ARRAYS

The most natural representation for a region on a raster is the membership predicate (or characteristic function of the region) \( p \), defined by

\[
p(x, y) = \begin{cases} 
1 & \text{if pixel at } (x, y) \text{ is in the region} \\
0 & \text{otherwise.}
\end{cases}
\]

A raster of size \( n \times n \) is represented by a binary square matrix of order \( n \), where 0 indicates the background and 1, the presence of the object pixel or foreground. When the image is colored or has gray shades, different indices may be given.

Queries regarding point membership problem and area can be answered immediately. Two regions can be merged easily and operations of intersection and union, performed with the use of pixel-wise AND and OR, respectively, are trivial. Space requirement for the storage of an image in an \( n \times n \) matrix is easily seen to be \( O(n^2) \). But if one employs any programming language that offers bit-wise operators, the space requirement can be reduced drastically. To compute the perimeter, not stored here explicitly, one has to take the aid of contour/boundary tracing algorithms. [Rosenfeld and Kak, 82].

2.1.2.2 Y-AXIS

To reduce the high space requirement of the spatial occupancy arrays without sacrificing nicety of the algorithms for merging, union and intersection, the Y-axis [Merrill 73] (also called run length code [Rutowitz 68]) representation is proposed. This is a list of lists. Each element in the main list corresponds to a row in the raster image. For each row, as one
moves in the increasing X direction, the X pairs of coordinates at which the image starts and ends are encoded. If there is a lone pixel at \((x_0,y_0)\), it is encoded as \((y_0 \ x_0 \ x_0)\). A region and its Y-axis representation is shown in Fig. 2.8. Here the first element of each sublist is the Y-coordinate, followed by pair(s) of 'into' and 'outof' X-coordinates.

( (2 2 3) (3 2 5) (4 2 6) (5 2 2 6 6) )

Fig. 2.8 : The Y-axis representation of a region.

The intersection and union are implemented as merge-like operations, which take time linearly proportional to the number of non-empty rows. The union operation amounts to a merge of X-pixels along rows organized within a merge of rows themselves. Two regions A and B and the regions \(A \cup B\) and \(A \cap B\), with their Y-axis representations are shown in Fig. 2.9.

If the region is thin, long and parallel to the Y-axis, then its representation is not space efficient. In such cases, one may use X-axis representation, a variation of the current scheme, with the obvious meaning. At the cost of convenience, one may work using a combination of X-axis and Y-axis modes, which
Fig. 2.9: Two regions A and B and the set operations of $A \cup B$ and $A \cap B$ and their y-axis representations.
presents no conceptual difficulties. Even then, to represent a 'chess-board' like pattern takes much space.

2.1.2.3 BINTREES

The bintree [Knowlton 80], is a hierarchical representation of 2D region that uses a binary tree. The region is contained in a spatial occupancy array of size $N$, ($N = 2^n$, for some $n$).

A bintree is based on recursive subdivision of the image array into halves alternating between X and Y axes. If the array is not consisted entirely of 1's or entirely 0's, it is subdivided into halves, subhalves etc. until the blocks so obtained are of only 1's or 0's. i.e., a block is either completely inside or disjoint from it.

Fig. 2.10: An object (a) and its bintree representation (b).
The root node corresponds to the entire array. Each son of
a node represents half of the region represented by that node.
The leaf nodes of the tree correspond to those blocks for which
no further subdivision is necessary. A leaf node is said to be
black or white, depending on whether its corresponding block is
entirely inside or entirely outside the region. A non-leaf node
is said to be a gray node.

A node in a bintree has got three fields: two pointers for
the sons and a field TYPE, which is one of the three values
black, white or gray. To represent an $N \times N$ image, in the worst
case, the height of the bintree would be $\log N$, i.e., $n$. A chess-
board like image gives rise to a complete bintree.

All the algorithms for answering various queries boil down
to tree traversals. The intersection and union of two regions are
obtained by traversing the corresponding trees parallelly.
Intersection procedure is explained here.

The intersection is done by examining the nodes of each
input tree once, visiting nodes in each tree with a stride chosen
to keep the two scans 'in step'. Thus if at a certain stage, the
next node in each of the trees is a leaf node, the leaves are
intersected. If one has a nonterminal node and the other a
terminal, then the first scan recurses while the second waits;
the intersection in these cases is carried out between nodes of
different sizes. If both trees have non-terminal nodes, then both
scans recurse.

Complementation is trivial. For each leaf node, the TYPE
is changed from black to white and vice-versa. The internal nodes
are not modified. Samet and Tamminen[84] present an algorithm for
efficiently labeling the connected components in a region using bintrees.

2.1.2.4 PYRAMIDS

The Pyramid [Tanimoto and Pavlidis 74] is a hierarchical representation of an image organized into layers, each successive layer representing a finer resolution. To represent an \( N \times N \) image array, \( N = 2^n \), a pyramid is a sequence of arrays \( \{P(i)\} \) such that \( P(i-1) \) is a version of \( P(i) \) at half the resolution of \( P(i) \), for \( 0 \leq i \leq n \). \( P(0) \) is a single pixel.

In view of the hierarchical structure of the pyramid it is more convenient to think of the pyramid as a complete quadtree [Knuth 68] (Section 2.1.2.5). Starting from the \( N \times N \) image, a recursive decomposition into quadrants is performed till we reach

![Diagram of a pyramid](image)

Fig 2.11: The successive layers of a pyramid. At the top level of the pyramid, the single pixel represents the entire image. At the next level, the four cells represent the partitioning of the image into four equal parts and so on.
the individual pixels. The leaf nodes of the resulting tree represent the pixels. In the non-terminal nodes, the average gray level value of its four sons is stored.

If $A$ is the area of the image in terms of pixels, then the space requirement for the pyramid is $4A/3$ and for production of the full set of lower resolution arrays, $A$ additions and $A/3$ shifts are necessary. Pyramids are used for feature detection and extraction since they can be used to limit the scope of the search. Once a piece of region of interest is found at a coarse level, the finer resolution levels can be searched. Davis and Roussopoulos [80] used this approach for approximating visual pattern matching. Pyramids are also used for encoding information about curves, lines and edges in an image [Shneier 81a]. Algorithms for edge finding, region growing, and texture analysis have been defined as sequences of parallel operations applied up and down the levels of the tree. The pyramid being a multiresolution representation, enables one to design algorithms that work on several levels of resolution either in parallel or in a controlled sequence.

2.1.2.5 REGION QUADTREES

In the recent times, no other data structure has received as much attention as did region quadtrees. The region quadtree, (hereafter referred to as quadtree) a tree with outdegree four, is based on the principle of recursive decomposition, a divide and conquer method [Aho et al 74]. The term quadtree was first used by Finkel and Bentley [74], where they used the quadtree for partitioning the space into rectangular quadrants.
But the concept of recursively partitioning the space into four equal parts was used earlier by Warnock [69] for implementing a hidden surface elimination algorithm for display of images on the monitor. But later, the meaning of quadtree has changed. The term as it is understood today may be attributed to Klinger [71], Klinger and Dyer [76] who used the term quadtrie, whereas it was Hunter [78] who used the specific term quadtree in such contexts as we use today.

The study of quadtrees may be broken up into three parts
i) pointered quadtrees
ii) pointerless quadtrees
iii) semi-pointered quadtrees

2.1.2.5.1 POINTERED QUADTREES

Owing to the inherent recursive process of dividing the region, the tree structure is best suited for storing a quadtree. In this subsection, some of the tree structures with explicit pointers between nodes are considered. Here we mean by quadtree, the pointered quadtree.

The following terminology is being used for the present work.

The term region quadtree, due to Samet [84a], is used to distinguish this type of quadtree from the point quadtree (Section 2.1.0.1). By image, we mean the 2D object we are interested in. The border of the image is the outer boundary of the square, of side $N$, corresponding to the array. $n (= \log_2 N)$ is the resolution. Two pixels are 4-adjacent if they are adjacent to each other either in the vertical or horizontal directions. Two
pixels are 8-adjacent, if the concept of adjacency includes diagonal adjacency too. A black region is a maximal 4-connected set of black pixels. i.e., a set \( R \) such that for any two pixels \( p \) and \( q \) in \( R \), we can find a sequence of black pixels \( p = p_0, p_1, p_2, \ldots, p_l = q \) in \( R \) such that \( p_i \) is 4-adjacent to \( p_{i+1}, 0 \leq i < l \). A black region is also called a connected component. A white region is a maximal eight-connected set of white pixels. A pixel is a quadrant of unit length. The boundary of the black region consists of the set of edges of its constituent pixels.

The construction of a quadtree is based on the successive subdivision of the raster image into four equal-sized quadrants. The image is embedded in a square array of size \( N, N = 2^n \) for some \( n \), the resolution, where 1's represent the image and 0's the background. (Here our discussion are limited to binary images i.e., a pixel is either black or white). A block or quadrant is said to be homogeneous if it consists entirely of 1's or entirely of 0's. Otherwise it is said to be non-homogeneous. If the original binary array is non-homogeneous, then it is subdivided into four quadrants. Then each of these four quadrants is checked for homogeneity. The non-homogeneous quadrants, if any, are again subdivided and this process continues till all the quadrants are homogeneous. The root node corresponds to the entire array. Each son of a node represents a quadrant (labeled in order of NW, NE, SW and SE) of the region represented by that node. The leaves represent those quadrants which are homogenous. A leaf node is said to be black or white, according as its corresponding block in the image is made up of 1's or 0's. All internal nodes are said to be gray. Samet [80a] presents an algorithm for
Fig. 2.12 : The ordering of the quadrants (a), an object (b) and its region quadtree (c).

constructing a quadtree from the array representation of a binary image. The execution time is of the order of number of pixels in the image.

