Chapter 7
Data Structures for Computer Graphics

(This chapter was written for programmers - option in lecture course)

- Any computer model of an **Object** must comprise three different types of entities:

1. Data - basic elements that consist of numerical values, characters, instructions, representation of attributes
2. Algorithm - data manipulation
3. Structure - data organization

Structuring of data is a very important aspect of a *database*, which is a “bank” of the information to be processed and of the results, stored for future use.

- General databases

The most popular data base models are relational, hierarchical, and network.

1. Relational database accesses data in a sequential form. (Figures 7.1, 7.2)

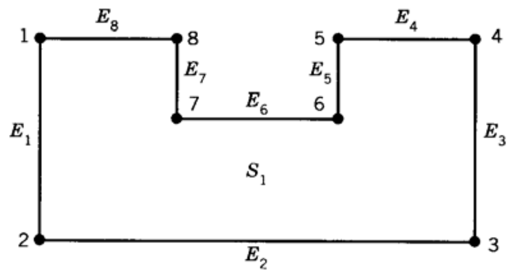


FIGURE 7.1 Object to be described using various types of database.

oint	x	y	Line	Beginning Point	Ending Point	Surface	Line
1	x_1	y_1	E_1	1	2		E_1
2	x_2	y_2	E_2	2	3		E_2
3	x_3	y_3	E_3	3	4		E_3
4	x_4	y_4	E_4	4	5		E_4
5	x_5	y_5	E_5	5	6	S_1	E_5
6	x_6	y_6	E_6	6	7		E_6
7	x_7	y_7	E_7	7	8		E_7
8	x_8	y_8	E_8	8	1		E_8

Figure 7.2 Example of a relational database

2. Hierarchical database is a *tree* structure composed of a hierarchy of elements called nodes. (Figure 7.3)

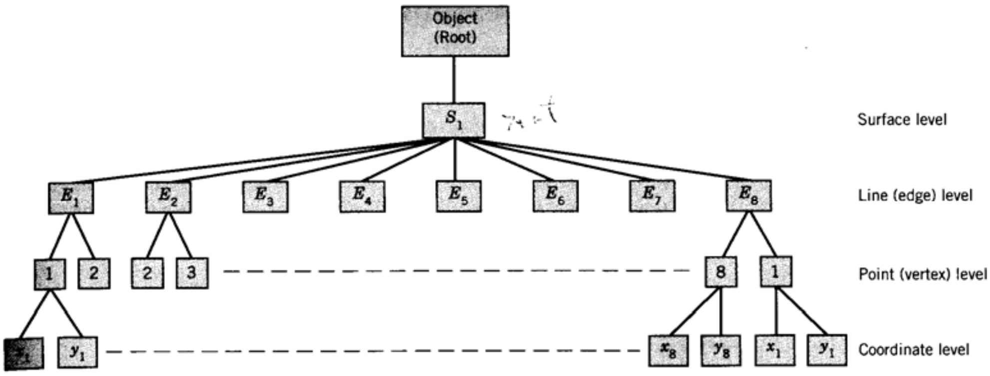


FIGURE 7.3 Example of a hierarchical database.

Hierarchical models are usually simple and fast, but only few relations in the real world are purely hierarchical. The other disadvantages of hierarchical structure are the hierarchical implementation usually creates redundancy and a danger of inconsistency (i.e. cannot implement non-manifold models)

3. Network database has a “many-to-many” relationship among its elements; elements at each level can be connected to many elements of the level above. (Figure 7.4)

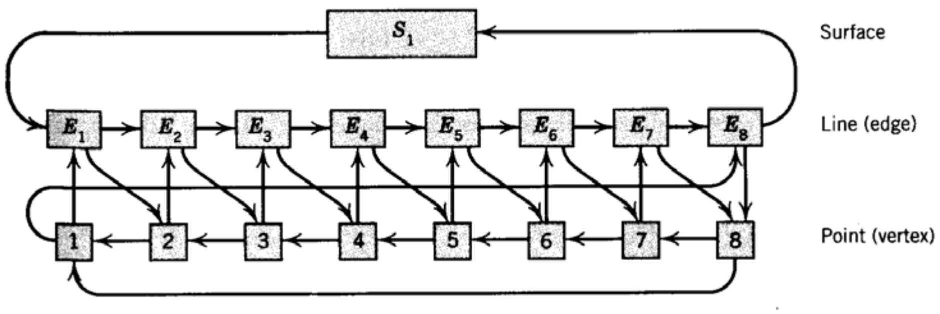


FIGURE 7.4 Example of a network database.

