Tulsiramji Gaikwad-Patil College of Engineering & Technology

Department of Master of Computer Application

Subject Notes

Academic Session: 2018 – 2019

Subject: Computer Graphics

Semester: IV

Q1. (a) What is polygon? How polygon is represented? Write an algorithm for entry of an absolute Polygon into the display file.

Ans:- Polygon:- A polygon may be represented as a number of line segments connected end to end to form a closed figure. It may be represented as the points where the sides of the polygon are connected. The line segments which make up the polygon boundary are called sides of edges. The endpoints of the sides are called the polygon vertices.

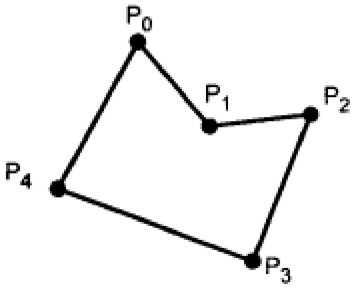


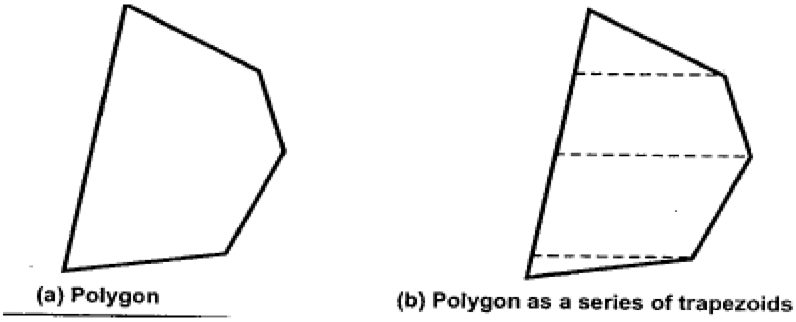
Fig:- Polygon

Representation of Polygon

The closed polyline is a polygon. Each polygon has sides and edges. The end points of the sides are called polygon vertices. To add polygon to our graphics system, we must first decide how to represent it. There are three approached to represent polygon according to the graphics system:

- Polygon drawing primitive approach
- Trapezoid primitive approach
- Line and point approach.

Some graphics devices supports polygon drawing primitive approach. They can directly draw the polygon shapes. On such devices polygon are saved as a unit. Some graphics devices support trapezoid primitive. In such device, trapezoid are formed from two scan lines and two line segments as shown in fig. Here trapezoid are drawn by stepping down the line segments with two vector generator and, for each scan line, filling in all the pixels between them. Therefore every polygon is broken up into trapezoid and its represented as series of trapezoids.

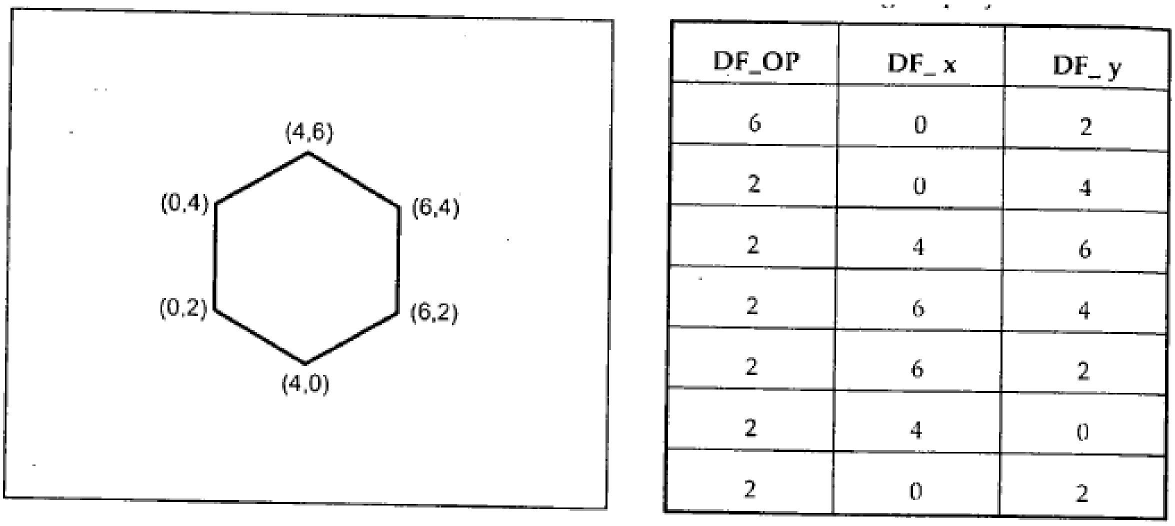


(a) Polygon

(b) Polygon as a series of trapezoids

Fig. Representation of Polygon

Most of the other graphics devices do not provide any polygon supports at all. In such cases polygon are represented using line and points. A polygon is represented as a unit and it is stored in the display file. In a display file polygon cannot be stores only with series of line commands because they do not specify how many of the following line commands are the part of the polygon. Therefore new command is used in the display file to represent polygon. The opcode for new command itself specify the number of line segments in the polygon. The fig shows the polygon and its representation using display file.



DF_OP	DF_ x	DF_ y
6	0	2
2	0	4
2	4	6
2	6	4
2	6	2
2	4	0
2	0	2

Fig:- Polygon and its representation using display file

An Algorithm For Entry Of An Absolute Polygon Into The Display File

Algorithm POLYGON-ABS-2(AX, AY, N) Entry of an absolute polygon into the display file

Arguments AX,AY array containing the vertices of the polygon

N the number of polygon sides

```
Global          DF-PEN-X, DE-PEN-Y the current pen position

Local           I for stepping through the polygon sides

BEGIN

    IF N<3 RETURN ERROR ‘ POLYGON SIZE ERROR’;

    Enter the polygon instruction

    DF-PEN-X ⟵AX[N]

    DF-PEN-Y ⟵ AY[N];

    DISPLAY-FILE-ENTER(N);

    Enter the instruction for the sides

    FOR I = 1 TO N DO LINE-ABS-2(AX[I], AY[I]);

RETURN;

END;
```

(B) Write a BRESENHAM’S algorithm for changing pixel values of the frame buffer along a line segment with integer end points.

Ans:- A **BRESENHAM’S algorithm for changing pixel values of the frame buffer along a line segment with integer end points**

Algorithm BRESENHAM(XA, YA, XB, YB, INENSITY) for changing pixel values of the frame buffer along a line segment with integer endpoints

```
Arguments       XA and YA are the coordinates of one endpoint

                XB and YB are the coordinates of other endpoint

                INTENSITY is the intensity setting to be used for the vector

Global          FRAME the two dimensional frame buffer array

Local           DX, DY the vector to be drawn

                R, C the row and column indices for the pixel

                F the final row and column
                G for testing for a new row or column
                INC1 increment of G when row or column is unchanged
                INC2 increment for G when row or Column changes
                POS-SLOPE a flag to indicate if the slope is positive
BEGIN
        Determine the components of the vector
        DX ⟵ BX – XA ;
        DY ⟵ YB – YA;
        Determine the sign of the slope.
        POS-SLOPE ⟵ (DX>0);
        IF DY < 0 THEN POS-SLOPE ⟵ NOT POS-SLOPE;
        Decide on whether to step across columns or up rows
        IF |DX| > |DY|
         BEGIN
        This is the gentle slope case
        IF DX >0
                BEGIN
                C ⟵ XA;
                R ⟵ YA;
                F ⟵ XB;
                END;
        ELSE
                BEGIN
                C ⟵ XB;
                R ⟵ YB;
                F ⟵ XA;
                END;
```

```
INC1 ⟵ 2 * |DY|;
G ⟵ 2 * |DY| - |DX|;
INC2 ⟵ 2 * (|DY| - |DX|)
IF POS-SLOPE THEN
    BEGIN
    Now step across the line segment
    WHILE C <= F DO
    BEGIN
    Set the nearest pixel in the frame buffer
    FRAME[C, R] ⟵ INTENSITY;
    Next column
    C ⟵ C + 1;
    IF C>= 0 THEN
    BEGIN
    R ⟵ R +1;
    G ⟵ $G + INC1$;
    END;
    END;
END;
ELSE
    BEGIN
    WHILE C <= F DO
    FRAME[C, R] ⟵ INTENSITY;
    C ⟵ C+1;
    IF G >0 THEN
        BEGIN
        R ⟵ R + 1;
```

```
                    G ← INC2;
                    END;
                ELSE G ← G + INC1;
        END;
        END;
END;
ELSE
        BEGIN
        This is the sharp slope case
        Here the above steps are repeated
        With the roles of x and y interchanged
        End;
RETURN;
END;
```

(C) Explain segment table. Write an algorithm to create a named segment.

Ans:- Segment Table:- We would like to organised our display file to reflect this subpicture structure. We would like to divide our display file into segments, each segment corresponding to a component of the overall display. We must give each segment its own unique name so that we can specify it. If we are to change the visibility of a segment, we must have some way to distinguish that segment from all the others. When we refer to display a file segment, we must know which display-file instructions belong to it. This may be determines by knowing where the display file instructions for the segment being and how many of them there are. In our system we shall do this by forming a segment table. We use a number for the mane of the segment. Simple array will serve to hold segment properties, and the segment name will be used as the index into these arrays. We shall have one array containing display file starting location. A second array will hold segment size information, while a third will indicates visibility, and so on.

SEGMENT	SEGMENT –	SEGMENT –	VISIBILITY	SCALE – X	. . . .

NAME	START	SIZE			
0					
1					
2					
3					
4					
.					
.					
.					
.					
.					

Fig:- Segment Table

An Algorithm to Create A Named Segment

Algorithm CREATE-SEGMENT(SEGMENT – NAME) User routine to create a names segment

```
Argument        SEGMENT-NAME the segment name
Global          NOW-OPEN the segment currently open
                FREE the index of the next free display file cell.
                SEGMENT-START, SEGMENT-SIZE, VISIBILITY, ANGLE, SCALE-X,
                SCALE – Y, TRANSLATE – X, TRANSLATE- Y arrays that make up the
                segment tbale
Constant        NUMBER-OF-SEGMENT size of the segment table.
BEGIN
    IF NOW-OPEN > 0 THEN RETURN ERROR ‘ SEGMENT STILL OPEN’;
    IF SEGMENT-NAME < 1 OR SEGMENT-NAME > NUMBER-OF-SEGMENTS
    THEN
        RETURN ERROR ‘ INVALID SEGMENT NAME’
    IF SEGMENT-SIZE[SEGMENT-NEME] > 0 THEN
        RETURN ERROR ‘SEGMENT ALREADY EXISTS’;
    SEGMENT-START[SEGMENT-NAME] ← FREE;
    SEGMENT-SIZE[SEGMENT-NAME] ← 0;
    VISIBILITY [SEGMENT-NAME] ← VISIBILITY[0];
    ANGLE[SEGMENT-NAME] ← ANGLE[0];
    SCALE-X[SEGMENT-NAME] ← SCALE – X[0];
    SCALE-Y[SEGMENT-NAME] ← SCALE – Y[0];
```

```
        TRANSLATE-X[SEGMENT-NAME] ← TRANSLATE – X[0];

        TRANSLATE -Y[SEGMENT-NAME] ← TRANSLATE – Y[0];

        NOW-OPEN  ← SEGMENT-NAME

RETURN

END;
```

(D) Explain

(i) Homogeneous Co – ordinates and Translation

(ii) Rotation about an arbitrary point.

Ans: (i) **Homogeneous co – ordinates and Translation :** In homogeneous coordinates we use 3 x 3 matrices instead of 2 x 2, introducing an additional dummy coordinate w; points are specified by three numbers instead two. The first homogeneous coordinate will be the product of x and w, the second will be the product of y and w, and third will be the just w. A coordinate point (x,y) will be represented by triple (xw, yw, w). The x and y coordinate can easily be recovered by dividing the first and second numbers by the third. We will not really used extra number w until we consider three-dimensional perspective transformations. In two dimensional its value is usually kept at 1 for simplicity. Still, we will discuss it in its generality in anticipation of the three-dimensional transformations.

In homogeneous coordinates our scaling matrix

$$\begin{vmatrix} s_x & 0 \\ 0 & s_y \end{vmatrix}$$

Becomes

$$s = \begin{vmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

If we apply this to the point (xw, yw, w), we obtain

$$\begin{vmatrix} xw & yw & w \end{vmatrix} \begin{vmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} s_x xw & s_y yw & w \end{vmatrix}$$

Dividing by the third number w gives

$(s_x x, s_x y)$

Which is the correctly scaled point.

The counter clockwise rotation matrix

$$\begin{vmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{vmatrix}$$

Becomes, using homogeneous coordinates,

$$R = \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Applying it to point (x,y) with homogeneous coordinates (xw, yw, w) gives

$$[xw \quad yw \quad w] \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} = [(xw\cos\theta - yw\sin\theta) \quad (xw\sin\theta + yw\cos\theta) \quad w]$$

For the correctly rotated point

$(x\cos\theta - y\ \sin\theta),(x\sin\theta + y\cos\theta)$

The homogeneous coordinates transformation matrix for a translation of t_x, t_y is

$$T = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{vmatrix}$$

To show that this is so, we apply the matrix

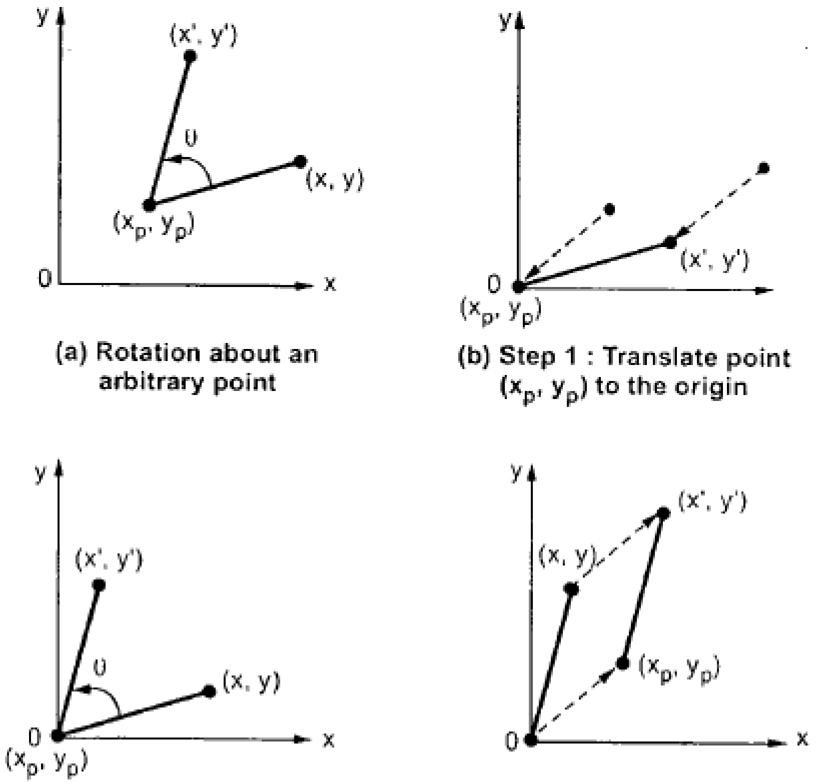
$$[xw \quad yw \quad w] \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{vmatrix} = [\ (xw + t_x w) \quad (yw+t_y w) \quad w]$$

For the translated point ($x + t_x$, $y + t_y$)

(ii) Rotation about an arbitrary point

Let’s determine the transformation matrix for a couterclockwise rotation about point (x_p, y_p)

We shall do this by three transformation steps. We shall translate the point (x_p, y_p) to the origin, rotate about the origin, and then translate the center of rotation back where it belong.



(a) Rotation about an arbitrary point

(b) Step 1 : Translate point (x_p, y_p) to the origin

(c) Step 2 : Rotate it about the origin

(d) Step 3 : Translate back to the original position

The translation matrix to move point (x_p, y_p) to the origin is given by

$$T_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_p & -y_p & 1 \end{bmatrix}$$

The rotation matrix for counterclockwise rotation of point about the origin is given as

$$R = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The translation matrix to move the center point back to its original position is given as

$$T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_p & y_p & 1 \end{bmatrix}$$

Therefore, the overall transformation matrix for a counter clockwise rotation by an angle Θ about the point (x_p, y_p) is given as

$$T_1 \cdot R \cdot T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_p & -y_p & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_p & y_p & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -x_p\cos\theta + y_p\sin\theta & -x_p\sin\theta - y_p\cos\theta & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_p & y_p & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -x_p\cos\theta + y_p\sin\theta + x_p & -x_p\sin\theta - y_p\cos\theta + y_p & 1 \end{bmatrix} \quad \ldots (4.18)$$

This is the overall transformation for a rotation by Θ counterclockwise about the point (x_p,y_p).

Q1. (A) Write and Explain DDA algorithm for line generation.

Ans:- Digital Differential Analyser

We know that the slope of a straight line is given as

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \quad \text{-----------} (1)$$

The above differential equation ca be used to obtain a rasterized straight line, For any given x interval dx along a line, we can compute the corresponding y interval dy for the equation (1) as

$$\Delta y = \frac{y_2 - y_1}{x_2 - x_1} \Delta x$$

---------------(2)

Similarly, we can obtain the x interval dx corresponding to a specific dy as

$$\Delta x = \frac{x_2 - x_1}{y_2 - y_1} \Delta y$$

---------------------(3)

Once the intervals are known the values for next a and next y on the straight line can be obtained as follows

$$\begin{aligned} x_{i+1} &= x_i + \Delta x \\ &= x_i + \frac{x_2 - x_1}{y_2 - y_1} \Delta y \end{aligned}$$

--------------- (4)

And

$$\begin{aligned} y_{i+1} &= y_i + \Delta y \\ &= y_i + \frac{y_2 - y_1}{x_2 - x_1} \Delta x \end{aligned}$$

--------------- (5)

The equation 4 and 5 represents a recursion for successive values of x and y along the required line. Such a way of rasterizing a line is called a **Digital differential analyzer (DDA).** For simple DDA either dx or dy, whichever is larger, is chosen as one raster unit i.e.

If | dx | >= | dy | then

dx = 1

else dy = 1

With this simplification, if dx = 1 then

We have

$$y_{i+1} = y_i + \frac{y_2 - y_1}{x_2 - x_1} \text{ and}$$

$$x_{i+1} = x_i + 1$$

If dy = 1 then

We have

$$y_{i+1} = y_i + 1 \text{ and}$$
$$x_{i+1} = x_i + \frac{x_2 - x_1}{y_2 - y_1}$$

Let us see the DDA algorithm

```
1. Read the line end points (x1, y1) and (x2,y2) such that they are not equal.
2. dx= | x2 – x1|  and  dy = | y2 – y1|
3. if (dx >= dy) then
           length = dx
   else
           length = dy
   end if
4. dx = (x2 – x1) / length
   dy = (y2 – y1) / length
   [ This makes a either dx or dy equal to 1 because length is either | x2 – x1| or | y2 – y1|.
   Therefore, the incremental value for either x or y is one]
5. x = x1 + 0.5 * (sign (dx)
   y = y1 + 0.5 * sign (dy)
   [ Here, sign function makes the algorithm work in all quadrant. It returns -1, 0, 1
   depending on whether its arguments is <0, =0,>0 respectively. The factor 0.5 makes it
   possible to round the values in the integer function rather than truncating them]
6. i =1
   While (i <= length)
   {
           Plot (Integer (x), integer (y))
           X = x + dx
           Y = y + dy
           I = i + 1
```

```
    }
7.  Stop
```

(B) What is frame buffer? Write an algorithm to display the contents of a frame buffer.

Ans :- Frame Buffer :- A frame buffer is a large, contiguous piece of computer memory. At a minimum there is one memory bit for each pixel in the rater; this amount of memory is called a bit plane. The picture is built up in the frame buffer one bit at a time. You know that a memory bit has only two states, therefore a single bit plane yields a black-and white display. You know that a frame buffer is a digital device and the CRT is an analog device. Therefore, a conversion from a digital representation to an analog signal must take place when information is read from the frame buffer and displayed on the raster CRT graphics device. For this you can use a digital to analog converter (DAC).Each pixel in the frame buffer must be accessed and converted before it is visible on the raster CRT.

Algorithm to display the contents of a Frame buffer

Algorithm Display This displays the contents of the frame buffer

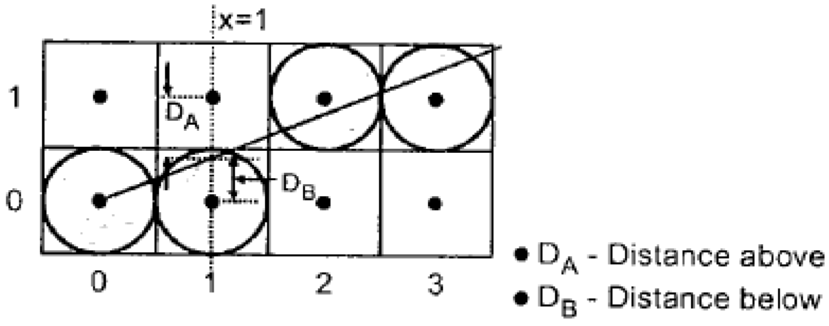
```
Global : FRAME the frame buffer array
        WIDTH-START and HEIGHT-START the starting indices of FRAME
        WIDTH-END and HEIGHT-END the ending indices of FRAME
Local: X, Y the pixel being displayed
BEGIN
        FOR Y = HEIGHT-END TO HEIGHT-START DO
            PRINT FOR X = WIDTH-START TO WIDTH-END, FRAME[X,Y];
        RETURN;
END;
```

(C) Write and explain Bresenham’s line generation algorithm with suitable example.

Ans:- Bresenham's line algorithm uses only integer addition and subtraction and multiplication by 2, and we know that the computer can perform the operations of integer addition and subtraction very rapidly. The computer is also time-efficient when performing integer multiplication by powers of 2. Therefore, it is an efficient method for scan-converting straight lines.

Although developed originally for use with digital plotters, Bresenham's algorithm is equally suited for use with CRT raster devices. The basic principle of Bresenham's line algorithm is to select the optimum raster locations to represent a straight line. To accomplish this, the algorithm always increments either x or y by one unit depending on the slope of line. The increment in the other variable is determined by examining the distance between the actual line location and the nearest pixel. This distance is called decision variable or the error.

This is illustrated in the Fig. below.



As shown in the figure above, the line does not pass through all raster points (pixels). It passes through raster point (0, 0) and subsequently crosses three pixels. It is seen that the intercept of line with the line x = 1 is closer to the line y = 0, i.e. pixel (1, 0) than to the line y = 1 i.e. Pixel (1, 1). Hence, in this case, the raster point at (1, 0) better represents the path of the line than that at (1. 1). The intercept of the line with the line x = 2 is close to the line y = 1. i.e. pixel (2, 1) than to the line y = 0. i.e. pixel (2,0). Hence, the raster point at (2, 1) better represents the path of the line, as shown in the figure above.

Bresenham's Line algorithm

1. Read the line end points (x1, y1) and (x2,y2) such that they are not equal.

[if equal then plot that point and exit]

```
2. $\Delta x = |x2 - x1|$ and $\Delta y = |y2 - y1|$
3. [Initialize starting point]
    x = x1 and y = y1
4. $e = 2 * \Delta y - \Delta x$
   [Initialize value of decision variable or error to compensate for nonzero intercepts]
5. i=1[ Initialize counter ]
6. Plot(x , y)
7. whi1e( $e \geq 0$)
   {
     y = y + 1
     $e = e - 2 * \Delta x$
   }
     x = x + 1
    $e = e + 2 * \Delta y$
8. i = i + 1
9. if($i \leq \Delta x$) then go to step6.
10. Stop.
```

(D) Write a transformation routine in terms of scaling, rotation and translation.

Ans:- We construct routines for translating, rotating and scaling. The routines that create the transformation will modify a homogeneous coordinate's transformation matrix.