A node in a quadtree has the following structure.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>FATHER</th>
<th>NW</th>
<th>NE</th>
<th>SW</th>
<th>SE</th>
</tr>
</thead>
</table>

TYPE is one of black, white or gray depending on whether the quadrant of the image which the node represents is completely within the image, disjoint from it or neither. FATHER is a pointer to the father node. NW, NE, SW and SE are pointers to the north-west, north-east, south-west and south-east sons of the
node, respectively.

Given a node P and a son I, these fields are referenced as FATHER(P) and SON (P,I) respectively. We use the boolean functions BLACK, WHITE and GRAY to access the TYPE field of a node, which is true if the function name and the value accessed are same.

The quadtree, in view of its tree structure, is well suited for performing many recursive algorithms. This not only makes it easier to program but also allows clear visualization of the concepts. Most of the algorithms, to follow, reduce to tree traversals. Now we present algorithms for set operations (union, intersection and complement), computing areas, moments and centroid, connected component labeling and counting, transformation and determining the perimeter.

Set Operations

Given two images R1 and R2, represented by the corresponding quadtrees, QT1 and QT2, algorithms are presented [Shneier 81b, Hunter 78, Hunter and Steiglitz 79a] to generate the resulting quadtrees obtained by performing union and intersection of QT1 and QT2. Also, to generate the quadtree of set complement of QT1. It is to be noted here that given complementation and one of intersection and union, the other operations can be derived. For any two sets A and B, it is well known that

\[ A \cap B = B - (B - A) \]
\[ A \cup B = (A^c \cap B^c)^c \]
\[ A - B = A \cap (A \land B)^c \]
'SONS' is used to denote the set, \(\{ NW, NE, SW, SE \}\), of the four sons of a node.

**Complementation** Generating the complement of an image involves changing black pixels into white and vice-versa, i.e., changing black nodes in the corresponding quadtree to white and vice-versa.

\[
\text{function COMPLEMENT (QT1)} \\
\text{begin} \\
\quad \text{if GRAY(QT1) then} \\
\quad \quad \text{for i in SONS do} \\
\quad \quad \quad \text{COMPLEMENT (SON(QT1,i))} \\
\quad \text{else} \\
\quad \quad \text{if (BLACK(QT1)) then TYPE (QT1)=white} \\
\quad \quad \quad \text{else TYPE (QT1)=black.} \\
\text{end} \\
\text{end}
\]

Algorithm 2.5

If the numerical values of 0, 1 and 2 are given to denote white, black and gray types respectively, in TYPE field, the above algorithm becomes simple, which is presented in Alg. 2.6.

\[
\text{COMPLEMENT1 (QT1)} \\
\text{begin} \\
\quad \text{if TYPE (QT1)=2 then} \\
\quad \quad \text{for i in SONS do} \\
\quad \quad \quad \text{COMPLEMENT 1 (SON(QT1,i))} \\
\quad \text{else} \\
\quad \quad \text{TYPE(QT1)=1-TYPE(QT1)} \\
\text{end} \\
\text{end.}
\]

Algorithm 2.6

**Intersection**

This algorithm finds the logical AND of two images \(R1\) and \(R2\) represented by their quadtrees \(QT1\) and \(QT2\) respectively,
traversing the two trees in parallel. When one tree has a black son and the other a non-black son, then the black son is replaced by the corresponding node. If one of tree has a white node, the intersection will have a white leaf at the corresponding position. Finally, if both the trees have gray nodes in the corresponding positions, the algorithm recurses. At the end, the pointer to the root of the resulting quadtree is returned.

function INTSECT (QT1, QT2)
begin
  if BLACK(QT1) then return (COPY(QT2))
  else if BLACK(QT2) then return (COPY(QT1))
  INTSECT = CREAT-NODE().
  for i in SONS do
    SON(INTSECT,i) = INTSECT(SON(QT2,i))
    FATHER(SON(INTSECT,i)) = INTSECT
  end.
  return (INTSECT)
end.

Algorithm 2.7

The function INTSECT returns a pointer to a node in a quadtree. The function COPY(QT) creates a tree structure identical to QT and returns the pointer of the root of the new tree.

function COPY(QT)
begin
  NEWQT = CREAT-NODE()
  TYPE(NEWQT) = TYPE(QT)
  for i in SONS do
    if (SON(QT,i)) = NULL then SON(NEWQT,i) = NULL.
  else
    begin
      SON(NEWQT,i) = COPY(SON(QT,i))
      FATHER(SON(NEWQT,i)) = NEWQT.
    end.
  return (NEWQT)
end.

Algorithm 2.8
Union

The union algorithm is identical to the intersection algorithm with the roles of white and black nodes swapped. Here too, the trees are parallelly traversed and a decision is made when a leaf node is reached.

```
function UNION (QT1,QT2)
begin
   if BLACK (QT1) then COPY (QT2)
   else if BLACK (QT2) then COPY (QT1)
   UNI = CREATE-NODE ( )
   for I in SONs do
      begin
         SON (UNI,I) = UNION (SON(QT1,I),SON(QT2,I))
      end.
   FATHER (SON(UNI,1))=UNI
   end.
return (UNI)
end.
```

Algorithm 2.9

The union and intersection algorithms can be generalized to handle any number of quadtrees [Shneier 81b]. The intersection algorithm takes time proportional to the traversal time of the smallest tree whereas the union algorithm requires that of the second largest tree (when more than two trees are considered). The ability to perform set operations quickly is one of the major reasons for the popularity of the quadtrees over other representations [Samet 84a].

Burton et al [87] have presented a single quadtree overlay function that can perform a number of common quadtree operations including union, intersection, difference, masking, copy, complement and map generalization. They assert that a slightly more complicated version of this function ensures the full
advantage of the special characteristics of each of their operations and performs the computations in a manner which is optimal to within a constant factor with respect to both space and time.

**Area and Centroid**

Area and centroid [Shneier 81b] for images represented by quadtree is simple to compute. It involves the postorder traversal of a quadtree. For computation of area, as the tree is traversed, the areas of black leaves are accumulated.

```c
function AREA (QT,n)
begin
  if GRAY (QT) then
    for I in SONS do
      black_area = black_area + AREA (SON(QT,I),n-1)
  else
    if BLACK (QT) then
      black_area=black_area + 2^n
    return (black_area)
end.
```

**Algorithm 2.10**

The arithmetic mean of the coordinates of all the black pixels gives the centroid, which is same as the arithmetic mean of the centroids of all black leaves. The position of each black block is easy to ascertain from the path that was taken to reach that block.

Both the algorithms have a time complexity that is of \( O(T) \), where \( T \) is the total number of nodes in the quadtree.

**Geometric Transformations**

A major motivation for the development of the quadtree concept is the desire to provide an efficient data structure for computer graphics. Quadtrees are also used in graphics and
animation [Hunter 77, Negroponte 77, Newman et al 84] which are oriented towards construction of images from polygons and superposition of images. Encoded pictures are specially useful for display [Hunter 78] if the encoding is well suited for processing of the image.

Suppose we have the quadtree representation of an image and a general linear transformation

\[ F : (x, y) \rightarrow (a_1x + a_2y + a_3, b_1x + b_2y + b_3) \]

where \( a_i, b_i, 1 \leq i \leq 3 \), are real constants. The aim is to efficiently generate the quadtree of the transformed image. In the transformation \( F \), setting appropriate values to the constants, the particular transformations of translation, scaling and rotation are obtained.

To achieve this aim, a brute force method would be to decode the quadtree to get the raster image [Samet 84b], apply

![Input Picture](image1)

![Output Picture](image2)

**Fig. 2.13**: The inverse of the transformation is applied to the input picture and the part that is of interest is clipped. The dotted square is the inverse image of the transformation.
the transformation pixel-wise and then encode the raster in a quadtrees form [Samet 80a, 81b]. This, as easily observed, is a lengthy and time taking process.

Hunter and Steiglitz [79b] have given an algorithm that requires time and space of $O(T+p)$, where $T$ is the total number of nodes in the quadtrees and $p$ is the perimeter of the non-background visible portions of the multicolored image. Here, the image may consist of more than one, possibly overlapping, polygon and may have holes. The algorithm is briefly explained below.

We have an $N \times N$ square array of pixels, the corresponding quadtrees and a general transformation $F$. First the nodes which are 'visible' after the transformation are identified by computing the edge of visibility i.e., the inverse image of the output picture boundary. All the visible edges of nodes are stacked. Pop the stack until a leaf-side is found with a visible portion which lies between two different colors or which lies on the edge of the picture and has a foreground-colored leaf. Now, following the boundary of the connected component containing this leaf, generate the polygon formed by edges of leaves. For each polygon, called primary polygon, so outlined, each edge is transformed by $F$ and the transformed polygon is encoded into a quadtrees. Coloring is done by recursive spreading of color from boundary leaves to neighboring leaves. The interior polygons are outlined, transformed and a quadtrees for each is generated. Now the interior polygons are cut out of the primary polygon by superposing them as holes in the quadtrees for the primary polygon.
The process of popping the stack, finding an inter color edge and building quadtrees, with holes if any, for polygons found is repeated till the stack is empty. Superposing the quadtrees for the transformed polygons will result in the desired transformed quadtree.

In this paper by [Hunter and Steiglitz 79b], the algorithm is presented for a netted quadtree. A net is a linked list whose elements are all the nodes that are adjacent along a given side of a node. But the central idea of the algorithm holds good even for quadtrees, where the neighbors can be determined by neighbor finding techniques, proposed by Samet [82a, 85a].

Peters [85] restricted the class of transformations that transform squares into convex quadrangles (operations like translation, scaling and rotation own this property), and proposed an algorithm which has $O(T+m(n+1))$ time and $O(T+n+m)$ space complexities, where $T$ and $m$ are the total number of nodes in the input quadtree, and an intermediate quadtree and $n$ is the resolution factor. The number of nodes in the resultant quadtree does not exceed $m$.

