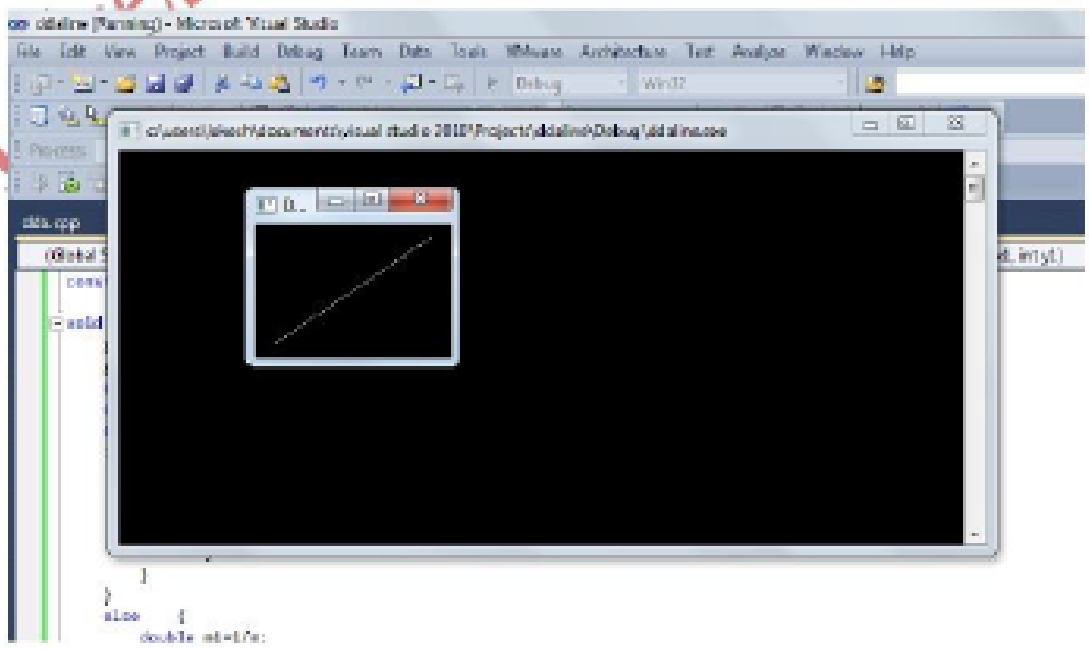
Q.1. Write a program in C/C++ to generate line segment between two points, by using DDA line generation Algorithm. Your program should map each and every step of pseudo algorithm, in the form of comments. What are the advantages and disadvantages of using DDA algorithm for line generation? List the steps of the algorithm. Use this algorithm to draw a line with endpoints (2, 3) and (9, 8). Ans:-

```
#include <windows.h>        // Header File For The Windows Library
#include <gl/glut.h>        // Header File For The OpenGL32 Library
#include <math.h>           // Header File For The Math Library
#include <stdio.h>          // Header File For Standard Input/Output
const float PI=3.14;

void drawLine(int x0,int y0,int x1,int y1){        // input the line endpoint and
    glBegin(GL_POINTS);                            store the left endpoint in (x0, y0)
    glColor3f(1.0,1.0,1.0);                         and right endpoint (x1,y1)
    double m=(double)(y1-y0)/(x1-x0);              // calculate the values of Δx and Δy
    double y=(double)y0;                            using Δx = x1 - x0, Δy = y1 - y0
    double x=(double)x0;
    if(m<1)        {                               // if the value of Δx <= Δx1 assign values of
        while(x<=x1)            {                  steps as Δx otherwise the values of steps as Δy

            glVertex2d(x,floor(y));
            printf("%f %f\n",floor(y),x);
            y=y+m;                                 // calculate the values of x and y increment
            x++;                                      and assign the value x++ and y=y+m
        }
    }
```

```
        else        {
                double m1=1/m;
                while(y<=y1)                {                       // while (y <= y1) compute the pixel and
                        glVertex2d(floor(x),y);                     plot the pixel with y increment and x=x+m1
                        y++;
                        x=x+m1;
                }
        }
        glEnd();
}

void init(void){                                        // init() method to set the color and projection
        glClearColor(0.0,0.0,0.0,0.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0.0,100.0,0.0,100,0.0,100.0);
}
void display(void)                                      // method to draw the line
{
        glClear(GL_COLOR_BUFFER_BIT);
        drawLine(10,10,90,90);                          // value of the line according to coordinates
        glutSwapBuffers();
}

int main(int argc, char** argv){                        // main function
        glutInit(&argc,argv);                           // calling glutInit() method
        glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB); // calling gultInitDisplayMode() method
        glutInitWindowSize(100,100);                    // Setting of the windows size
        glutInitWindowPosition(200,100);                // setting windows position
        glutCreateWindow("DDA Line Drawing!");          // creating windows header name
        init();                                         // calling init() method
        glutDisplayFunc(display);                       // calling display method to draw the line
        glutMainLoop();
        return 0;
}
```

Output:



Advantages of using DDA algorithm for line generation:

(1) It is the simplest algorithm and it does not require special skills for implementation. DDA uses float numbers and uses operators such as division and multiplication in its calculation. Bresenhams algorithm uses ints and only uses addition and subtraction.
(2) It is a faster method for calculating pixel positions than the direct use of equation y = mx + b. It eliminates the multiplication in the equation by making use of raster characteristics, so that appropriate increments are applied in the x or v direction to find the pixel positions along the line path.
(3) Fixed point DDA algorithms are generally superior to Bresenhams algoritm on modern computers. The reason is that Bresenhams algoritm uses a conditional branch in the3 loop and this result in frequent branch mis-predictions in the CPU.
(4) Fixed point DDA also has fewer instructions in the loop body (one bit shift, one increment and one addition to be exact. In addition to the loop instructions and the actual plotting. As CPU pipelines become deeper mis predictions penalties will become more severe.
(5) It Due to the use of only addition, subtraction and bit shifting Bresenhams algorithm is faster than DDA in producing the line.
(6) Fixed point DDA does not require conditional jumps, you can compute several lines in parallel with SIMD (Single Instruction Multiple Data techniques.

Disadvantages of using DDA algorithm for line generation:

(1) The accumulation of round of error is successive addition of the floating point increments is used to find the pixel position but it take lot of time to compute the pixel position.
(2) The disadvantage of such a simple algorithm is that it is meant for basic line drawing. The "advanced" topic of antialiasing isn't part of Bresenham's algorithm, so to draw smooth lines, you'd want to look into a different algorithm.
(3) Floating point arithmetic in DDA algorithm is still time-consuming.
(4) The algorithm is orientation dependent. Hence end point accuracy is poor.

List the steps of the algorithm. Use this algorithm to draw a line with endpoints (2, 3) and (9, 8).

We know that the general equation of the line is given below:

y=mx+c where m=(y1-y0)/(x1-x0)
Given (x0,y0)→(2,3); (x1,y1) →(9,8)

⟹ m=(8-3)/(9-2) =5/7
c = y1-mx1 = 8-(5/7)*9=11/7

So by equation of line (y=mx=c) we have…
Y=(5/7)x+(11/7)

DDA Algorithm Two case:

Case 1: $m < 1$

$$x_{i+1} = x_i + 1$$
$$y_{i+1} = y_i + m$$

Case 2: $m > 1$

$$x_{i+1} = x_i + (1/m)$$
$$y_{i+1} = y_i + 1$$

As $0 < m < 1$ so according to DDA algorithm case 1

$$x_{i+1} = x_i + 1 \qquad y_{i+1} = y_i + m$$

Given (x0,y0)=(2,3)

1) X1=x0+1 = 2+1 = 3
   Y1=y0+m= 3+(5/7)=26/7

   Put pixel (x0, round y, Color) i.e.., put (3,5)

2) X2=x1+1 = 3+1 = 4
   Y2=y1+m= 26/7+5/7=31/7

   Put pixel (x0, round y, Color) i.e.., put (4,5)

   Similarly go on till (9,8) reached.

Q.2. Write the Bresenham Circle Generation Algorithm and use it to draw a circle with radius r = 10, determine positions along the circle octants in 1st Quadrant from x = 0 to x = y. (10 Marks)

Ans:-

Midpoint Circle Algorithm

(a) Input radius r and circle, center (x_c, y_c) and obtain the first point on the circumference of the circle centered on origin as

$$(x_0, y_0) = (0, r)$$

(b) Calculate initial value of decision parameter as

$$p_0 = \frac{5}{4} - r \sim 1 - r$$

(c) At each x_k position starting at $k = 0$ perform following test.