The following three algorithms describe user routines for saving transformation parameters until it is time interpret the display file.

```
Algorithm TRANSLATE(TX,TY) User routine to set the translation parameters

Arguments:    TX,TY the translation amount

Global:       TRNX, TRNY storage for the translation parameters.

Begin

    TRNX<- TX;

    TRNY <- TY;

    CALL NEWFRAME;

    RETURN;

END;
```

```
Algorithm SCALE (SX, SY) User routine to set the scaling parameters

Argument:     SX, SY the scaling factors

Global:       SCLX, SCLY storage for the scale parameters.

Begin

    SCLX <- SX;

    SCLY <- SY;

    CALL NEWFRAME;

    RETURN;

END;
```

```
Algorithm ROTATE(A) User routine to set the rotation angle.

Arguments:    A the rotation angle

Global:       ANGL a place to save the rotation angle.
```

```
BEGIN
        ANGL <- A
        CALL NEWFRAME;
        RETURN
END;
```

1. **(a) Write an algorithm for vector generation of a line and explain in detail.**

Ans:- Vector Generation of a Line

Digital Differential Analyser

We know that the slope of a straight line is given as

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \quad \text{----------- (1)}$$

The above differential equation ca be used to obtain a rasterized straight line, For any given x interval dx along a line, we can compute the corresponding y interval dy for the equation (1) as

$$\Delta y = \frac{y_2 - y_1}{x_2 - x_1} \Delta x \quad \text{---------------(2)}$$

Similarly, we can obtain the x interval dx corresponding to a specific dy as

$$\Delta x = \frac{x_2 - x_1}{y_2 - y_1} \Delta y \quad \text{---------------------(3)}$$

Once the intervals are known the values for next a and next y on the straight line can be obtained as follows

$$\begin{aligned} x_{i+1} &= x_i + \Delta x \\ &= x_i + \frac{x_2 - x_1}{y_2 - y_1} \Delta y \end{aligned} \quad \text{--------------- (4)}$$

And

$$\begin{aligned} y_{i+1} &= y_i + \Delta y \\ &= y_i + \frac{y_2 - y_1}{x_2 - x_1}\Delta x \end{aligned}$$ --------------- (5)

The equation 4 and 5 represents a recursion for successive values of x and y along the required line. Such a way of rasterizing a line is called a **Digital differential analyzer (DDA).** For simple DDA either dx or dy, whichever is larger, is chosen as one raster unit i.e.

If | dx | >= | dy | then

dx = 1

else dy = 1

With this simplification, if dx = 1 then

We have

$$y_{i+1} = y_1 + \frac{y_2 - y_1}{x_2 - x_1} \text{ and}$$

$$x_{i+1} = x_i + 1$$

If dy = 1 then

We have

$$\begin{aligned} y_{i+1} &= y_i + 1 \text{ and} \\ x_{i+1} &= x_i + \frac{x_2 - x_1}{y_2 - y_1} \end{aligned}$$

Let us see the DDA algorithm

```
1. Read the line end points (x1, y1) and (x2,y2) such that they are not equal.
2. dx= | x2 – x1|  and  dy = | y2 – y1|
3. if (dx >= dy) then
           length = dx
   else
           length = dy
```

```
    end if
4.  dx = (x2 – x1) / length

    dy = (y2 – y1) / length

    [ This makes a either dx or dy equal to 1 because length is either | x2 – x1| or | y2 – y1|.
    Therefore, the incremental value for either x or y is one]

5.  x = x1 + 0.5 * (sign (dx)

    y = y1 + 0.5 * sign (dy)

    [ Here, sign function makes the algorithm work in all quadrant. It returns -1, 0, 1
    depending on whether its arguments is <0, =0,>0 respectively. The factor 0.5 makes it
    possible to round the values in the integer function rather than truncating them]

6.  i =1
    While (i <= length)
    {
            Plot (Integer (x), integer (y))
            X = x + dx
            Y = y + dy
            I = i + 1
    }
7.  Stop
```

(b) Explain segment creation in detail.

Ans:- Segment Table:- We would like to organised our display file to reflect this subpicture structure. We would like to divide our display file into segments, each segment corresponding to a component of the overall display. We must give each segment its own unique name so that we can specify it. If we are to change the visibility of a segment, we must have some way to distinguish that segment from all the others. When we refer to display a file segment, we must know which display-file instructions belong to it. This may be determines by knowing where the display file instructions for the segment being and how many of them there are. In our system we shall do this by forming a segment table. We use a number for the mane of the segment. Simple array will serve to hold segment properties, and the segment name will be used as the index into these arrays. We shall have one array containing display file starting location. A second array will hold segment size information, while a third will indicates visibility, and so on.

SEGMENT NAME	SEGMENT – START	SEGMENT – SIZE	VISIBILITY	SCALE – X	. . . .
0					
1					
2					
3					
4					
.					
.					
.					
.					
.					

Fig:- Segment Table

An Algorithm to Create A Named Segment

Algorithm CREATE-SEGMENT(SEGMENT – NAME) User routine to create a names segment

Argument SEGMENT-NAME the segment name

Global NOW-OPEN the segment currently open

FREE the index of the next free display file cell.

SEGMENT-START, SEGMENT-SIZE, VISIBILITY, ANGLE, SCALE-X, SCALE – Y, TRANSLATE – X, TRANSLATE- Y arrays that make up the segment tbale

Constant NUMBER-OF-SEGMENT size of the segment table.

BEGIN

```
        IF NOW-OPEN > 0 THEN RETURN ERROR ‘ SEGMENT STILL OPEN’;
        IF SEGMENT-NAME < 1 OR SEGMENT-NAME > NUMBER-OF-SEGMENTS
        THEN
                RETURN ERROR ‘ INVALID SEGMENT NAME’
        IF SEGMENT-SIZE[SEGMENT-NEME] > 0 THEN
                RETURN ERROR ‘SEGMENT ALREADY EXISTS’;
        SEGMENT-START[SEGMENT-NAME] ← FREE;
        SEGMENT-SIZE[SEGMENT-NAME] ← 0;
        VISIBILITY [SEGMENT-NAME] ← VISIBILITY[0];
        ANGLE[SEGMENT-NAME] ← ANGLE[0];
        SCALE-X[SEGMENT-NAME] ← SCALE – X[0];
        SCALE-Y[SEGMENT-NAME] ← SCALE – Y[0];
        TRANSLATE-X[SEGMENT-NAME] ← TRANSLATE – X[0];
        TRANSLATE -Y[SEGMENT-NAME] ← TRANSLATE – Y[0];
        NOW-OPEN ← SEGMENT-NAME
RETURN
END;
```

2. (a) Write an algorithm for vector generation of a line and explain in detail.

Ans:- Vector Generation of a Line

Digital Differential Analyser

We know that the slope of a straight line is given as

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \quad \text{----------- (1)}$$

The above differential equation ca be used to obtain a rasterized straight line, For any given x interval dx along a line, we can compute the corresponding y interval dy for the equation (1) as

$$\Delta y = \frac{y_2 - y_1}{x_2 - x_1} \Delta x$$

---------------(2)

Similarly, we can obtain the x interval dx corresponding to a specific dy as

$$\Delta x = \frac{x_2 - x_1}{y_2 - y_1} \Delta y$$

---------------------(3)

Once the intervals are known the values for next a and next y on the straight line can be obtained as follows

$$\begin{aligned} x_{i+1} &= x_i + \Delta x \\ &= x_i + \frac{x_2 - x_1}{y_2 - y_1} \Delta y \end{aligned}$$

--------------- (4)

And

$$\begin{aligned} y_{i+1} &= y_i + \Delta y \\ &= y_i + \frac{y_2 - y_1}{x_2 - x_1} \Delta x \end{aligned}$$

--------------- (5)

The equation 4 and 5 represents a recursion for successive values of x and y along the required line. Such a way of rasterizing a line is called a **Digital differential analyzer (DDA).** For simple DDA either dx or dy, whichever is larger, is chosen as one raster unit i.e.

If | dx | >= | dy | then

dx = 1

else dy = 1

With this simplification, if dx = 1 then

We have

$$y_{i+1} = y_i + \frac{y_2 - y_1}{x_2 - x_1} \text{ and}$$

$$x_{i+1} = x_i + 1$$

If dy = 1 then

We have

$$y_{i+1} = y_i + 1 \text{ and}$$
$$x_{i+1} = x_i + \frac{x_2 - x_1}{y_2 - y_1}$$

Let us see the DDA algorithm

```
8.  Read the line end points (x1, y1) and (x2,y2) such that they are not equal.
9.  dx= | x2 – x1|  and  dy = | y2 – y1|
10. if (dx >= dy) then

            length = dx

    else
            length = dy
    end if
11. dx = (x2 – x1) / length

    dy = (y2 – y1) / length

    [ This makes a either dx or dy equal to 1 because length is either | x2 – x1| or | y2 – y1|.
    Therefore, the incremental value for either x or y is one]

12. x = x1 + 0.5 * (sign (dx)

    y = y1 + 0.5 * sign (dy)

    [ Here, sign function makes the algorithm work in all quadrant. It returns -1, 0, 1
    depending on whether its arguments is <0, =0,>0 respectively. The factor 0.5 makes it
    possible to round the values in the integer function rather than truncating them]

13. i =1
    While (i <= length)
    {
            Plot (Integer (x), integer (y))
            X = x + dx
            Y = y + dy
            I = i + 1
      }
  14. Stop
```

(b) Explain segment creation in detail.

Ans:- Segment Table:- We would like to organised our display file to reflect this subpicture structure. We would like to divide our display file into segments, each segment corresponding to a component of the overall display. We must give each segment its own unique name so that we can specify it. If we are to change the visibility of a segment, we must have some way to distinguish that segment from all the others. When we refer to display a file segment, we must know which display-file instructions belong to it. This may be determines by knowing where the display file instructions for the segment being and how many of them there are. In our system we shall do this by forming a segment table. We use a number for the mane of the segment. Simple array will serve to hold segment properties, and the segment name will be used as the index into these arrays. We shall have one array containing display file starting location. A second array will hold segment size information, while a third will indicates visibility, and so on.

SEGMENT NAME	SEGMENT – START	SEGMENT – SIZE	VISIBILITY	SCALE – X	. . . .
0					
1					
2					
3					
4					
.					
.					
.					

.					
.					

Fig:- Segment Table

An Algorithm to Create A Named Segment

Algorithm CREATE-SEGMENT(SEGMENT – NAME) User routine to create a names segment

```
Argument        SEGMENT-NAME the segment name

Global          NOW-OPEN the segment currently open

                FREE the index of the next free display file cell.

                SEGMENT-START, SEGMENT-SIZE, VISIBILITY, ANGLE, SCALE-X,
                SCALE – Y, TRANSLATE – X, TRANSLATE- Y arrays that make up the
                segment tbale

Constant        NUMBER-OF-SEGMENT size of the segment table.

BEGIN

        IF NOW-OPEN > 0 THEN RETURN ERROR ‘ SEGMENT STILL OPEN’;

        IF SEGMENT-NAME < 1 OR SEGMENT-NAME > NUMBER-OF-SEGMENTS

        THEN

                RETURN ERROR ‘ INVALID SEGMENT NAME’

        IF SEGMENT-SIZE[SEGMENT-NEME] > 0 THEN

                RETURN ERROR ‘SEGMENT ALREADY EXISTS’;

        SEGMENT-START[SEGMENT-NAME] ← FREE;

        SEGMENT-SIZE[SEGMENT-NAME] ← 0;

        VISIBILITY [SEGMENT-NAME] ← VISIBILITY[0];
        ANGLE[SEGMENT-NAME] ⟵ ANGLE[0];

        SCALE-X[SEGMENT-NAME] ⟵ SCALE – X[0];

        SCALE-Y[SEGMENT-NAME] ⟵ SCALE – Y[0];

        TRANSLATE-X[SEGMENT-NAME] ⟵ TRANSLATE – X[0];

        TRANSLATE -Y[SEGMENT-NAME] ⟵ TRANSLATE – Y[0];

        NOW-OPEN  ⟵ SEGMENT-NAME

RETURN

END;
```

UNIT - II

Q2.(a) Write CLIP_POLYGON_EDGE (OP, X, Y) algorithm to close and enter a clipped polygon into the display file.

Ans:- Clipping polygon routine will remove some of the polygon sides, and it will insert a move command instead of a line command along the window boundary. This change in the number of sides in the polygon must be reflected in our initial polygon-drawing operation code. The algorithm for clip polygon edge is as follow

Algorithm CLIP_POLYGON_EDGE (OP, X, Y) close and enter a clipped polygon into the display file.

```
Argument        OP, X, Y a display-file instruction

Global          PFLAG indicates that a polygon is being drawn

                COUNT-IN the number of sides remaining to be processed

                COUNT-OUT the number of sides to be entered in the display file.

                IT, XT, YT temporary storage arrays for a polygon

                NEEDFIRST array of indictors for saving the first command

                FIRSTOP, FIRSTX, FIRSTY arrays for saving the first command

                CLOSSING indicates the stage in polygon

Local           I for stepping through the polygon sides.

BEGIN

        COUNT-IN ← COUNT-IN – 1;

        CLIP-LEFT (OP, X, Y);

        IF COUNT-IN ≠ 0 THEN RETURN;

        Close the clipped polygon

        CLOSING ← 1;

        IF NOT NEEDFIRST[1] THEN CLIP-LEFT(FIRSTOP[1], FIRSTX[1], FIRST[1]);

        CLOSING ← 2;
```

(b) Write an algorithm ENABLE_GROUP (CLASS) to enable an input devices class.

Ans:-

Algorithm ENABLE_GROUP (CLASS) Routine to enable an input device class.

Argument CLASS the code for the class to be enabled

Global BUTTON, PICK, KEYBOARD, LOCATOR, VALUATOR device flags

```
BEGIN
    IF CLASS = 1 THEN
    BEGIN
    PERFOMR ALL OPERATIONS NEEDED TO PERMIT INPUT FROM THE BUTTON DEVICES;
    BUTTON ← TRUE;
    END;
    IF CLASS = 2 THEN
    BEGIN
    PERFOMR ALL OPERATIONS NEEDED TO PERMIT INPUT FROM THE PICK DEVICES;
    PICK ← TRUE;
    END;
    IF CLASS = 3 THEN
    BEGIN
    PERFOMR ALL OPERATIONS NEEDED TO PERMIT INPUT FROM THE KEYBOARD DEVICES;
    KEYBOARD ← TRUE;
    END;
    IF CLASS = 4 THEN
    BEGIN
    PERFOMR ALL OPERATIONS NEEDED TO PERMIT INPUT FROM THE LOCATOR DEVICES;
    LOCATOR ← TRUE;
    END;
    IF CLASS = 5 THEN
    BEGIN
    PERFOMR ALL OPERATIONS NEEDED TO PERMIT INPUT FROM THE VALUATOR DEVICES;
    VALUATOR ← TRUE;
    END;
    RETURN;
END;
```

(C) Explain:

(i) Simulating a locator with a pick

(ii) Echoing.

Ans:- (i) Simulating a locator with a pick :- While the light pen can be used to select an object on the screen, it does not give that object’s position. Nor it can be used to indicate a position where there is no object, as can be done with a joystick or tablet. In order to use a light pen for position information, a tracking cross is employed. A tracking cross is a small cross made of four or more separate line segment. This is placed at some known position on the screen and selected so that it is detectable by the light pen (see fig.)

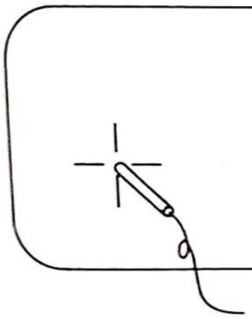


Fig:- Light pen and tracking cross

The pen is positioned at the center of the cross. Then a s the pen is moved, it will encounter one of the cross arm. If the pen is moved slightly to the right, it will encounter the right arm of the cross. This information is used to move the cross slightly to the right. After each refresh the cross will be moved until the light pen in once again centered. (see fig.)

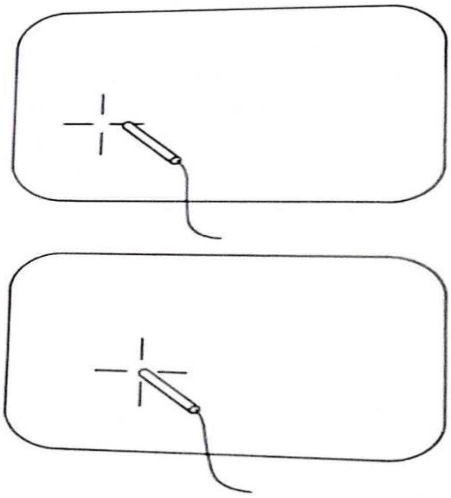


Fig:- if the light pen encounters one of the arms of the cross, the position of the cross is shifted.

A similar action is taken for all of the other arms of the cross so that the tracking cross will follow the movements of the light pen. Since the position of the center of the tracking cross is known, positional information can be entered by “grabing” the cross with the light pen and moving it to the desired location.

(ii) ECHOING

An important part of an interactive is echoing. Echoing provides the user with information about his actions. This allows the user to compare what he has done against what he wanted to do. For keyboard input, echoing usually takes the form of displaying the typed characters. Locator may be echoed by a screen cursor displayed at the current locator position. This allows the user to see the current locator setting and to relate its position to the object on the display. (see fig.)

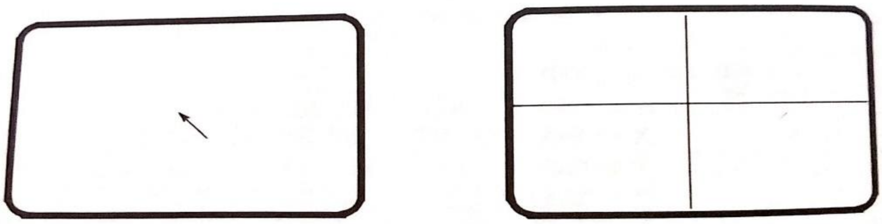


Fig:- Echoing a location

A pick may be echoed by identifying the selected objects on the display. The selected objects may be flashed or made brighter or, perhaps, altered in color. This will allow the user to determine whether or not he has selected the intended objects (see fig.)

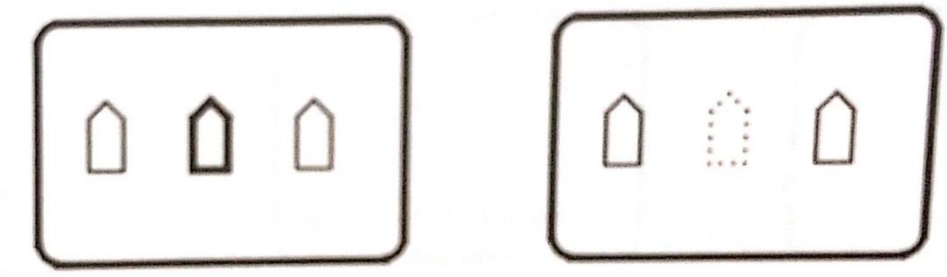


Fig:- Echoing a pick

Buttons, when used to select menu items, can be echoed by flashing or highlighting the selected items. (see fig.)

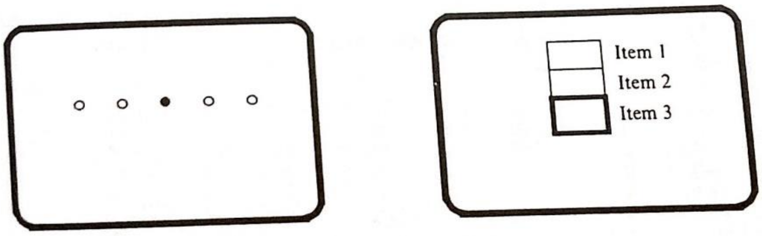


Fig:- Echoing buttons and menu selection

For a valuator, display of the current setting in numerical form or as a position on a scale is possible. (See. Fig.)

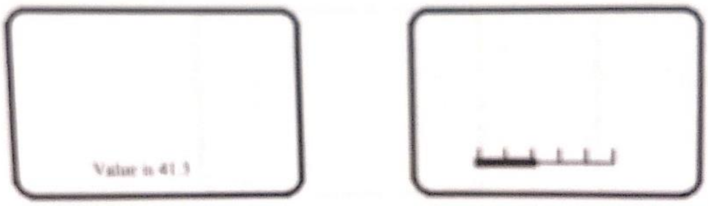


FIGURE 7-37

Fig.:- Echoing a value

It is important that some form of echoing be present, no matter what form it takes.

(d) What is viewing transformation? Write algorithms for setting the viewport and window dimensions.

Ans:- **Viewing Transformation :-** There is a object model and there is the image of the object which appears on the display. When we speak of the object, we are actually referring to a model of the object stored in the computer. The object model is said to reside in object space. This model represents the object using the physical units of length. In the object space, lengths of the object may be measured in any units from light years to angstroms. In the object space, length of the image on the screen, however, must be measured in screen coordinates. (see fig)

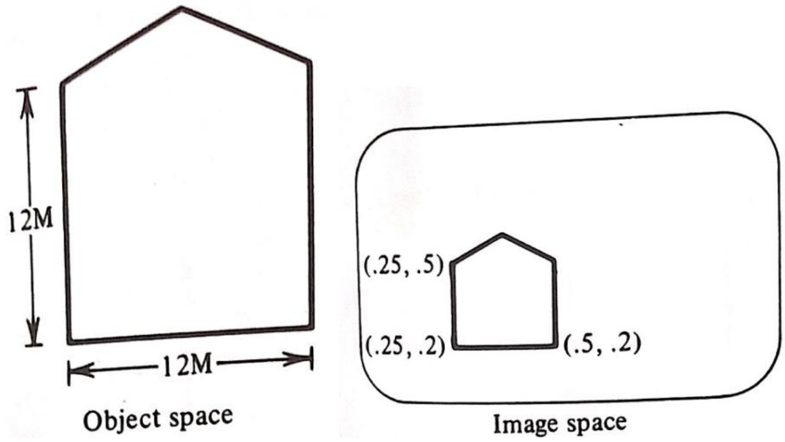


Fig:- In the object space, position is measured in physical units, such as meter. In the image space, position is given in normalized screen coordinates

We must have some way of converting form the object space units of measure to those of the image space. By scaling, we can uniformly reduce the size of the object until its dimension lie between 0 and 1. Very small objects can be enlarged until their overall dimension is almost 1 unit. The physical dimensions of the object are scaled until they are suitable for display (see fig.).

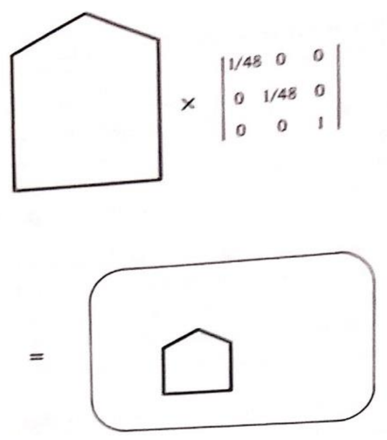


Fig:- A scaling transformation will convert object coordinates units to normalized screen coordinate

The object is too complex to show in its entirely or that we are particularly interested in just a portion of it. We would like to imagine a box about a portion of the object. We would only display what is enclosed in the box. Such box is called a window. (see fig.)

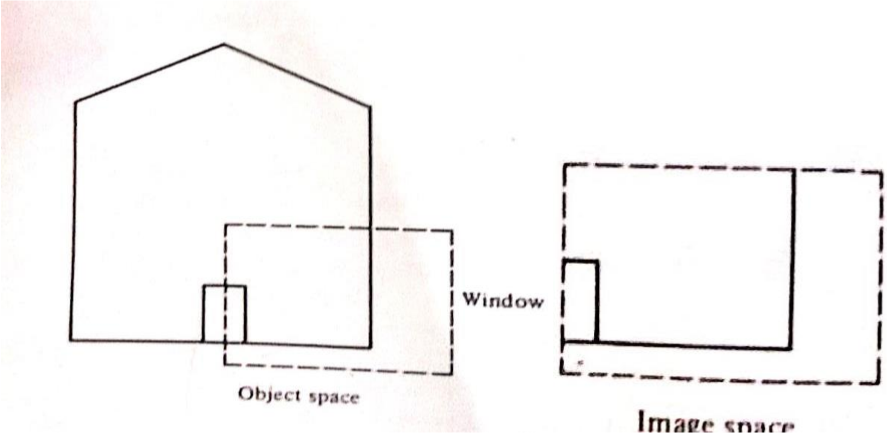


Fig:- A window to view only part of an object

We would like to imagine a box on the screen and have the image confirmed to that box. Such a box in the screen space is called a viewport. (see fig)

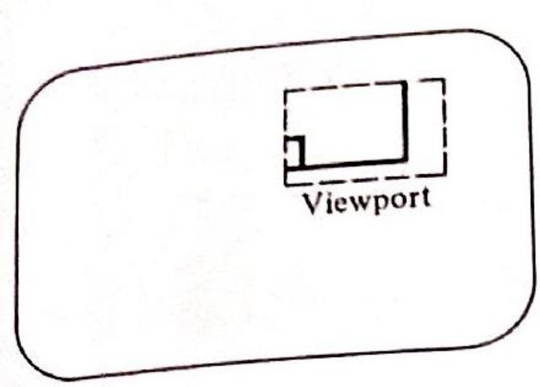


Fig:- A viewport to define the part of the screen to be used

When the window is changed, we see a different part of the object shown at the same position on the display (see fig).