Scaling the image by a power of two is an easy process when using quadtrees, for, it is simply a modification of the resolution [Tanimoto 76]. So is the rotation by multiples of $90^\circ$, since it involves a recursive rotation of sons at each level of the quadtree.

Barring the few transformations (scaling by a power of two, translations and rotations in multiples of $90^\circ$), all the other transformations have an inherent shortcoming in them. The resulting transformed picture is only an approximation. Straight
lines are not necessarily transformed into straight lines. This is because of the underlying digitization process. It manifests itself, in any representation scheme involving raster graphics.

Perimeter

The computation of perimeter [Samet 81a] involves the postorder traversal of the quadtree and requires $O(T)$ time, $T$, the total number of nodes in the quadtree.

In the postorder tree traversal, for each black node that is encountered, its four adjacent sides are explored to determine if any adjacent nodes are white. For each of the existing adjacent white nodes, if any, the length of the corresponding shared side is accumulated in the perimeter.

This will result in a certain amount of redundant effort because each adjacency between two black blocks is explored twice, without any further addition to the perimeter. A way out is to explore only for southern and eastern neighbors. In this modified approach [Samet 84a], the northern and western boundaries of the image are never explored, and this is alleviated by embedding the image in white region.

Jackins and Tanimoto [83] have developed an asymptotically faster algorithm for perimeter computation that works for an arbitrary number of dimensions.

Connected Component Labeling and Counting

Connected component labeling is one of the crucial operations for any image processing system. Samet [81c] presents a three step process to label the components.

(i) Traverse the tree in postorder. For each black node,
say BN, determine all the adjacent black nodes on the southern and eastern sides and assign the same label as that of BN. (It is possible that a node may be assigned the label more than once).

(ii) Merge all the equivalences generated in the first step.

(iii) Traverse the tree again and update the labels on the nodes to reflect the equivalences generated in the first two steps.

The algorithm has an average execution time that is of $O(B \log B)$, B, the number of black nodes in the quadtree. Component counting is a consequence of labeling. The number of different equivalence classes resulting is step (ii) is the required number.

A few remarks are made below on the space and time complexities of quadtrees. To reduce the amount of space necessary to store data through the use of aggregation of homogeneous blocks is the prime motivation for the development of quadtrees. For a simple polygon, i.e., a polygon with non-intersecting edges, of perimeter $p$ and a resolution $n$, the number of nodes in the quadtree is of $O(p+n)$ [Hunter 78]. Moreover, the quadtree grows linearly in number of nodes as the resolution is doubled, whereas a binary array representation leads to quadrupling of the number of pixels. The amount of space occupied by a quadtree is very sensitive to its orientation and position.

Dyer [82] has shown that the average numbers of white, gray and total nodes in a quadtree representation of $2^m \times 2^m$ square image in a $2^n \times 2^n$ region, are each of $O(2^{m+2+n-m})$. 
If $B$, $W$ and $G$ represent the number of black, white and gray nodes respectively, then $T$, the total number of nodes is given by

$$T = B + W + G.$$  

The following relations hold for any quadtree [Knuth 75].

\begin{align*}
G &= (B + W - 1)/3 \\
B &= 3G - W + 1 \\
W &= 3G - B + 1 \\
T &= 4G + 1.
\end{align*}

Weng and Ahuja [87] (in the context of octrees) have given an elegant proof for $T$ in terms of $G$. We imitate their proof here.

**Theorem**  \[ T = 4G + 1. \]

**Proof**  Let $L$ be the number of leaves.

Suppose that each gray node corresponds to a 4-person game. Originally there are $L$ players to take part in the tournament. Three players lose after each game. Only the winners participate in the rest of the games. So we have

\begin{align*}
L - 3G &= 1 \\
L &= 3G + 1 \\
L+G &= 4G+1 \\
T &= 4G+1
\end{align*}

Hence in a quadtree, there will be an additional $(B+W-1)/3$ gray nodes besides $B$ black and $W$ white nodes, which makes the space requirement of $O(4/3(B+W))$.

To reduce the space requirement of the pointer quadtrees, the Autumnal quadtrees is proposed by Fabrini et al [86], in which the leaf nodes in a region quadtree are dropped. Instead,
the leaf node values are stored in place of the pointers to them. The sign bit is used to distinguish pointers from leaf values, with the convention being that positive numbers denote pointers. The Autumnal quadtrees gives a space saving of 75% for a complete quadtree.

Since the quadtree of an image is greatly affected by its location, orientation and size, methods have been suggested to overcome this dependency. The effect of translating a region by a unit is shown in Fig. 2.14. The first step in this direction was made by Li et al [82], to eliminate the effect due to translation, to define a normal form of quadtree. Supposing that the size of the image lies between $2^{l-1}$ and $2^l$, the image is moved around in a region of size $2^{l+1}$ to find a minimal cost quadtree in terms of number of nodes. This can be done in $O(2^{2l})$ space and $O(1.2^{2l})$ time. They assert that this quadtree representation is unique for any image over the class of translations.

Chien et al [84] have proposed a scheme, the normalized quadtree representation, which is invariant to rotation, scaling and translation. A normalized quadtree is generated for each object in the image rather than for the entire image. The object is normalized to an object centered coordinate system, with its centroid as the origin and its principal axes as the coordinate axes. Then the object is scaled to a standard size. This ensures that the normalized quadtree of an object is dependent only on the shape of the object but not on its orientation, location or size. This is an information preserving shape descriptor [Pavlidis 78] and hence can be used for object identification.
2.1.2.5.2 POINTERLESS QUADTREES

The space requirement of $4(B+W)/3$ nodes for a pointered quadtree where $B$ and $W$ are the number of black and white nodes respectively, to represent an image in an $N \times N$, $N = 2^n$, binary array is high for many real-life applications. Two of the drawbacks of the pointered quadtree are

(i) A large portion of the space requirement is taken up by gray nodes and pointers.

(ii) Individual leaf nodes are located by following a chain of pointers from the root to the desired node requiring many pointer references.

The pointerless data structure to store a quadtree has been proposed to overcome the above shortfalls.

2.1.2.5.2.1. DF-EXPRESSION

Kawaguchi and Endo [80] represent a tree in the form of preorder tree traversal of the nodes of the quadtree called DF-expression. The result is a word over the alphabet set $\{B, W, (, )\}$ corresponding to black, white and gray respectively. The DF-expression corresponding to the quadtree is shown in Fig. 2.15.

The original image can be reconstructed from the DF-expressions by using the fact that the degree of any non-terminal node is always four. Kawaguchi et al [83] show how a number of basic image processing operations can be performed using DF-expressions. In particular, centroid computation, rotation, scaling, shifting and set operations, are presented. If the length of the DF-expression is $T$ and if each symbol takes 2 bits for storage, the total requirement will be of $O(T)$ bits.
2.1.2.5.2.2. LINEAR QUADTREES

The reduction of space requirement and elimination of pointers is the prime motivation for the development of linear quadtrees. In this scheme, each quadrant is given a locational numerical code which corresponds to a sequence of directional codes that locate the leaf along a path from the root of the tree. The location of the quadrant manifests itself inherently in the code. The set of only black nodes, stored either in an array or a linked list is termed a linear quadtree. [Gargantini 82a]. Gargantini was the first to propose such a model, but later Abel and Smith [83] and Samet [85b] have proposed similar models and used it for rectangle retrieval, and approximation and compression, respectively.
Gargantini's Model

There are two phases in the construction of the linear quadtree (LQT).

(i) all black pixels are encoded and transformed into numerical codes in number system with base 4 i.e., quaternary codes.

(ii) condense the codes. (as is done in [Samet 81b])

The coordinate system and the quadrant encoding is as shown in Fig.2.16. The encoding process is explained below.

![Diagram of coordinate system and quadrant encoding](image)

**Fig. 2.16**: The coordinate system (a), and the ordering of the quadrants (b).

For a black pixel at position (I,J), the binary equivalents of I and J are interleaved and then converted to quaternary system. For example, for a black pixel at (6,5)

- \( I = 6_{10} = 110_2 \)
- \( J = 5_{10} = 101_2 \)

After interleaving the corresponding bits, we have 111001_2 ,i.e., 321_4. The location of the pixel in the image space is clear from the quaternary code. The pixel with code 321 is in the SE(3) quadrant at first division, in the SW (2) quadrant in the second
and NE (1) in the final subdivision. A code will have \( n \) quaternary digits, \( n = \log_2 N \).

After all the black pixels have been transformed into their corresponding quaternary codes, the condensation process starts. If any four pixels or quadrants have the same code but for the last quait (quaternary digit), the four is replaced by a code of the first \((n-1)\) quaits followed by 'X'. For instance, if pixels with codes 210, 211, 212 and 213 are all present, they are replaced by 21X. Moreover, if 20X, 21X, 22X and 23X are all present, they are substituted by 2XX and so on. The linear codes sorted in ascending order correspond to the post-order traversal of the black nodes of the corresponding region quadtree.

A region and its LQT based on Gargantini's Model are given in Fig. 2.17.

Fig. 2.17: An object (a), and its linear quadtree based on Gargantini's model (b).

All the operations on linear quadtrees boil down to manipulation of numbers, which is relatively easy compared to tree traversals using pointers. Given a code, the adjacent quadrant(s), if any, can be found easily [Gargantini 82a].
Let $T$, $B$, and $W$ be as defined before. Let $NP$ be the total number of black pixels. A linear quadtree can be stored in $(3(n-1)+2)B$ bits. In terms of number of nodes, the space complexity for a LQT is $B$ whereas for a regular quadtree it can be as high as $4nB+1$ [Samet 80b]. Savings of space more than 66% is achieved.

**Abel and Smith's Model**

A similar model for generation of LQT has been proposed by Abel and Smith [83]. The root of the tree, at level 0, has the code $5^n$ ($n$ is the resolution). The keys are defined recursively. For a node at level 1, the key $k$, can be computed from the key of the father, $k'$, by

$$
k = k' + 5^{n-1}.
$$

where $i$ is 1, 2, 3 or 4 according as the node is the SW, NW, SE or NE son of its father. Three level decomposition with locational keys to base 5 is shown in Fig. 2.18.