1) If $p_k < 0$ then next point along the circle centered on (0, 0) is (x_{k+1}, y_k) and $p_{k+1} = p_k + 2x_{k+1} + 1$

2) Else the next point along circle is $(x_{k+1}, y_k - 1)$ and

$$p_{k+1} = p_k + 2x_{k+1} - 2y_{k-1}$$

where $2x_{k+1} = 2(x_k + 1) = 2x_k + 2$ and $2y_{k-1} = 2(y_k - 1) = 2y_k - 2$

(d) Determine symmetry points in the other seven octants.

(e) Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c) and plot coordinate values $x = x + x_c, y = y + y_c$

(f) Repeat step (c) through (e) until $x \geq y$.

Solution: An initial decision parameter $p_0 = 1 - r = 1 - 10 = -9$

For circle centered on coordinate origin the initial point $(x_0, y_0) = (0, 10)$ and initial increment for calculating decision parameter are:

$$2x_0 = 0, 2y_0 = 20$$

Using mid point algorithm point are:

k	p_k	(x_{k+1}, y_{k-1})	$2x_{k+1}$	$2y_{k-1}$
0	– 9	(1, 10)	2	20
1	– 6	(2, 10)	4	20
2	– 1	(3, 10)	6	20
3	6	(4, 9)	8	18
4	– 3	(5, 9)	10	18
5	8	(6, 8)	12	16
6	5	(7, 7)	14	14

Q.3. What is line clipping? Compare Cohen Sutherland Line Clipping Algorithm with the Cyrus Beck line clipping algorithm. Explain the Cyrus Beck line clipping algorithm with the help of an example. How Cyrus Back line clipping algorithm, clips a line segment, if the window is non-convex? (10 Marks)

Ans:- Clipping: Clipping may be described as the procedure that identifies the portions of a picture lie inside the region, and therefore, should be drawn or, outside the specified region, and hence, not to be drawn. The algorithms that perform the job of clipping are called clipping algorithms there are various types, such as:

- Point Clipping
- Line Clipping
- Polygon Clipping
- Text Clipping
- Curve Clipping

Line clipping: In computer graphics, 'line clipping' is the process of removing lines or portions of lines outside of an area of interest. Typically, any line or part thereof which is outside of the viewing area is removed.

Line is a series of infinite number of points, where no two points have space in between them. So, the above said inequality also holds for every point on the line to be clipped. A variety of line clipping algorithms are available in the world of computer graphics, but we restrict our discussion to the following Line clipping algorithms, name after their respective developers:

1) Cohen Sutherland algorithm
2) Cyrus-Beck of algorithm

Compare Cohen Sutherland Line Clipping Algorithm with the Cyrus Beck line clipping algorithm:

Cohen Sutherland Line Clippings: This algorithm is quite interesting. The clipping problem is simplified by dividing the area surrounding the window region into four segments Up, Down, Left, Right (U,D,L,R) and assignment of number 1 and 0 to respective segments helps in positioning the region surrounding the window. How this positioning of regions is performed can be well understood by considering Figure 3.

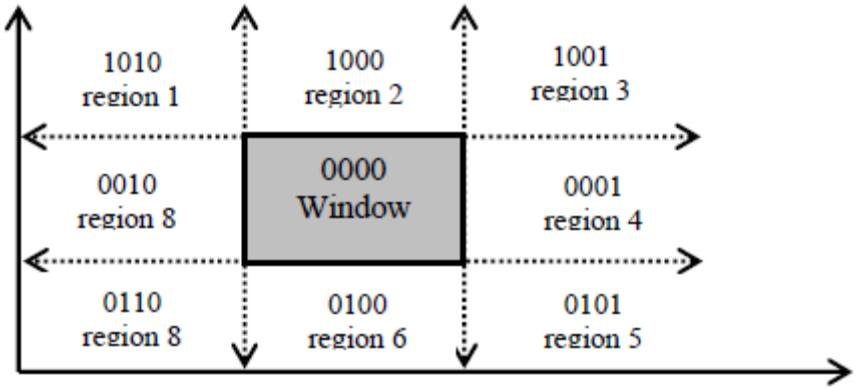
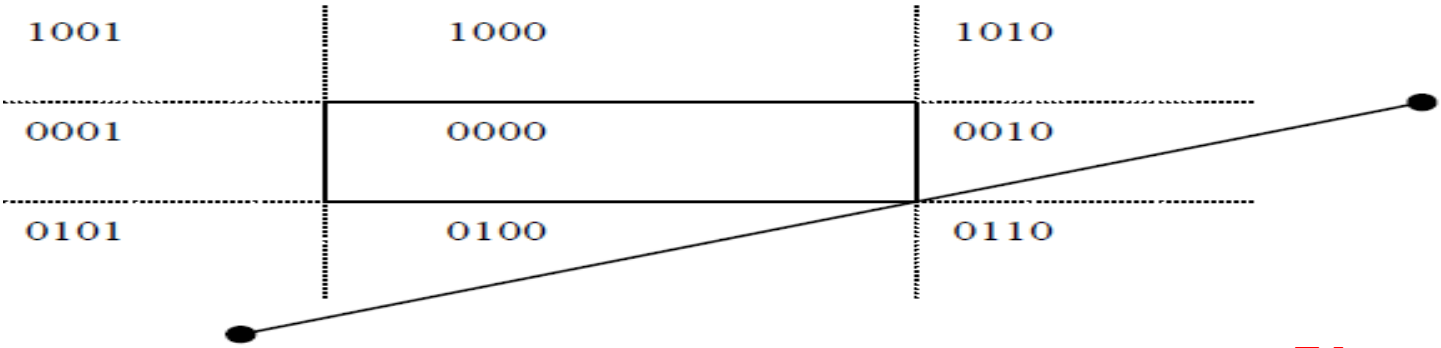


Figure 3: Positioning of regions surrounding the window

1) Clipping window region can be rectangular in shape only and no other polygonal shaped window is allowed.
2) Edges of rectangular shaped clipping window has to be parallel to the x-axis and y-axis.
3) If end pts of line segment lies in the extreme limits i.e., one at R.H.S other at L.H.S., and on one the at top and other at the bottom (diagonally) then, even if the line doesn't pass through the clipping region it will have logical intersection of 0000 implying that line segment will be clipped but infact it is not so.



Cyrus Beck line clipping algorithm: Cyrus Beck Line clipping algorithm is infact, a parametric line-clipping algorithm. The term parametric implies that we need to find the value of the parameter t in the parametric representation of the line segment for the point at which the segment intersects the clipping edge. For better understanding, consider the Figure 9(a), where P Q is a line segment, which is intersecting at the two edges of the convex window.

The Cyrus–Beck algorithm is a generalized line clipping algorithm. It was designed to be more efficient than the Sutherland–Cohen algorithm which uses repetitive clipping.[1] Cyrus–Beck is a general algorithm and can be used with a convex polygon clipping window unlike Sutherland-Cohen that can be used only on a rectangular clipping area.

Here the parametric equation of a line in the view plane is:

$$p(t) = tp_1 + (1-t)p_0$$

$$= p_0 + t(p_1 - p_0)$$

where $0 \leq t \leq 1$.

Now to find intersection point with the clipping window we calculate value of dot product. Let pE be a point on the clipping plane E.

Calculate $n \cdot (p(t) - p_E)$.

if > 0 vector pointed towards interior

if = 0 vector pointed parallel to plane containing p

if < 0 vector pointed away from interior

Here n stands for normal of the current clipping plane (pointed away from interior).

By this we select the point of intersection of line and clipping window where (dot product = 0) and hence clip the line.

Note: The algorithm is applicable to the “convex polygonal window”.