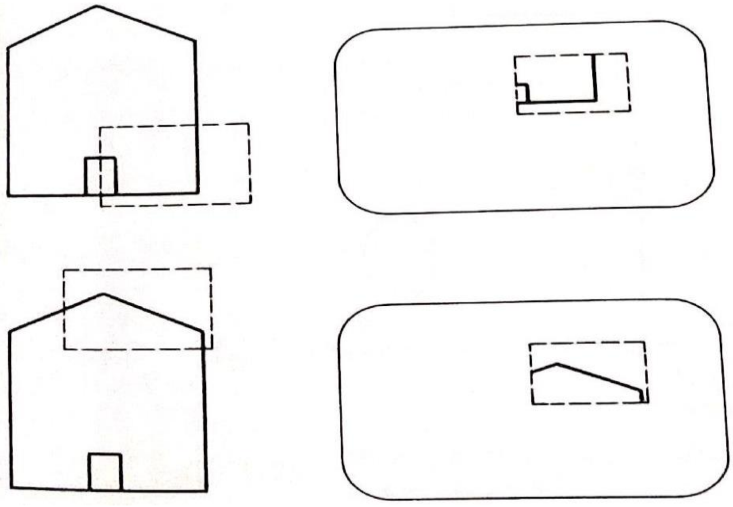


Fig:- Different window, same viewport

If we change the viewport, we see the same part of the object drawn at a different place on the display. (see fig.)

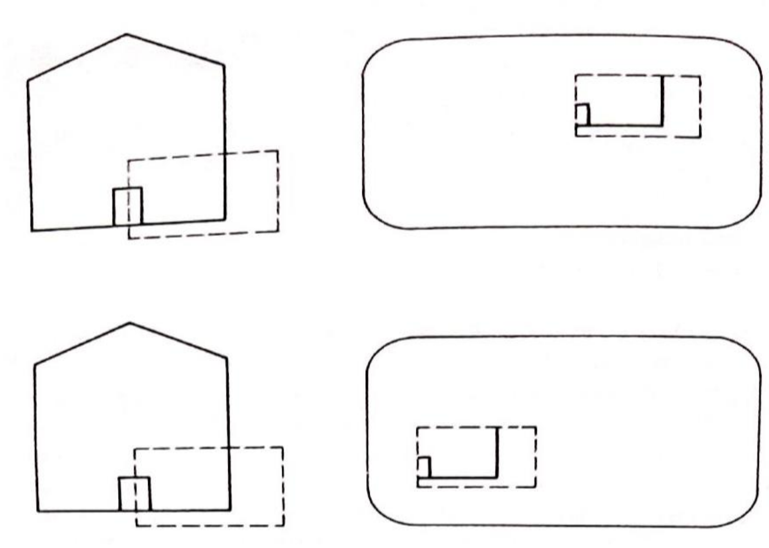


Fig:- same window, different viewport

This can be done with the following three steps. First, the object together with its window is translated until the lower-left corner of the window is at the origin. Second, the object and window we scaled until the window has the dimension of the viewport. In effect, this converts objects and window into image and viewport. The final transformation step is another translation to move the viewport to its correct position on the screen. (see fig.)

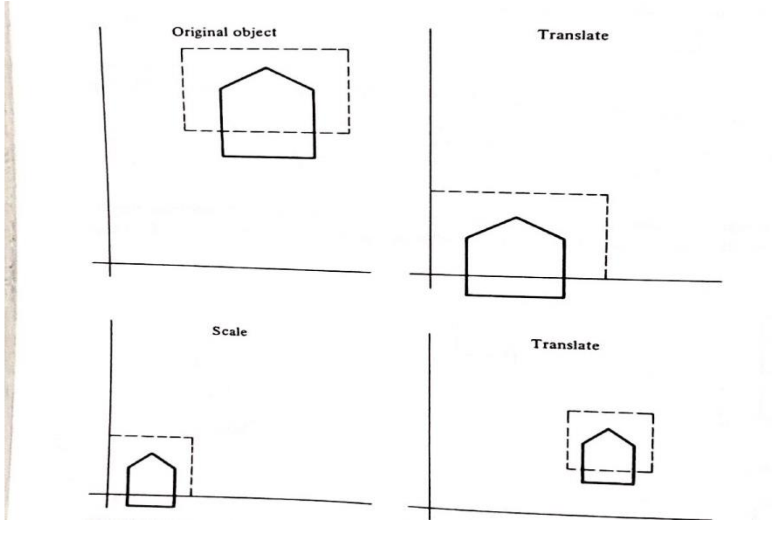


Fig:- Steps in the viewing transformation

We are really trying to do two things. We are changing the window size to become the size of the viewport (scaling) and we are positioning it at the desired location on the screen (translating).

The overall transformation which performs these three steps. We shall call the viewing transformation.

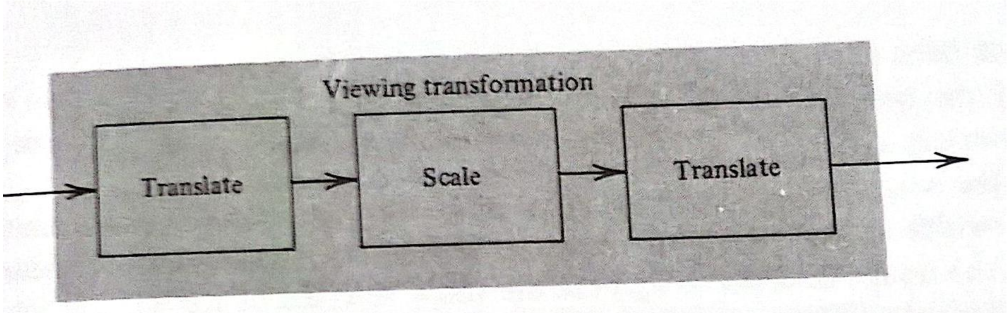


Fig:- A viewport to define the part of the screen to be used.

Algorithms for setting the viewport and window dimensions are given below:

Algorithm SET-VIEWPORT(XL, XH, YL, YH) User routine foe specifying the viewport

```
Arguments    XL, XH the left and right viewport boundaries
             YL, YH the bottom and top viewport boundaries.
Global       VXL-HOLD, VXH-HOLD, VYL-HOLD, VYH-HOLD storage for the viewport
             boundaries
BEGIN
             IF XL >= XH OR YL >= YH THEN RETURN ERROR ‘BAD VIEWPORT’;
             VXL-HOLD ← XL;
             VXH-HOLD ← XH;
             VYL-HOLD ← YL;
             VYH-HOLD ← YH;
             RETURN;
END;
```

Algorithm SET-WINDOW(XL, XH, YL, YH) user routine for specifying the window

```
Arguments    XL, XH the left and right viewport boundaries
             YL, YH the bottom and top viewport boundaries.
```

```
Global          WXL-HOLD, WXH-HOLD, WYL-HOLD, WYH-HOLD storage for the
                window boundaries

BEGIN

                IF XL >= XH OR YL >= YH THEN RETURN ERROR ‘BAD VIEWPORT’;

                WXL-HOLD ← XL;

                WXH-HOLD ← XH;

                WYL-HOLD ← YL;

                WYH-HOLD ← YH;

                RETURN;

END;
```

2. (A) What is clipping? Explain Sutherland-Hodgman algorithm.

Ans:- Clipping :- The primary use of clipping in computer graphics is to remove objects, lines, or line segments that are outside the viewing pane. The viewing transformation is insensitive to the position of points relative to the viewing volume − especially those points behind the viewer − and it is necessary to remove these points before generating the view.

Sutherland-Hodgman algorithm.

A polygon can be clipped by processing its boundary as a whole against each window edge. This is achieved by processing all polygon vertices against each clip rectangle boundary in turn. beginning with the original set of polygon vertices, we could first clip the polygon against the left rectangle boundary to produce a new sequence of vertices. The new set of vertices could then be successively passed to a right boundary clipper, a top boundary clipper and a bottom boundary clipper, as shown in figure (l). At each step a new set of polygon vertices is generated and passed to the next window boundary clipper. This is the fundamental idea used in the Sutherland - Hodgeman algorithm.

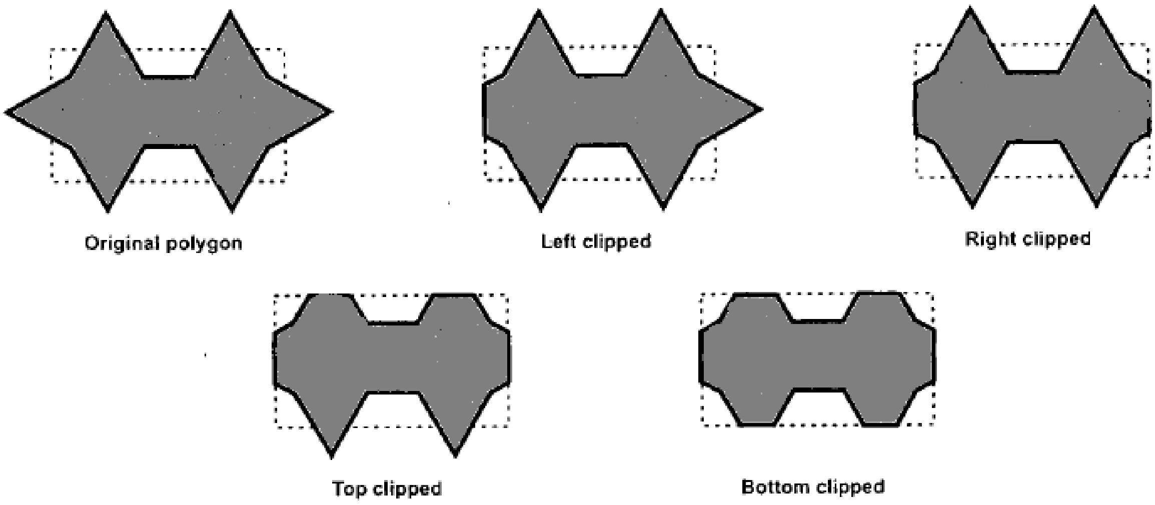


Fig. (l) Clipping a polygon against successive window boundaries

The output of the algorithm is a list of polygon vertices all of which are on the visible side of a clipping plane. Such each edge of the polygon is individually compared with the clipping plane. This is achieved by processing two vertices of each edge of the polygon around the clipping boundary or plane. This results in four possible relationships between the edge and the clipping boundary or Plane. (See Fig. m).

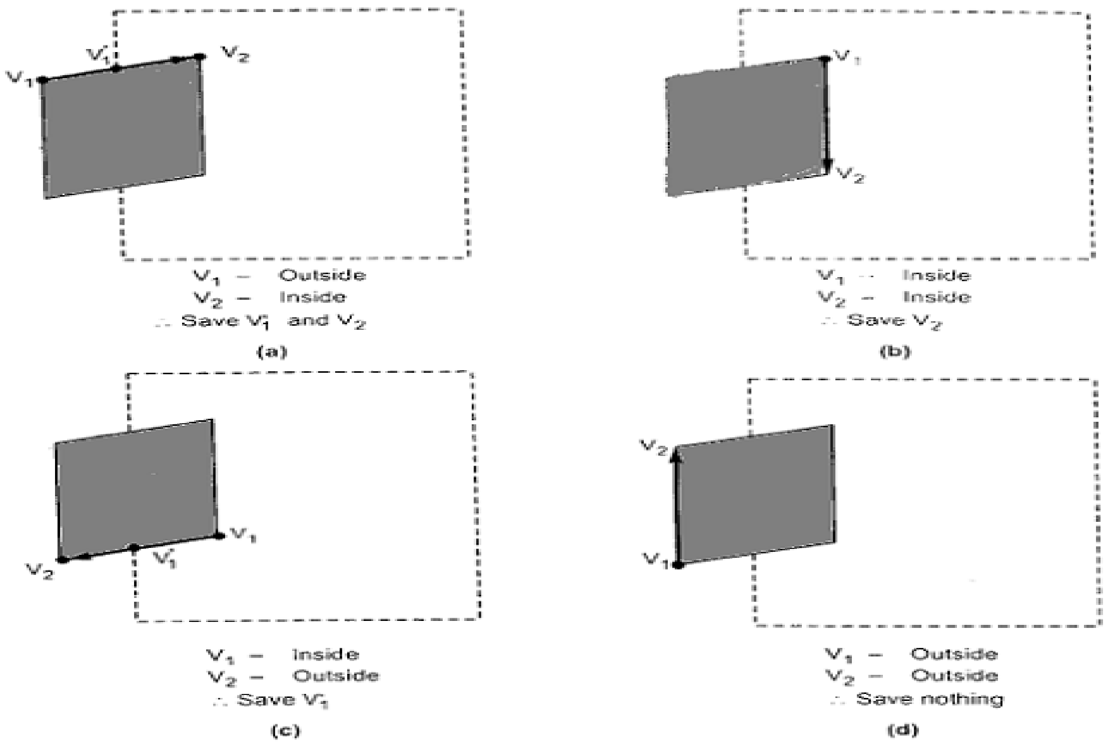


Fig. (m) Processing of edges of the polygon against the left window boundary

1. If the first vertex of the edge is outside the window boundary and the second vertex of the edge is inside then the intersection point of the polygon edge with the window boundary and the second vertex are added to the output vertex list (See Fig. m (a)).
2. If both vertices of the edge are inside the window boundary, only the second vertex is added to the output vertex list. (See Fig. m (b)).
3. If the first vertex of the edge is inside the window boundary and the second vertex of the edge is outside, only the edge intersection with the window boundary is added to the output vertex list. (See Fig. m (c)).
4. If both vertices of the edge are outside the window boundary, nothing is added to the output list. (See Fig. m (d)).

Once all vertices are processed for one clip window boundary, the output list of vertices is clipped against the next window boundary. Going through above four cases we can realize that there are two key processes in this algorithm.

1. Determining the visibility of a point or vertex (lnside - Outside test) and
2. Determining the intersection of the polygon edge and the clipping plane.

One way of determining the visibility of a point or vertex is described here. Consider that two points A and B define the window boundary and point under consideration is V, then these three points define a plane. Two vectors which lie in that plane are AB and AV. If this plane is considered in the xy plane, then the vector cross product AV x AB has only a component given by

$$(x_V - x_A)\ (y_B - y_A) - (y_V - y_A)\ (x_B - x_A)$$

The sign of the z component decides the position of Point V with respect to window boundary.

If z is:

Positive - Point is on the right side of the window boundary.

Zero - Point is on the window boundary.

Negative - Point is on the left side of the window boundary.

Sutherland-Hodgeman Polygon Clipping Algorithm:-

1. Read coordinates of all vertices of the Polygon.
2. Read coordinates of the dipping window
3. Consider the left edge of the window
4. Compare the vertices of each edge of the polygon, individually with the clipping plane.
5. Save the resulting intersections and vertices in the new list of vertices according to four possible relationships between the edge and the clipping boundary.
6. Repeat the steps 4 and 5 for remaining edges or the clipping window. Each time the resultant list of vertices is successively passed to process the next edge of the clipping window.
7. Stop.

Example :- For a polygon and clipping window shown in figure below give the list of vertices after each boundary clipping.

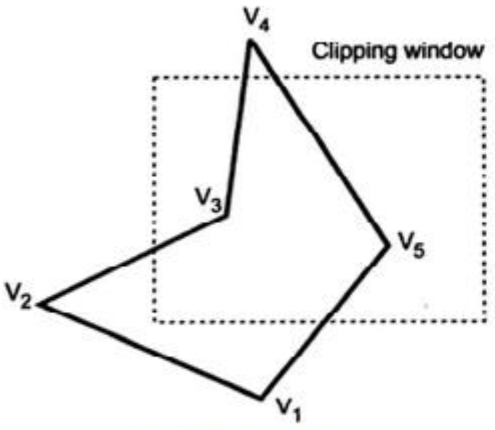


Fig. Before Clipping

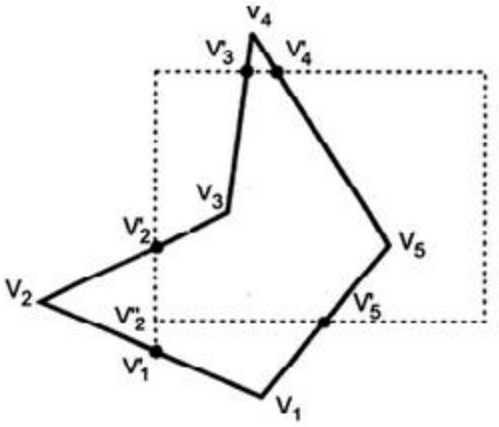
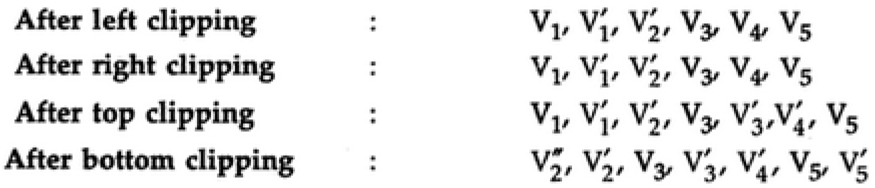


Fig. After Boundary Clipping

Solution:- Original polygon vertices are V1, V2, V3, V4, and V5. After clipping each boundary the new vertices are as shown in figure above.



After left clipping	:	$V_1, V_1', V_2', V_3, V_4, V_5$
After right clipping	:	$V_1, V_1', V_2', V_3, V_4, V_5$
After top clipping	:	$V_1, V_1', V_2', V_3, V_3', V_4', V_5$
After bottom clipping	:	$V_2'', V_2', V_3, V_3', V_4', V_5, V_5'$

(B) Explain Input Device handling algorithm for enable group.

Ans:- **Input Device handling algorithm for enable group**.

Algorithm ENABLE-GROUP(CLASS) Routine to enable an input device class

Arguments: CLASS the code for the class to be enables

Global: BUTTON PICK, KEYBOARD, LOCATOR, VALUATOR device flags

```
BEGIN
    IF CLASS = 1 THEN
    BEGIN
        PERFORM ALL OPERATION NEEDED TO [PERMIT INPUT FROM THE BUTTOM DEVICE
        BUTTON <- TRUE;
    END;
    IF CLASS = 2 THEN
    BEGIN
        PERFORM ALL OPERATION NEEDED TO PERMIT INPUT FROM THE PICKDEVICE
        PICK <- TRUE;
    END;
IF CLASS = 3 THEN
    BEGIN
        PERFORM ALL OPERATION NEEDED TO [PERMIT INPUT FROM THE KEYBOARD DEVICE
        KEYBOARD <- TRUE;
    END;
IF CLASS = 4 THEN
    BEGIN
        PERFORM ALL OPERATION NEEDED TO [PERMIT INPUT FROM THE LOCATOR DEVICE
        LOCATOR <- TRUE;
    END;
IF CLASS = 5 THEN
    BEGIN
        PERFORM ALL OPERATION NEEDED TO PERMIT INPUT FROM THE VALUATOR DEVICE
        VALUATOR <- TRUE;
    END;
```

```
        RETURN
END
```

(C) Explain locator, selector and multiple windowing in detail.

ANS:- Locator:- Locator are devices which gives position information. The computer typically receives from a locator the coordinator for a point. Using locator we can indicate position on the screen.

Let us first consider some locator devices. One example of a locator is a pair of thumbwheels such as is found on the Tektronix 400 graphical terminal. These are two potentiometer mounted on the keyboard, which the user can adjust. One potentiometer is used for x direction, the other for the y direction. The two potentiometer readings together form the coordinates of a point.

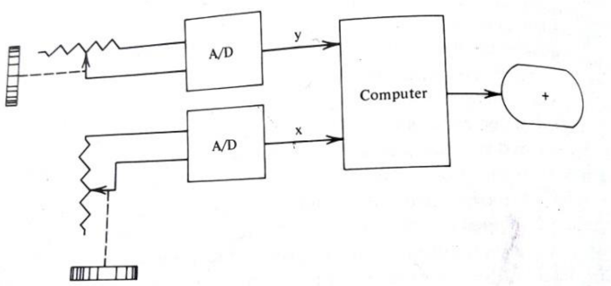


Fig: Thumbwheel entry of position.

Another locator device is a joystick. A joystick has two potentiometers, just like a pair of thumbwheels. They have been attached to a single lever. Moving the lever forward or back changes the setting on one potentiometer. Moving it left or right changes the setting on the other potentiometer. Thus with a joystick both x and y coordinate positions can be simultaneously altered by the motion of a single lever. Joysticks are inexpensive and are quite common on display where only rough positioning is needed.

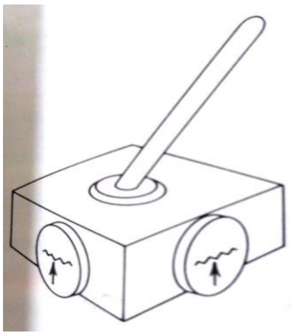


Fig 2: Joystick.

Some locator devices use switches attached to wheels instead of potentiometers. As the wheels are turned, the switches produce pulses which may be counted. This mechanism is often found in mice and track balls. As the mouse is pushed across a surface, the wheels are turned, providing distance and direction information. A mouse may also come with one or more buttons which may ne sensed. A track ball is essentially a mouse turned upside down. The ball which turns the wheels is large and is moved directly by the operator.

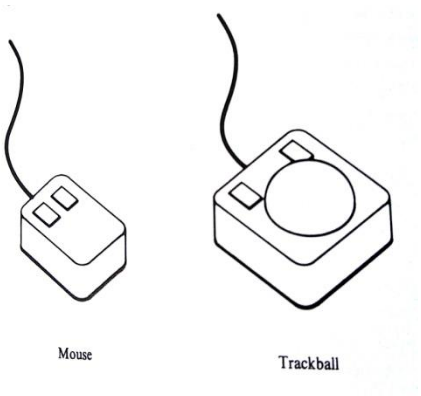


Fig 3: Mouse and Track Ball

If we had a paper drawing or a blueprint which we wished to enter onto the machine, we would find that the joystick was not very useful. Although the joystick could indicate a position on the screen. For applications such as tracing we need a device called digitizer or a tablet. A tablet is composed of a flat surface and a penlike stylus or windowlike tablet cursor on the surface. If tablet entries are to be coordinated with items already on the screen, then some form of feedback, such as a screen cursor, is useful.

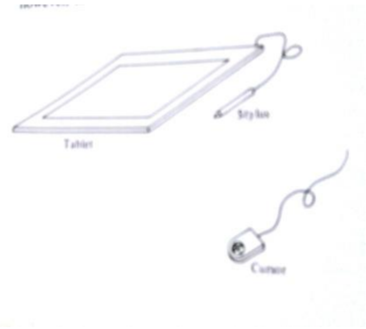


Fig 4: Tablet

Selector: Selector devices are used to select a particular graphical object. A selector may pick a particular item but provide no information about where that item is located on the screen.

An example of selector device is a light pen. A light pen is composed of a photocell mounted in a penlike case. This pen may be pointed at the screen on a refresh display. The pen will send a pulse whenever the phosphor below it is illuminated. While the image on a refresh display may appear to be stable, it is in fact blinking on a off faster than the eye can detect. When the light pen senses the phosphor beneath it being illuminated, it can interrupt the display processor interpreting of the display file.

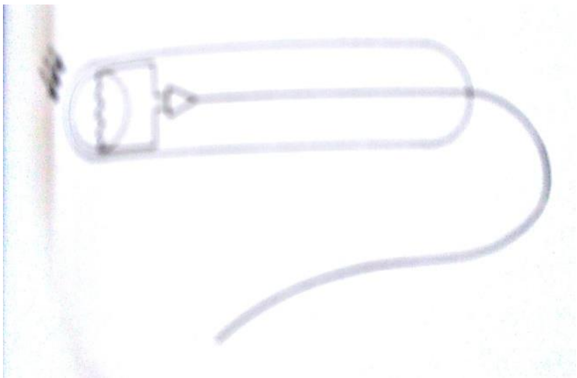


Fig 5: Light Pen

(D) Explain how to add a new picture element to the image. Write a routine to interactively select a point.

Ans:- Point Plotting method gives the user the capability of selecting a particular point on the screen. This is usually done by a combination of a locator and a button. The locator is used to tell which point the user selects. The button indicates when the locator is correctly positioned. The algorithm to perform point plotting must await the button event; as soon as it occurs, the locator may be read. The selection of point can be used in many different ways.