The LQT for the image in Fig. 2.17 is

1100, 1320, 1341, 1342, 1410, 1431, 1432, 1441

The locational keys sorted in ascending order gives the preorder traversal of the black nodes of the corresponding region quadtree. Here too, the keys have $n$ digits. The size and location of a quadrant can be determined from the key. Further, given two keys, it can be easily ascertained if one is contained in the other. Abel and Smith [83] have used this method for rectangle retrieval i.e., to retrieve from a set of spatially defined entities those relevant to a certain subregion. When the LQT is large, for efficient and fast operations, a $B^+$ tree structure has been proposed [Abel 84].
Fig. 2.18: The ordering of quadrants (a), and the complete numbering of quadrants (b) in Abel and Smith's model, with a resolution of 3.

Samet's Model

Hanan Samet has used his version of LQT for quadtree approximation and compression [Samet 85b]. The directional codes are obtained by traversing the path from the root to the node. The code for the root is 0 and for the other quadrants, it is defined in a recursive process. The code, $z$, can be computed by

$$z = 5z' + i$$

where $z'$ is code of the father and $i$ is one of 1, 2, 3 or 4 depending on whether the node is the NW, NE, SW or SE son of its father. The LQT based on Samet's Model for the image in Fig. 2.17 is

$$3, 13, 21, 63, 71, 73, 111, 113$$

The nodes sorted in ascending order, gives the breadth-first traversal of the black portion of the corresponding region quadtree.
Samet's model of encoding differs from the others (i.e., Gargantini, and Abel and Smith) in the following ways [Samet 85b].

(i) For a grid of size $N \times N$, $N=2^n$, the locational codes, irrespective of the size of the quadrant, are $n$ digits long in case of others. In Samet's model, larger the quadrant, smaller is the code.

(ii) In the 'others' coding scheme, it is more complex to decode the locational code to yield the path from the root of a quadtree to the root of the subquadtree.

(iii) Increasing the resolution of the image requires recoding in the other models. In particular, their codes must be multiplied by 5 to a power equal to the increase in resolution.

(iv) Samet's model has the progressive approximation property, i.e., as the list grows, one gets a better approximation of the image i.e., successive nodes in the list lead to a better approximation.

All the operations that can be done using pointer quadtrees can be done as well, if not better, by pointerless quadtrees.

Algorithms for intersection, union and pairwise difference of two images represented by their respective linear quadtrees is given by Bauer [85]. The procedures for translation, rotation and superposition of linear quadtrees are presented in [Gargantini 83]. Detection of the connectivity for images represented by LQT
is proposed by Gargantini [82b]. Van Lierop [86] describes how the leaf codes of geometrically transformed pictures can be derived from the original leaf codes, in a time of $O(MO^2(n+\log MI))$, where MI and MO are the number of input and output nodes respectively, and $n$ is the resolution. But Walsh [88] has presented a nonrecursive translation algorithm, which also has the same time complexity as above, but turns out to execute faster.

Bhaskar et al [88] have presented algorithms for the computation of area, perimeter, center of gravity, moments of images and the basic set operations by means linear quadtrees using parallel processing. If there are $p$ processors and $n_1$ and $n_2$ nodes in the two linear quadtrees, then the intersection and union can be done in $O(t(n_1+n_2)/p+\log p)$, where $t$ is the depth of the smallest node. Complementation can be done in $O(nt/p)$, $n$ is the number of nodes.

Besides the linear quadtree proposed by Samet [85b], Gargantini [82a] and Abel and Smith [83], there are some other schemes which can be said as the close variants of the form. Some of them are presented here.

The Explicit quadtree [Woodwark 82] is proposed to improve the speed of interrogation and modification. Basic operations and an efficient addressing scheme are presented by the author.

Oliver and Wiseman [83a] have defined treecodes, that are based on the depth-first traversal of the quadtree, for performing operations like merging, masking, construction of a quadtree from polygon, rotation, reflection and translation.

The Squarecodes [Oliver et al 83b] is the set of
specifications of the colour, size and position of image square. In the conventional quadtree model, the image is broken up into squares, that have sides which are powers of 2 long, whereas in the squarecode representation, the image is broken up into arbitrary square areas, each one of uniform colour. Algorithm for operations of translation, scaling, 90° rotations, restricted reflection and rotation through an arbitrary angle about the image center are presented.

For progressive transmission of images using linear quadtree, largely used in satellite image processing, a method of storing linear quadtrees, termed 'P-compressed quadtrees' was proposed by Anedda and Felician [88]. This is based on the Gargantini's model. In the new scheme, all the distinct prefixes are stored once; each of them will be followed by the number of pixels having that prefix and by the corresponding suffixes. For example, the five locational codes 001312010, 001312011, 001312020, 001312023 and 001312031 are replaced by 0013120, 5 and 10, 11, 20, 23, 31.

Unnikrishnan et al [87] have presented an improvement over Gargantini's model of linear quadtrees called Linear Hierarchical Quadtree, LHQT. Here, 4 is used for the 'don't-care' digit. For a grid of size $N \times N$, $N=2^n$, there will be an extra $n$ arrays, where the $i^{th}$ array will contain the locational codes of quadrants of size $2^i \times 2^i$. Since the level of hierarchy indicates the size of the black nodes, the digit 4 is redundant and hence deleted. As a consequence, the codes at level $k$ will contain $(n-k)$ digits only. These modified codes are termed linear hierarchical Q-
codes. An algorithm for labeling the connected components by bottom-up approach is presented.

The Threaded linear hierarchical quadtrees, TLHQT [Unnikrishnan et al 88] is an enhancement of LHQT. In the TLHQT, links to neighbor nodes along the four directions at the same or higher levels of hierarchy are provided. Connected component labeling, perimeter and Euler number computation are addressed in this paper.

2.1.2.5.3 SEMI POINTERED QUADTREES

The fully pointered quadtrees offer the maximum flexibility, for, these quadtrees can be traversed is any order, but they require the maximum storage space. On the other hand, the pointerless quadtrees are the most compact, but have to be traversed in the order of their creation, which reduces the speed of some algorithms. In between these two extremes of the space-time trade-off enter the semi-pointered quadtree representations.

2.1.2.5.3.1 ONE-TO-FOUR QUADTREES

The one-to-four quadtree, or just one-to-four, proposed by Stewart [86] reduces the number of pointer fields in a complete quadtree. In the one-to-four, a node has five fields: four fields contain the node types of the four sons, (as against pointers to the four sons, as in pointered quadtrees) and the fifth field is a single pointer to all the sons within that quadrant. See. Fig 2.19. The one-to-four is so called because one pointer is used to determine where the sons’ records are for the four quadrant descriptors. The node structure is shown below.
Fig. 2.19: A 2D object (a), the first division (b), second division (c) and the final division (d). In (d) all the quadrants are homogeneous. The one-to-four quadtree is shown in (e).
Address of the record for the son $i$ can be computed using the expression

$$\text{pointer} + [\text{record length} \times i]$$

where $i$ is 1, 2, 3 or 4 according as the son is NW, NE, SW or SE son. Here one pointer replaces 16 pointers and hence in a complete quadtree gives a saving of 93% memory.

This structure is quite useful when the quadtree is complete, but this fails when the quadtree is not. In such cases, the following remedies are helpful.

(i) Inspecting all the preceding quadrant descriptors to determine the number of sons’ records that are there in front of the desired record, at the cost of extra time for traversal.

(ii) By padding out the one-to-four with extra empty records, the sons’ records’ address calculation will remain simple. This padding results in each pointer pointing to four records whether they are used or not. Space is wasted, but time is gained for various operations.

(iii) With a premium of very little space and no additional time, a way out is to add one more field, 'offset', which gives the offset from the pointer for each son’s record. And the address of a son’s record is given by

$$\text{pointer} + [\text{quadrant offset} \times \text{record length}].$$
The modified node structure is

<table>
<thead>
<tr>
<th>offset</th>
<th>pointer to sons of quadrant</th>
<th>NW son</th>
<th>NE son</th>
<th>SW son</th>
<th>SE son</th>
</tr>
</thead>
</table>

This tree structure maintains traversal flexibility and reduces memory requirement. Algorithms for one-to-four to raster conversion, 90° rotations and reflections about the two diagonals and the horizontal and vertical axes through the center point are presented.

2.1.2.5.3.2 GOBLIN QUADTREE

The goblin quadtree [Williams 88] is a new and simple data structure for storing spatial information, consisting of a root and a number of branches. Each branch is stored as a record in a direct access disc file. A branch contains five fields: the node values for the NW, NE, SW and SE sons and a pointer. For the node values, a positive value indicates a terminal node, with the value being either an attribute or a pointer to a list of attributes. A negative value indicates a non-terminal node, with the magnitude being the dominant value of the quadrant it represents. See Fig. 2.20.

The pointer field contains the number of branch that would be processed next, if the detail for the current branch were to be skipped. If all the four quadrants of a given branch are homogeneous, the pointer will point to the next branch. If skipping detail for a particular branch would complete the traversal, the value of the pointer is irrelevant. Such a pointer is given a value of one greater than the final branch number.
Here 2 indicates black and 1 white. The numbers in the quadtree correspond to the branches in the goblin quadtree.

The order in which the branches are stored is depth-first. The first branch contains the first subdivision of the object space. The second branch contains the subdivision of the first non-homogeneous quadrant in the first branch, or if there is none, then it contains the subdivision for the next non-homogeneous quadrant. The conventional NW, NE, SW, SE is followed when ordering of sons is performed.