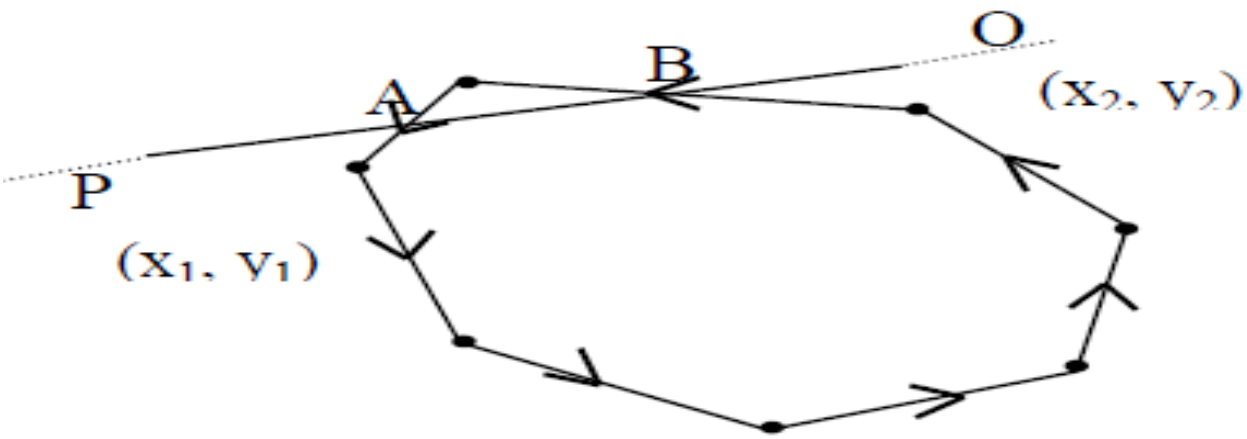


Figure 9(a): Interaction of line PQ and Window

Now, just recall the parametric equation of line segment PQ, which we have studied in the course CS-60.

It is simply $P + t(Q - P)$ $0 \leq t \leq 1$

Where, t → linear parameter continuously changes value.

$\therefore P + t(Q - P) \Rightarrow (x1, y1) + t(x2 - x1, y2 - y1) = (x, y)$ be any point on PQ. ------ (1)

For this equation (1) we have following cases:

1) When t = 0 we obtain the point P.
2) When t = 1 we get the point Q.
3) When t varies such that $0 \leq t \leq 1$ then line between point P and Q is traced. For t = ½ we get the mid-point of PQ.
4) When t < 0 line on LHS of P is traced.
5) When t > 1 line on RHS of Q is traced.

So, the variation in parameter t is actually generating line in point wise manner .The range of the parameter values will identify the portion to be clipped by any convex polygonal region having n-vertices or lattice points to be specified by the user. One such clipping situation is shown in Figure 9(a).

Cyrus Back line clipping algorithm, clips a line segment, if the window is non-convex?

Solution: Consider the *Figure 13*, here, the window is non convex in shape and PQ is a line segment passing through this window .Here too the condition of visibility of the line is $t_{max} < t_{min}$ and the line is visible from $P + t_{max}(Q - P)$ to $P + t_{min}(Q - P)$, if $t_{max} \not< t_{min}$ then reject the line segment. Now, applying this rule to the *Figure 13,* we find that when PQ line segment passes through the non convex window, it cuts the edges of the window at 4 points. $1 \rightarrow P_E$; $2 \rightarrow P_L$; $3 \rightarrow P_E$; $4 \rightarrow P_L$. In this example, using the algorithm we reject the line segment PQ but it is not the correct result.

Condition of visibility is satisfied in region 1-2 and 3-4 only so the line exists there but in region 2-3 the condition is violated so the line does not exists.

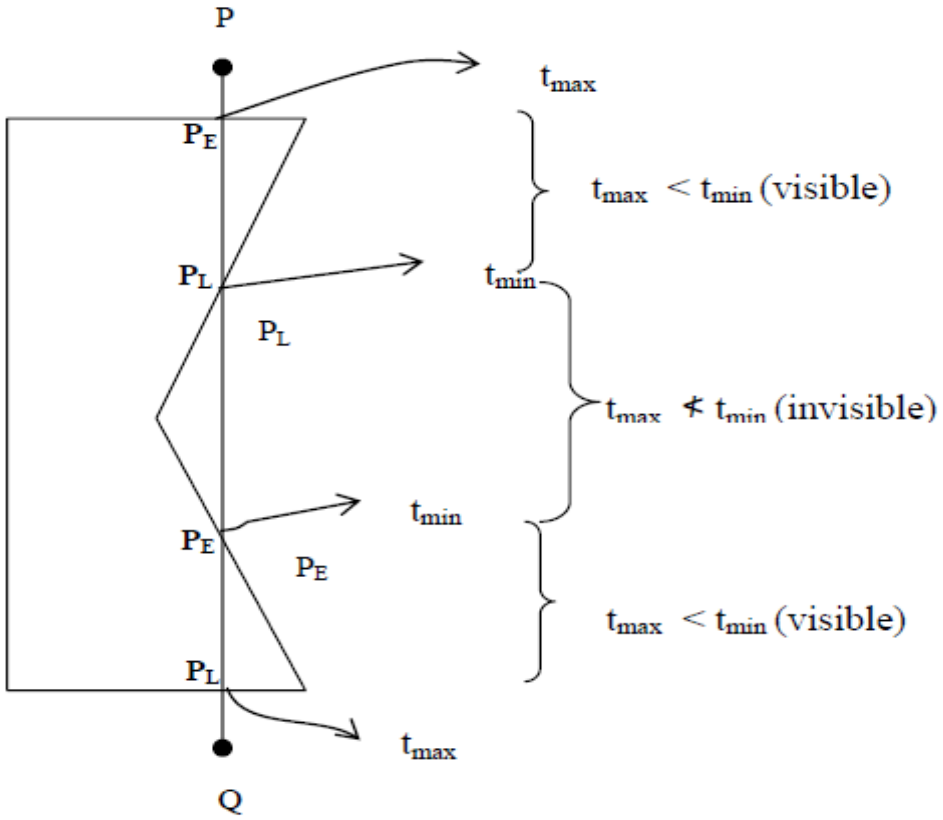


Figure 13: Example Cyrus Beck Clipping

Q.4. Explain the Homogeneous Coordinate System with the help of an example. Assume that a triangle ABC has the coordinates A(0, 0), B(5,8), C(4,2). Find the transformed coordinates when the triangle ABC is subjected to the clockwise rotation of 45° about the origin and then translation in the direction of vector (1, 0). You should represent the transformation using Homogeneous Coordinate System.

Ans:- Homogeneous Coordinate System: In mathematics, homogeneous coordinates or projective coordinates, introduced by August Ferdinand Möbius in his 1827 work *Der barycentrische Calcül*,[1][2] are a system of coordinates used in projective geometry, as Cartesian coordinatesare used in Euclidean geometry. They have the advantage that the coordinates of points, including points at infinity, can be represented using finite coordinates. Formulas involving homogeneous coordinates are often simpler and more symmetric than their Cartesian counterparts. Homogeneous coordinates have a range of applications, including computer graphics and 3D computer vision, where they allow affine transformations and, in general, projective transformations to be easily represented by a matrix.

Let P(x,y) be any point in 2-D Euclidean (Cartesian) system.

In Homogeneous Coordinate system, we add a third coordinate to a point. Instead of (x,y), each point is represented by a triple (x,y,H) such that H≠0; with the condition that (x1,y1,H1)=(x2,y2,H2) ↔ x1/H1 = x2/H2 ; y1/H1 = y2/H2.

(Here, if we take H=0, then we have point at infinity, i.e., generation of horizons).

Thus, (2,3,6) and (4,6,12) are the same points are represented by different coordinate triples, i.e., each point has many different Homogeneous Coordinate representation.

2-D Euclidian System	Homogeneous Coordinate System
Any point (x,y) ⟶	(x,y,1)
	If (x,y,H) be any point in HCS(such that H≠0); Then (x,y,H)=(x/H,y/H,1)
(x/H,y/H) ⟵	(x,y,H)

Now, we are in the position to construct the matrix form for the translation with the use of homogeneous coordinates.

For translation transformation (x,y)□(x+tx,y+ty) in Euclidian system, where tx and ty are the translation factor in x and y direction, respectively. Unfortunately, this way of describing translation does not use a matrix, so it cannot be combined with other transformations by simple matrix multiplication. Such a combination would be desirable; for example, we have seen that rotation about an arbitrary point can be done by a translation, a rotation, and another translation. We would like to be able to combine these three transformations into a single transformation for the sake of efficiency and elegance. One way of doing this is to use homogeneous coordinates. In homogeneous coordinates we use 3x3 matrices instead of 2x2, introducing an additional dummy coordinate H. Instead of (x,y), each point is represented by a triple (x,y,H) such that H≠0; In two dimensions the value of H is usually kept at 1 for simplicity.
Thus, in HCS (x,y,1) → (x+tx,y+ty,1), now, we can express this in matrix form as:

$$(x',y',1)=(x,y,1)\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ t_x & t_y & 1\end{pmatrix}$$

The advantage of introducing the matrix form of translation is that it simplifies the operations on complex objects, i.e., we can now build complex transformations by multiplying the basic matrix transformations.