- Basic data structure for graphics

1. Primitive data types - INTEGER, REAL, BOOLEAN (LOGICAL), CHARACTER
2. Static data structure (array) - fixed, predefined values and occupies fixed memory locations.
3. Dynamic data structure (pointer) - dynamic storage allocation

• Static arrays (Homogeneous data structure) allow random access, is widely used in computer graphics.

DIMENSION A (100)	(FORTRAN)
a = array[1..100] of real	(PASCAL)
float a[100];	(C)

• Static array operations:

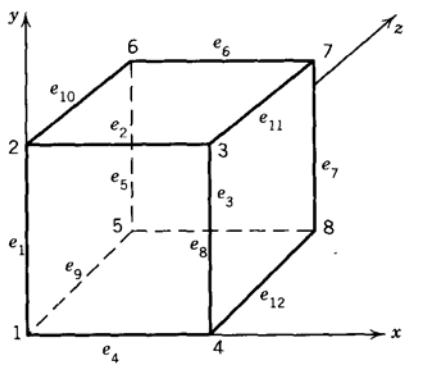
a. Traversal - accessing and processing each element of the array exactly once. (indexing operation)
b. Insertion - adding a new element to an existing list.
c. Deletion - removing an element from an existing list.

• Representation of polyhedral objects using static arrays (Figures 7.6, 7.7, 7.8)

The three arrays used in the description of the cube are:

1. A vertex array listing the x, y, z coordinates of each vertex.
2. An edge array, pointing to the vertex array, indicating the two vertices at the end of each edge
3. A face array, pointing to the edge array, listing the edges that form each face

Arrays: Face ➔ Edge ➔ Vertex



	Edges	Vertices
Face 1	1, 2, 3, 4	1, 2, 3, 4
Face 2	1, 9, 5, 10	1, 5, 6, 2
Face 3	8, 7, 6, 5	5, 8, 7, 6
Face 4	3, 11, 7, 12	3, 7, 8, 4
Face 5	4, 9, 8, 12	1, 5, 8, 4
Face 6	2, 10, 6, 11	2, 6, 7, 3

FIGURE 7.6 Representation of a unit cube.

Vertex (8 × 3)

	x	y	z
1	0.0	0.0	0.0
2	0.0	1.0	0.0
3	1.0	1.0	0.0
4	1.0	0.0	0.0
5	0.0	0.0	1.0
6	0.0	1.0	1.0
7	1.0	1.0	1.0
8	1.0	0.0	1.0

Edge (12 × 2)

	Vertices	
1	1	2
2	2	3
3	3	4
4	4	1
5	5	6
6	6	7
7	7	8
8	8	5
9	1	5
10	2	6
11	3	7
12	4	8

Face (6 × 4)

	Edges			
1	1	2	3	4
2	1	9	5	10
3	8	7	6	5
4	3	11	7	12
5	4	9	8	12
6	2	10	6	11

Face (6 × 4)

	Vertices			
1	1	2	3	4
2	1	5	6	2
3	5	8	7	6
4	3	7	8	4
5	1	5	8	4
6	2	6	7	3

FIGURE 7.7 Arrays used in the description of the unit cube in Figure 7.6.

Vertex Connectivity Matrix

	1	2	3	4	5	6	7	8
1	0	1	0	1	1	0	0	0
2	1	0	1	0	0	1	0	0
3	0	1	0	1	0	0	1	0
4	1	0	1	0	0	0	0	1
5	1	0	0	0	0	1	0	0
6	0	1	0	0	1	0	1	0
7	0	0	1	0	0	1	0	1
8	0	0	0	1	1	0	1	0

Face Connectivity Matrix

	1	2	3	4	5	6
1	0	1	0	1	1	1
2	1	0	1	0	1	1
3	0	1	0	1	1	1
4	1	0	1	0	1	1
5	1	1	1	1	0	0
6	1	1	1	1	0	0

FIGURE 7.8 Connectivity matrices for the unit cube of Figure 7.6.