Algorithm AWAIT-BUTTON-GET-LOCATOR (WAIT, BUTTON-NUM,X,Y) User routine to inractively select a point.

```
Arguments     WAIT the time to wait for a button event.

              BUTTON-NUM for return of the user's button selection.

              X,Y the point the user selects

BEGIN
```

```
        AWAIT-BUTTON(WAIT, BUTTON-NUM(;

        READ-LOCATOR(X,Y);

        RETURN;

END;
```

UNIT - III

Q3. (a) Write the three dimensional LINE and MOVE algorithm

Ans:- The three dimensional LINE and MOVE algorithm are as follows:

1) **Algorithm MOVE-ABS-3(X,Y,Z)** the 3D absolute move

```
Arguments    X,Y,Z world coordinates of the point to move the pen to
Global       DF-PEN-X, DE-PEN-Y, DF-PEN-Z current pen position
Begin
        DF-PEN-X ← X;
        DE-PEN-Y ← Y;
        DF-PEN-Z ← Z;
        DISPLAY-FILE-ENTER(1);
        RETURN;
END
```

2) **Algorithm MOVE-REL-3(DX,DY,DZ)** the 3D relative move

```
Arguments    DX, DY, DZ change to be made to the pen position
Global       DF-PEN-X, DE-PEN-Y, DF-PEN-Z current pen position
Begin
        DF-PEN-X ← DF-PEN-X + DX;
        DE-PEN-Y ← DF-PEN-X + DY;
        DF-PEN-Z ← DF-PEN-X + DZ;
        DISPLAY-FILE-ENTER(1);
        RETURN;
END
```

3) **Algorithm LINE-ABS-3(X,Y,Z)** the 3D absolute line drawing routine

```
Arguments    X,Y,Z world coordinates of the point to draw the line to
Global       DF-PEN-X, DE-PEN-Y, DF-PEN-Z current pen position
Begin
        DF-PEN-X ← X;
        DE-PEN-Y ← Y;
        DF-PEN-Z ← Z;
        DISPLAY-FILE-ENTER(2);
        RETURN;
END
```

4) **Algorithm LINE-REL-3(DX,DY,DZ)** the 3D relative line drawing routine

```
Arguments   DX, DY, DZ displacement over which a line is to be drawn
Global      DF-PEN-X, DE-PEN-Y, DF-PEN-Z current pen position
Begin
        DF-PEN-X ← DF-PEN-X + DX;
        DE-PEN-Y ← DF-PEN-X + DY;
        DF-PEN-Z ← DF-PEN-X + DZ;
        DISPLAY-FILE-ENTER(2);
        RETURN;
END
```

(b) Explain:-

i) Isometric Projection

ii) Dimetric Projection

iii) Trimetric Projection

iv) Cavalier Projection

Ans:- Isometric Projection:- There is a particular direction of projection for which all edges will appear shortened form there three dimensional length by the same factor. This special direction is called an isometric projection.

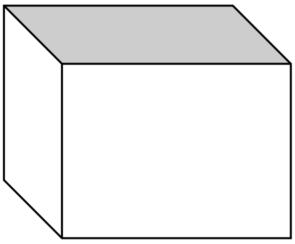


Fig:- Isometric projection shortens all equally

An isometric projection of a cube will show a corner of the cube in the middle of the image surrounded by three identical faces. From the symmetry of the situation, we can see that the commands needed for an isometric projection of an object with sides parallel to the axex are:

SET-PARALLEL (1,1,1);

SET-VIEW-PLANE-NORMAL(-1,-1,-1);

This projection will focus on the upper right-front corner of the object.

ii) Dimetric Projection:- If a viewing transformation is chosen such that edges parallel to only two of the axes are equally shortened, then projection is called diametric projection.

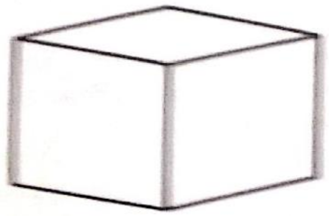


Fig:- Dimetric projection shortens two axes equally.

iii) Trimetric Projection:- A trimetric projection is one in which none of the three edge directions is equally shortened. There are no symmetries in the angles between the direction of projection and the directions of the object edges.

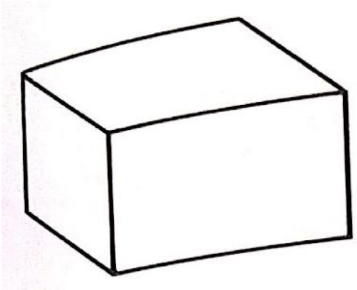


Fig:- Trimetric projection shortens all axes differently.

iv) Cavalier Projection:- If the direction of parallel projection is not parallel to the view plane normal, then we have what is called an oblique projection. We shall describe two special types of oblique projections called **cavalier projection** and **cabinet** projection.

In a cavalier projection let us assume that we are viewing an object with edges parallel to the coordinate axes. The view plane will be parallel to the front face. For a cavalier projection, the direction of projection is slanted so that points with positive z coordinates will be projected down and to the left on the view plane. Points with negative z coordinates will be projected up and to the right. The angle of the projected z axis can be whatever we desire, but the distance the point shifts in the projected z direction must equal the actual three-dimensional z distance from the view plane. The projection command which will create a cavalier projection at angle A is

SET-PARALLEL (COS(A), SIN(S), 1);

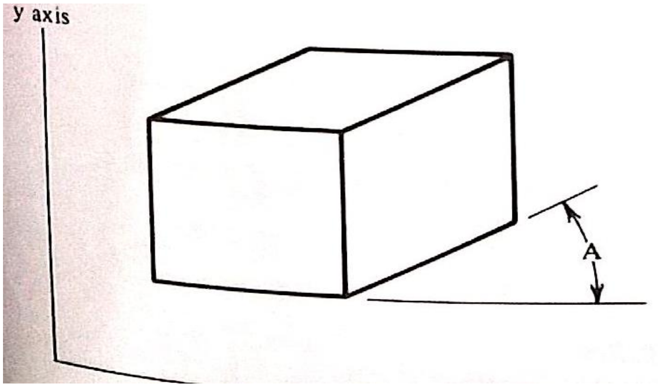


FIG:- A cavalier projection.

(c) Explain perspective projection in detail, also write an algorithm for perspective projection of a point.

Ans:- In a perspective projection, the away an object is from the viewer, the smaller its appears. This provides the viewer with a depth cue, an indication of which portions of the image correspond to parts of the object which are close or far away. In a perspective projection, the lines of projection are not parallel. They all converge at a single point called center of projection. It is the intersection of these converging lines with the plane of the screen that determines the projected image. The projection gives the image which would be seen if the viewer's eye were located at the center of projection. The lines of projection would correspond to the paths of the light rays coming from the object to the eye.

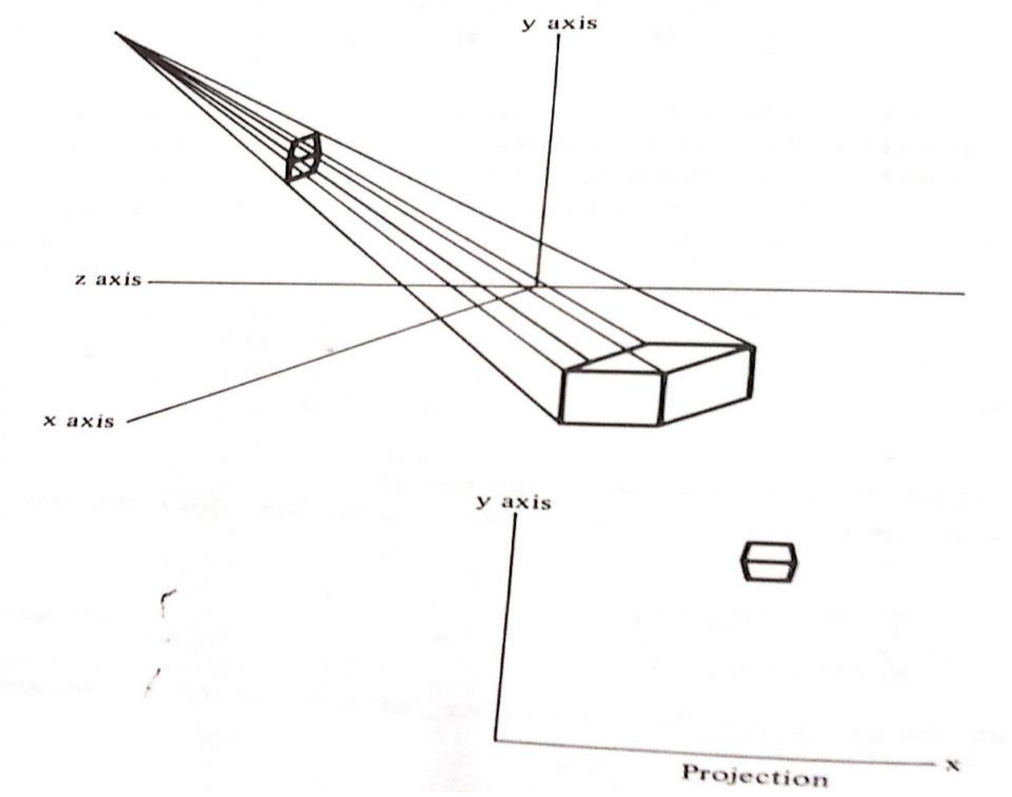


Fig:- A perspective projection

If the center of projection is at (x_c, y_c, z_c) and the point on the object is (x_1, y_1, z_1), then the projection ray will be the line containing these points and will be given by

$x = x_c + (x_1 - x_c)u$

$y = y_c + (y_1 - y_c)u$

$z = z_c + (z_1 - z_c)u$

the projected point (x2, y2) will be the point where this line intersects the xy plane. The third equation tells us that u, for this intersection point (z=0), is

$$u = -\frac{z_c}{z_1 - z_c}$$

Substituting into the first two equations gives

$$x_2 = x_c - z_c \frac{x_1 - x_c}{z_1 - z_c}$$

$$y_2 = y_c - z_c \frac{y_1 - y_c}{z_1 - z_c}$$

With the little algebra, we can rewrite this as

$$x_2 = \frac{x_c z_1 - x_1 z_c}{z_1 - z_c}$$

$$y_2 = \frac{y_c z_1 - y_1 z_c}{z_1 - z_c}$$

This projection can be put into the form of a transformation matrix if we take full advantage of the properties of homogeneous coordinates. The form of the matrix is

$$P = \begin{vmatrix} -z_c & 0 & 0 & 0 \\ 0 & -z_c & 0 & 0 \\ x_c & y_c & 0 & 1 \\ 0 & 0 & 0 & -z_c \end{vmatrix}$$

To show that this transformation works, consider the point (x1, y1, z1). In homogeneous coordinates we would have

$$[x_1 w_1 \quad y_1 w_1 \quad z_1 w_1 \quad w_1]$$

Multiplying by the transformation matrix gives

$$[x_2w_2 \;\; y_2w_2 \;\; z_2w_2 \;\; w_2] = [x_1w_1 \;\; y_1w_1 \;\; z_1w_1 \;\; w_1] \begin{vmatrix} -z_c & 0 & 0 & 0 \\ 0 & -z_c & 0 & 0 \\ x_c & y_c & 0 & 1 \\ 0 & 0 & 0 & -z_c \end{vmatrix}$$

$$= [-x_1w_1z_c + z_1w_1x_c \quad -y_1w_1z_c + z_1w_1y_c \quad 0 \quad z_1w_1 - z_cw_1] \quad (8.47)$$

So

$$w_2 = z_1w_1 - z_cw_1$$

And

$$Z_2w_2 = 0$$

Which gives

$$x_2 = \frac{x_c\, z_1 - x_1\, z_c}{z_1 - z_c}$$

And

$$Y_2w_2 = -\, y_1w_1z_c + z_1w_1y_c$$

Which gives

$$y_2 = \frac{y_c\, z_1 - y_1\, z_c}{z_1 - z_c}$$

The resulting point (x2, y2) is then indeed the correctly projected point.

Algorithm for perspective projection of a point

Algorithm PERSPECTIVE-TRANSFORM(X, Y, Z) For perspective transformation of a point.

Arguments X, Y, Z the view plane coordinates of the point

Global XC, YC, ZC the conter of projection

```
Local           D the denominator in the calculations

Constant        ROUNDOFF some small number greater that any round-off error

                VERY-LARGE a very large number approximating infinity

BEGIN

     D ← ZC – Z

     IF |D| < ROUNDOFF THEN

     BEGIN

     X ← ( X – XC) * VERY-LARGE

     Y ← ( Y – YC) * VERY-LARGE

     Z X ← VERY-LARGE

     END

     ELSE

     BEGIN

     X ← (X * ZC – XC * Z) / D;

     Y ← (Y * ZC – YC * Z) / D;

     Z ← Z / D;

END

RETURN;

END;
```

3. (A) What is projection? Explain special projection with suitable example.

Ans:- we have talked about creating and transforming three-dimensional objects, but since our viewing surface is only two-dimensional, we must some way of projecting our three-dimensional object onto the two dimensional screen.

Special Projection

Isometric Projection:- There is a particular direction of projection for which all edges will appear shortened form there three dimensional length by the same factor. This special direction is called an isometric projection.

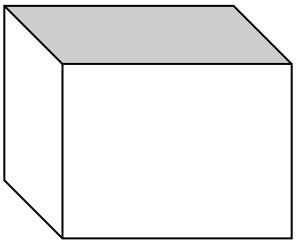


Fig:- Isometric projection shortens all equally

An isometric projection of a cube will show a corner of the cube in the middle of the image surrounded by three identical faces. From the symmetry of the situation, we can see that the commands needed for an isometric projection of an object with sides parallel to the axex are:

SET-PARALLEL (1,1,1);

SET-VIEW-PLANE-NORMAL(-1,-1,-1);

This projection will focus on the upper right-front corner of the object.

ii) Dimetric Projection:- If a viewing transformation is chosen such that edges parallel to only two of the axes are equally shortened, then projection is called diametric projection.

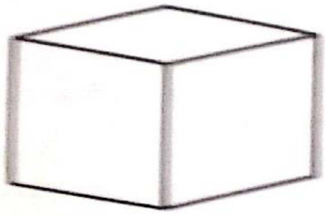


Fig:- Dimetric projection shortens two axes equally.

iii) Trimetric Projection:- A trimetric projection is one in which none of the three edge directions is equally shortened. There are no symmetries in the angles between the direction of projection and the directions of the object edges.

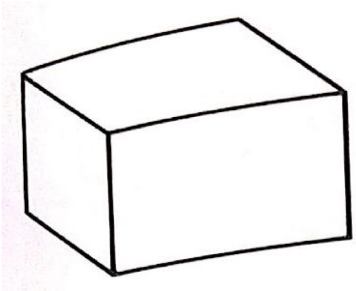


Fig:- Trimetric projection shortens all axes differently.

iv) Cavalier Projection:- If the direction of parallel projection is not parallel to the view plane normal, then we have what is called an oblique projection. We shall describe two special types of oblique projections called **cavalier projection** and **cabinet** projection.

In a cavalier projection let us assume that we are viewing an object with edges parallel to the coordinate axes. The view plane will be parallel to the front face. For a cavalier projection, the direction of projection is slanted so that points with positive z coordinates will be projected down and to the left on the view plane. Points with negative z coordinates will be projected up and to the right. The angle of the projected z axis can be whatever we desire, but the distance the point shifts in the projected z direction must equal the actual three-dimensional z distance from the view plane. The projection command which will create a cavalier projection at angle A is

SET-PARALLEL (COS(A), SIN(S), 1);

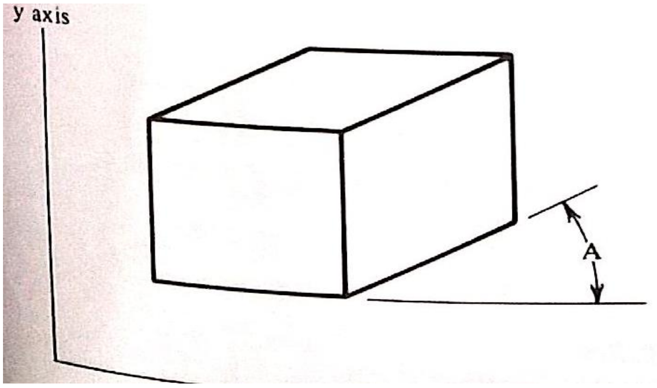


FIG:- A cavalier projection.

(B) Write an algorithm for BACK-FACE-CHECK (Polysize).

Ans:- The algorithm BACK-FACE-CHECK function to actually decide whether the polygon is a back face and enters it into the display file. It computes the z component of the vector normal to the polygon according to equation

Algorithm BACK-FACE-CHECK (POLYSIZE) Filters out polygon drawn clockwise

Arguments	POLYSIZE the number of sides on the polygon
Global	XT, YT, ZT T- buffer array storage of the vertex points
Local	C z component of a vector for the normal to the plane of the polygon
	I, J for stepping through the vertices

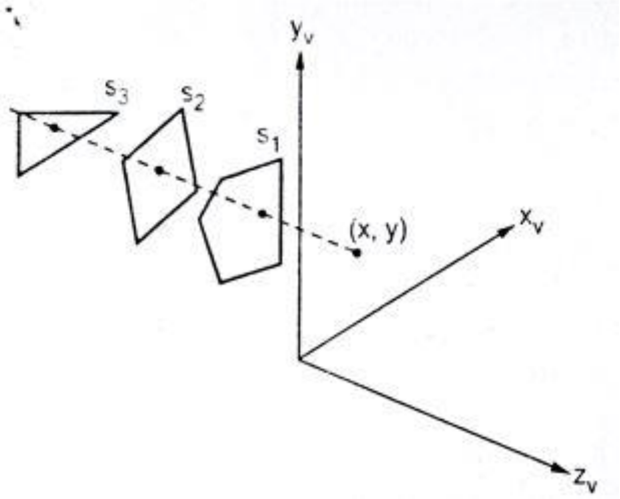
```
BEGIN
    C <- 0
    FOR I =1 TO POLYSOZE DO
        BEGIN
            IF I = POLYSOZE THEN J <- 1
            ELSE J <- I +1
            C <- C + ((XT[I] – XT[J]) * (YT[I] + YT[J]))
        END
    IF C<= 0 THEN RETURN;
    VIEWING-TRANSFORM (POLYSOZE, XT[POLYSIZE], YT[POLYSIZE]);
    FOR I = 0 TO POLYSIZE DO VIEWING-TRANSFORM( IT[I], XT[I], YT[I]);
    RETURN
```

END

(C) Explain BACK-FACE-REMOVAL in detail.

Ans:- One of the simplest and commonly used image space approach to eliminate hidden surfaces is the **Z-buffer or Depth Buffer** algorithm. It is developed by Catmull. This algorithm compares surface depths at each pixel position on the on the projection plane. The surface depth is measured from the view plane along the z axis of a viewing system. When object description is converted to projection coordinates (x,y,z) each pixel position on the view plane is specified by x and y coordinate and z value gives the depth information. Thus object depths can be compared by comparing the z-values.

The Z-buffer algorithm is usually implemented in the normalized coordinates, so that z values range from at the back clipping plane to at the front clipping plane. The implementation requires another buffer memory called Z-buffer along with the frame buffer memory required for raster display devices. A Z-buffer is used to store depth values for each (x,y) position as surfaces are proceeds and the frame buffer stores the memory values for each position.



If the calculated depth values is greater than the value stored in the Z-buffer, the new depth value is stored, and the surfaced intensity at that position is determined and placed in the same xy location in the frame buffer

For Example, among three surfaces, surface S1S1 has the smallest depth at view position (x,y) and hence highest z value. So it is visible at that position.

(D) Explain rotation about an arbitrary axis in detail.

Ans:- Rotation About An Arbitrary Axis

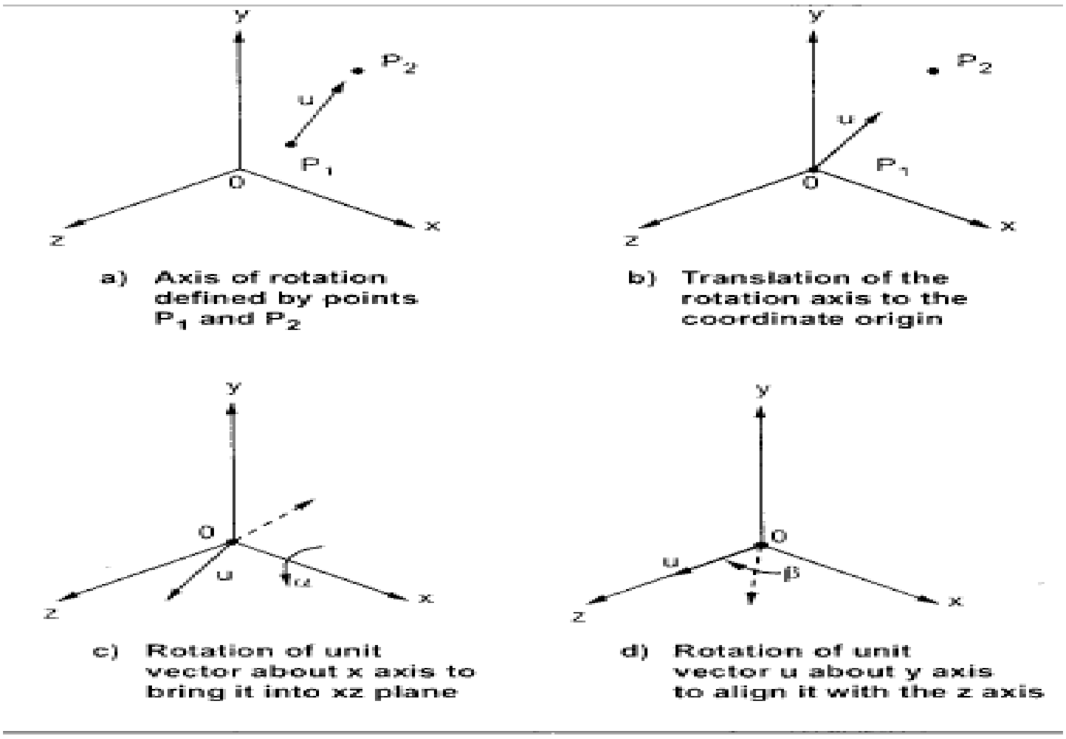
A rotation matrix for any axis that does not coincide with a coordinate axis can be set up as a composite transformation involving combination of translation and the coordinate-axes rotation.

In a special case where an object is to be rotated about an axis that is parallel to one of the coordinate axes we can obtain the resultant coordinates with the following transformation sequence.

1. Translate the object so that the rotation axis coincides with the parallel coordinate axis.
2. Perform the specified rotation about that axis.
3. Translate the object so that the rotation axis is moved back to its original position.