In other words, we can say, that a sequence of transformation matrices can be concatenated into a single matrix. This is an effective procedure as it reduces the computation because instead of applying initial coordinate position of an object to each transformation matrix, we can obtain the final transformed position of an object by applying composite matrix to the initial coordinate position of an object. Matrix representation is standard method of implementing transformations in computer graphics.

Thus, from the point of view of matrix multiplication, with the matrix of translation, the other basic transformations such as scaling, rotation, reflection, etc., can also be expressed as 3x3 homogeneous coordinate matrices. This can be accomplished by augmenting the 2x2 matrices with a third row (0,0,x) and a third column. That is

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Example 9: Show that the order in which transformations are performed is importan by applying the transformation of the triangle ABC by:

(i) Rotating by 45° about the origin and then translating in the direction of the vecto (1,0), and

(ii) Translating first in the direction of the vector (1,0), and then rotating by 45° abou the origin, where A = (1, 0) B = (0 ,1) and C = (1, 1).

Solution: We can represent the given triangle, as shown in *Figure (a)*, in terms of Homogeneous coordinates as:

$$[ABC] = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

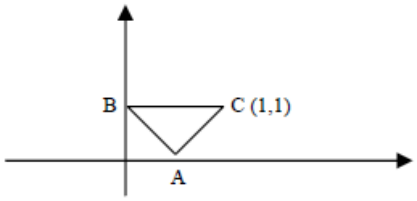


Figure (a)

Suppose the rotation is made in the counter clockwise direction. Then, the transformation matrix for rotation, $R_{45°}$, in terms of homogeneous coordinate syste is given by:

$$R_{45°} = \begin{bmatrix} \text{Cos}\,45° & \text{Sin}\,45° & 0 \\ -\text{Sin}\,45° & \text{Cos}\,45° & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and the Translation matrix, T_v, where V = 1I + 0J is:

$$T_v = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

where t_x and t_y is the translation factors in the x and y directions respectively.

i) Now the rotation followed by translation can be computed as:

$$R_{45°} \,.\, T_v = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

So the new coordinates A′B′C′ of a given triangle ABC can be found as:

$$[A'B'C'] = [ABC] \,.\, R_{45°} \,.\, T_v$$

$$= \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} . \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \left(1/\sqrt{2}+1\right) & 1/\sqrt{2} & 1 \\ \left(-1/\sqrt{2}+1\right) & 1/\sqrt{2} & 1 \\ 1 & \sqrt{2} & 1 \end{bmatrix} \qquad \text{(I)}$$

implies that the given triangle A(1,0), B (0, 1) C (1, 1) be transformed into $A'\left(\frac{1}{\sqrt{2}}+1, \frac{1}{\sqrt{2}}\right)$, $B'\left(\frac{-1}{\sqrt{2}}+1, \frac{1}{\sqrt{2}}\right)$ and $C'\left(1, \sqrt{2}\right)$, respectively, as shown in *Figure (b)*.

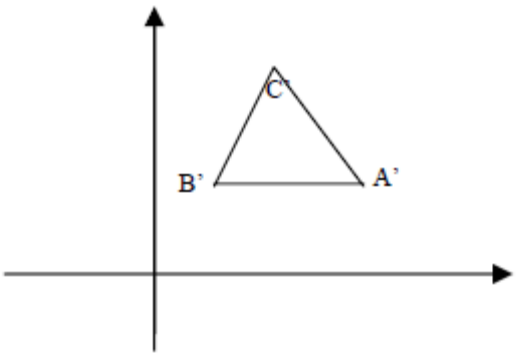


Figure (b)

Similarly, we can obtain the translation followed by rotation transformation as:

$$T_v \,.\, R_{45^\circ} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 1 \end{bmatrix}$$

And hence, the new coordinates $A'B'C'$ can be computed as:

$$[A'B'C'] = [ABC].T_v \; R_{45^\circ}$$

$$= \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} . \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 1 \end{bmatrix} = \begin{bmatrix} 2/\sqrt{2} & 2/\sqrt{2} & 1 \\ 0 & 2/\sqrt{2} & 1 \\ 1/\sqrt{2} & 3/\sqrt{2} & 1 \end{bmatrix} \qquad \text{(II)}$$

Thus, in this case, the given triangle A(1,0), B(0, 1) and C(1,1) are transformed into $A''\left(2/\sqrt{2}, 2/\sqrt{2}\right)$, $B''\left(0, 2/\sqrt{2}\right)$ *and* $C''\left(\frac{1}{\sqrt{2}}, \frac{3}{\sqrt{2}}\right)$, respectively, as shown in *Figure (c).*

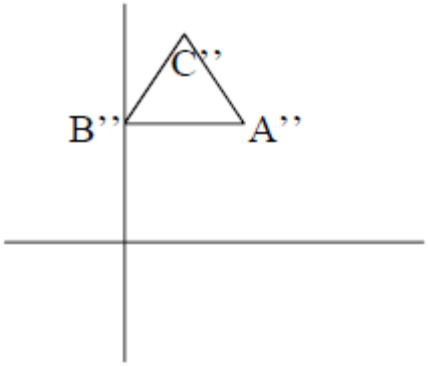


Figure (c)

By (I) and (II), we see that the two transformations do not commute.

Q.5. What are Bezier Curves? Briefly discuss their properties, with the help of the proof for each of the property. How Bezier surfaces contributes to the world of computer games and simulations? Given p0(1, 1); p1(2, 3); p2(4, 3); p3(3, 1) as vertices of Bezier Curve. Determine 3 points on Bezier Curve.

Ans:-BezierCurves:- Bezier curves are used in computer graphics to produce curves which appear reasonably smooth at all scales. This spline **approximation method** was developed by French engineer Pierre Bezier for automobile body design. Bezier spline was designed in such a manner that they are very useful and convenient for curve and surface design, and are easy to implement Curves are trajectories of moving points. We will specify them as functions assigning a location of that moving point (in 2D or 3D) to a parameter t, i.e., parametric curves.

Curves are useful in geometric modeling and they should have a shape which has a clear and intuitive relation to the path of the sequence of control points. One family of curves satisfying this requirement are Bezier curve.

The Bezier curve require only two end points and other points that control the endpoint tangent vector.

Bezier curve is defined by a sequence of N + 1 control points, *P0, P1, . . . , Pn.* We defined the Bezier curve using the algorithm (invented by ***DeCasteljeau***), based on recursive splitting of the intervals joining the consecutive control points. A purely geometric construction for Bezier splines which does not rely on any polynomial formulation, and is extremely easy to understand. The DeCasteljeau method is an algorithm which performs repeated bi-linear interpolation to compute splines of any order.

Properties of Bezier Curves:

A very useful property of a Bezier curve is that it always passes through the first and last control points. That is, the boundary conditions at the two ends of the curve are

P(0) = p0

P(1) = pn

Values of the parametric first derivatives of a Bezier curve at the end points can be calculated from control-point coordinates as

P'(0) = −np0 + np1

P'(1) = −npn 1 + npn −

Thus, the slope at the beginning of the curve is along the line joining the first two control points, and the slope at the end of the curve is along the line joining the last two endpoint. Similarly, the parametric second derivatives of a Bezier curve at the endpoints are calculated as

$$p''(0) = n)n-1)[(p_2 - p_1) - (p_1 - p_0)]$$
$$p''(1) = n)n-1)[(p_{n-2} - p_{n-1}) - (p_{n-1} - p_n)]$$

Another important property of any Bezier curve is that it lies within the convex hull (convex polygon boundary) of the control points. This follows from the properties of Bezier blending functions: They are all positive and their sum is always 1,

$$\sum_{k=0}^{n} B_{k,n}(u) = 1$$

so that any curve position is simply the weighted sum of the control-point positions. The convex-hull property for a Bezier curve ensures that the polynomial will not have erratic oscillations.