The goblin quadtree is similar to the autumnal quadtrees. [Fabrini et al 86] but for the pointers and the average or dominating
values. The advantages of goblin quadtree over autumnal quadtrees are

(i) the average values are stored one level higher in the goblin quadtree, so a truncated traversal requires a reading of far fewer records.

(ii) goblin quadtree does not mix pointers and leaf values, so that space is not wasted when the pointer requires more bits than the leaf value.

Oliver et al [84] have proposed an algorithm that converts a linear quadtree into run length coding via an intermediate semipointed quadtree called sextree. In the sextree, the pointers allow for quadrants 0 and 1 or 2 and 3 to be missed out while walking over the tree, which is not possible with a linear quadtree. Since any scan line either passes through quadrants 0 and 1, or 2 and 3, not both or any other combination, this is feasible.

2.1.2.6 SKELETONS AND MEDIAL AXIS TRANSFORM

Most of the representation schemes are highly dependent on the location of the image in the region i.e., a slight translation of the image may lead to a drastic modification of the representation. A structure that is invariant to the location of the image is the skeleton [Pfaltz et al 67]. Here the image is regarded as a union of maximal neighborhoods of its points, and can be specified by the centers and radii of these neighborhoods. The set of centers is termed skeleton, see Fig. 2.21, since the locus of centers of maximal neighborhoods often takes the form of centrally located stick figure. It may be observed that the
neighborhoods may overlap.

Depending on the nature of the figure, one of the following metrics may be used to specify the neighborhoods.

\[ d_E(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2} \]

The ball, \( d_E(p, x) \leq r \), is a circular disc centered at \( p \) and radius \( r \).

**Euclidean Metric**

Let \( p = (p_x, p_y) \) and \( q = (q_x, q_y) \). Then

The ball, \( d_E(p, x) \leq r \), is a circular disc centered at \( p \) and radius \( r \).

**Absolute Value Metric**

\[ d_A(p, q) = |p_x - q_x| + |p_y - q_y| \]

The ball, \( d_A(p, x) \leq r \), is a diamond centered at \( p \) and side length \( r\sqrt{2} \). This is also called city block distance.

**Maximum Value Metric**

This is also called chess board metric.

\[ d_M(p, q) = \max \{ |p_x - q_x|, |p_y - q_y| \} \]

The ball, \( d_M(p, x) \leq r \), is a square centered at \( p \) and side \( 2r \).

The set of points comprising the skeletons and their associated radii is called the Medial Axes Transform, MAT. The
operations such as translation and scaling are easy. The former involves modification of the skeletons and latter the radii. The Algorithms for answering the point-membership problem and for performing intersections and union of two images represented by their MATs is given by Pfaltz and Rosenfeld [67]. Wu et al [88] present algorithms for the computation of geometric properties using MATs by parallel processing.

2.1.2.7 QMAT

The skeleton and MAT concepts used in traditional image processing representations, when adapted to the quadtree representations, yield the new data structure, Quadtree Medial Axis Transform or QMAT [Samet 83]. In a QMAT, black nodes in the original quadtree are allowed to expand to absorb adjacent smaller black nodes. At times, they overlap the boundary of the grid. An object and its QMAT are shown in Fig. 2.22.

![Diagram](image)

Fig. 2.22: An object (a), its quadtree (b) and QMAT (c). In QMAT representation, the radius is given in parentheses below the node.
Similar to the skeleton of MAT, there is a quadtree skeleton in QMAT. Before defining the quadtree skeleton some definitions are in order.

Since the ball centered at \( p \) and radius \( r \), i.e., \( d_M(p,x) \leq r \), is a square in chess-board metric, it is well suited over other metrics, for the quadtree operations [Samet 82b]. The function \( \text{DIST} \), yields for each black block in the quadtree the chess-board distance transform from the center of the block to the nearest pixel which is on a black-white border i.e., if \( p \) is the center of the black block \( B \) and \( z \) is a point on the border, \( \text{bord}(W) \), of the white block \( W \), we have

\[
F(B,W) = \min_{z \in \text{bord}(W)} d_M(p,z)
\]
i.e., \( F(B,W) \) gives the distance between \( p \), the center of \( B \) and the nearest point on border of \( W \). The minimum taken over all such \( W \)'s gives \( \text{DIST}(B) \) i.e.,

\[
\text{DIST}(B) = \min_W F(B,W)
= \min_W \min_{z \in \text{bord}(W)} d_M(p,z)
\]

\( \text{DIST}() \) for a white block is defined to be zero. Now we define quadtree skeleton.

Let \( BB \) be the set of all black blocks in the image. For each \( B_i \) \( BB \), let \( S(B_i) \) be the part of the image spanned by a square with side width \( 2 \cdot \text{DIST}(B_i) \), centered about \( B_i \). The set \( T \), of black blocks, having the following properties is the quadtree skeleton.

(i) area (\( BB \)) = \( \cup_{i \in T} S(t_i) \)
(ii) for any \( t_i \in T, \exists B_i \in BB, (b_i \neq t_i), \exists S(t_i) \subseteq S(B_i), \)
(iii) \( \forall B_i \in BB, \exists t_i \in T \ni S(B_i) \subseteq S(t_i) \)
Samet [82b] has shown that the quadtree skeleton of an image is unique. The QMAT of an image is the quadtree whose black nodes correspond to the black blocks comprising the quadtree skeleton and their associated chess-board distance transform values. All the remaining nodes in QMAT are white and gray with distance value zero.

A QMAT results in a partitioning of the image into set of non-disjoint squares having sides whose lengths are sums of powers of two, whereas, in the region quadtrees the image is made up of a set of disjoint squares having sides whose lengths are powers of two. The number of nodes in a QMAT for a region is less than that of the corresponding quadtree. For a certain class of images, the QMAT requires a minimum number of nodes regardless of the image resolution. Specially in cases where $2^n \times 2^n$ grid consists of square with side $2^{n-1}$. For the QMAT representation, the number of nodes necessary to represent an image is not as shift sensitive as in the region quadtree.

2.1.2.8 TID

A data structure that is not sensitive to the location of the image, is the Translation Invariant Data structure, TID [Scott et al 86]. This is a slight variation of the MAT. In the MAT, maximal blocks are stored by their centers and radii. A TID is made up of maximal squares and each square is stored as a triplet $(i,j,s)$, where $(i,j)$ is the north west corner of the square and $r$ the length of the side. In Fig. 2.23, an object and its TID are presented.

The following steps are involved in the construction of a
TID of an image.

(i) For each black pixel, compute the distance of the nearest white pixel. Let $D(i,j)$ be the distance from $(i,j)$ to the nearest white pixel.

(ii) Locate all maximal black squares. A maximal black square is defined to be a square not contained in any other black square. The largest square centered $(i,j)$ is of the odd order of size $(2*D(i,j)-1)$.

(iii) Check for redundant maximal black squares and eliminate if any.

(iv) Identify each non-redundant maximal square by the triple $N(i,j,s)$, where $(i,j)$ is the north-west corner of the square and $s$ is the size.

In the TID, as with MAT, answering the point membership question is easily done. Translation and scaling are also easy. Rotation by $90^\circ$ clockwise involves changing $(i,j,s)$ to $(-j,i+s,s)$ for each maximal block. Union of two regions represented by their TIDs is also direct.

Fig. 2.23: A 2D object (a), and its TID (b).
Scott and Iyengar [86] have shown that on the average, TID provides a 56% space reduction over linear quadtrees and an 86% over quadtrees, forests of quadtrees and QMATs. They prove that the number of nodes required to store a TID does not exceed the minimum number required by optimally located quadtrees, forest of quadtrees or QMATs.

Scott and Iyengar [84] note that the time required to construct a TID is of $O(rc \log (\min(r,c)))$, where $r$ and $c$ are the number of rows and columns in the embedded region. In principle, the TID for a $2^n \times 2^n$ image requires $3n$ bits for each maximal square, but this space requirement can be reduced by having the minimal number of maximal squares.

Kim et al [83] designed a method, rectangular coding, that partitions the image into disjoint rectangles. Each rectangle is stored by means of the coordinates of the corner nearest to the origin of the coordinate system, its length and width.
2.2 3D OBJECT REPRESENTATION

The 3D object representation is crucial for computer vision. A great majority of objects in the real life are 3D and to make the robot an efficient companion to humans, 3D objects or the important attributes of it should be represented so that they may be learned, compared, retrieved and used for image analysis and pattern recognition. After the advent of 2D graphics, the next natural step is to explore the storage, the manipulation and the display of 3D objects in a realistic manner. In fact, this was necessitated by the introduction of present data collection techniques such as Computer Axial Tomography (CAT), Scanning Electron Microscope (SEM) and Digital Stereoscopy. These methods make possible the computation of 3D structure of scenes, ranging from inner parts of the human body to the rock microstructures in the form of a 3D array of numbers. Developing the algorithms for collection, storage, display and manipulation of these images has revealed the need for developing new data structures, and more generally, for developing spatial-knowledge representation schemata.

The task of representing the 3D objects is more intricate than that of the 2D objects, mainly because of the extra dimension involved. On the face of it, using spatial occupancy arrays, it requires $O(N^2)$ space in case of 2D, whereas, it is
of $O(N^3)$ for 3D. Besides the phenomenal rise in the space requirement, we look at a 2D object, from outside the 2D space containing it, which enables us to grasp the complete information. But the same cannot be done to 3D objects.

In this section some of the 3D object representation schemes are presented. Requicha [80] has classified the 3D object representation schemes into six categories: primitive instancing, spatial enumeration, cell decomposition, constructive solid geometry, sweep representation and boundary representation. In primitive instancing, components of objects are defined parametrically. A shape type and a limited set of parameter values specify an object. This is best suited for well-defined objects i.e., objects that obey a fixed mathematical equation. The details of the other categories will be considered.