- Structured data types

record	(PASCAL)
struct	(C)

```
Object =   RECORD
             no_of_edges, no_of_vertices, } : integer ;
             no_of_faces                  }
             vertices : array [1.. NOVERTS] of
                          [RECORD
                          [   x,y,z : real
                          [END;
             edges : array [1.. NOEDGES] of
                          [RECORD
                          [   x1, y1, z1 : real
                          [   x2, y2, z2 : real
                          [END;
             face : array [1.. NOFACES] of array
             [1.. NOEDGES] of integer;
           END;
```

Figure 7.9 (a) PASCAL record implementation

```
struct  point
      {
            float  x,y,z;
       };

struct  edge
      {
            struct point v1 , v2 ;
       };

struct  object
      {
            int no_of_edges,no_of_vertices,
            no_of_faces;
            struct   point   vertices[NUM_VERTS];
            struct   edge   edges[NUM_EDGES];
            int   faces[NUM_FACES][NUM_EDGES];
      };
```

Figure 7.9 (b) C struct implementation

- Pointers

  struct edge { struct point *pv1, *pv2; }

  e1.pv1 = &v1;

  e1.pv2 = &v2;

- Linked lists

In C, a vertex array of variable size n can be dynamically allocated as

vertex_header = (struct point*) calloc (n.sizeof(stuct point));

Single linked list has two fields: data and next link.

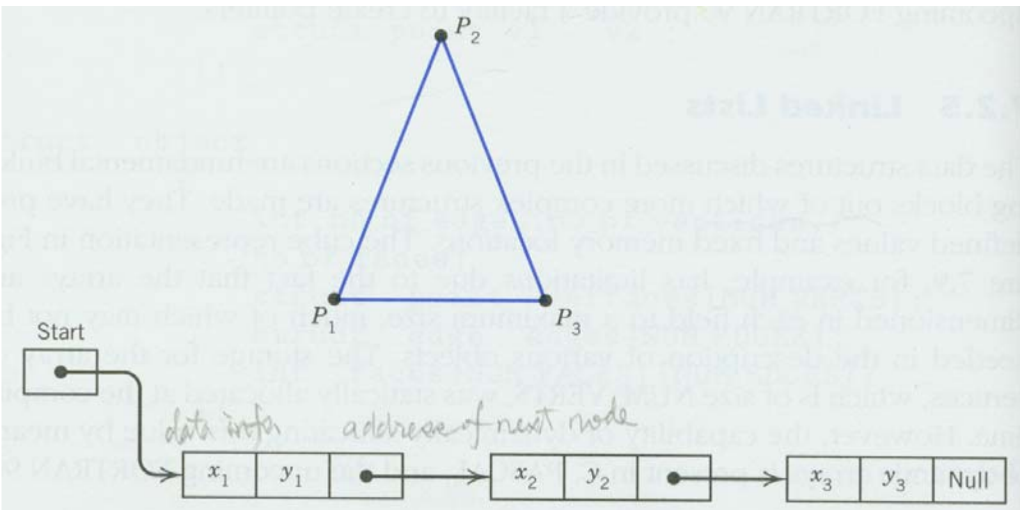


Figure 7.10 Schematic diagram of a linked list

The original list would have to be expanded by the insertion of the new node through three operations:

1. Creation of the new node
2. Linking P2 to the new node
3. Linking the new node to the node P2 was originally point to

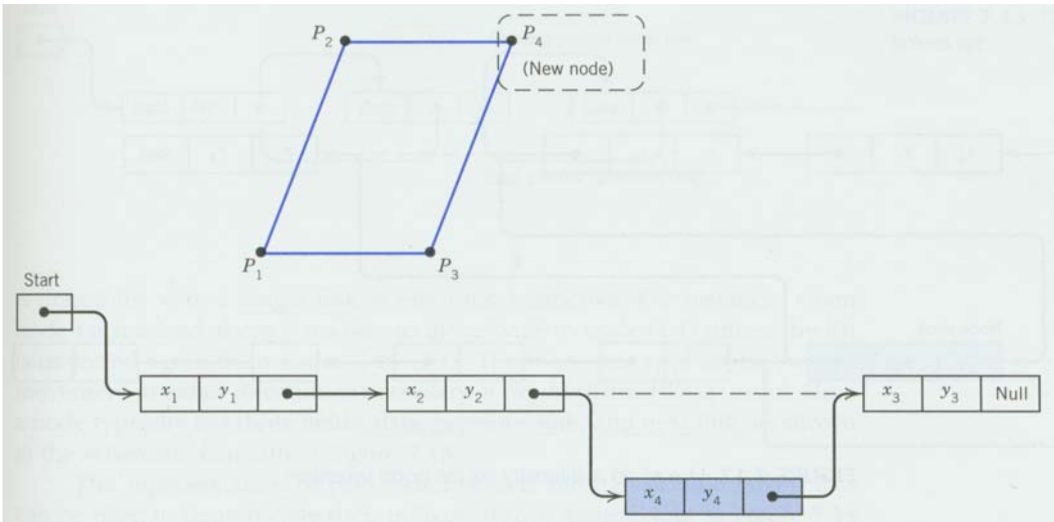


Figure 7.11 Node insertion in a linked list

These unused memory cells would also be linked to form a linked list of available nodes called AVAIL, which uses AVAIL as it list pointer variable.

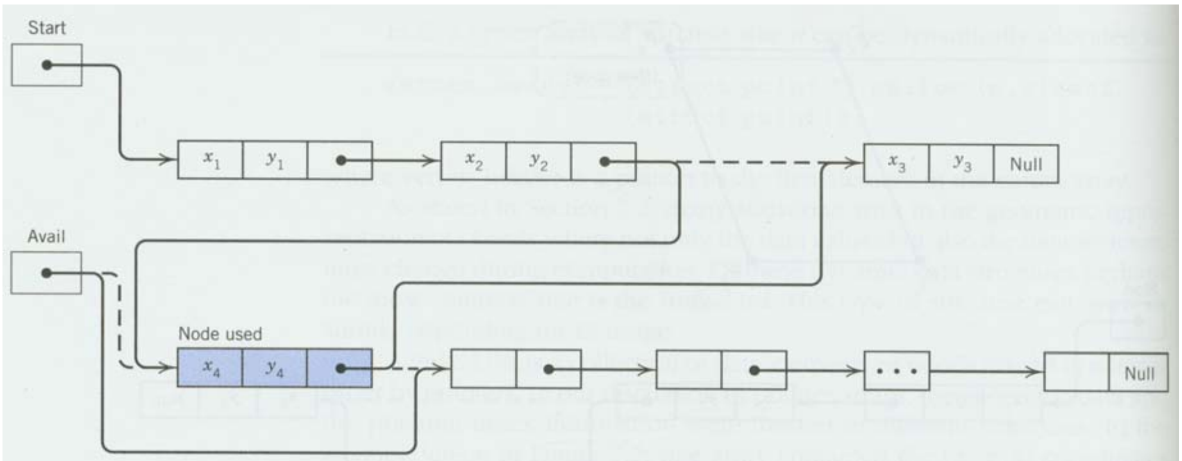


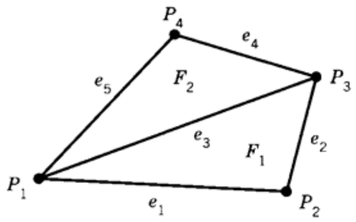
Figure 7.12 Use of an availability list for node insertion

In C, the implementation could be

```
struct node { int x, y;    struct node *next_link; };
struct node *temp, *nlink;
temp = (struct node*) malloc (sizeof(struct node));
temp->x = XNEW;
```

```
temp->y = YNEW;
temp->next_link = NLINK;
NLINK = temp;
```

Double linked list has three fields: data, previous link, and next link.



List of vertices:

$P_1\ (x_1, y_1\ z_1)$
$P_2\ (x_2, y_2, z_2)$
$P_3\ (x_3, y_3, z_3)$
$P_4\ (x_4, y_4, z_4)$

List of edges:

$e\text{list} = \text{ptr}e_1$

$e_1 = (1, \text{ptr}P_1, \text{ptr}P_2, \text{ptr}F_1, \text{null}, \text{ptr}e_2)$
$e_2 = (1, \text{ptr}P_2, \text{ptr}P_3, \text{ptr}F_1, \text{null}, \text{ptr}e_3)$
$e_3 = (2, \text{ptr}P_1, \text{ptr}P_3, \text{ptr}F_1, \text{ptr}F_2, \text{ptr}e_4)$
$e_4 = (1, \text{ptr}P_3, \text{ptr}P_4, \text{ptr}F_2, \text{null}, \text{ptr}e_5)$
$e_5 = (1, \text{ptr}P_1, \text{ptr}P_4, \text{ptr}F_2, \text{null}, \text{null})$