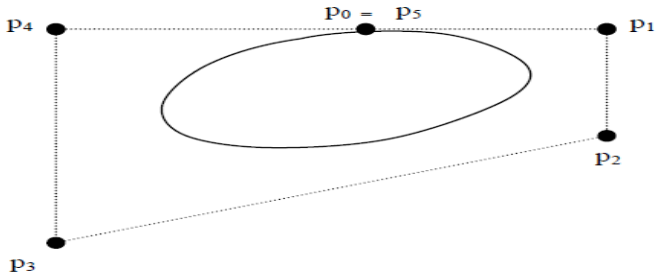


Figure 13 (a): Shows closed Bezier curve generated by specifying the first and last control points at the same location

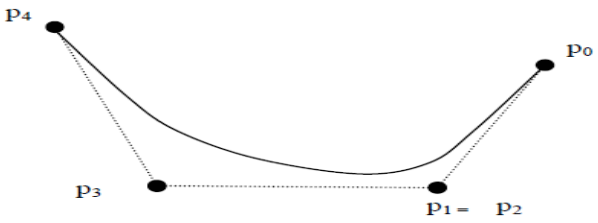


Figure 13 (b): Shows that a Bezier curve can be made to pass closer to a given coordinate position by assigning multiple control points to that position.

Note:

1) Generalising the idea of Bezier curve of degree at n based on n+1 control point p0,…..pn

P(0)= P0

P(1) =pn

Values of parametric first derivates of Bezier curve at the end points can be calculated from control point

Coordinates as

P'(0) = -nP0 +n P1

P'(1) = -nPn-1 + nPn

Thus, the slope at the beginning of the curve is along the line joining two control points, and the slope at the end of the curve is along the line joining the last two endpoints.

2) **Convex hull**: For $0 \leq t \leq 1$, the Bezier curve lies entirely in the convex hull of its control points. The convex hull property for a Bezier curve ensures that polynomial will not have erratic oscillation.

3) Bezier curves are invariant under affine transformations, but they are not invariant under projective transformations.

4) The vector tangent to the Bezier curve at the start (stop) is parallel to the line connecting the first two (last two) control points.

5) Bezier curves exhibit a *symmetry* property: The same Bezier curve shape is obtained if the control points are specified in the opposite order. The only difference will be the parametric direction of the curve. The direction of increasing parameter reverses when the control points are specified in the reverse order.

6) Adjusting the position of a control point changes the shape of the curve in a "predictable manner". Intuitively, the curve "follows" the control point.

There is *no local control* of this shape modification. Every point on the curve (with

the exception of the first and last) move whenever any interior control point is moved.
Following examples prove the discussed properties of the Bezier curves

Example 1: To prove: P (u = 0) = p0

Solution: $\because P(u) = \sum_{i=0}^{n} pi\ B_{n,i}(u)$

$= p_0 B_{n,0}(u) + p_1 B_{n,1}(u) + \ldots\ldots + p_n B_{n,n}(u) \ldots\ldots\ldots\ldots\ldots (1)$

$B_{n,i}(u) = {}_{n}C_{i}\, u^i (1-u)^{n-i}$

$B_{n,0}(u) = {}_{n}C_{0}\, u^0 (1-u)^{n-0} = \frac{n!}{0!(n-0)!} . 1. (1-u)^n = (1-u)^n$

$B_{n,1}(u) = {}_{n}C_{1}\, u' (1-u)^{n-1} = \frac{n!}{1!(n-1)!} . u. (1-u)^{n-1}$

We observe that all terms expect $B_{n,0}(u)$ have multiple of u^i (i = 0 to n) using these terms with u = 0 in (1) we get,

$P(u=0) = p_0 (1-0)^n + p_1 . 0. (1-0)^{n-1} . n + 0 + 0 + \ldots\ldots + 0$

$P(u=0) = p_0$ Proved

Example 2: To prove $\bar{P}(1) = p_n$

Solution: As in the above case we find each term except $B_{n,n}(u)$ will have multiple of $(1-u)^i$ (i = 0 to n) so using u = 1 will lead to result = 0 of all terms except of $B_{n,n}(u)$.

$B_{n,n}(u) = \frac{n!}{n!\,(n-n)!}\, u^n (1-u)^{n-n} = u^n$

$P(u-1) = p_0 . 0 + p_1 . 0 + \ldots\ldots + p_n . 1^n$

$= p_n$

Example 3: Prove: $\sum_{i=0}^{n} B_{n,i} = 1$

Solution: By simple arithmetic we know,

$[(1-u)+u]^n = 1^n = 1$ .....(1)

expending LHS of (1) binomially we find

$[(1-u)+u]^n = {}^{n}C_0 (1-u)^n + {}^{n}C_1 u (1-u)^{n-1} + {}^{n}C_2 u^2 (1-u)^{n-2} + ... + {}^{n}C_n u^n$

$$= \sum_{i=0}^{n} {}^{n}C_i u^i (1-u)^{n-i}$$

$$[(1-u)+u]^n = \sum_{i=0}^{n} B_{n,i}(u) \quad .....(2)$$

by (1) & (2) we get

$$\boxed{\sum_{i=0}^{n} B_{n,i}(u) = 1}$$

Bezier Surfaces Contribution in Computer Games:

Two sets of Bezier curve can be used to design an object surface by specifying by an input mesh of control points. The Bézier surface is formed as the cartesian product of the blending functions of two Bézier curves.

$$P(u,v) = \sum_{j=0}^{m} \sum_{k=0}^{n} p_{j,k} B_{j,m}(v) B_{k,n}(u)$$

with $p_{j,k}$ specifying the location of the (m + 1) by (n + 1) control points.

The corresponding properties of the Bézier curve apply to the Bézier surface.

The surface does not in general pass through the control points except for the corners of the control point grid.

The surface is contained within the convex hull of the control points.

Figures (a), (b), (c) illustrate Bezier surface plots. The control points are connected by dashed lines, and the solid lines show curves of constant *u* and constant *v*. Each curve of constant *u* is plotted by varying *v* over the interval from 0 to 1, with u fixed at one of the values in this unit interval. Curves of constant *v* are plotted similarly.

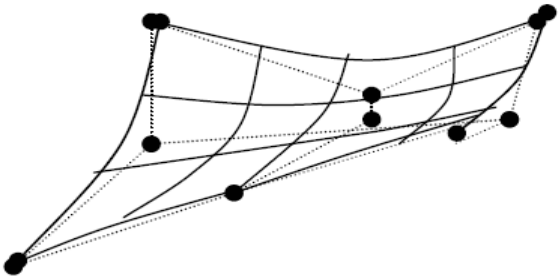


Figure 18 (a)
Bezier surfaces constructed for m=3, n=3, Dashed lines connect the control points

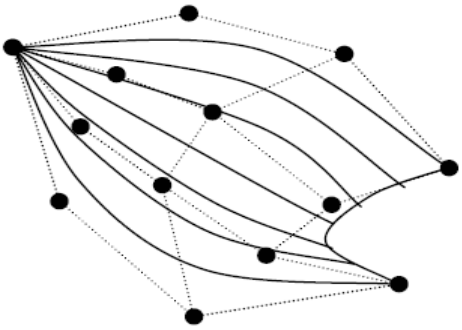


Figure 18 (b)
Bezier surfaces constructed for m=4, n=4. Dashed lines connect the control points.

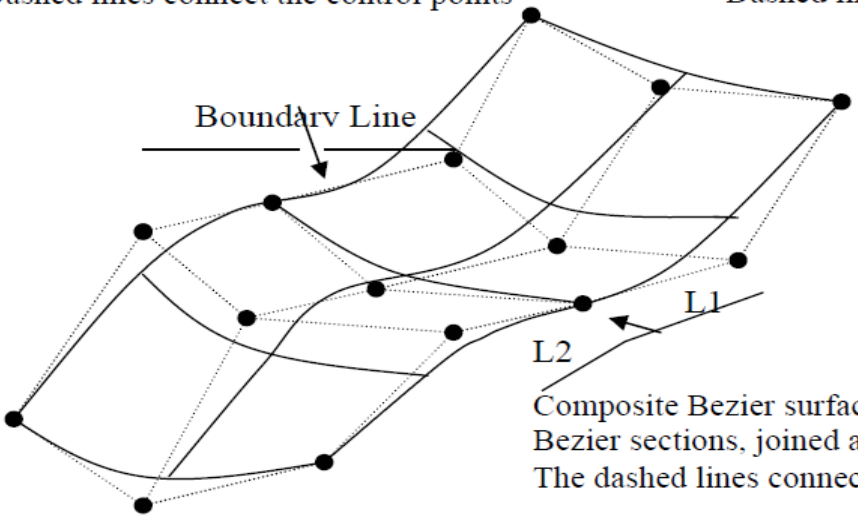


Figure 18 (c)
Composite Bezier surface constructed with two Bezier sections, joined at the indicated boundary line. The dashed lines connect specified control points.