When an object is to be rotated about an axis that is not parallel to one of the coordinate axes, we have to perform some additional transformations. The sequence of these transformations is given below:

1. Translate the object so that rotation axis specified by unit vector u passes through the coordinate origin. (See fig.)
2. Rotate the object so that the axis of rotation coincides with one of the coordinate axes. To coincide the axis of rotation of z axis we have to first perform rotation of unit vector u about x axis to bring it into xz plane and then perform rotation about y axis to coincide it with z axis (See fig.)
3. Perform the desired rotation 0 about the z axis.
4. Apply the inverse rotation about y axis and then about x axis t bring the rotation axis back to its original orientation.
5. Apply the inverse translation to move the rotation axis back to its original position.



a) Axis of rotation defined by points P_1 and P_2

b) Translation of the rotation axis to the coordinate origin

c) Rotation of unit vector about x axis to bring it into xz plane

d) Rotation of unit vector u about y axis to align it with the z axis

As shown in the above fig. the rotation axis is defined with two coordinate points P1 and point P2 and unit vector u is defined along the rotation of axis as

$$u = \frac{V}{|V|} = (a, b, c)$$

Where V is the axis vector defined by two points P1 and P2 as

$$\begin{aligned} V &= P_2 - P_1 \\ &= (x_2 - x_1, y_2 - y_1, z_2 - z_1) \end{aligned}$$

The components a, b, c of unit vector u are the direction cosines for the rotation axis and they can be defined as

$$a = \frac{x_2 - x_1}{|V|}, \quad b = \frac{y_2 - y_1}{|V|}, \quad c = \frac{z_2 - z_1}{|V|}$$

As mention earlier, the first step in the transformation sequence is to translate the object to pass the rotation axis through the coordinate origin. This can be accomplished by moving point P1 to the origin. The translate is as given below

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_1 & -y_1 & -z_1 & 1 \end{bmatrix}$$

Now we have to perform the rotation of unit vector u about x axis. The rotation of u around the x axis into the xz plane is accomplished by rotating u’ (0, b, c) through angle α into the z axis and the cosine of the rotation angle α can be determined from the dot product of u’ and the unit vector u_z (0,0,1) along the z axis.

$$\cos\alpha = \frac{u' \cdot u_z}{|u'||u_z|} = \frac{c}{d} \quad \text{where } u'(0, b, c) = bJ + cK \text{ and}$$

$$u_z(0, 0, 1) = K$$

$$= \frac{c}{|u'||u_z|}$$

$$= \frac{c}{|u'|} \qquad \text{Since } |u_z| = 1$$

$$= \frac{c}{d}$$

Where d is the magnitude of u':

$$d = \sqrt{b^2 + c^2}$$

Similarly, we can determine the sine of α from the cross product of u' and u_z

$$u' \times u_z = u_x \, |u'| \; |u_z| \sin\alpha$$

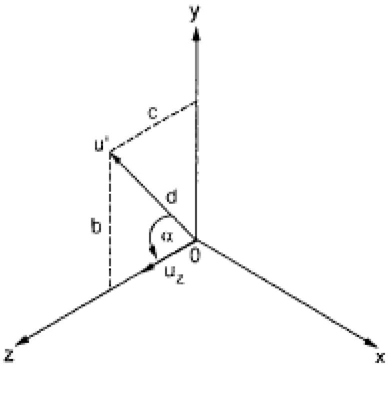
And the Cartesian form for the cross product given us

$$u' \times u_z = u_x \cdot b$$

Equating the right sides of equations we get

$$\begin{aligned} u_x \, |u'| \; |u_z| \sin\alpha &= u_x \,.\, b \\ |u'| \; |u_z| \sin\alpha &= b \\ \therefore \qquad \sin\alpha &= \frac{b}{|u'||u_z|} \\ &= \frac{b}{d} \qquad \text{since } |u_z| = 1 \text{ and } |u'| = d \end{aligned}$$

This can also be verified graphically as shown in fig.



By substituting values of cos α and sin α the rotation matrix R, can be given as

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & b/d & 0 \\ 0 & -b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Next we have to perform the rotation of unit vector about y axis. This can be achieved by rotating u’’(a,0,d) through angle β onto the z axis. Using similar equations we can determine cosβ and sin β as follows.

We have angle of rotation = - β

$$\therefore \qquad \cos(-\beta) = \cos\beta = \frac{u'' \cdot u_z}{|u''|\,|u_z|} \text{ where } u'' = aI + dK \text{ and}$$

$$u_z = K$$

$$= \frac{d}{|u''|\,|u_z|}$$

$$= \frac{d}{|u''|}$$

$$= \frac{d}{\sqrt{a^2 + d^2}}$$

Consider cross product of u’’ and u_z

$$\begin{aligned} u'' \times u_z &= u_y\, |u''|\ |u_z|\ \sin(-\beta) \\ &= -u_y\, |u''|\ |u_z|\ \sin\beta \end{aligned}$$

Cartesian form of cross product gives us

$$u'' \times u_z = u_y\,(+a)$$

Equating above equations,

$$-|u''|\ |u_z|\ \sin\beta = a$$

$$\therefore \qquad \sin\beta = \frac{-a}{|u''||u_z|}$$

$$= \frac{-a}{|u''|} \qquad \because\ |u_z| = 1$$

$$= \frac{-a}{\sqrt{a^2 + d^2}}$$

but we have, $$d = \sqrt{b^2 + c^2}$$

$$\therefore \qquad \cos\beta = \frac{d}{\sqrt{a^2 + d^2}}$$

$$= \frac{\sqrt{b^2 + c^2}}{\sqrt{a^2 + b^2 + c^2}}$$

and $$\sin\beta = \frac{-a}{\sqrt{a^2 + d^2}}$$

$$= \frac{-a}{\sqrt{a^2 + b^2 + c^2}}$$

By substituting values of cos β and sin β in the rotation matrix Ry can be given as

$$R_y = \begin{bmatrix} \frac{\sqrt{b^2 + c^2}}{\sqrt{a^2 + b^2 + c^2}} & 0 & \frac{+a}{\sqrt{a^2 + b^2 + c^2}} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{-a}{\sqrt{a^2 + b^2 + c^2}} & 0 & \frac{\sqrt{b^2 + c^2}}{\sqrt{a^2 + b^2 + c^2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Let $$\lambda = \sqrt{b^2 + c^2} \text{ and } |V| = \sqrt{a^2 + b^2 + c^2}$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{\lambda} & \frac{+b}{\lambda} & 0 \\ 0 & \frac{-b}{\lambda} & \frac{c}{\lambda} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R_y = \begin{bmatrix} \frac{\lambda}{|V|} & 0 & \frac{+a}{|V|} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{-a}{|V|} & 0 & \frac{\lambda}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$\therefore$ Resultant rotation matrix $R_{xy} = R_x \cdot R_y$

$$\therefore \qquad R_{xy} = \begin{bmatrix} \frac{\lambda}{|V|} & 0 & \frac{a}{|V|} & 0 \\ \frac{-ab}{|V|\lambda} & \frac{c}{\lambda} & \frac{b}{|V|} & 0 \\ \frac{-ac}{|V|\lambda} & \frac{-b}{\lambda} & \frac{c}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We have,

$$t_{ij}^{-1} = \frac{(-1)^{i+j} \det M_{ji}}{\det T}$$

Using above equation we get inverse of R_{xy} as

$$R_{xy}^{-1} = \begin{bmatrix} \frac{\lambda}{|V|} & \frac{-ab}{|V|\lambda} & \frac{-ac}{|V|\lambda} & 0 \\ 0 & \frac{c}{\lambda} & \frac{-b}{\lambda} & 0 \\ \frac{a}{|V|} & \frac{b}{|V|} & \frac{c}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inverse of translation matrix can be given as

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_1 & y_1 & z_1 & 1 \end{bmatrix}$$

With transformation matrices T and R_{xy} we can align the rotation axis with the positive z axis. Now the specified rotation with angle θ can be achieved by rotation transformation as given below

$$R_z = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To complete the required rotation about the given axis, we have to transform the rotation axis back to its original position. This can be achieved by applying the inverse transformations T^{-1} and R_{xy}^{-1} . The overall transformation matrix for rotation about an arbitrary axis then can be expressed as the concatenation of five individual transformations.

$$R(\theta) = T \cdot R_{xy} \cdot R_z \cdot R_{xy}^{-1} \cdot T^{-1}$$

i.e.

$$R(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_1 & -y_1 & -z_1 & 1 \end{bmatrix} \begin{bmatrix} \frac{\lambda}{|V|} & 0 & \frac{a}{|V|} & 0 \\ \frac{-ab}{\lambda|V|} & \frac{c}{\lambda} & \frac{b}{|V|} & 0 \\ \frac{-ac}{\lambda|V|} & \frac{-b}{\lambda} & \frac{c}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \frac{\lambda}{|V|} & \frac{-ab}{\lambda|V|} & \frac{-ac}{\lambda|V|} & 0 \\ 0 & \frac{c}{\lambda} & \frac{-b}{\lambda} & 0 \\ \frac{a}{|V|} & \frac{b}{|V|} & \frac{c}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_1 & y_1 & z_1 & 1 \end{bmatrix}$$

Q3. (a) What do you mean by projection? Explain parallel projection in detail with example.

Ans:- we have talked about creating and transforming three-dimensional objects, but since our viewing surface is only two-dimensional, we must some way of projecting our three-dimensional object onto the two dimensional screen.

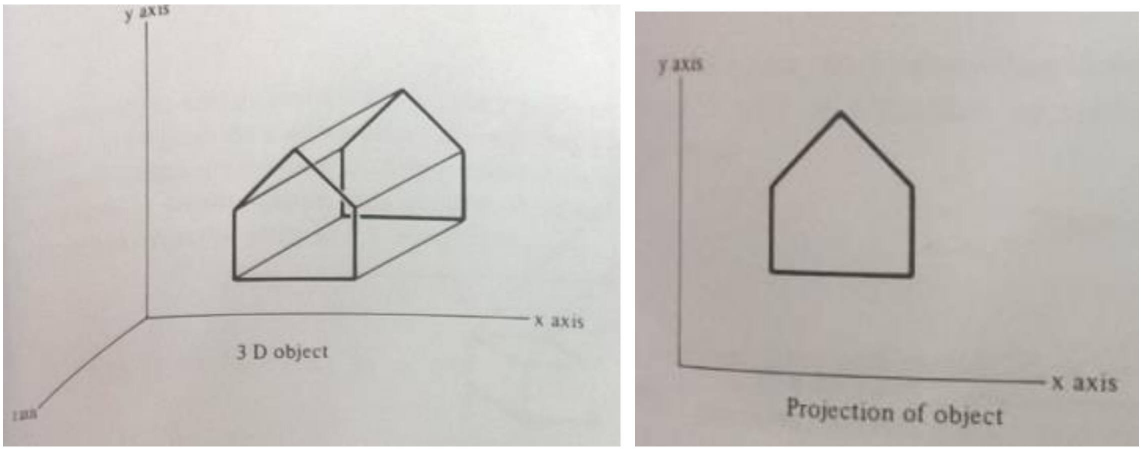


Fig:- A three – dimensional object and its projection.

Parallel Projection : The simplest way of doing this is just to discard the z coordinate. This is a special case of a method known as parallel projection. A parallel projection is formed by extending parallel lines from each vertex on the object until they intersect the plane of the screen. The point of intersection is the projection of the vertex. We connect the projected vertices by line segments which correspond to connections on the original object.

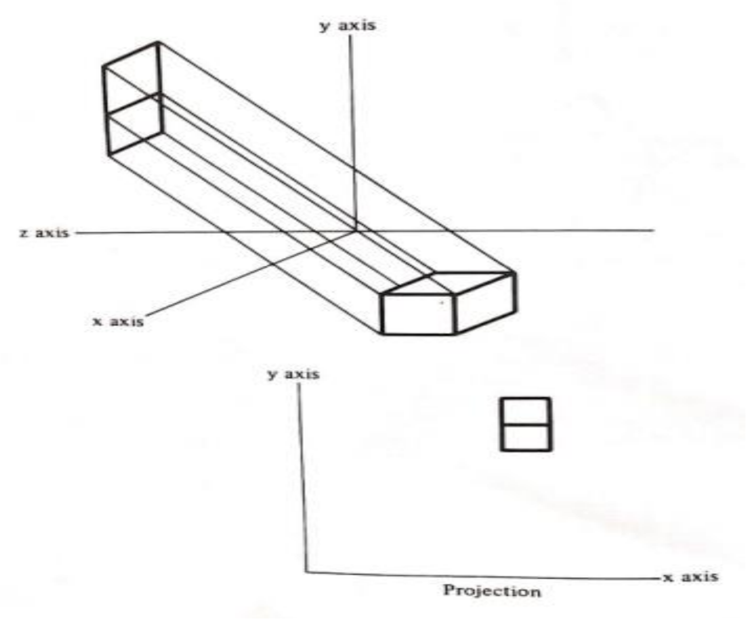


Fig:- A parallel projection

In a general parallel projection, we may select any direction for the lines of projection. Suppose that the direction of projection is given by the vector [xp yp zp] and that the image is to be projected onto the xy plane. If we have a point on the object at (x1,y1,z1), we wish to determine where the projected point (x2,y2) will lie. Let us begin by writing the equations for a line passing through the point (x,y,z) and in the direction of projection. This is easy to do using the parametric form.

$X = x_1 + x_p u$

$y = y_1 + y_p u$

$z = z_1 + z_p u$

If z is 0, the thirs equation tell us that the parameter u is

$u = - z_1 / z_p$

substituting this into the first two equations gives

$x_2 = x_1 - z_1 (x_p/z_p)$

$y_2 = y_1 - z_1 (y_p/z_p)$

This projection formula is in fact a transformation which may be written in matrix form

$$[x_2 \quad y_2] = [x_1 \quad y_1 \quad z_1] \begin{vmatrix} 1 & 0 \\ 0 & 1 \\ -\frac{xp}{zp} & -\frac{yp}{zp} \end{vmatrix}$$

Or in full homogeneous coordinates

$$[x2 \quad y2 \quad z2 \quad 1] = [x_1 \quad y_1 \quad z_1] \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ -\frac{xp}{zp} & -\frac{yp}{zp} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

This transformation always gives 0 for z_2, the position fo the view plane, but it is often useful to maintain the z information. A transformation that includes determining a z-coordinate value z_2 is as follows:

$$[x2 \quad y2 \quad z2 \quad 1] = [x_1 \quad y_1 \quad z_1] \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ -\frac{xp}{zp} & -\frac{yp}{zp} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

We will just ignore the z value when drawing the projected image.

(b) Write and explain painters algorithm.

Ans:- The painters algorithm gets its name from the manner in which an oil painting is created. The artist begin with the background scene. He can, if he wishes fill the entire canvas with the background scene. The artist then paints the foreground objects. There is no need to erase portion of the background; the artist simply paints on top of them. The new paint covers the old so that only the newest layer of paint is visible. A frame buffer has this same property. We enter a filled polygon into the frame buffer by changing the proper pixel to values corresponding to the polygon's interior style. If we then enter a second polygon "on top of" the first, some of that same pixel will be changed to correspond to the second polygon interior style. Wherever the second polygon lies, the first polygon's pixel setting have been forgotten. The second polygon has covers up the first. The painters algorithm tell us to enter first those polygon which are farthest from the viewer and enter last the objects closet to the viewer. Hidden surface can be covered up by choosing the correct order to draw them and taking advantage of the properties of frame buffer.

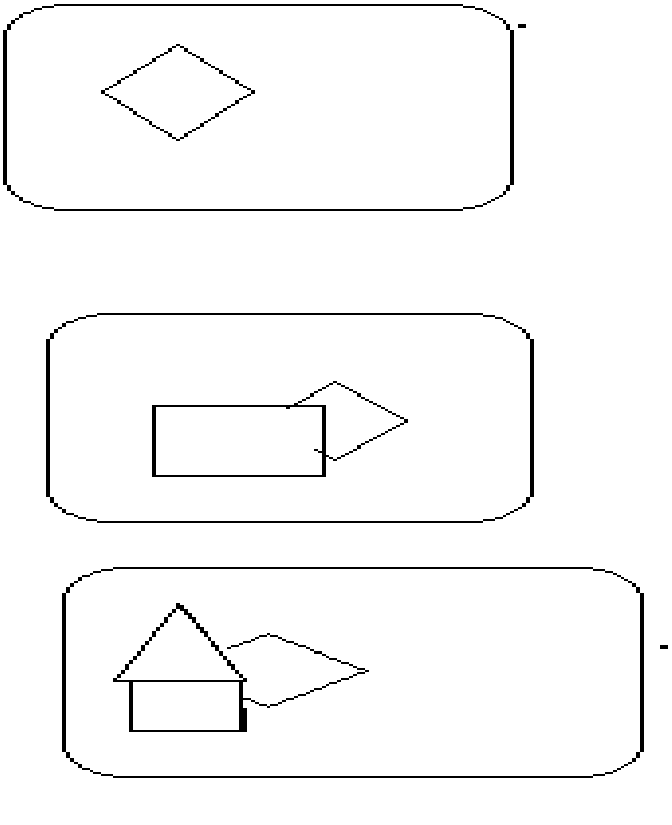


Fig:- The most recent filled polygon overwrites previous pixel values.

(c) Discuss any three 3D primitives.

Ans:- 3D primitives is as follows algorithms

1) **Algorithm MOVE-ABS-3(X,Y,Z)** the 3D absolute move

```
Arguments    X,Y,Z world coordinates of the point to move the pen to
Global       DF-PEN-X, DE-PEN-Y, DF-PEN-Z current pen position
Begin
        DF-PEN-X ← X;
        DE-PEN-Y ← Y;
        DF-PEN-Z ← Z;
        DISPLAY-FILE-ENTER(1);
        RETURN;
END
```

2) **Algorithm MOVE-REL-3(DX,DY,DZ)** the 3D relative move

```
Arguments    DX, DY, DZ change to be made to the pen position
Global       DF-PEN-X, DE-PEN-Y, DF-PEN-Z current pen position
Begin
        DF-PEN-X ← DF-PEN-X + DX;
        DE-PEN-Y ← DF-PEN-X + DY;
        DF-PEN-Z ← DF-PEN-X + DZ;
        DISPLAY-FILE-ENTER(1);
        RETURN;
END
```

3) **Algorithm LINE-ABS-3(X,Y,Z)** the 3D absolute line drawing routine

```
Arguments    X,Y,Z world coordinates of the point to draw the line to
Global       DF-PEN-X, DE-PEN-Y, DF-PEN-Z current pen position
Begin
        DF-PEN-X ← X;
        DE-PEN-Y ← Y;
        DF-PEN-Z ←  Z;
        DISPLAY-FILE-ENTER(2);
        RETURN;
    END
```

(D) How is conversion to view plane coordinates achieved? Write an algorithm for Make – View – Plane – Transformation.

Ans:- Conversion To View Plane Coordinates Achieved :The user enters the description of the object in terms of the object coordinates. But our particular point of view corresponds to expressing the object in the view plane coordinates. The process of generating a particular view of the object is one of transforming form one coordinate system to another. The steps to be taken are the same as those which were used for a rotation about an arbitrary axis. The first step is a translation to move the origin to the correct position for the view plane coordinates system. This is a shift first to the view reference point, and then along the view plane normal by the VIEW-DISTANCE . After the origin is place, we align the Z axis. In the rotation problem, we saw how to rotate a line onto the z axis. the line to be the object coordinates z axis, and we shall rotate it into he view plane coordinates z axis. This is done in two steps. First, a rotation about the x asis place the line in the view plane coordinates xz plane. The entire transformation sequence is given by:

TMATRIX = $TR_XR_YR_Z$

Where

$$T = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -(XR + DXN * \text{VIEW-DISTANCE}) & -(YR + DYN * \text{VIEW-DISTANCE}) & -(ZR + DZN * \text{VIEW-DISTANCE}) \end{vmatrix}$$

$$R_x = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & -DZN/V & -DYN/V & 0 \\ 0 & DYN/V & -DZN/V & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

And

$$V = (DNY^2 + DZN^2)^{1/2}$$

Also

$$R_y = \begin{vmatrix} V & 0 & -DXN & 0 \\ 0 & 1 & 0 & 0 \\ DXN & 0 & V & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

And

$$R_z = \begin{vmatrix} YUP\text{-}VP/RUP & XUP\text{-}VP/RUP & 0 & 0 \\ -XUP\text{-}VP/RUP & YUP\text{-}VP/RUP & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Where

$[XUP\text{-}VP \quad YUP\text{-}VP \quad Z \quad 1] = [DXUP \quad DYUP \quad DZUP \quad 1]\, R_xR_y$

And $RUP = (XUP\text{-}VP_2 + YUP\text{-}VP_2)_{1/2}$

Algorithm for Make **– View – Plane – Transformation**

Algorithm Make **– View – Plane – Transformation** for making the viewing transformation

Global XR,YR,ZR the view reference point

DXN,DYN,DZN the view plane normal

```
          DXUP,DYUP,DZUP the view-up direction

          TMATRIX a 4 X 3 transformation matrix array

          PERSPECTIVE-FLAG the perspective projection flag

          VIEW-DISTANCE distance between view reference point and view plane

Local     V,XUP-VP,YUP-VP,RUP for storage of partial results

Constant            ROUNDOFF some small number greater than any round-of  error.

Begin

          Start with  the identity matrix

          NEW-TRANSFORM-3;

          Translate so that view plane center is new origin

          TRANSLATE-3(-(XR+DXN*VIEW-DISTANCE), -(YR + DYN * VIEW-DISTANCE),
          -( ZR + DZN * VIEW-DISTANCE)

          Rotate so that view plane normal is z axis

          V ← SQRT(DYN ↑ 2 + DZN ↑2);

          IF V > ROUNDOFF THEN ROTATE – X-3 (- DYN /V, - DZN /V);

          ROTATE-Y-3(DXN, V)

          Determine the view-up direction in these new coordinates

          XUP-VP  ←  DXUP* TMATRIX [1, 1] + DYUP* TMATRIX [2, 1] + DZUP * TMATRIX [3, 1]

          YUP-VP  ←  DXUP* TMATRIX [1, 2] + DYUP* TMATRIX [2, 2] + DZUP * TMATRIX [3, 2]

          Determine rotation needed to make view-up vertical

          RUP ← SQRT(XUP-VP ↑ 2 + YUP-VP ↑2);

          If rup < roundoff then
```

```
                RETURN ERROR ‘SET-VIEW-UP- ALONG VIEW PLANE NORMAL’;
        ROTATE-Z-3(XUP-VP / RUP, YUP-VP / RUP);
        IF PERSPECTIVE-FLAG THEN MAKE-PERSPECTIVE-TRANSFORMATION
        ELSE MAKE-PARALLEL-TRANSFORMATION;
        REUTN;
END
```

UNIT - IV

Q4. (a) Give short notes on transparency, reflection and shadows

Ans:- Transparency:- In the shading models we have not considered the transparent objects. A transparent surface, in general, produces both reflected and transmitted light. It has a transparency coefficient T as well as values for reflectivity and specular reflection. The coefficient of transparency depends on the thickness of the object because the transmission of light depends exponentially on the distance which the light ray must travel within the object. The expression for coefficient of transparency is given as

$T = te^{-ad}$

Where t is the coefficient of property of material which determines how much of the light is transmitted at the surface instead of reflected, a is the coefficient of property of material which tells how quickly the material absorbs or attenuates the light, d is the distance the light must travel in the object.

When light crosses the boundaries between two media it changes the direction as shown in fig.

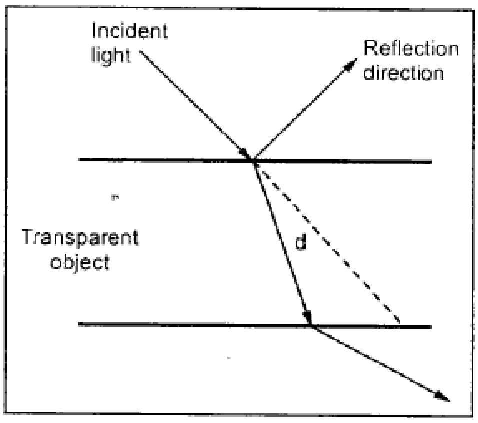


Fig:- Refraction

This effect is called refraction. The effect of refraction is observed because the speed of light is different in different material resulting different path for refracted light from that of incident light. The direction of the refracted light is specified by the angle of refraction (Θ_r). It is the function of the property materials called the index of refraction (n). The angle of refraction Θ_r is calculated from the angle of incident Θ_i , the index of refraction n_i of the incident material, and the index of refraction n_r of the refracting material according to snell' law:

$$\sin\theta_r = \frac{n_i}{n_r}\sin\theta$$

The transparency and absorption coefficient are also depend on colour. Therefore, when we are dealing with colour objects we require three pairs of transparency and absorption coefficients.

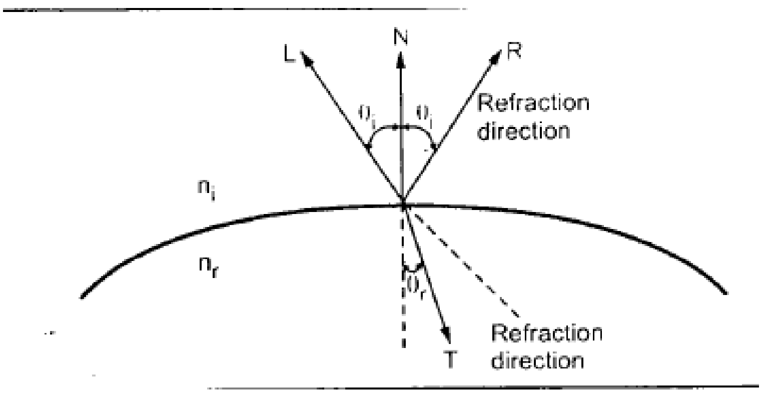


Fig:- Refraction

For modelling of transparent surface we have to consider contribution from the light reflected from the surface and the light coming from behind the object. If we assume for a given surface that

i. The transparency coefficient for the object is a constant
ii. Refraction effects are negligible and
iii. No light source can be seen directly through the object,

Then the light coming through the object is given as,

$v = v_r + t\ v_1$

where v is the total amount of light

v_r is the amount of light reflected from the surface,

t is the transparency coefficient and,

v_t is the light coming from behind the object.