List of faces:

$F\text{list} = (\text{ptr}F_1, \text{ptr}F_2, \text{null})$

$F_1 = (\text{ptr}e_1, \text{ptr}e_2, \text{ptr}e_3, \text{null})$
$F_2 = (\text{ptr}e_3, \text{ptr}e_4, \text{ptr}e_5, \text{null})$

Figure 7.14 Linked lists describing faces of representation of the structure a polyhedral object

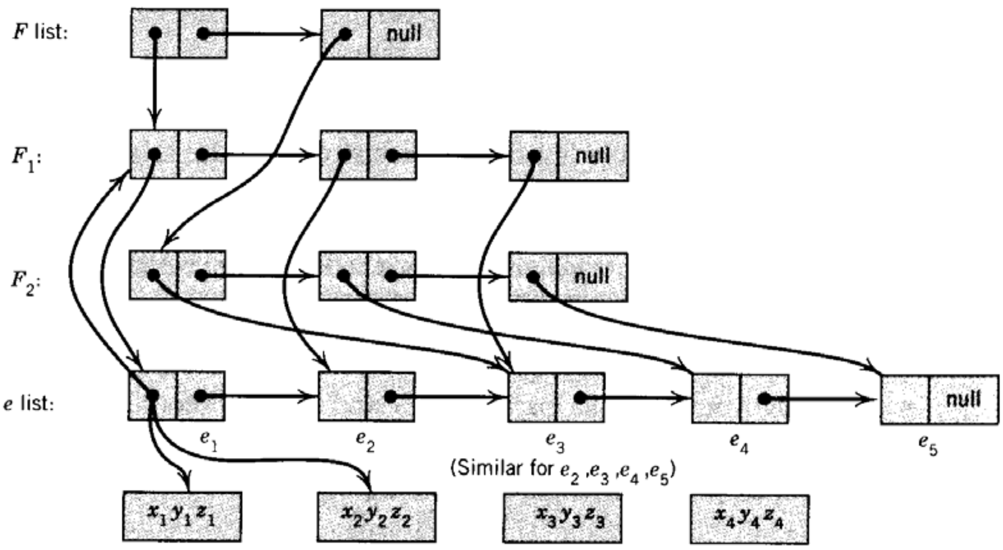


Figure 7.15 Schematic of Figure 7.14

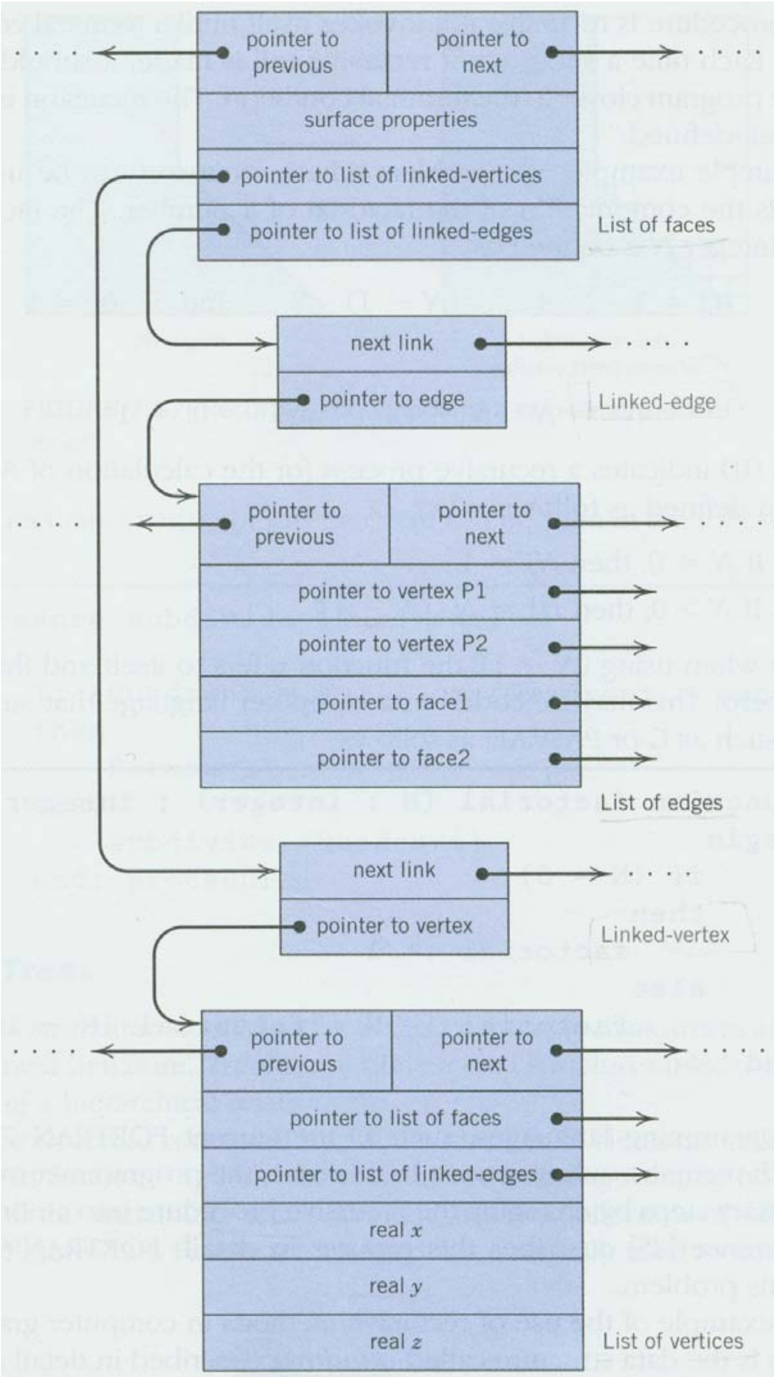


Figure 7.16 A more general hierarchical linked data structure for polyhedral geometry