Given p0 (1, 1): p1 (2, 3); p2 (4, 3); p3 (3, 1) as vertices of Bezier curve determine 3 points on Bezier curve?

Solution: We know Cubic Bezier curve is

$$P(u) = \sum_{i=0}^{3} pi\, B_{3,i}(u)$$

$$\Rightarrow P(u) = p_0 (1-u)^3 + 3p_1 u (1-u)^2 + 3p_2 u^2 (1-u) + p_3 u^3$$

$$P(u) = (1, 1)(1-u)^3 + 3(2, 3)u(1-u)^2 + 3(4, 3)u^2(1-u) + (3, 1)u^3.$$

we choose different values of u from 0 to 1.

$$u = 0: \quad P(0) = (1, 1)(1-0)^3 + 0 + 0 + 0 = (1, 1)$$

$$u = 0.5: \quad P(0.5) = (1, 1)(1 - 0.5)^3 + 3(2, 3)(0.5)(1 - 0.5)^2 + 3(4, 3)(0.5)^2(1 - 0.5) + (3,1)(0.5)^3$$

$$= (1, 1)(0.5)^3 + (2, 3)(0.375) + (0.375)(4, 3) + (3, 1)(0.125)$$

$$= (0.125, 0.125) + (0.75, 1.125) + (1.5, 1.125) + (1.125, 0.125)$$

$$P(0.5) = (3.5, 2.5)$$

$$u = 1: \quad P(1) = 0 + 0 + 0 + (3, 1).\, 1^3$$

$$= (3, 1)$$

Three points on Bezier curve are , P (0) = (1, 1); P (0.5) = (3.5, 2.5) and P (1) = (3, 1).

Q.6. Why do you need to use visible-surface detection in Computer Graphics? Explain Scan Line method along with the algorithm for the visible-surface detection with the help of an example. How scan line method is different to z-buffer method?

Ans-: Need of visible-surface detection: As you know for the generation of realistic graphics display, hidden surfaces and hidden lines must be identified for elimination. For this purpose we need to conduct visibility tests. Visibility tests try to identify the visible surfaces or visible edges that are visible from a given viewpoint. Visibility tests are performed by making use of either i) *object-space* or ii) *image-space* or iii) both *object-space* and *image-spaces*. *Object-space* approaches use the directions of a surface normal w.r.t. a viewing direction to detect a back face. *Image-space* approaches utilize two buffers: one for storing the pixel intensities and another for updating the depth of the visible surfaces from the view plane.

A method, which uses both *object-space* and *image-space*, utilizes depth for sorting (or reordering) of surfaces. The methods in this category also use image-space for conducting visibility tests. While making visibility tests, coherency property is utilized to make the method very fast.

Scan Line method: In contrast to z-buffer method, where we consider one surface at a time, scan-line method deals with multiple surfaces. As it processes each scan-line at a time, all polygon intersected by that scan-line are examined to determine which surfaces are visible. The visibility test involves the comparison of depths of each overlapping surface to determine which one is closer to the view plane. If it is found so, then it is declared as a visible surface and the intensity values at the positions along the scanline are entered into the refresh-buffer.

Assumptions:
1. Plane of projection is Z=0 plane.
2. Orthographic parallel projection.
3. Direction of projection, d = (0,0, –1)
4. Objects made up of polygon faces.

Scan-line algorithm solves the hidden- surface problem, one scan-line at a time, usually processing scan lines from the bottom to the top of the display. The scan-line algorithm is a one-dimensional version of the depth – Buffer. We require two arrays, intensity [x] & depth [x] to hold values for a single scan-line.

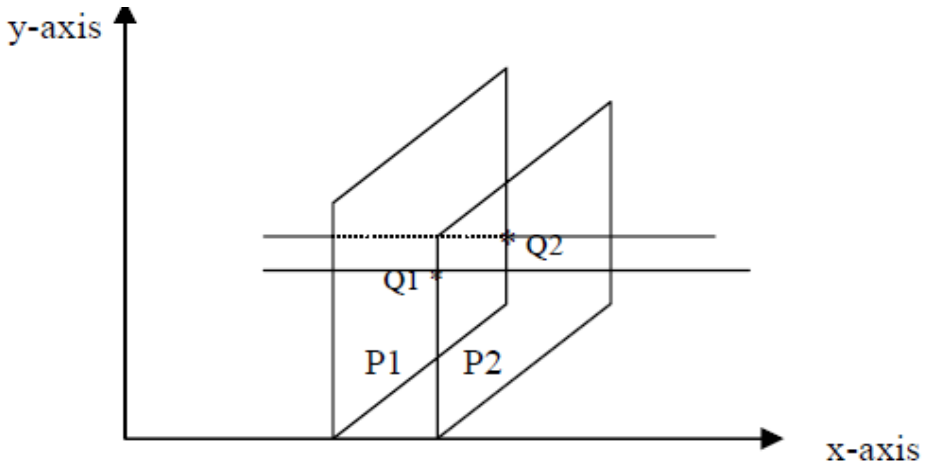


Figure 7

Here at $Q_1$and Q_2 both polygons are active (i.e., sharing).

Compare the z-values at Q_1 for both the planes (P_1 & P_2). Let $z_1^{(1)}$,$z_1^{(2)}$ be the z-value at Q_1 ,corresponding to P_1& P_2 polygon respectively.

Similarly

$z_2^{(1)}$, $z_2^{(2)}$ are the z-values at Q_2, corresponding to P_1 & P_2 polygon respectively.

Case1: $z_1^{(1)} < z_1^{(2)}$, $z_2^{(1)} < z_2^{(2)}$ → Q1,Q2 is filled with the color of P2.

Case2: $z_1^{(2)} < z_1^{(1)}$, $z_2^{(2)} < z_2^{(1)}$ → Q1,Q2 is filled with the color of P2.

Case3: Intersection is taking place.

In this case we have to go back pixel by pixel and determine which plane is closer. Then choose the color of the pixel.

Algorithm (scan-line):

For each scan line perform step (1) through step (3).

1) For all pixels on a scan-line, set depth [x]=1.0 (max value) & Intensity [x] = background-color.
2) For each polygon in the scene, find all pixels on the current scan-line (say S1) that lies within the polygon. For each of these x-values:
   a. calculate the depth z of the polygon at (x,y)
   b. if z < depth [x], set depth [x]=z & intensity corresponding to the polygon's shading.
3) After all polygons have been considered, the values contained in the intensity array represent the solution and can be copied into a frame-buffer.

Advantages of Scan line Algorithm:
Here, every time, we are working with one-dimensional array, i.e., x[0…x_max] for color not a 2D-array as in depth buffer algorithm.

scan line method is different to z-buffer method: In z-buffer algorithm every pixel position on the projection plane is considered for determining the visibility of surfaces w. r. t. this pixel. On the other hand in scan-line method all surfaces intersected by a scan line are examined for visibility. The visibility test in z-buffer method involves the comparison of depths of surfaces w. r. t. a pixel on the projection plane. The surface closest to the pixel position is considered visible. The visibility test in scan-line method compares depth calculations for each overlapping surface to determine which surface is nearest to the view-plane so that it is declared as visible.

Q.7. Explain the following terms in the context of computer Graphics using suitable diagram and /or mathematical equations or one example. (10 Marks)

Ans:-
i) Depth Buffer Method: Depth-buffer method is a fast and simple technique for identifying visible-surfaces. This method is also referred to as the *z*-buffer method, since object depth is usually measured from the view plane along the z-axis of a viewing system. This algorithm compares surface depths at each pixel position (x,y) on the view plane. Here we are taking the following assumption:

- Plane of projection is z=0 plane
- Orthographic parallel projection.

For each pixel position (x,y) on the view plane, the surface with the smallest zcoordinate at that position is visible. For example, *Figure 3* shows three surfaces S1, S2, and S3, out of which surface S1 has the smallest z-value at (x,y) position. So surface S1 is visible at that position. So its surface intensity value at (x,y) is saved in the refresh-buffer.