To get more realistic images we have to consider the angular behaviour of the reflection v_s transmission at the surface and also the attenuation due to thickness. The simple approximation for this behaviour can be given as

$t = (t_{max} – t_{min}) (N.E)^{\alpha} + t_{min}$

Therefore, we can say that cosine of angle is maximum when surface is viewed straight on and it drops off for glancing views. The power of angle represented by α enhance the effect.

Reflection: - The effect of light coming directly from illumination sources, but in a realistic scene, every object can act as a light source. We should be able to see the specular reflections of objects, not just of lights. We look at a points p1 on a object and ask what light is coming from that point and from where does it originate. We have seen how to find the light coming from diffuse illumination and directly from point sources. We can also follow a reflected ray back from our eye to the point and determine the incident ray. If we move back along the incident ray and reach a point p2 on some other surface, then we know that light (I) from this second point will be reflected at p1. (see fig)

This will add a term to equation for the contribution of p2 of the form

$$\frac{IF'/(E.N)}{C+U}$$

Where F' is calculated according to equation just f, and only c used in this calculation is given by

$$C = E . N$$

The light (I) for the point p2 should be calculated as if located at p1. For such calculations, all surfaces must be considered, even back faces.

Shadow:- A shadowed object is one which is hidden from the light source. It is possible to use hidden surface algorithms to locate the areas where light sources produce shadows. In order to achieve this we have to repeat the hidden surface calculation using light source as the viewpoint. This calculation divides the surface into shadowed and unshadowed groups. The surface that are visible from the light source are not in shadow; those that are not visible from the light source are in shadow. Surface which are visible and which are also visible from the light source are shown with both background illumination and the light-source illumination. Surfaces which are visible but which are hidden form the light source are displayed with only the background illumination as shown in fig.

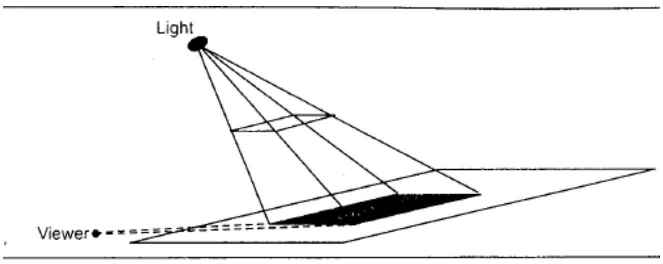


Fig:- Shadow

(B) Write an algorithm BEZIER_ABS_3 to adds a Bezier curve section to display file.

Ans:- Algorithm for adding Bezier curves to our system are given below. They assume that the first control point is at the current pen position, so only three additional points must be specified. They use the recursive subdivision approach.

Algorithm BEZIER-ABS-3(XB, YB, ZB, XC, YC, ZC, XD, YD, ZD, N) Adds a Bezier curve section to the display file.

```
Arguments    XB, YB, ZB, XC, YC, ZC, XD, YD, ZD control points
              N the desired degree of subdivision
Global       DF-PEN-X, DE-PEN-Y, DE-PEN-Z the current pen position
Local        XAB, YAB, ZAB, XBC, YBC, ZBC, XCD, YDC, ZCD first level mid points
             XABC, YABC, ZABC, XBCD, YBCD, ZBCD second level midpoints
             XABCS, YABCD, ZABCD THIRS LEVEL MIDPOINT
BEGIN
    IF N=0 THEN
    LINE – ABS-3(XB, YB, ZB);
    LINE – ABS-3(XC, YC, ZC);
    LINE – ABS-3(XD, YD, ZD);
    END;
ELSE
```

```
BEGIN
XAB ← (DF-PEN-X + XB) / 2;
YAB ← (DF-PEN-Y + YB) / 2;
ZAB ← (DF-PEN-Z + ZB) / 2;
XBC ← ( XB+ XC) / 2;
YBC ← ( YB+ YC) / 2;
ZBC ← ( ZB+ ZC) / 2;
XCD ← ( XC+ XD) / 2;
YCD ← ( YC+ YD) / 2;
ZCD ← ( ZC+ ZD) / 2;
XABC ← ( XAB+ XBC) / 2;
YABC ← ( YAB+ YBC) / 2;
ZABC ← ( ZAB+ ZBC) / 2;
XBCD ← ( XBC+ XCD) / 2;
YBCD ← ( YBC+ YCD) / 2;
ZBCD ← ( ZBC+ ZCD) / 2;
XABCD ← (XABC+ XBCD) / 2;
YABCD ← (YABC+ YBCD) / 2;
ZABCD ← (ZABC+ ZBCD) / 2;
BEZIER-ABS-3(XAB, YAB, ZAB, XABC, YABC, ZABC, XABCS, YABCS, ZABCS, N -1);
BEZIER-ABS-3(XBCD, YBCD, ZBCD, XCD, YCD, ZCD, XD, YD, ZD, N – 1 );
```

```
        END;
RETURN;
END;
```

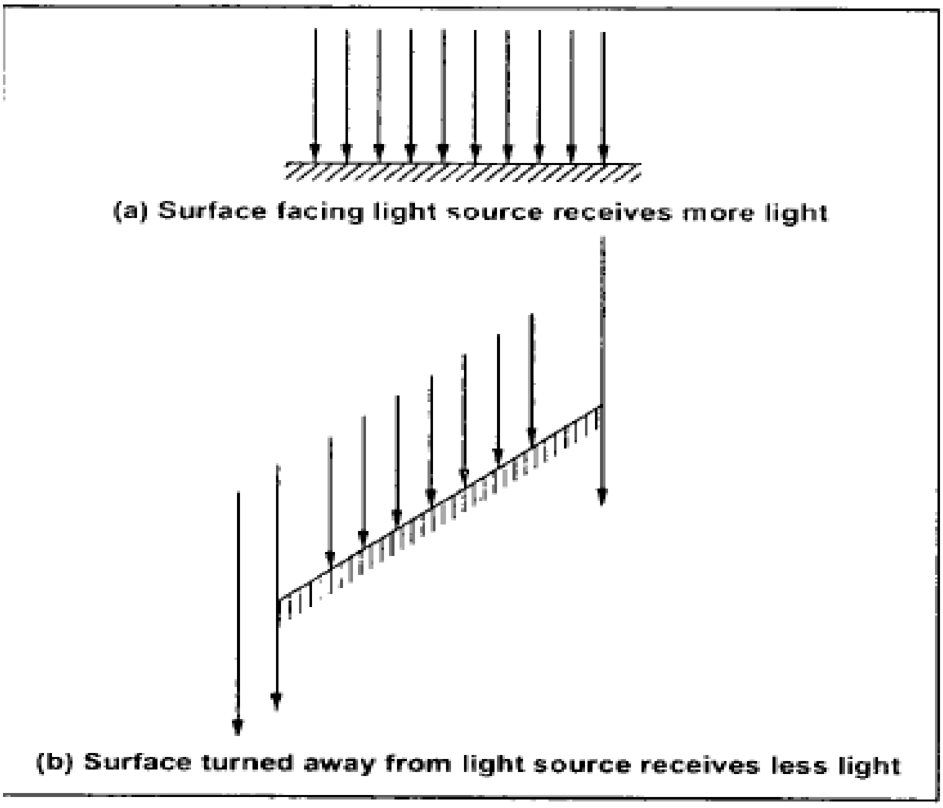
(C) Explain the following: -

(i) Diffuse Illumination

(ii) Point source illumination

Ans:- (i) Diffuse Illumination:-

(ii) Point Source Illumination:-Point source are abstraction of real-world sources of light such as light bulbs, candles, or the sun. The light originates at a particular place, it comes from a particular direction over a particular distance. For point source, the position and orientation of the objects surface relative to the light source will determine how much light the surface will receive and in turn how bright it will appear. Surfaces facing toward and positioned near the light source will receive more light than those facing away from or far removed from the illumination. Let us begin by considering the effects of orientation. By arguments similar to those we made earlier, we can see that as a face is turned away from the light source, the surface area on which a fixed-sized beam of light falls increases. This means that there is less light for each surface point. The surface is less brightly illuminated (see fig.).


(a) Surface facing light source receives more light
(b) Surface turned away from light source receives less light

The illumination is decreased by a factor of COS I, where I is the angle between the direction of the light and the direction normal to the plane. The angle I is called the angle of incidence (see fig.)

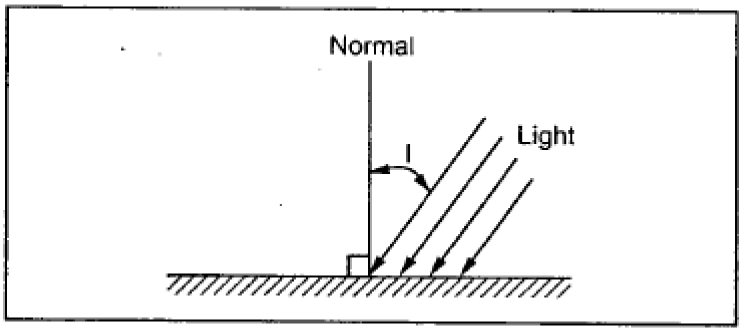


Fig.:- The angle of incidence

If we have a vector of length 1 called L pointing towards the light source and a vector of length 1 called N in the direction normal to the surface, then the vector dot product gives

COS I = L . N

Suppose that P amount of light comes from the point source. Then the shade of a surface of an object will be given by

V = BR + PR(L . N)

Q4 (A) Explain B-spline curves generation in detail.

Ans:-

(B) Discuss:

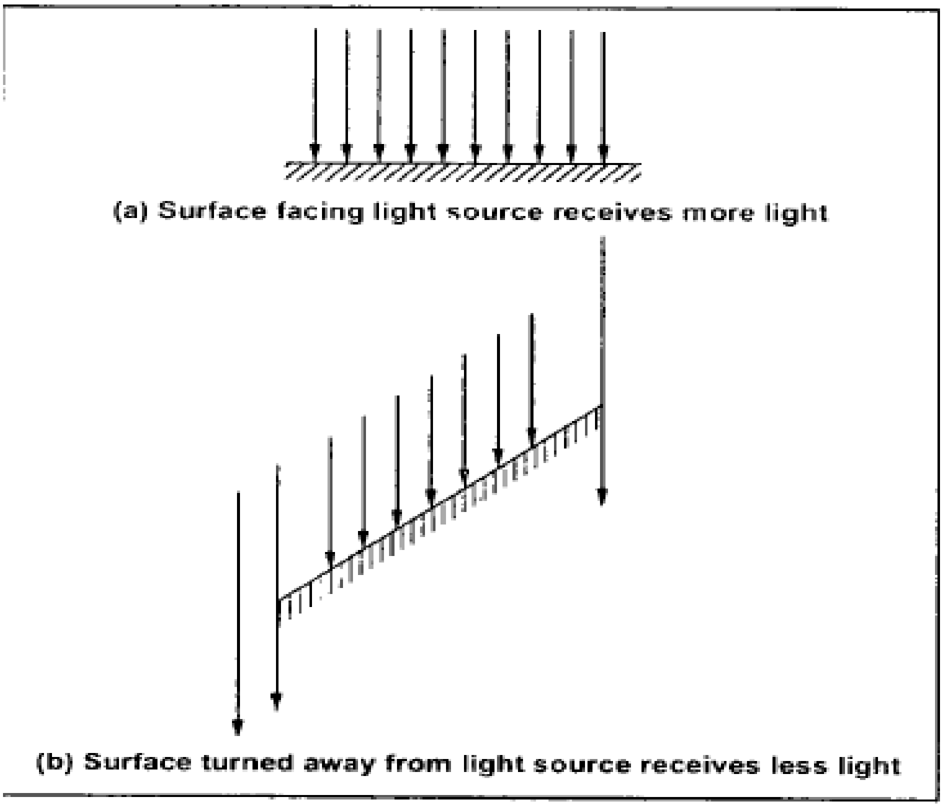
(i) Diffused illustration.

(ii) Point source illustration.

Ans:- (i) Diffuse Illumination:- An object illumination is as important as its surface properties in computing its intensity. The object may be illuminated by light which does not come from any particular source but which comes from all directions. When such illumination is uniform from all directions, the illumination is called diffuse illumination. Usually, diffuse illumination is a background light which is reflected form walls, floor, and ceiling.

When we assume that going up, down, right and left is of same amount then we can say that the reflections are constant over each surface of the object and they are independent of the viewing direction. Such a reflection is called diffuse reflection. In practice, when object is illuminated, some part of light reflected from the surface of the object, while the reset is reflected. The ratio of the light reflected from the surface to the total incoming light to the surface is called coefficient of reflection or the reflectivity. It is denoted by R. The value of R varies from 0 to 1. It is closer to 1 for white surface and closer to 0 for black surface. This is because white surface reflects nearly all incident light whereas black surface absorbs most of the incident light. Reflection coefficient for gray shades is in between 0 to 1.

(ii) Point Source Illumination:-Point source are abstraction of real-world sources of light such as light bulbs, candles, or the sun. The light originates at a particular place, it comes from a particular direction over a particular distance. For point source, the position and orientation of the objects surface relative to the light source will determine how much light the surface will receive and in turn how bright it will appear. Surfaces facing toward and positioned near the light source will receive more light than those facing away from or far removed from the illumination. Let us begin by considering the effects of orientation. By arguments similar to those we made earlier, we can see that as a face is turned away from the light source, the surface area on which a fixed-sized beam of light falls increases. This means that there is less light for each surface point. The surface is less brightly illuminated (see fig.).


(a) Surface facing light source receives more light
(b) Surface turned away from light source receives less light

The illumination is decreased by a factor of COS I, where I is the angle between the direction of the light and the direction normal to the plane. The angle I is called the angle of incidence (see fig.)

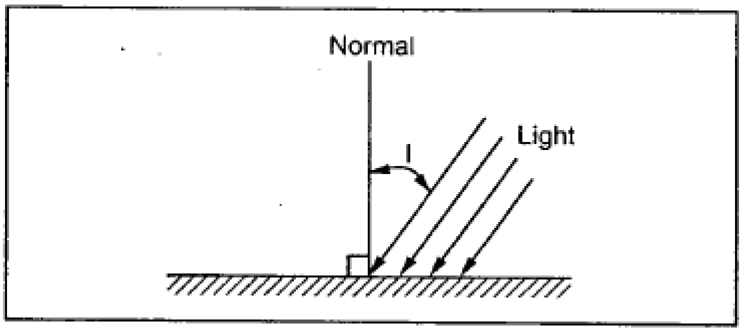


Fig.:- The angle of incidence

If we have a vector of length 1 called L pointing towards the light source and a vector of length 1 called N in the direction normal to the surface, then the vector dot product gives

COS I = L . N

Suppose that P amount of light comes from the point source. Then the shade of a surface of an object will be given by

V = BR + PR(L . N)

(C) Write short notes on :

(i) Transparency

(ii) Shadows

Ans:- (i) Transparency :- In the shading models we have not considered the transparent objects. A transparent surface in general, produces both reflected and transmitted light. It has a transparency coefficient T as well as values for reflectivity and specular reflection. The coefficient of transparency depends on the thickness of the object because the transmission of light depends exponentially on the distance which the light ray must travel within the object. The expression for coefficient of transparency is given as

$$T = te^{-ad}$$

Where t is the coefficient of property of material which determines how much of the light is transmitted at the surface instead of reflected, a is the coefficient of property of material which tells how quickly the material absorbs or attenuates the light, d is the distance the light must travel in the object.

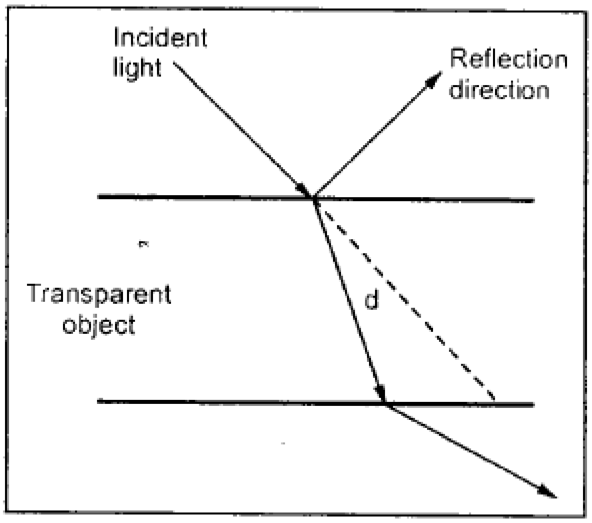


Fig:- Reflection

When light crosses the boundary between two media it changes the direction as shown in the fig. This effect is called refraction. The effect of refraction is observed because the speed of light is different in different materials resulting different path for refracted light from that incident light. The direction of the refracted light is specified by angle of refraction (θ_r). It is the function of the property materials called the index of refraction (n). The angle of refraction θ_r is calculated from the angle of incidence θ_i , the index of refraction n_i of the incident material (usually air), and the index of refraction n_r of the refracting material according to Snell's law:

$$\sin\theta_r = \frac{n_i}{n_r}\sin\theta_i$$

In practice, the index of refraction of a material is a function of the wave length of the incident light, so that the different color components of a light ray refracts at different angles. The transparency and absorption coefficients are also depend on color. Therefore, when we are dealing with color objects we require three pairs of transparency and absorption coefficients.

(ii) Shadow: A shdowed abject is one which is hidden from the light source. It is possible to use hidden surface algorithms to locate the areas where light sources produce shadows. In order to achieve this we have to repeat the hidden-surface calculation using light source as the viewpoint. This calculation divides the surfaces into shadowed and unshadowed groups. The surfaces that are visible from the light source are not in shadow; those that are not visible from the light source are in shadow. Surfaces which are visible and which are also visible from the light source are shown with both the background illumination and the light-source illumination. Surfaces which are visible but which are hidden from the light source are displayed with only the background illumination, as shown in fig.1.

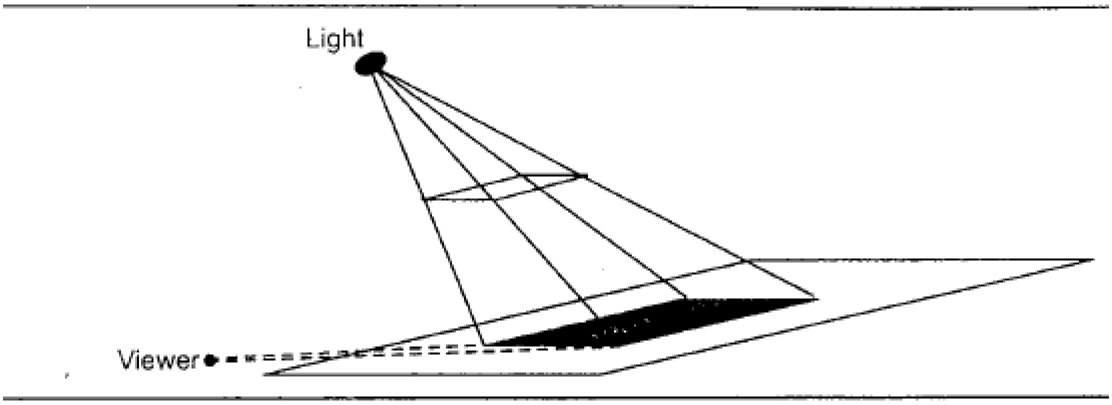


Fig 1: Shadow

Another way to locate shadow areas is the use of shadow volumes. A shadow volume is define by a set of invisible shadow polygons as shown in fig.2 This volume is also known as polygon’s shadow volume. By comparing visible polygon with this volume we can identify the portions which lie inside of the volume and which are outside of the volume. The portions which lie inside of the volume are shadowed, and their intensity calculations do not include a term from the light source. The polygon or portion of polygon which lie outside the shadow volume are not shaded by this polygon, but might be

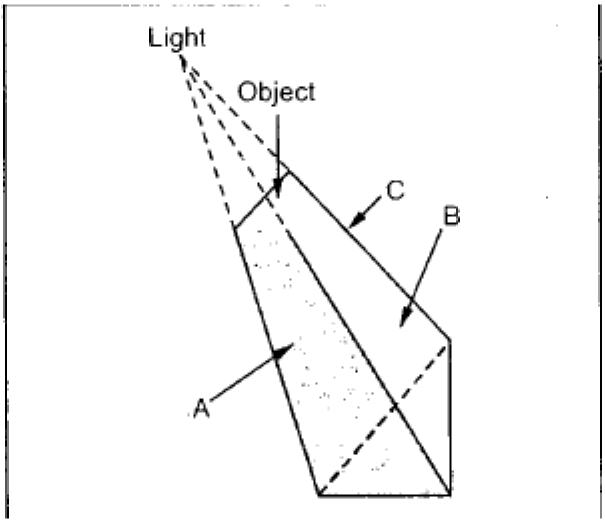


Fig 2

shaded by some other polygon so they still must be shaded by some other polygon so they still must be checked against the other shadow volume.

(D) Explain Specular Reflection.

Ans:- **Specular Reflection:-** When we illuminate a shiny surface such as polished metal or an apple with a bright light, we observe highlight or bright spot on the shiny surface. This phenomenon of reflection of incident light in a concentrated region around the specular reflection angle is called specular reflection. Due to specular reflection, at the highlight, the surface appears to be not in its original color, but white, the color of incident light.

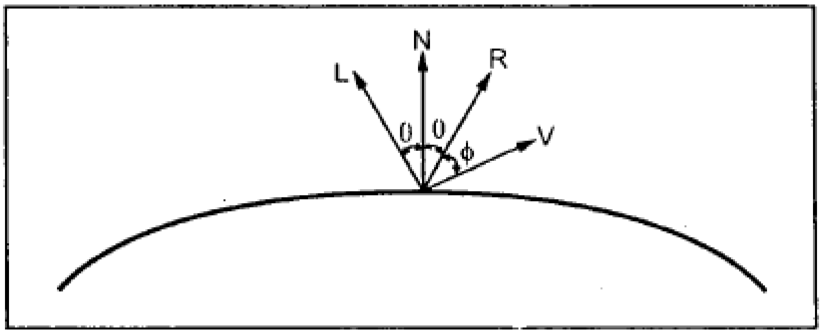


Fig. Specular Reflection

The fig shows the specular reflection direction at a point on the illuminated surface. The specular reflection angle equals the angle of the incident light, with the two angles measured on opposite sides of the unit normal surface vector N. As shown in the fig2. R is the unit vector in the direction of ideal specular reflection, L is the unit vector directed towards the point light source and V is the unit vector pointing to the viewer from the surface position.

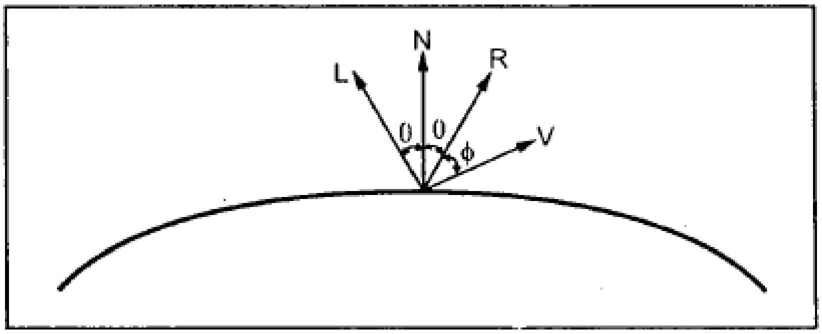
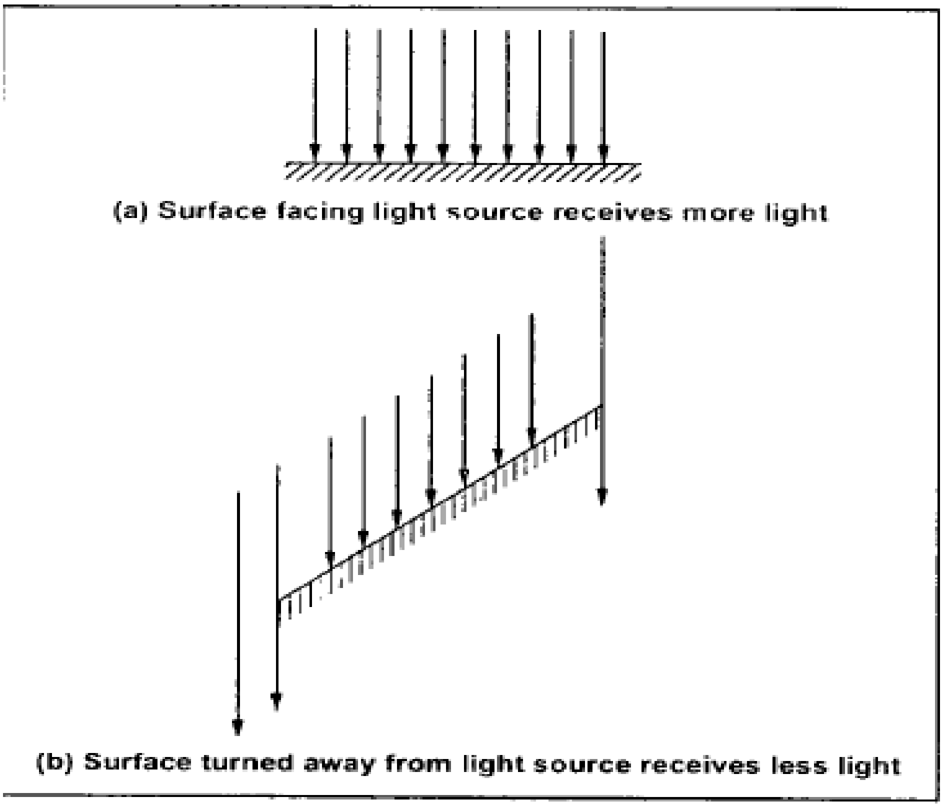


Fig2: Specular Reflection

The angle ϕ between vector R and vector V is called viewing angle. For an ideal reflector, incident light is reflected only in the specular reflection direction. In such case, we can see reflected light only when vector V and R coincide i.e. $\phi = 0$.