Specific merits and demerits of each have been studied and presented by Requicha et al [80]. In the remaining part of this section, an object is assumed to be a three dimensional one. The object is made up of volume elements or voxels, (Meagher[82a] calls them 'obels' for object elements). Here objects are assumed to be monochrome and voxels inside the object have a value of 1 and outside, 0.

2.2.1 SPATIAL ENUMERATION

In this scheme, an object is represented by a list of cubical cells which it occupies.

2.2.1.1 SPATIAL OCCUPANCY ARRAYS

Just as in the 2D case, this is the most simplest method to represent a 3D object. In low resolution tasks where the
objects are highly irregular, this is quite useful.

Using this method, it is easy to perform the geometric operations, set operations, and the computation of volume and moments. Further, by the use of this method, it is easy to generate 'slices' i.e., cross-sections of objects. The spatial occupancy arrays are used in CAT because, here, the objects are highly irregular. And the fast generation of slices facilitates on-line diagnosis. With the declining cost of computer memory, one can afford this representation model, in spite of its high space requirement of $O(N^3)$.

2.2.1.2 MAT

The Medial Axis Transform (MAT) representation, besides being compact, is translation invariant. In the 3D version of the MAT, the object is decomposed into a collection of overlapping balls [O'Rourke et al. 79]. The metric used is the $d_M$, the maximum-value metric, where balls are cubes. Given the centers and radii, it is possible to generate the boundary representation by choosing those voxels that are not in the interior.

Since the 'medial axis' in the case of a 3D object is not a space curve, it may be referred to as the medial layer [Srihari 81]. An voxel $v$ is in the medial layer if

$$\text{card} \{ u | (u,v) = \min_{w \in B(S)} d_M(v,w) \} > 1$$

where $S$ is the object and $B(S)$, its boundary. This definition means that $M(S)$, the medial layer, consists of those voxels of $S$ whose distances from $S^C$, the complement of $S$, are local maxima.
Thus, $S$ is the union of all maximal digital balls centered at the voxels of $M(S)$.

### 2.2.1.3 OCTREES

The cube of side $N$, $N=2^n$, enclosing the object is our universe. The recursive partitioning of this universe into eight equal octants, suboctants and so on, till the ultimate octants so obtained are homogeneous is the underlying concept for the octree. This is the extension of the quadtree to three dimensions and the terminology of quadtrees is carried over to the octrees. The octree was first briefly described by Hunter [78], and later was independently developed by Reddy et al [78], Jackins et al [80], Meagher [80] and Srihari [80]. The octree is also referred to as octtree, oct-tree [Jackins et al 80] and octal-tree [Srihari 80].

![Image](image_url)

**Fig. 2.24**: The ordering of octants (a), a 3D object (b) and its octree (c).
The octree is formally defined now. In the universe, as defined above, each voxel has a value 0 or 1 depending on whether it is outside or inside the object. A cube is homogeneous if all the voxels that are inside have the same attribute. If the universe is non-homogeneous, it is divided into eight equal octants. All the non-homogeneous octants are divided again. This process is repeated till all the octants are homogeneous or the voxel level is reached. In the tree structure, the root corresponds to the universe, its eight sons, to the eight octants obtained at the first division and so on. The height of the octree, in the worst case, is n. Let T, B, W and G denote the number of the total, the black, the white and the gray nodes in the octree, respectively. Then the following relations hold [Knuth 68].

\[ T = 8G + 1 \]
\[ B = 7G - W + 1 \]
\[ W = 7G - B + 1 \]
\[ G = (B + W - 1)/7. \]

Further for a cube of side m, m \(>\) 3, at any position and with any orientation, the following relations hold [Weng et al 87].

\[ B \leq 24.25.4^m - 200.2^m + 1454. \]
\[ G \leq 5.76.4^m + 17.2^m + 8n + 76m - 277. \]

and \[ T \leq 47.4^m + 136.2^m + 64n + 608m + 2215. \]

An object and the corresponding octree are given in Fig 2.24. The octants are ordered as shown in Fig. 2.24a.

Meagher[80] asserts that the memory and processing computation required for the construction of an octree are on the order of the surface area of the object. Many operations on
objects benefit from the tree structure of an octree and can be simply implemented as tree traversal algorithms. The hierarchy of the data structure, spatial accessibility and pre-sorted nature of the octree simplify operations like detecting intersection among objects, hidden surface elimination, locating a point or block in space, connected component labeling and neighbor finding. The octrees find their application in solid modeling, computer graphics, CAD, computer vision, image processing and robotics.

The set operations of union and intersection of two objects and set complement, can be carried out as described for quadtrees in Section. 2.1.2.5, with minor amendments. But the quintessential concept of traversing the two trees in parallel remains the same. Meagher [82b] has developed efficient algorithms without the use of floating-point operations, multiplications and divisions. The computation of moments, volume, surface area, component counting, connectivity labeling and neighbor finding can easily be extended from the quadtree and hence not described here.

Given a transformation and the octree of an object, the aim is to build the octree of the transformed object. Separate algorithms exist for translations [Meagher 82a, Jackins et al 80] and rotations [Meagher 82a, Weng et al 87]. Weng et al [87] have given a general algorithm that takes care of both translation and rotation. Their algorithm is described below.

Let $X$ and $X'$ be the coordinate vectors before and after the transformation, $T$, the translation vector and $R$ be a three
dimensional rotation matrix for rotation by an angle about the
unit vector \( \mathbf{n} = (n_x, n_y, n_z) \) through the origin.
Then

\[
X' = RX + T
\]

Where

\[
R = \begin{pmatrix}
(n_x^2 - 1)a + 1 & n_y n_x a - n_z b & n_x n_z a - n_y b \\
n_y n_x a + n_z b & (n_y^2 - 1)a + 1 & n_y n_z a - n_x b \\
n_z n_x a - n_y b & n_z n_y a - n_x b & (n_z^2 - 1)a + 1
\end{pmatrix}
\]

\[
a = 1 - \cos \theta
\]

\[
b = \sin \theta
\]

Let \( OT_1 \) and \( OT_2 \) represent the source octree and target octree respectively. In \( OT_2 \), an voxel is treated black if and only if its center is on or inside some displaced black cube from \( OT_1 \). The accuracy is lost by coloring partially occupied voxels as black or white. If they are colored white, they may create holes in the transformed object. On the other hand coloring them black would tend to increase the volume.

Informally, the algorithm is described below. For each black leaf, \( BL \), in \( OT_1 \), the following are performed to generate \( OT_2 \).

(i) generate the new position and orientation of \( BL \). If this is outside of the universe, report error and stop.

(ii) generate the nodes in \( OT_2 \) by testing their corresponding upright cubes for intersection with the transformed \( BL \).

An octant/cube is upright if its sides are parallel to the coordinate axes. Else, it is tilted. The intersection test leads to three cases, each leading to different action about the nature of the target tree node(s).
(i) no intersection: the upright cube is unoccupied and is left alone

(ii) the upright cube is inside the transformed BL: the upright cube is occupied and hence colored black.

(iii) otherwise: generate all the eight children of the upright cube and carry out the intersection test recursively. If all the eight children are of a single color, delete the children and make the father the same colour.

A formal description of the algorithm follows.

```
function FDLEAF (node)
/* traverses OT1 in post order and for each black node checks if this leaf node goes out of the universe after displacement */

begin
if BLACK (node) then
   if (out-of-space) then error-stop
   else INTERSECT (node).
else
   if GRAY (node) then
      for each child, i of node do FDLEAF (i).
end
```

Algorithm 2.11

In a universe with voxel resolution $n$, the level of the root node is defined to be $n$ and for a non-root node, its level is one less than the level of its father.

For the functions INTERSECT and PARTIAL to be time effective, a bounding sphere circumscribing the transformed OT1_node is used to help identify the target octants that are inside or intersect the source octant.
function INTERSECT (OT1_node, OT2_node)
    /* l1 and l2 are levels of OT1_node and OT2_node respectively */
    begin
        if l1 < l2 then
            if PARTIAL (OT1_node, OT2_node) then
                for each child i of OT2_node do INTERSECT (OT1_node, i)
            else
                if l2 = 0 then
                    if (center of OT2_node is inside OT1_node) then
                        TYPE (OT2_node) = BLACK
                    else TYPE (OT2_node) = WHITE
                else if INSIDE (OT2_node, OT1_node) then
                    TYPE (OT2_node) = BLACK
                else if PARTIAL (OT2_node, OT1_node) then
                    for each child i of OT2_node do INTERSECT (OT1_node, i)
                end.
            end.
        end.
    end.

Algorithm 2.12

The average space and time of the algorithm are both bounded by the number of nodes in the source octree multiplied by the resolution of the universe.

Using octrees, it is easy to display objects by eliminating the hidden surfaces. Since the octree maintains all the elements of an object in a spatially pre-sorted order, display of objects after hidden surface removal is rendered easy. One can display the object by following a particular visiting sequence of octants. There are two types of visiting sequences front-to-back [Meagher 82b, Gibson 83] and back-to-front [Doctor et al 81 and Frieder et al 85]. In the front-to-back approach, octree nodes closer to the viewer are projected prior to those that are farther away. If a region of the image plane is painted, any other nodes projected onto it are ignored. More than one visiting sequence of nodes exists and in this approach any of the following can be used.

15043726, 13507246, 13570246 and 10537246.
Whereas in the back-to-front visiting sequence, octants farther away from the viewer are visited before the closer ones. An octant visited later in the sequence overwrites the painted region of any octant visited earlier. Frieder et al [85] assert that on the average, the front-to-back approach is more efficient of the two, since if a node is obscured, then the entire subtree rooted on it can be ignored.

There are different paradigms for the construction of an octree for an object. Since, it is not always possible to obtain the spatial occupancy array of the objects, octrees are also constructed from other representations/methods. They are

- volume intersection

- converting other representation to octrees.