- Recursion

E. g. Quadtree and octree data structures use of recursive methods in computer graphics applications.

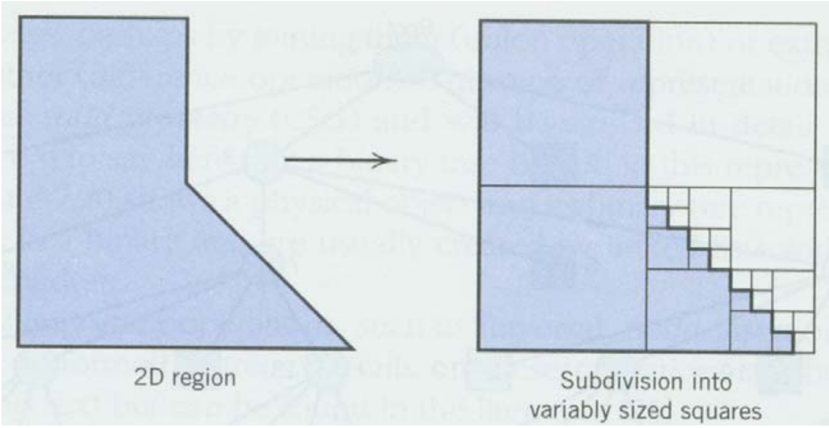


Figure 7.17 Recursive subdivision of a two-dimensional region

• Trees

Binary Tree - Each node of the tree has at most two children. Usually, binary tree is used in CSG (<u>Constructive Solid Geometry</u>) representation. The nodes in a binary tree are usually created by linked lists with pointers to its two children.