Q4 (a) Write a note on Point source illumination.

Ans:- Point source are abstraction of real-world sources of light such as light bulbs, candles, or the sun. The light originates at a particular place, it comes from a particular direction over a particular distance. For point source, the position and orientation of the objects surface relative to the light source will determine how much light the surface will receive and in turn how bright it will appear. Surfaces facing toward and positioned near the light source will receive more light than those facing away from or far removed from the illumination. Let us begin by considering the effects of orientation. By arguments similar to those we made earlier, we can see that as a face is turned away from the light source, the surface area on which a fixed-sized beam of light falls increases. This means that there is less light for each surface point. The surface is less brightly illuminated (see fig.).


(a) Surface facing light source receives more light
(b) Surface turned away from light source receives less light

The illumination is decreased by a factor of COS I, where I is the angle between the direction of the light and the direction normal to the plane. The angle I is called the angle of incidence (see fig.)

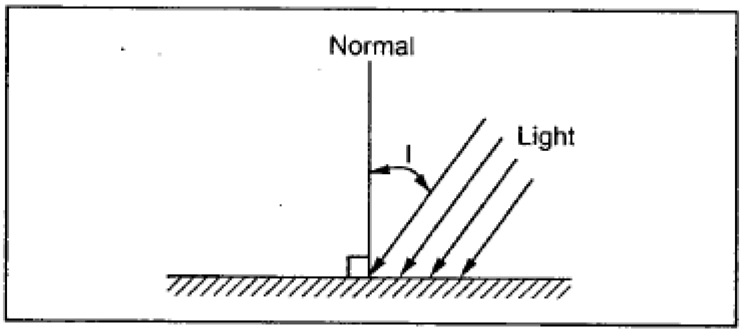


Fig.:- The angle of incidence

If we have a vector of length 1 called L pointing towards the light source and a vector of length 1 called N in the direction normal to the surface, then the vector dot product gives

COS I = L . N

Suppose that P amount of light comes from the point source. Then the shade of a surface of an object will be given by

V = BR + PR(L . N)

(b) what is polygon interpolation? Write an algorithm for polygon smoothing?

Ans:- **Polygon Interpolation :-** The sides of a polygon can also be rounded by means of our blending function. In fact, a polygon is conceptually easier to deal with, since no special initial or final section occurs. We just step around the polygon, smoothing out each side by replacing it with several small line segments. We start out with a polygon that has only a few sides and end up with a polygon which has many more sides and appears smoother.

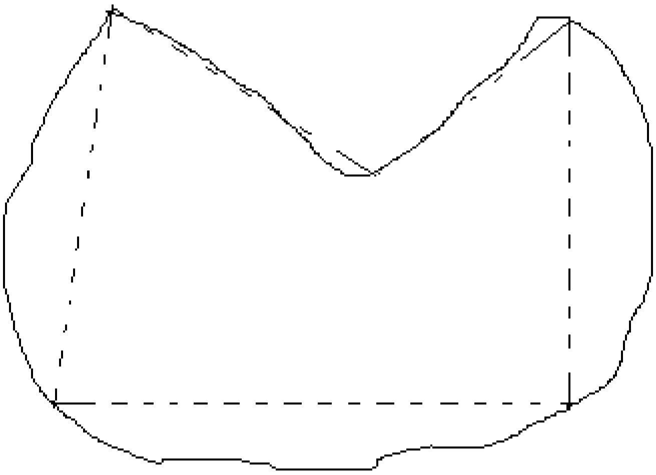


Fig:- Smoothing of a polygon

Algorithm for polygon Smoothing

Algorithm Smooth-Poly-Abs-3(Ax, Ay, Az, M) User routine to draw a smoothed polygon

Arguments AX, AY, AZ arrays containing vertices of original polygon

M the original number of polygon sides

Global BLEND the array of blending function values

LINE-PER-SECTION the number of line segments per original polygon side

```
Local        BX, BY, BZ arrays to hold smothed polygon sertices
             NSIDES number of sides for the smoothed polygon
             J index variable for saving the smoothed polygon sides
             K to step thorough the small segments for each polygon side
             I, I1 variables for stepping through the four sample points
             L for stepping through the four blending functions
BEGIN
    IF M<3 THEN RETURN ERROR ‘POLYGON SIZE ERROE’;
    NSIDES ⟵ LINES-PER-SECTION * M;
    IF LINES-PER-SECTION = 1 THEN POLYGON-ABS-3(AX, AY, AZ, M)
    ELSE
        BEGIN
            J ⟵ 1
            FOR I1 = 1 TO M DO
            Smooth all sides
            FOR K = 1 TO LINE-PER-SECTION DO
            Smooth a side of the original polygon
            BEGIN
            BX[J] ⟵ 0;
            BY[J] ⟵ 0;
            BZ[J] ⟵ 0
            I ⟵ I1;
            FOR L = 1 TO 4 DO
```

```
                    One side of the smooth polygon
                    BEGIN
                    BX[J] ⟵ BX[J] + AX[I] * BLEND [L, K]
                    BY[J] ⟵ BY[J] + AY[I] * BLEND [L, K]
                    BZ[J] ⟵ BZ[J] + AZ[I] * BLEND [L, K]
                    END
                J  ] ⟵ J +1;
        END;
Draw the result
POLYGON-ABS-3(BX,BY,BZ,NSIDES);
END
RETURN
END;
```

(C) Explain Specular reflection in detail.

ANS:- When we illuminate a shiny surface such as polished metal or an apple with a bright light, we observe highlight or bright spot on the shiny surface. This phenomenon of reflection of incident light in a concentrated region around the specular reflection angle is called specular reflection. Due to specular reflection, at the highlight, the surface appears to be not in its original colour, but white, the colour of incident light.

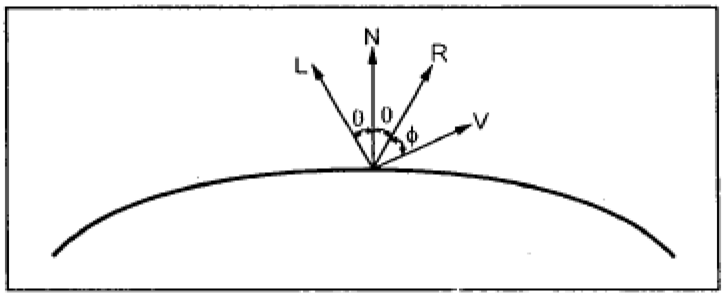


Fig:- Specular Reflection

The fig shows the specular reflection direction at a point on the illuminated surface. The specular reflection angle equals the angle of the incident light, with the two angles measured on opposite sides of the unit normal surface vector N. As shown in the fig, R is the unit vector in the direction of ideal specular reflection, L is the unit vector directed toward the point light source and V is the unit vector pointing to the viewer form the surface position.

The angle ϕ between vector R and vector V is called viewing angle. For an ideal reflector (perfect mirror), incident light is reflected only in the specular reflection direction. In such case, er can see reflected light only when vector V and R coincide, i.e. $\phi = 0$.

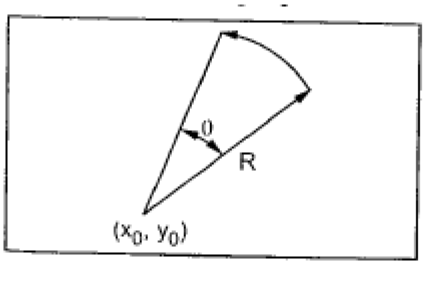
(d) Write and Explain an algorithm for counter clock wise circular arc generation.

Ans:- Circular arc generation: - Digital differential analyzer algorithm uses the differential equation of the curve. The differential equations for simple curve such as circle is fairly easy to solve. Let us see the DDA algorithm for generating circular arcs. The equation for an arc in the angle parameters can be gives as

$X = R \cos\Theta + x_0$

$Y = R \sin\Theta + y_0$

Where (x_0,y_0) is the center of curvature, and R is the radius of arc.



Differentiaitng equation, we get

$Dx = -R \sin\Theta \, d\Theta$

$Dy = R \cos\Theta \, d\Theta$

From above equation we can solve for R cosΘ and R sin Θ as follows.

$X = R \cos\Theta + x_0$

$R \cos\Theta = x - x_0$ and

$R \sin \Theta = y - y_0$

Substituting values of R cosΘ and R sin Θ from equation we get,

$Dx = -(y - y_0)\, d\Theta$

$Dy = (x - x_0)\, d\Theta$

The values of dx and dy indicates the increment in x and y increment, respectively, to be added in the current point on the arc to get the next point on the arc. Therefore, we can write

$X_2 = x_1 + dx = x_1 - (y_1 - y_0)\, d\Theta$

$y_2 = y_1 + dy = y_1 - (x_2 - x_0)\, d\Theta$

the above equation is the basic for arc generation.

Algorithm for counter clock wise circular arc generation.

Algorithm ARCDDA(X0, Y0, A, X, Y, INTENSITY) counter clock wise circular arc generation

Arguments	X0, Y0 the center of curvature
	A the arc angle
	X, Y the starting point for the arc drawing
Global	FRAME the frame buffer array
Local	XARC, YARC the next point to draw
	A-DRAWN the amount of angle currently coveres
	DA the angle increment
Constant	ROUNDOFF some small number greater than any round-of error

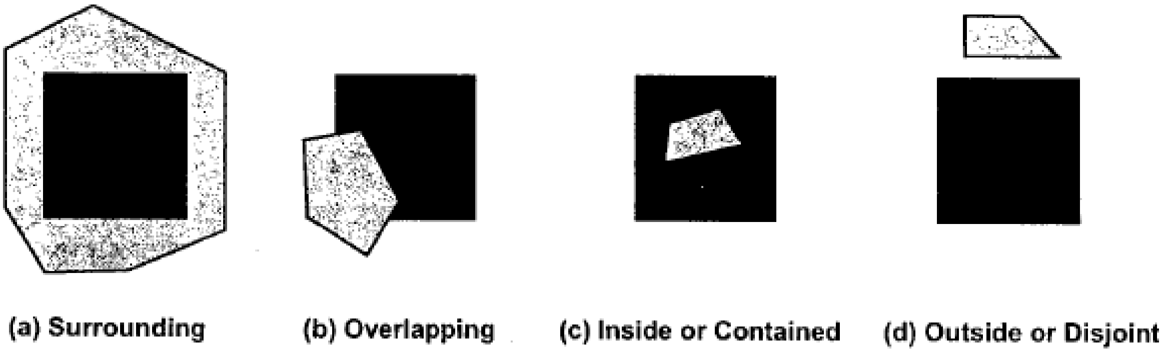
```
BEGIN
    IF | X – X0| + |Y – Y0| < ROUNDOFF THEN RETURN
    A-DRAWN ⟵ 0;
    Find a suitable angle increment
    DA ⟵ MIN(0.01 / (3.2 *(| X – X0| + |Y – Y0|)));
    Set the first point of the arc
    XARC ⟵ X;
    YARC ⟵ Y;
    Generating the arc until desired angle is covered
    WHILE A=DRAWN < A DO
        BEGIN
        Find new point
        XARC ⟵ XARC + (Y0 – YARC) * DA;
        YARC ⟵ YARC + (XARC – X0) *DA;
        A-DRAWN ⟵ A-DRAWN + DA;
        Set the corresponding pixel
        FRAME[INT(XARC), INT (YARC)] ⟵ INTENSITY;
    END;
RETURN;
END;
```

UNIT - V

Q5 (a) Explain Area – subdivision method in detail

ANS:- Area – subdivision method:- An interesting approach to the hidden-surface problem was developed by Warnock. He developed area subdivision algorithm which subdivides each area into four equal squares. At each stage in the recursive- subdivision process, the relationship between projection of each polygon and the area of interest is checked for four possible relationships:

1. Surrounding Polygon – one that completely encloses the (shaded) area of interest (se fig (a))
2. Overlapping or Interesting Polygon – one that is partly inside and partly outside the area (see fig b)
3. Inside or Contained Polygon – one that is completely inside the area (see fig c)
4. Outside or Disjoint Polygon – one that is completely outside the area (see fig d)



(a) Surrounding

(b) Overlapping

(c) Inside or Contained

(d) Outside or Disjoint

Fig:- Possible relationships with polygon surfaces and the area of interest

After checking four relationships we can handle each relationship as follows:

1. If all the polygons are disjoint from the area, then the background colour is displayed in the area.
2. If there is only one interesting or only one contained polygon, then the area is first filled with the background colour, and then the part of the polygon contained in the area is filled with colour of polygon.
3. If there is a single surrounding polygon, but no intersecting or contained polygons, then the area is filled with the colour of the surrounding polygon.

4. If there are more than one polygon intersections of surrounding polygon are all closer to the viewpoint that any of the other intersections. Therefore, the entire area is filled with the colour of the surrounding polygon.

(b) Explain scan – line method for visible surface detection.

Ans:- **Scan – Line Method For Visible Surface Detection:-** A scan line method of hidden surface removal is an another approach of image space method. If is an extension of the scan line algorithm for filling polygon interiors. Here, the algorithm deals with more than one surfaces. As each scan line is processed, it examines all polygon surfaces intersecting that line to determine which are visible. It then does the depth calculation and finds which polygon is nearest to the view plane. Finally, it enters the intensity value of the nearest polygon at that position into the frame buffer.

We know that scan line algorithm maintains the active edge list. This active edge list contains only edges that cross the current scan line, sorted in order of increasing x. The scan line method of hidden surface removal also stores a flag for each surface that is set on or off to indicate whether a position along a scan line is inside or outside of the surface. Scan lines are processed from left to right. At the leftmost boundary of a surface, the surface flag is turned ON and at the rightmost boundary, it is turned OFF.

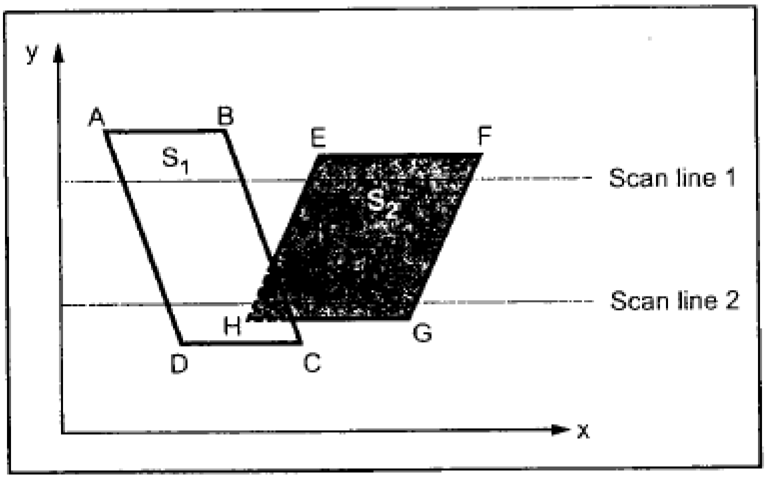


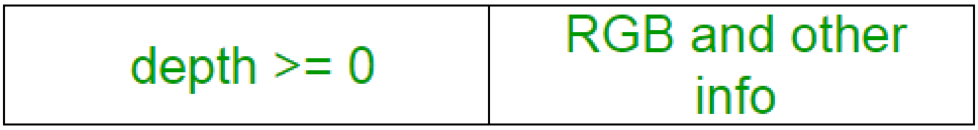
Fig:- Illustration of scan line method

The fig illustrates the scan line method for hidden surface removal. As shown in fig the active edge list for scan line 1 contains the information for edges AD, BC, EH and FG. For the position along this scan line between edges AD and BC, only the flag for surface S1 is ON. Therefore, no depth calculations are necessary, and intensity information for surface s_1 is entered into the frame buffer. Similarly, between edges EH and FG, only the flag for surface S2 is ON and during that portion of scan line the intensity information for surface S2 is entered into the frame buffer.

For scan line 2 in the fig, the active edge list contains edges AD, EH, EC and FG. Along the scan line 2 from edge AD to edge EH, only the flag surface S1 is ON. However, between edges EH and BC, the flags for both surface are ON. In this portion of scan line 2, the dept calculations are necessary. Here we have assumed that the depth of s1 is less than the depth s2 and hence the intensities of surface S1 are loaded into the frame buffer. Then, for edge BC to edge FG portion of scan line 2 intensities of surface S2 are entered into the frame buffer because during that portion only for s2 is ON.

5. (A) Explain :
(i) A-Buffer method
(ii) Depth Buffer method.
Ans:- A- Buffer Method:-

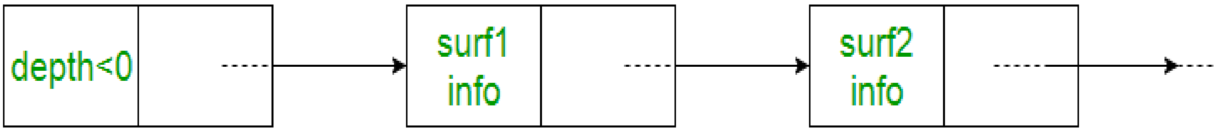
A-Buffer method in computer graphics is a general hidden face detection mechanism suited to medium scale virtual memory computers. This method is also known as **anti-aliased** or **area-averaged** or **accumulation buffer**. This method extends the algorithm of depth-buffer (or Z Buffer) method. As the depth buffer method can only be used for opaque object but not for transparent object, the A-buffer method provides advantage in this scenario. Although the A buffer method requires more memory, but different surface colors can be correctly composed using it. Being a descendent of the Z-buffer algorithm, each position in the buffer can reference a linked list of surfaces. The key data structure in the A buffer is the accumulation buffer.
Each position in the A buffer has 2 fields :
1) Depth field
2) Surface data field or Intensity field
A depth field stores a positive or negative real number. A surface data field can stores surface intensity information or a pointer to a linked list of surfaces that contribute to that pixel position.



depth >= 0	RGB and other info

(a) When a pixel overlap by only one surface

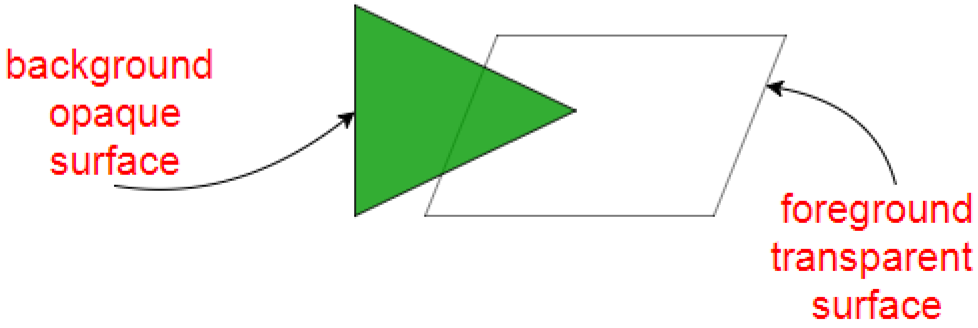
As shown in the above figure, if the value of depth is >= 0, the number stored at that position is the **depth of single surface**overlapping the corresponding pixel area. The 2nd field, i.e, the intensity field then stores the RGB components of the surface color at that point and the percent of pixel coverage.



(b) When a pixel overlaps by multiple surfaces

As shown in the above figure, multiple-surface contributions to the pixel intensity is indicated by depth < 0. The 2nd field, i.e, the intensity field then stores a pointer to a linked list of surface data.

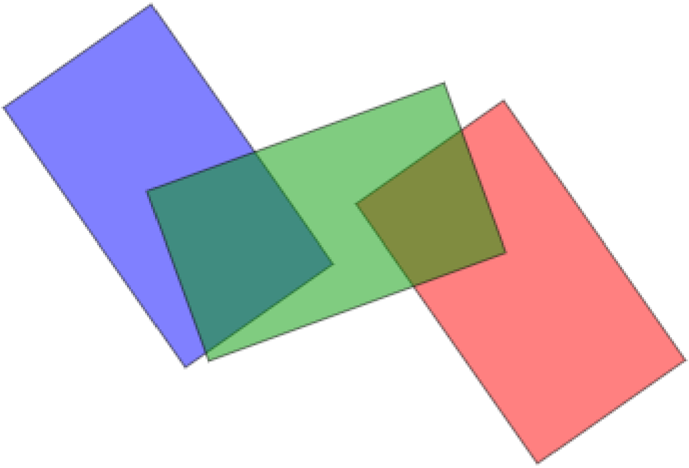
A buffer method is slightly costly than Z-buffer method because it requires more memory in comparison to the Z-buffer method. It proceeds just like the depth buffer algorithm. Here, the depth and opacity are used to determine the final color of the pixel. As shown in the figure below, the A buffer method can be used to show the transparent objects.



The surface buffer in the A buffer method includes :

1. Depth
2. Surface Identifier
3. Opacity Parameter
4. Percent of area coverage
5. RGB intensity components
6. Pointer to the next surface

The other advantage of A buffer method is that it provides anti-aliasing in addition to what Z-buffer does. The usage of A-buffer algorithm for the transparent surfaces is as shown below :



On applying the A-buffer method on all the six surfaces indicated below, the corresponding colors are as :

In A-buffer method, each pixel is made up of a group of sub-pixels. The final color of a pixel is computed by summing up all of its sub-pixels. Due to this accumulation taking place at sub-pixel level, A-buffer method gets the name **accumulation buffer**.

(ii) Depth Buffer method.:-

One of the simplest and commonly used image space approach to eliminate hidden surfaces is the Z-buffer or depth buffer algorithm. It is developed by Catmull. This algorithm compares surface depths at each pixel position on the projection plane. The surface depth is measured from the view plane along the z-axis of a viewing system. When object description is converted to projection co-ordinates (x, y, z), each pixel position on the view plane is specified by x and y coordinate, and z-value gives the depth information. Thus object depths can be compared by comparing the z- values.

The Z-buffer algorithm is usually implemented in the normalized coordinates, so that z-values range from 0 at the back clipping plane to 1 at the front clipping plane. The implementation requites another buffer memory called Z-buffer along with the frame buffer memory required for raster display devices. A Z-buffer is used to store depth values for each (x, y) position as surfaces are processed, and the frame buffer stores the intensity values for each position. At the beginning Z-buffer is initialized to zero, representing the z-value at the back clipping plane, and the frame buffer is initialized to the background colour. Each surface listed in the display file is then processed, one scan line at a time, calculating the depth (z-value) at each (x, y) pixel position. The calculated depth value is compared to the value previously stored in the Z-buffer at that position. If the calculated depth values are greater than the value stored in the Z-buffer, the new depth value is stored, and the surface intensity at that position is determined and placed in the same xy location in the frame buffer.

For example, in figure (e) shown below, among three surfaces, surface S1 has the smallest depth at view position (x, y) and hence highest z value. So it is visible at that position.