In the volume intersection method, different 2D silhouettes of the object are generated; the silhouettes are swept along the viewing direction to obtain the 3D swept volume and these swept volumes are intersected to construct the resulting 3D object representation. Chien et al [86a] present an algorithm for generating the octree of the object from three orthogonal silhouettes. Veenstra and Ahuja [85] describe an algorithm to construct the octree representation of a 3D object from nine silhouettes: three 'face-on' views and six 'edge-on' views of an upright cube, in time linear in the number of nodes in the octree.
A depth (range) image of object is similar to intensity image, except that for each pixel of the image, the depth of the voxel rather than the intensity are stored. Connolly [84] constructs an octree from multiple views of the corresponding range quadtrees. Brunet et al [87] present an extended octree representation for storing free form surfaces.

One can also convert other representation to the octree representation. Yau et al [83] describe an algorithm for the construction of an octree from quadtrees of serial-sections.

The notion of a pointerless structure to store a tree is extended to the octrees too [Gargantini 82c]. As with linear quadtrees, many an operation can be performed using linear octrees - connected component separation [Gargantini et al 82b] and determining the 3D border [Atkinson et al 85b]. Gargantini [82c] presents the other operations that can be done on linear octrees.

2.2.2 CELL DECOMPOSITION

This is generalized form of spatial enumeration in which the disjoint cells are not necessarily cubical or even identical. In this model, the object is partitioned into rectangular parallelepipeds.

2.2.2.1 RECTANGULAR PARALLELEPIPED CODING

Reddy et al [78] have proposed an extension of the point quadtree to the 3D case, wherein the universe is decomposed into rectangular parallelepipeds by planes perpendicular to the X, Y and Z axes. They have also proposed another method of partitioning the universe into rectangular parallelepipeds. But,
here the rectangular parallelepipeds are not necessarily aligned with any axis, as in the above method. A tree structure is used to represent the object. At the root level, the tree has $N$ branches and the $i^{th}$ branch contains the transformation matrix $T_i$, $1 \leq i \leq N$. The 4x4 matrices $T_i$ are the transformations that convert the object space point into the local coordinates of its parallelepiped. Each parallelepiped is recursively subdivided into parallelepipeds in the local coordinates of the enclosing parallelepiped. This method facilitates easy display of the objects, but requires the spatial occupancy array as the input. Methods that overcome this short fall are those that are based on the volume intersection technique.

Kim et al [86] have extended the 2D rectangular coding scheme [Kim et al 83] for storing binary images. In the rectangular coding, elements in the set of rectangles which partition the binary image, are represented by the coordinates of

![Diagram of parallelepiped](image)

**Fig. 2.25** The rectangle in the front surface is stored as $(X, Y, W, H)$ and the rectangular parallelepiped is stored as $(X, Y, I, W, H, N)$. 
the vertex closest to the origin of the image coordinate system, its width and height, i.e., \((X,Y,W,H)\).

The rectangular parallelepiped coding or RP-coding, of an object is the set of rectangular parallelepipeds that partition the object volume. Each rectangular parallelepiped is represented by the \((X,Y,Z)\) coordinates of the vertex closest to the origin of the coordinate system, its width, height and depth i.e., \((X,Y,Z,W,H,D)\). See Fig. 2.25. The RP-code of an object is constructed from a set of three mutually orthogonal projections (silhouettes) of the object. First, for each of the silhouette, 2D rectangular codes are computed and are swept along the corresponding viewing direction to generate a set of rectangular parallelepipeds (Swept volumes). And then, the swept volumes are intersected to generate the RP-code of the object.

From the 2D rectangular coding, generating the 3D swept volume is a simple matter of rewriting the codes with insertion of '1' (the minimum value of coordinate in the direction of sweeping) and the image size \(N\). For instance, the swept volume of a rectangle \((X,Y,W,H)\) of the front silhouette becomes \((X,Y,1,W,H,N)\).

Let RP-Top, RP-Front and RP-side denote the swept volumes generated by the top, the front and the side silhouettes respectively. The RP-Code is generated in two steps.

(i) intersect RP-Front and RP-Top, to generate RP-I
(ii) intersect RP-Side and RP-I.
function GENERATE (RP-Front, RP-Top, RP-I)
/* generates the intermediate swept volume RP-I obtained by intersecting
RP-Front and RP-Top. */
begin
  for each F in RP-Front do
    for each T in RP-Top do
      if INTERSECT (X_F, W_F, X_T, W_T) then
        begin
          X_I = Max (X_F, X_T)
          Y_I = Y_F
          Z_I = Z_T
          W_I = min (X_F + W_F, X_T + W_T) - X_I
          H_I = H_F
          D_I = D_T
          add I to RP-I
        end.
      end.
end.

Algorithm 2.13

The function INTERSECT checks if two rectangular parallelepipeds intersect in 3D-space.

function INTERSECT(X_p, W_p, X_q, W_q)
begin
  INTERSECT = false
  if (X_p < X_q) and (X_p + W_p > X_q)
    then INTERSECT = true
  if (X_p > X_q) and (X_p - W_p < X_q)
    then INTERSECT = true
end.

Algorithm 2.14

The final step is to intersect RP-I and RP-side, which is performed by CONSTRUCT.
function CONSTRUCT(RP-1, RP-side, RP-R)
    /* generates the resulting RP-code, RP-R, by intersecting RP-1 and RP-side */
    begin
        for each I in RP-1 do
            for each S in RP-side do
                if INTERSECT(Y_I, H_I, Y_S, H_S) and INTERSECT(Z_I, D_I, Z_S, D_S) begin
                    X_R = X_I
                    Y_R = max (Y_I, Y_S)
                    Z_R = max (Z_I, Z_S)
                    W_R = W_I
                    H_R = min (Y_I + H_I, Y_S + H_S) - Y_R
                    D_R = min (Z_I + D_I, Z_S + D_S) - Z_R
                    add R to RP-R
                end
        end
    end

Algorithm 2.15

Kim et al [86] have presented a comparison of RP-codes and octrees. In RP-code, each rectangular parallelepiped is stored as a tuple of six integers and they are bounded by the size of the image. For an 128 x 128 image, if there are n rectangular parallelepipeds, the space requirement is 6n bytes.

The RP-codes take less storage than octrees because, after each intersection process in RP-coding scheme, the number of volume elements is reduced by merging rectangular parallelepipeds whereas in octrees, the volume element should always be a cube and merging is not possible i.e., one rectangular parallelepiped in RP-code can be an union of cubes, but a node in octree cannot be a union of more than one rectangular parallelepiped in the corresponding RP-code because there is no more scope for merging of rectangular parallelepipeds.

Using RP-codes, transformations are easy to perform and the properties of volume, surface area and moments can be computed easily, as also the generation of different views of the object [Kim et al 86].
2.2.2.2 TRANSLATION INVARIANT DATA STRUCTURE

The Translation Invariant Data structure, TID, proposed by Scott et al [86] for representing 2D objects has been extended to 3D by Iyengar and Gadagkar [88]. Since the same name, TID, is used to represent the 2D, as well as the 3D objects, we use 2TID and 3TID to distinguish these two.

In 2TID, the image is broken up into maximal squares, possibly overlapping, and each square is stored by a triple \((i,j,s)\), where \((i,j)\) is the first pixel of the square and \(s\) is the size of the square. More details can be found in Section 2.1.2.9.

The 3TID is constructed from multiple silhouettes of the object. For simple geometric objects (cubes, cones, cylinder etc.) and objects constructed from these primitives, three views are sufficient. The three views considered here are the top, the front and the right side views. The various phases in the construction of the 3TID are

(i) for each of the three views, the corresponding 2TID is generated.

(ii) Eliminate the redundant maximal squares that do not provide any useful information.

(iii) Represent each maximal square of the 2TID by a quadruple, \((v,i,j,s)\), where \(v\) is the view and \(i,j\) and \(s\) are as defined above.

The second phase, involving removal of redundant maximal squares needs to be elaborated. The front and right side views correspond to the last column and last row of the top view
respectively. Hence, if there are maximal squares of size 1 in the last column (corresponding to the right side) or last row (corresponding to the front) in the top view, they can be deleted. But the other maximal squares in the top view are retained. In case, the front and side views start with the largest maximal square and there are only two, then the following rules are applied.

(i) If the two maximal squares are in the front view then delete the second maximal square if the right side view is considered and the first, if the left side view is taken.

(ii) Retain the two if none of the side views are considered.

The authors ascertain that the number of nodes generated in the 3TID is less than or equal to the number of nodes in the corresponding octree. Equality may be attained for simple objects and it is less, when the image contains multiple objects.
Further, for the construction of an octree, the image required is a 3D image as obtained by a CAT scanner, whereas a 3TID needs only multiple 2D views. As the name suggests, translating an object is trivial using 3TID, as is the case with 90°-rotations.

The 3TID is compared to the octree generated from multiple slices of the object [Yau et al 83] and from three mutually orthogonal silhouettes [Chien et al 86a]. The time complexity in case of Yau et al is of \( O(2^n) \) where \( n \) and \( d \) are the resolution and the depth of the octree respectively. For the algorithm presented by Chien et al [86a], it is of \( O(S^{3\log S}) \), where \( S \) is the total number of nodes generated in the octree. But, to construct a 3TID, it takes time of \( O(S^2m \log S) \), where \( m \) is the total number of maximal squares. The time complexity for this method is much less than that of the other two because \( m \ll S \).

2.2.3 CONSTRUCTIVE SOLID GEOMETRY

The Constructive Solid Geometry, CSG for short, [Voelcker et al 77] is the method of representing objects as a composition of primitive solids. Here, the object is stored as a tree, in which internal nodes store the operations and the leaves, the primitives. See Fig. 2.27.

At the lowest level are the primitive solids, which are the intersection of closed half-spaces defined by \( F(X,Y,Z) \geq 0 \), where \( F \) is well-behaved, for instance, analytic. Generally, primitives are objects like arbitrarily scaled rectangular parallelepipeds, cylinders, cones and spheres of arbitrary radii. They may be positioned arbitrarily in space.