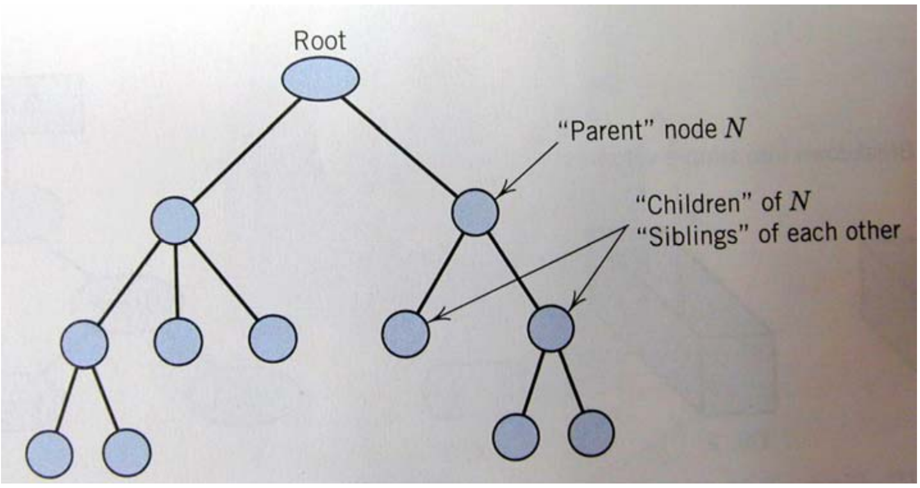


Figure 7.18 A general tree structure

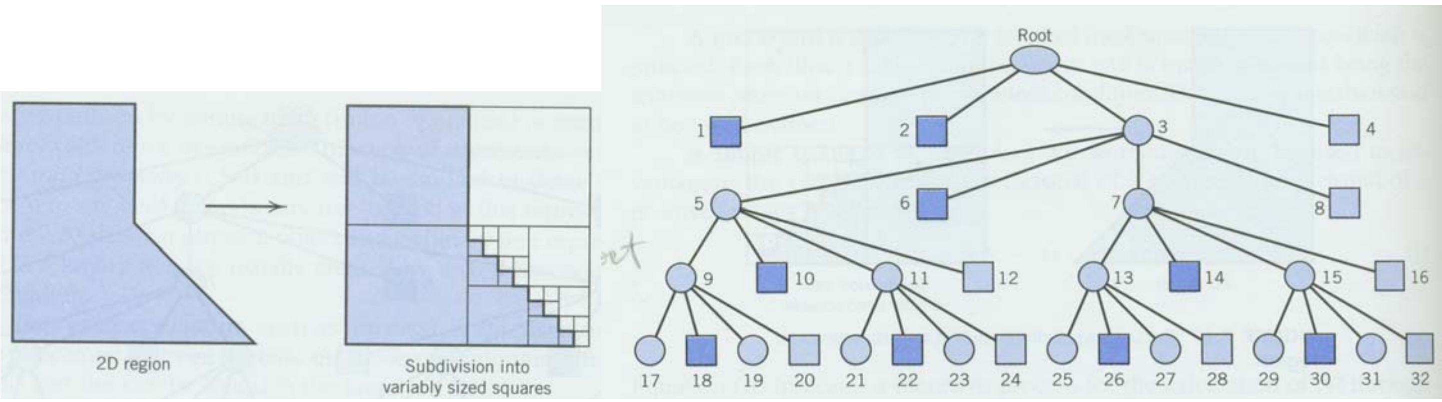


Figure 7.19 Three structure for a quadtree

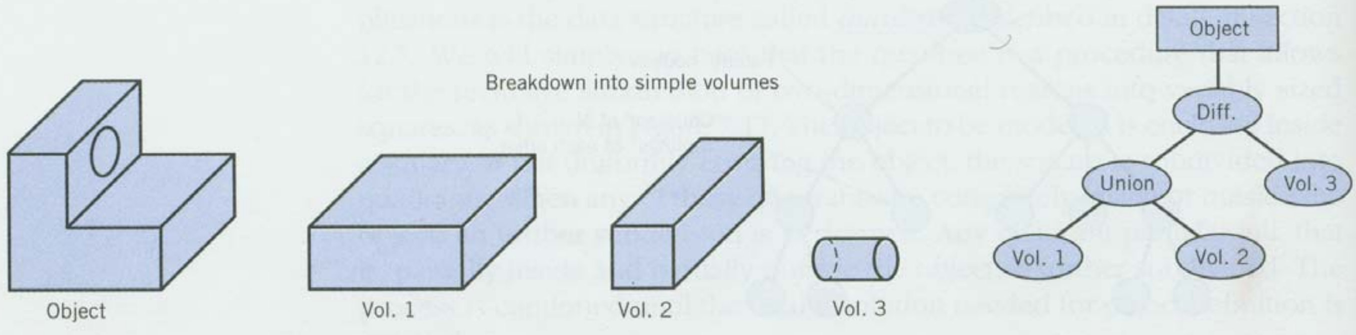


FIGURE 7.20 Binary tree representation.

- Geometry representations:

1. CSG (Constructive Solid Geometry)
2. B-Rep (Boundary Representation)