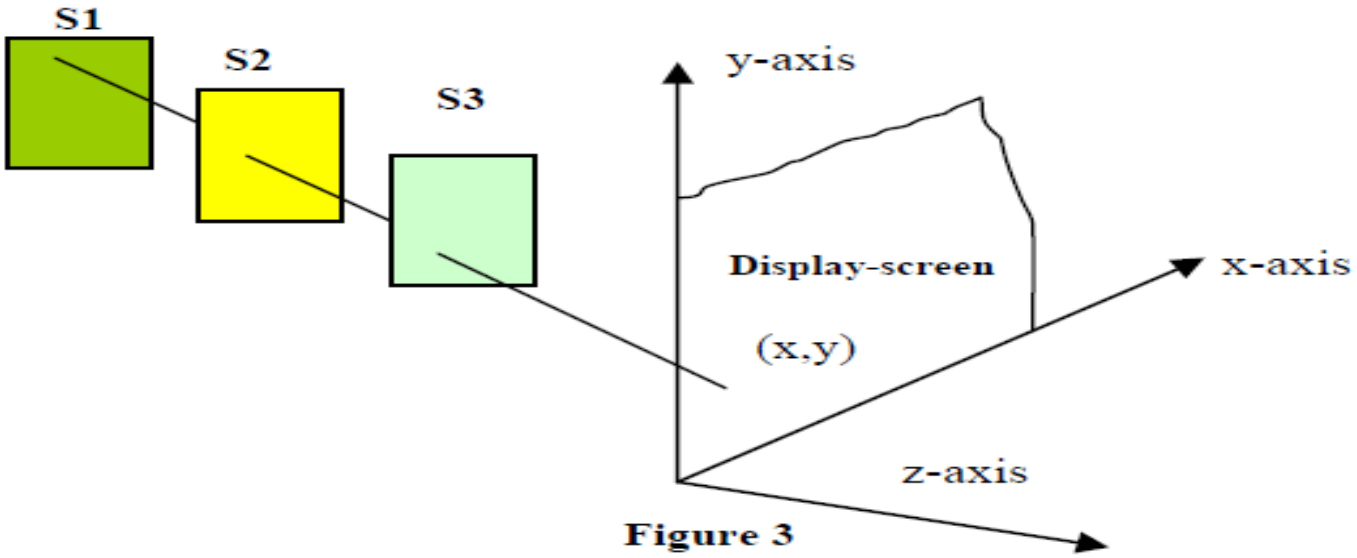


Figure 3

Here the projection is orthographic and the projection plane is taken as the xy-plane. So, each (x,y,z) position on the polygon surfaces corresponds to the orthographic projection point (x,y) on the projection plane. Therefore, for each pixel position (x,y) on the view plane, object depth can be compared by comparing z-values, as shown in *Figure 3*.

We summarize the steps of a depth-buffer algorithm as follows:

Given: A list of polygons {P1,P2,…..,Pn}.
Step1: Initially all positions (x,y) in the depth-buffer are set to 1.0 (maximum depth) and the refresh-buffer is initialized to the background intensity i.e.,
z-buffer*(x,y):=1.0; and*
COLOR*(x,y):= Background color.*
Step2: For each position on each polygon surface (listed in the polygon table) is then processed (scan-converted), one scan line at a time. Calculating the depth (zvalue) at each (x,y) pixel position. The calculated depth is then compared to the value previously stored in the depth buffer at that position to determine visibility.

a) If the calculated z-depth is less than the value stored in the depth-buffer, the new depth value is stored in the depth-buffer, and the surface intensity at that position is determined and placed in the same (x,y) location in the refresh-buffer, i.e.,

If z-depth< ***z-buffer***(x,y), then set
z-buffer*(x,y)=z-depth;*
COLOR*(x,y)=Isurf(x,y);* // where *Isurf(x,y)* is the projected intensity value of the polygon surface Pi at pixel position (x,y).

After all surfaces have been processed, the depth buffer contains depth values for the visible surfaces and the refresh-buffer contains the corresponding intensity values for those surfaces.

Example 1: How does the z-buffer algorithm determine which surfaces are hidden?
Solution: Z-buffer algorithm uses a two buffer area each of two-dimensional array, one z-buffer which stores the depth value at each pixel position (x,y), another framebuffer which stores the intensity values of the visible surface. By setting initial values of the z-buffer to some large number (usually the distance of back clipping plane), the problem of determining which surfaces are closer is reduced to simply comparing the present depth values stored in the z-buffer at pixel (x,y) with the newly calculated depth value at pixel (x,y). If this new value is less than the present z-buffer value, this value replaces the value stored in the z-buffer and the pixel color value is changed to the color of the new surface.

ii) Area Subdivision Method
This method is essentially an image-space method but uses object-space operations reordering (or sorting) of surfaces according to depth. This method takes advantage of area-coherence in a scene by locating those view areas that represent part of a single surface. In this method we successively subdivide the total viewing (screen) area, usually a rectangular window, into small rectangles until each small area is the projection of part of a single visible surface or no surface at all.
Assumptions:
- Plane of projection is z=0 plane
- Orthographic parallel projection
- Direction of projection d=(0,0,-1)
- Assume that the viewing (screen) area is a square
- Objects are made up of polygon faces.

Starting with the full screen as the initial area, the algorithm divides an area at each stage into 4 smaller area, as shown in Figure 8, which is similar to quad-tree approach.

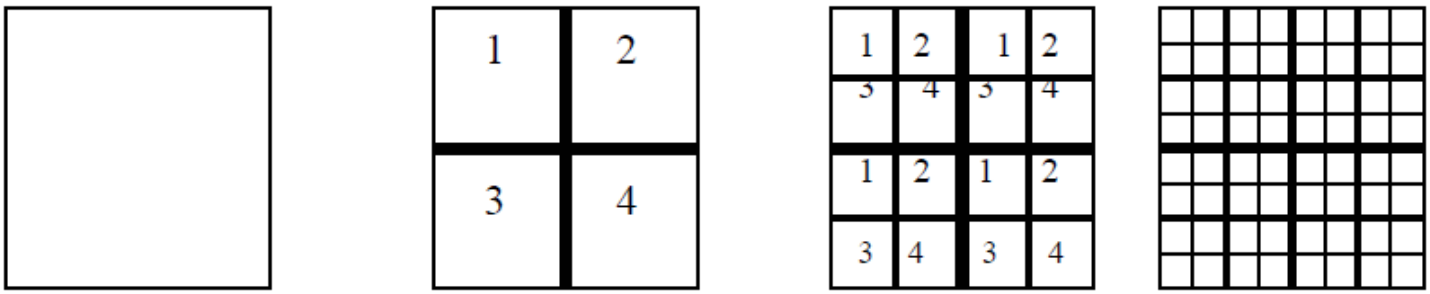


Figure 8 (a) Initial area (b) 1st subdivision (c) 2nd Subdivision (d) 3rd Subdivision

Subdivision Algorithm

1) Initialize the area to be the whole screen.

2) Create a PVPL w.r.t. an area, sorted on zmin (the smallest z coordinate of the polygon within the area). Place the polygons in their appropriate categories. Remove polygons hidden by a surrounding polygon and remove disjoint polygons.

3) Perform the visibility decision tests:

a)If the list is empty, set all pixels to the background color.

b) If there is exactly one polygon in the list and it is classified as intersecting (category 2) or contained (category 3), color (scan-converter) the polygon, and color the remaining area to the background color.

c) If there is exactly one polygon on the list and it is a surrounding one, color the area the color of the surrounding polygon.

d) If the area is the pixel (x,y), and neither a, b, nor c applies, compute the z coordinate z(x, y) at pixel (x, y) of all polygons on the PVPL. The pixel is then set to the color of the polygon with the smallest z coordinate.

4) If none of the above cases has occurred, subdivide the screen area into fourths.For each area, go to step 2.

Example 1: Suppose there are three polygon surfaces P,Q, R with vertices given by:

P: P1(1,1,1), P2(4,5,2), P3(5,2,5)

Q: Q1(2,2,0.5), Q2(3,3,1.75), Q3(6,1,0.5)

R: R1(0.5,2,5.5), R2(2,5,3), R3(4,4,5)

Using the Area subdivision method, which of the three polygon surfaces P, Q, R obscures the remaining two surfaces? Assume z=0 is the projection plane. Solution: Here, we have z=0 is the projection plane and P, Q, R are the 3-D planes. We apply first three visibility decision tests i.e. (a), (b) and (c), to check the bounding rectangles of all surfaces against the area boundaries in the xyplane. Using test 4, we can determine whether the minimum depth of one of the surrounding surface S is closer to the view plane.

iii) Basic Ray Tracing Algorithm:

The Hidden-surface removal is the most complete and most versatile method for display of objects in a realistic fashion. The concept is simply to take one ray at a time, emanating from the viewer's eye (in perspective projection) or from the bundle of parallel lines of sight (in parallel projection) and reaching out to each and every pixel in the viewport, using the laws of optics.