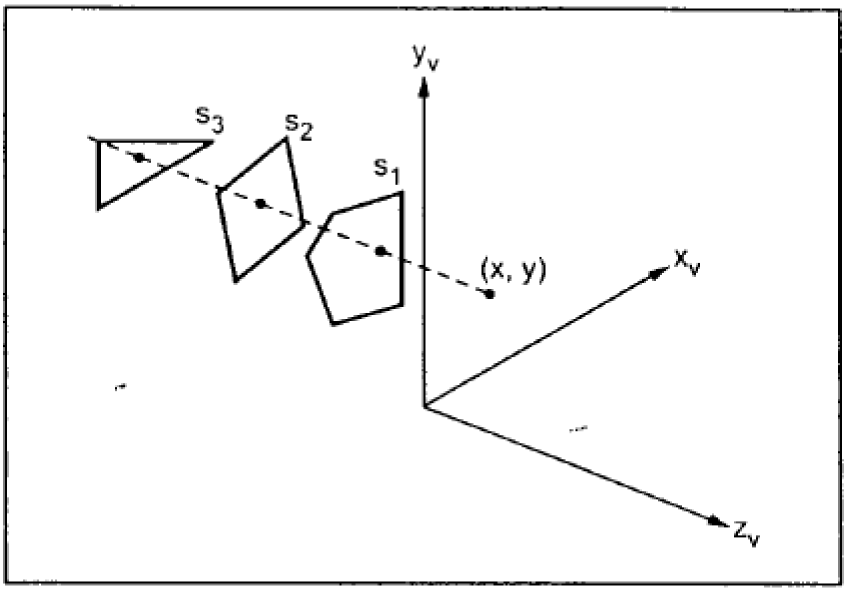


Fig. (e)

Z-buffer Algorithm:-

```
1. Initialize the Z-buffer and frame buffer so that for all buffer positions
   Z-buffer (x. y)  = 0 and frame-buffer (x, y) = Ibackground
2. During scan conversion process, for each position on each polygon surface, compare depth
   values to previously stored values in the depth buffer to determine visibility.
      Calculate z-value for each (x, y) position on the polygon
      If z > Z-buffer (x, y), then set
      Z-buffer (x, y) = z, frame-buffer (x, y) = Isurface(x, y)
3. Stop
```

Note that, Ibackground is the value for the background intensity, and Isurface is the projected intensity value for the surface at pixel position (x, y). After processing of all surfaces, the Z-buffer contains depth values for the visible surfaces and the frame buffer contains the corresponding intensity values for those surfaces.

To calculate z-values, the plane equation

$Ax + By + Cz + D =0$

is used where (x, y, z) is any point on the Plane, and the coefficient A, B, C and D are constants describing the spatial properties of the Plane.

Therefore, we can write

$$z = \frac{-Ax - By - D}{C}$$

Note, if at (x, y) the above equation evaluates to z1, then at (x + Δx, y) the value of z. is,

$$z_1 = \frac{A}{C} (\Delta x)$$

Only one subtraction is needed to calculate z(x + 1. y), given z(x, y). Since the quotient A/C is constant and Δx = 1. A similar incremental calculation can be performed to determine the first value of z on the next scan line, decrementing by B/C for each Δy.

Advantages:-

I. It is easy to implement.

1. It can be implemented in hardware to overcome the speed problem.
2. since the algorithm processes objects one at a time, the total number of polygons in a picture can be arbitrarily large.

Disadvantages:-

I. It requires an additional buffer and hence the large memory.

1. It is a time consuming process as it requires comparison for each pixel instead of for the entire polygon.

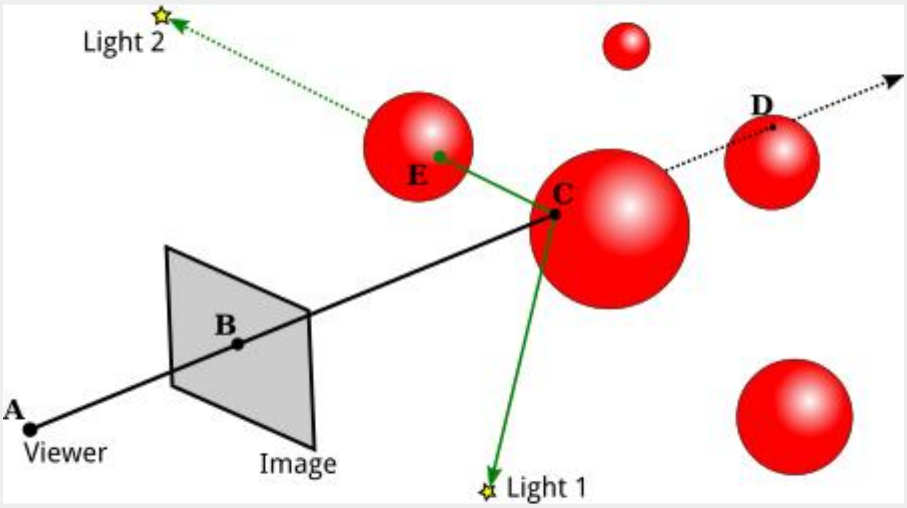
(B) Explain ray-casting method.

Ans:- Ray tracing is probably the best known technique for higher quality graphics. The idea behind it is not complicated: To find out what you see when you look in a given direction, consider a ray of light that arrives at your location from that direction, and follow that light ray backwards to see where it came from. Or, as it is usually phrased, cast a ray from your location in a given direction, and see what it hits. That's what you see when you look in that direction. The operation of determining what is hit by a ray is called ray casting. It is fundamental to ray tracing and to other advanced graphics techniques.

8.1.1 Ray Casting

We have already seen ray casting used by objects of type THREE.RayCaster in the three.js API (Subsection 5.3.2). A Raycastertakes an initial point and a direction, given as a vector. The point and vector determine a ray, that is, a half-infinite line that extends from a starting point, in some direction, to infinity. The Raycaster can find all the intersections of the ray with a given set of objects in a three.js scene, sorted by order of distance from the rays's starting point. In this chapter, we are interested in the first intersection, the one that is closest to the starting point.

To apply ray casting to rendering, let's say that we have a scene consisting of objects in three-dimensional space, using point lights and directional lights for illumination. We want to render an image of the scene from a given point of view. It's convenient to imagine the image as a kind of rectangular window through which the scene is being viewed. Given a point in the image, we need to know how to color that point. To compute a color for the point, we begin by casting a ray from the position of the viewer through the point and into the scene. We want to find the first intersection of that ray with an object in the scene:



In this illustration, the scene contains several red spheres and two point lights. A ray is cast from the viewpoint (A) through a point (B) in the image. The ray intersects two of the spheres, but we are only interested in the intersection point (C) that is closest to the viewpoint. That's the point that is visible at B in the image.

We need to compute the color that the viewer will see at point B. For that, just as in OpenGL, we need a normal vector at point C, and we need the material properties of the surface at C. That data has to be computable from the scene description. The visible color also depends on the light that is illuminating the surface. Consider a light source, and let L be the vector that points from C in the direction of the light. If the angle between L and the normal vector is greater than 90 degrees, then the light source lies behind the surface and so does not add any illumination. Otherwise, we can use ray casting again: Cast a ray from C in the direction L. If that ray hits an object before it gets to the light, then that object will block light from that source from reaching C. That's the case for Light 2 in the illustration: The ray from C in the direction of Light 2 intersects an object at point E before it gets to the light. On the other hand, the point C is illuminated by Light 1. A ray from a point on a surface in the direction of a light source is called a shadow ray, because it can be used to determine whether the surface point lies in the shadow of another object.

With all this information, we can color point B in the image using the same lighting equation that is used in OpenGL (Subsection 4.1.4). And, as a bonus, we get shadows, which are hard to do in OpenGL!

(If the ray from the viewpoint through B doesn't hit any objects in the scene, then B would be assigned a background color or a "sky" color that varies with different directions. Or maybe the entire scene is surrounded by a skybox, and a ray that doesn't hit any other object will hit the skybox.)

Ray casting is conceptually simple, but implementation details can be tricky. Spheres are actually fairly easy. There is a formula for finding the intersections of a line with a sphere, and a normal vector at a point on a sphere has the same direction as the line from the center of the sphere to that point. Assuming that the sphere has a uniform material, we have all the data we need.

However, surfaces are often given as triangular meshes, with properties specified only at the vertices of the triangles. Suppose that the intersection point found by our ray caster lies in one of those triangles. We will have to compute the properties of that intersection point by interpolating the property values that were specified at the vertices of the triangle.

The interpolation algorithm typically uses something called barycentric coordinates on the triangle: If A, B, and C are the vertices of a triangle, and P is a point in the triangle, then P can be written uniquely in the form a*A + b*B + c*C, where a, b, and care numbers in the range zero to one, and a + b + c = 1. The coefficients a, b, and c are called the barycentric coordinates of the point P in the triangle. If some quantity has values V(A), V(B), and V(C) at the vertices of the triangle, then we can use the barycentric coordinates of P to get an interpolated value at P:

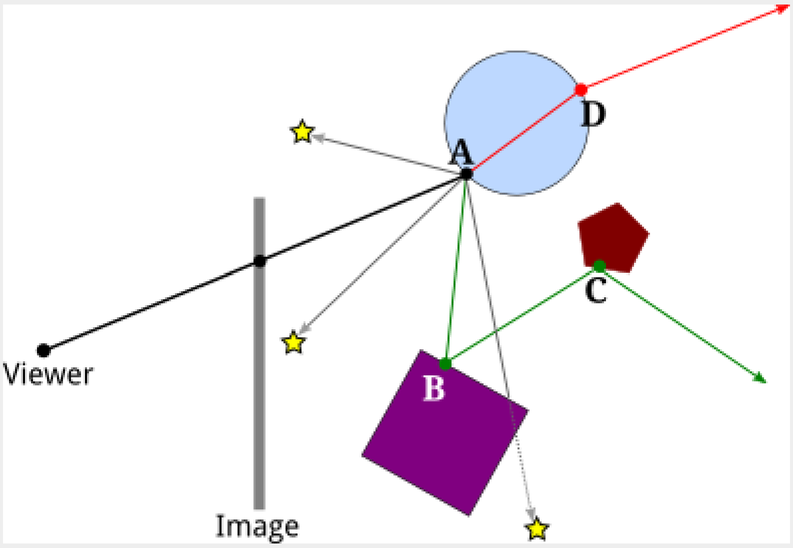
V(P) = a*V(A) + b*V(B) + c*V(C)

The quantity might be, for example, diffuse color, texture coordinates, or a normal vector. (Of course, I'm still leaving out a lot of the math here—how to test whether a line intersects a triangle and how to find barycentric coordinates of the point of intersection. There are formulas, but conceptually, they wouldn't add much to the discussion.)

8.1.2 Recursive Ray Tracing

Basic ray casting can be used to compute OpenGL-style rendering and, with the addition of shadow rays, to implement shadows as well. More features can be implemented by casting a few more rays. The improved algorithm is called ray tracing.

Consider specular reflection. In OpenGL, specular reflection can make an object look shiny in the sense that specular highlights can be seen where the object reflects light from a light source towards the viewer. But in reality, an object that has a mirror-like surface doesn't just reflect light sources; it also reflects other objects. If we are trying to compute a color for a point, A, on a mirror-like surface, we need to consider the contribution to that color from mirror-like reflection of other objects. To do that, we can cast a "reflected ray" from A. The direction of the reflected ray is determined by the normal vector to the surface at A and by the direction from A to the viewer. This illustration shows a 2D version. Think of it as a cross-section of the situation in 3D:



Here, the reflected ray from point A hits the purple square at point B, and the viewer will see a reflection of point B at A. (Remember that in ray tracing, we follow the path of light rays backwards from the viewer, to find out where they came from.)

To find out what the reflection of B looks like, we need to know the color of the the ray that arrives at A from B. But finding a color for B is the same sort of problem as finding a color for A, and we should solve it in the same way: by applying the ray-tracing algorithm to B! That is, we use the material properties of the surface at B, we cast shadow rays from B towards light sources to determine how B is illuminated, and—if the purple square has a mirror-like surface—we cast a reflected ray from B to find out what it reflects. In the illustration, the reflected ray from B hits a pentagon at point C, so the square reflects an image of the pentagon, and the disk reflects an image of the square, including its reflection of the pentagon. Ray-tracing can handle multiple mirror-like reflections between objects in a scene!

Because applying the ray-tracing algorithm at one point can involve applying the same algorithm at additional points, ray tracing is a recursive algorithm. To distinguish this from simple ray casting, ray tracing is often referred to as "recursive ray tracing."

Ray tracing can be extended in a similar way to handle transparency or, more properly, translucency. When computing a color for a point on a translucent object, we need to take into account light that arrives at that point through the object. To do that, we can cast yet another ray from that point, this time into the object. When a light ray passes from one medium, such as air, into another medium, such as glass, the path of the light ray can bend. This bending is called refraction, and the ray that is cast through a translucent object is called the "refracted ray." The above illustration shows the refracted ray from point A passing through the object and emerging from the object at D. To find a color for that ray, we would need to find out what, if anything, it hits, and we would need to apply ray tracing recursively at the point of intersection.

(The degree of bending of a light ray that passes from one medium to another depends on a property of the two media called the "index of refraction." More exactly, it depends on the ratio between the two indices of refraction. In practice, the index of refraction outside objects is often taken to be equal to one, so that the bending depends only on the index of refraction of an object. The index of refraction is another material property for translucent objects. It is often abbreviated IOR.)

We should look at the computations in a little more detail. The goal is to compute a color for a point on an image. We cast a ray from the viewpoint through the image and into the scene, and determine the first intersection point of the ray with an object. The color of that point is computed by adding up the contributions from different sources.

First, there are diffuse, specular, and possibly ambient reflection of light from various light sources. These contributions are based on the diffuse, specular, and ambient colors of the object, on the normal vector to the object, and on the properties of the light sources. Note that some color properties of the object, usually the ambient and diffuse colors, might come from a texture. The specular contribution can produce specular highlights, which are essentially the reflections of light sources. Shadow rays are used to determine which directional and point lights illuminate the object; aside from that, this part of the calculation is similar to OpenGL.

Next, if the surface has mirror-like reflection, then a reflected ray is cast and ray tracing is applied recursively to find a color for that ray. The contribution from that ray is combined with other contributions to the color of the surface, depending on the strength of the mirror reflection. For a perfect mirror, the reflected ray contributes 100% of the color, but in general the contribution is less. Mirror reflectivity is a new material property. It is responsible for reflections of one object on the surface of another, while specular color is responsible for specular highlights.

Finally, if the object is translucent, then a refracted ray is cast, and ray tracing is used to find its color. The contribution of that ray to the color of the object depends on the degree of transparency of the object, since some of the light can be absorbed rather than transmitted. The degree of transparency can depend on the wavelength of the light—as it does, for example, in colored glass.

And, of course, all of these calculations need to be done three times, for the red, the green, and the blue components of the color.

The ray tracing algorithm is recursive, and, as every programmer knows, recursion needs a base case. That is, there has to come a time when, instead of calling itself, the algorithm simply returns a value. A base case occurs whenever a casted ray does not intersect any objects. Another kind of base case can occur when it is determined that casting more rays cannot contribute any significant amount to the color of the image. For example, whenever a ray is reflected, some of that ray is absorbed rather than reflected, depending on the color of the object from which it is reflected. After reflecting many times, a ray would have very little color left to contribute to the final result. A ray can also lose energy because of attenuation of light with distance, and a ray-tracing algorithm might take that into account. In addition, a ray tracing algorithm should always be run with a maximum recursion depth, to put an absolute limit on the number of times the algorithm will call itself.

8.1.3 Limitations of Ray Tracing

Although ray tracing can produce very realistic images, there are some things that it can't do. For example, while ray tracing works well for point lights and directional lights, it can't handle area lights. An area light is one that has area. That is, it is an object that emits light from its entire surface area rather than from a single point. Of course, real lights are area lights. A fluorescent light emits light from the surface of a cylinder. A brightly lit window can be considered to be a kind of area light. Even a light bulb does not really radiate light from a single point. Ray tracing uses shadow rays to tell whether a light source illuminates an object. But a shadow ray is cast in only one direction, and can only hit one point on an area light. To implement area lights exactly, a different shadow ray would be needed for each point on the surface of the light.

A ray tracer can handle area lights in an approximate way, by replacing the area light with a grid of point lights. It can then cast a shadow ray towards each point in the grid. Using more point lights in the grid will give a better approximation. Of course, casting all those shadow rays can add significantly to the computation time.

Another problem with lighting in ray tracing is that it doesn't take into account illumination by reflected light. For example, light from a light source should reflect off a mirror, and the reflected light should illuminate other objects. Ray tracing uses shadow rays to tell whether a light source illuminates an object. But that won't work for reflected light, since the algorithm doesn't know where to aim the shadow ray—reflected light could come from any direction.

This is not just a problem with mirrors. Any reflected light, even from diffuse reflection, should illuminate nearby objects. For example, light that is reflected diffusely from a green object should add a green tint to nearby objects. This effect is called "color bleeding." In reality, light can be reflected and re-reflected many times, contributing a bit of color to each object that it hits. As with area lights, approximate methods can be added to ray tracing to simulate this effect. For example, "photon mapping" adds a pre-processing phase to ray tracing which simulates the emission of a large number of light rays, or "photons," from light sources, and tracks their paths through the scene to see how they add color to the objects that they hit. The information from this "photon map" is then used during the ray tracing phase to produce more realistic colors.

OpenGL uses ambient light as an approximation for light that has been reflected and re-reflected many times. Ambient light is assumed to illuminate every object equally. However, that is a poor approximation. A better approximation uses ambient occlusion, the idea that ambient light heading towards a surface can be blocked, or "occluded," by nearby objects. Like ambient light, ambient occlusion is not physically realistic, but in practice, it can make lighting look more realistic and objects look more three-dimensional. One technique for estimating ambient occlusion uses ray casting. We assume that the ambient light comes from the background of the scene. To estimate ambient occlusion at a point, cast a number of rays from that point in random directions, and count how many of those rays are blocked by geometry in the scene and how many reach the sky. The more rays that are blocked, the greater the degree of ambient occlusion at that point.

Explain Scan line method in detail.

Ans:- A scan line method of hidden surface removal is an another approach of image space method. It is an extension of the scan line algorithm for filling polygon interiors. Here, the algorithm deals with more than one surfaces. As each scan line is processed, it examines all polygon surfaces intersecting that line to determine which are visible. It then does the depth calculation and finds which polygon is nearest to the view plane. Finally, it enters the intensity value of the nearest polygon at that position into the frame buffer.

The scan line algorithm maintains the edge list in the edge table(ET). The AET(Active Edge Table) contains only edges that cross the current scan Line, sorted in order of increasing x. The scan line method of hidden surface removal also stores a flag for each surface that is set on or off to indicate whether a position along a scan line is inside or outside of the surface. Scan lines are processed from left to right. At the leftmost boundary of a surface, the surface flag is turned ON; and at the rightmost boundary, it is turned OFF.

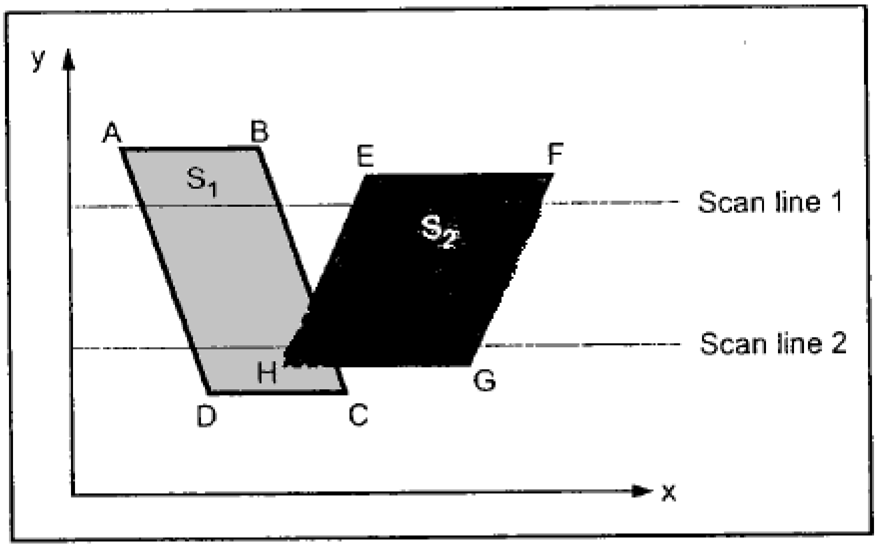


Fig. (f) Illustration of scan line method of hidden surface removal

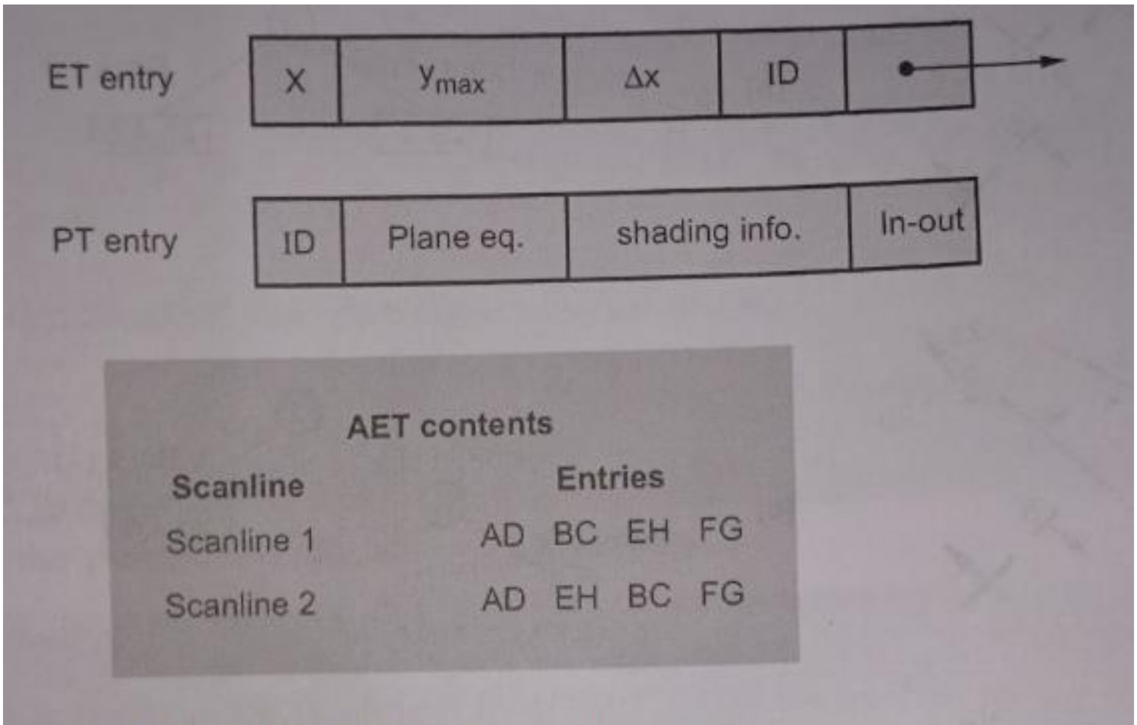
The figure (f) illustrates the scan line method for hidden surface removal. As shown in the Fig. (f), the active edge list for scan line 1 contains the information for edges AD, BC, EH and FG. for the positions along this scan line between edges AD and BC, only the flag for surface S1 is ON. Therefore, no depth calculations are necessary, and intensity information for surface S1 is entered into the frame buffer. Similarly, between edges EH and FG, only the flag for surface S2 is ON and during that portion of scan line the intensity information for surface S2, is entered into the frame buffer.

For scan line 2 in the figure (f), the active edge list contains edges AD, EH, BC and FG. Along the scan line 2 from edge AD to edge EH, only the flag for surfaces S1 is ON. However, between edges EH and BC, the flags for both surfaces are ON. In this portion of scan line 2, the depth calculations are necessary. Here we have assumed that the depth of S1 is less than the depth of S2 and hence the intensities of surface S1 are loaded into the frame buffer. Then, for edge BC to edge FG portion of scan line 2 intensities of surface S2 are entered into the frame buffer because during that portion only flag for S2 is ON.

To implement this algorithm along with AET we are required to maintain a polygon table (PT) that contains at least the following information for each polygon, in addition to ID.

1. The coefficient of the plane equation.
2. Shading or colour information for the polygon.
3. A in-out Boolean flag, initialized to false and used during scan line processing.

Fig. (g) Shows the ET, PT and AET for the scan line algorithm.



AET contents

Scanline	Entries
Scanline 1	AD BC EH FG
Scanline 2	AD EH BC FG

Fig. (g) ET, PT , AET, for the scan line algorithm.