Some of the point-set topology terms used in CSG are
defined below [Simmons 63]. Consider a topological space \((W,T)\).

The closure of a subset \(x\), denoted by \(kx\), is the union of \(x\) with the set of all its limit points.

The interior of a subset \(x\), denoted by \(ix\), is the set of all the interior points of \(x\).

The subset is regular if \(x = kix\) i.e., if \(x\) is the closure of its interior.

In the CSG scheme, to avoid dangling edges when set operations are performed, the regularized union, intersection, difference and complement defined below are considered and are denoted by superscripting them with a '*'.

\[
\begin{align*}
X \cup^* Y &= k_i (X \cup Y) \\
X \cap^* Y &= k_i (X \cap Y) \\
X -^* Y &= k_i (X - Y) \\
X^{c^*} &= k_i (X^c)
\end{align*}
\]

Fig. 2.27 : A 3D object (a), and its CSG representation (b). In the above tree, primitives are stored in the leaf nodes and the regular operators in the internal nodes.
A CSG representation is a Backus-Naur Form expression involving primitives and set operations (intersection, union and difference) for combination and motion.

\[ \text{<CSG rep> = <primitive solid>} \mid \text{MOVE <CSG rep> BY <motion parameters>} \mid \text{<CSG rep><operator><CSG rep>} \]

### 2.2.4 GENERALIZED CYLINDERS

When a 2D object is swept along a 3D space curve or rotated along an axis, different swept volumes are generated. When a 2D set is translated along a line, it is called a translational sweep. By rotating the 2D set around an axis, a rotational sweep is obtained. In the three dimensional sweeps, volumes, rather than 2D sets are swept. In a general sweep, a 2D set or a volume is swept along an arbitrary space curve and the set may vary parametrically along the curve [Soroka 79]. The general sweeps are called Generalized Cylinders, GC. (also called generalized cones).

A GC is a solid whose axis is a 3D space curve. At any point on the axis, a closed cross-section is defined. Usually, it is easier to think of a GC as a space curve with a cross sectional point set function, both parametrized by the arc length along the space curve. For a GC, there exist infinitely many axis-cross section pairs that define it.

A coordinate system that depicts the local behavior of the GC axis space curve is the Frenet Frame, defined at each point on the GC axis. The origin of the frame is the point on the axis itself and the three orthogonal vectors are given by
\((\xi, \nu, \zeta)\) where

\(\xi\) = unit vector tangent axis

\(\nu\) = unit vector direction of the center of the curvature of
axis normal curve, and

\(\zeta\) = unit vector direction of center of torsion of axis.

The Frenet frame gives good information about the GC axis, but has certain drawbacks. It is not well-defined when the curvature of the GC axis is zero. Further it may not reflect known underlying physical principles that generate the cross sections. For instance, consider the case of bolt threads. They can be described by a single cross section that stays fixed in a coordinate system that rotates as it moves along the straight axis of the bolt.

The cross section is defined as a point set in \(\nu, \zeta\) plane, using inequalities expressed in the \(\nu, \zeta\) coordinate system. It can also be described by the cross section boundary, parametrized by another parameter \(r\). Let this curve be given by \((x(r,s), y(r,s))\). The dependence on \(s\) shows that the cross sectional shapes can vary along the axis. The above expression, in world coordinates, is transformed to the local coordinates by

\[ B(r,s) = a(s) + x(r,s)\nu(s) + y(r,s)\zeta(s) \]

where \(B\) is the GC boundary and \(a\) is the axis.

The GC facilitates easy computation of many parameters of the solid.

- the perpendicular to a cross section, from \((x(r,s), y(r,s))\)

is given by

\[ \frac{dy}{ds}\nu - \frac{dx}{ds}\zeta \]
the area of a cross section is given by
\[ \int_0^P (x \frac{dy}{ds} - y \frac{dx}{ds}) ds \]
where P is the perimeter.

- the volume is given by the integral of the area as a
function of the axis parameter multiplied by the
incremental path length of the GC axis.
\[ \int_0^L \text{area}(s) \, ds \]
where L is the length of the GC axis.

2.2.5 BOUNDARY REPRESENTATIONS

In this scheme, the objects are represented by their
enclosing surfaces, such as planar, quadric surfaces, patches
etc. The representation of a space curve and a surface in three
dimensional space are distinguished.

2.2.5.1 SPACE CURVES

A digital space curve is defined as a connected set of
voxels all but two of which have exactly two neighbors in the set
and the end voxels have exactly one neighbor each. The 2D chain
coding is extended to the third dimension.

In 3D space, a voxel can have 6, 18 or 26 possible
directions to the neighboring voxel depending on whether the
adjacency considered is face, face and edge or face, edge and
vertex. These are also called 1-, 2- and 3-neighbors respectively
[Srihari 81]. See Fig. 2.28.

Thus each possible direction can be uniquely described by
3 bits for 1-connected space curves, and 5 bits for 2- or 3-
connected space curves. For a voxel at \((x, y, z)\), the 26
neighboring voxels are given by \((x \pm 1, y \pm 1, z \pm 1)\) and hence, a space curve can be stored as a word over the alphabet \([-1, 0, 1]\) and the coordinates of the starting voxel.

### 2.2.5.2 SURFACES

The representation of surfaces is vital for the display of the shaded surfaces of an object on the screen. The surface of an object \(S\), may be defined as the voxel faces that are at the interface of \(S\) and \(S^C\) (the set complement of \(S\) in the universe). If \(S\) is 1-connected, then \(S^C\) is 3-connected and vice-versa. The surface of an object \(S\) is formally defined as the set of voxel pairs

\[
Y(S) = \{(u,v) | u \in S, v \in S^C, u \text{ is a face neighbor of } v\}
\]

The surface of an object can be represented by specifying the set of constituent faces or indirectly by means of border voxels, medial axes and graphs (where vertices correspond to faces and edges to touching faces).
The border of $S$, $B(S)$, is defined by

$$B(S) = \{ \text{v} \mid \text{v} \in S \text{ and } N_3(\text{v}) \cap S^C \neq \emptyset \}$$

where $N_3(\text{v})$ denotes the set of 3-neighbors of $\text{v}$. By the very definition of $\gamma(s)$, it follows that the surface of $S$ is uniquely specified by $B(S)$.

To determine if a voxel $\text{v}$ belongs to $B(S)$ is to check if any voxel of $N_3(\text{v})$ belongs to $S^C$ i.e., if \{ $\text{v}^i$ \} are elements of $N_3(\text{v})$, then

$$\text{v} \in B(S) \iff \chi(\text{v}) \left( \prod_i (\text{v}^i) \right) = 1$$

where $\chi$ is the characteristic function of the object $S$.

2.3 CONCLUSIONS

A wide spectrum of data structures for the representation of objects has been presented. As mentioned earlier, no single data structure admits all the desirable properties one would ask for. At times, it would be convenient to convert one representation scheme to another for fast query processing and certain other operations. Among the 2D object representation schemes, the following algorithms are proposed for converting one representation to another: boundary to skeleton and vice-versa [Rosenfeld et al 66], boundary to bintrees [Diehl 88], chain codes to quadtrees [Samet 80b], quadtrees to chain codes [Dyer et al 80], arrays to quadtrees [Samet 80a], rasters to quadtrees [Samet 81b], quadtrees to rasters [Samet 84b], quadtrees to QMATS [Samet 83] and reverse [Samet 85c], quadtrees to edge quadtrees [Ayala et al 85], normalizing quadtrees [Chien et al 84], raster to linear quadtrees [Gargantini 82a, Abel et al 83, Samet 85b, Shaffer et al 87] and treecode to leafcode [Oliver et al 83b].
More details on 2D object representation schemes can be found in [Shapiro 79, Rosenfeld et al 82, Ballard et al 82, Overmars 83, Samet 84a, Samet et al 85].

The conversion algorithms for 3D objects are presented by different workers. Some of them are CSG to octree [Lee et al 82, Samet and Tamminen 85], boundary representation to octrees [Tamminen et al 84 and Tang et al 88] and cylinder representation to octree [Requicha 80] and boundary to CSG [Juan 88].

The 3D object representation can also be generated from the cross sections of the object, as well as from different silhouettes. Based on the paradigm of volume intersection, octrees can be generated from different types of input data. For example, octrees can be generated from quadtrees of serial sections [Kim et al 86], quadtrees of silhouettes [Chien et al 86b and Veenstra et al 86] and depth image quadtrees [Connolly 84].

In between 2D and 3D objects, there is another class of objects, known as $2\frac{1}{2}$ D objects. The two dimensional manifolds in three dimensional space are defined as $2\frac{1}{2}$ D objects. A two dimensional manifold may be thought of as a topological space with the property that every open set is diffeomorphic to an open set in $\mathbb{R}^2$. i.e., locally, it behaves like $\mathbb{R}^2$. The surfaces and boundaries of 3D objects are $2\frac{1}{2}$ D objects.

The judicious selection of proper data structure is difficult because of the plethora of the available data structures. Some of the aspects to be considered in the choice of an appropriate data structure are

(i) space: the amount of space required to store the data
structure, as well as the space required for any
temporary data/data structure is to be considered.

(ii) time: the construction of the data structure that
represents the object in real-time applications needs to
be very prompt.

(iii) fast query processing: one should be able to answer the
queries in a reasonable time. When the query processing
is the top priority, depending on the type of queries,
one can sacrifice the high time requirement to construct
the data structure.

(iv) nature of data: Depending on whether the data is static
or dynamic, i.e., some data structures allow easy
modification of the object.

v) display: a good data structure should enable one to
reconstruct the object from the representation exactly,
without any aberrations and approximations.

vi) translation invariance: in pattern recognition tasks,
it is desirable to have data structures that are
invariant to translations. The MAT, the TID, the RP-
codes and the chain codes are translation invariant.