If the ray encounters one or more objects, the algorithm filters out all but the nearest object, or when there is an overlap or a hole, the nearest visible portion of all the objects along the line of sight. Depending on the nature (attributes) of the surface specified by the user, the following effects are implemented, according to rules of optics.

a) Reflection (according to the angle of incidence and reflectivity of the surface).

b) Refraction (according to the angle of incidence and refraction index).
c) Display of renderings (texture or pattern as specified), or shadows (on the next nearest object or background) involving transparency or opacity, as the case may be.

Figure illustrates some of the general principles of ray tracing.

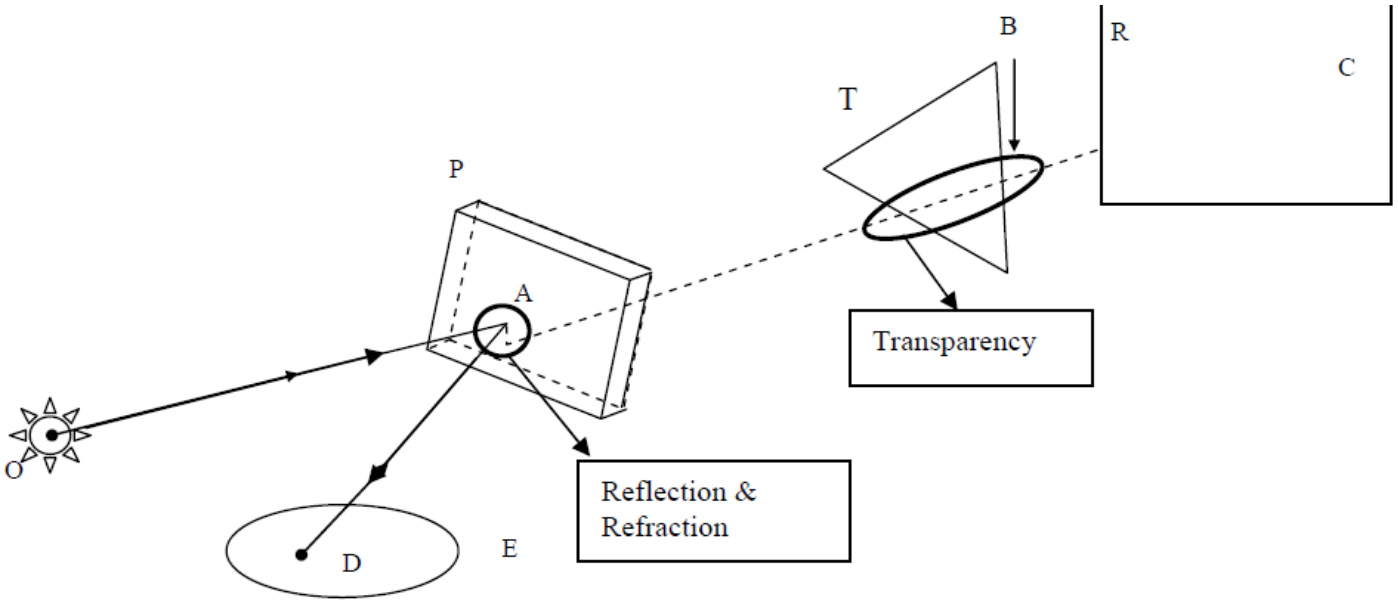


Figure 23

A ray starting at O hits the transparent glass plate P at an angle at A. It gets refracted in the plate (indicated by the kink within the plate thickness). The exiting ray happens to hit the edge of the triangle T, and casts a shadow on the opaque rectangular plate R at the point C. A part of the ray incident on the plate P gets reflected at A and the reflected ray hits the elliptical object E at the point D. If P is a green glass plate, the exiting ray AC will be assigned the appropriate green colour. If R or E has a textured surface, the corresponding point C or D will be given the attributes of the surface rendering.

If O is a point source of light, the ray OC will locate the shadow of the point B on the edge of the triangle T at the point C on the rectangle R. Different locations of light sources may be combined with differing view positions to improve the realism of the scene. The method is general also in the sense that it can apply to curved surfaces as well as to solids made of flat polygonal segments. Because of its versatile and broad applicability, it is a "brute force" method, involving massive computer resources and tremendous computer effort.

Algorithm

Often, the basic ray tracing algorithm is called a "recursive" (obtaining a result in which a given process repeats itself an arbitrary number of times) algorithm. Infinite recursion is recursion that never ends. The ray tracing algorithm, too, is recursive, but it is finitely recursive. This is important, because otherwise you would start an image rendering and it would never finish!

The algorithm begins, as in ray casting, by shooting a ray from the eye and through the screen, determining all the objects that intersect the ray, and finding the nearest of those intersections. It then recurses (or repeats itself) by shooting more rays from the point of intersection to see what objects are reflected at that point, what objects may be seen through the object at that point, which light sources are directly visible from that point, and so on. These additional rays are often called secondary rays to differentiate them from the original, primary ray. As an analysis of the above discussion we can say that we pay for the increased features of ray tracing by a dramatic increase in time spent with calculations of point of intersections with both the

primary rays (as in ray casting) and each secondary and shadow ray. Thus achieving good picture quality, is not an easy task, and it only gets more expensive as you try to achieve more realism in your image. One more concept known as Antialiasing is yet to be discussed, which plays a dominant role in achieving the goal of realism.

iv) Projections and its Types:

Given a 3-D object in a space, Projection can be defined as a mapping of 3-D object onto 2-D viewing screen. Here, 2-D screen is known as Plane of projection or view plane, which constitutes the display surface. The mapping is determined by projection rays called the projectors. Geometric projections of objects are formed by the intersection of lines (called projectors) with a plane called plane of projection /view plane. Projectors are lines from an arbitrary point, called the centre of projection (COP), through each point in an object. Figure 1 shows a mapping of point P(x,y,z) onto its image P′(x',y',z') in the view plane.

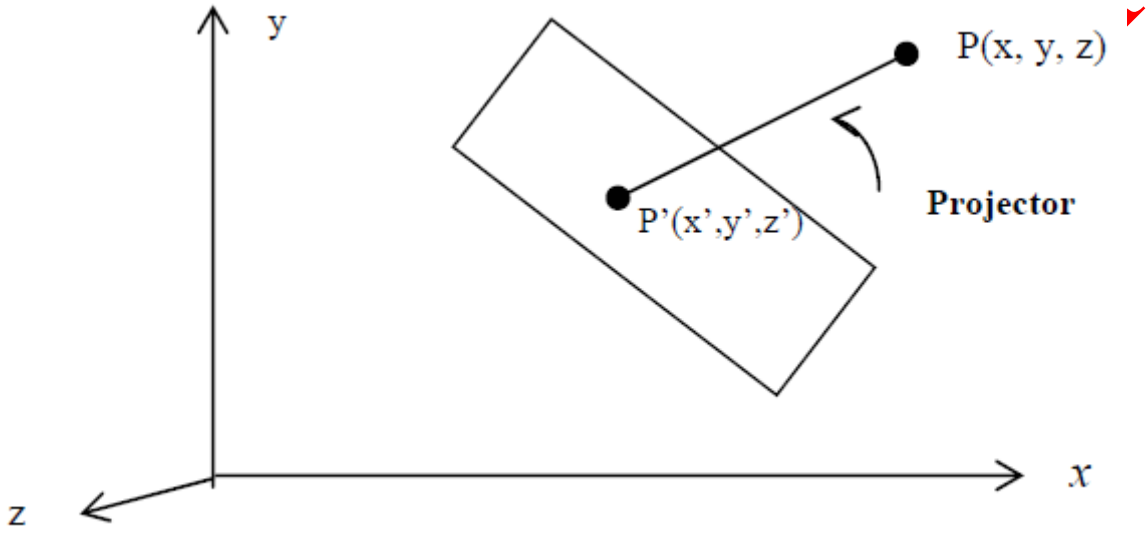


Figure 1

There are projection type:

Taxonomy of Projection

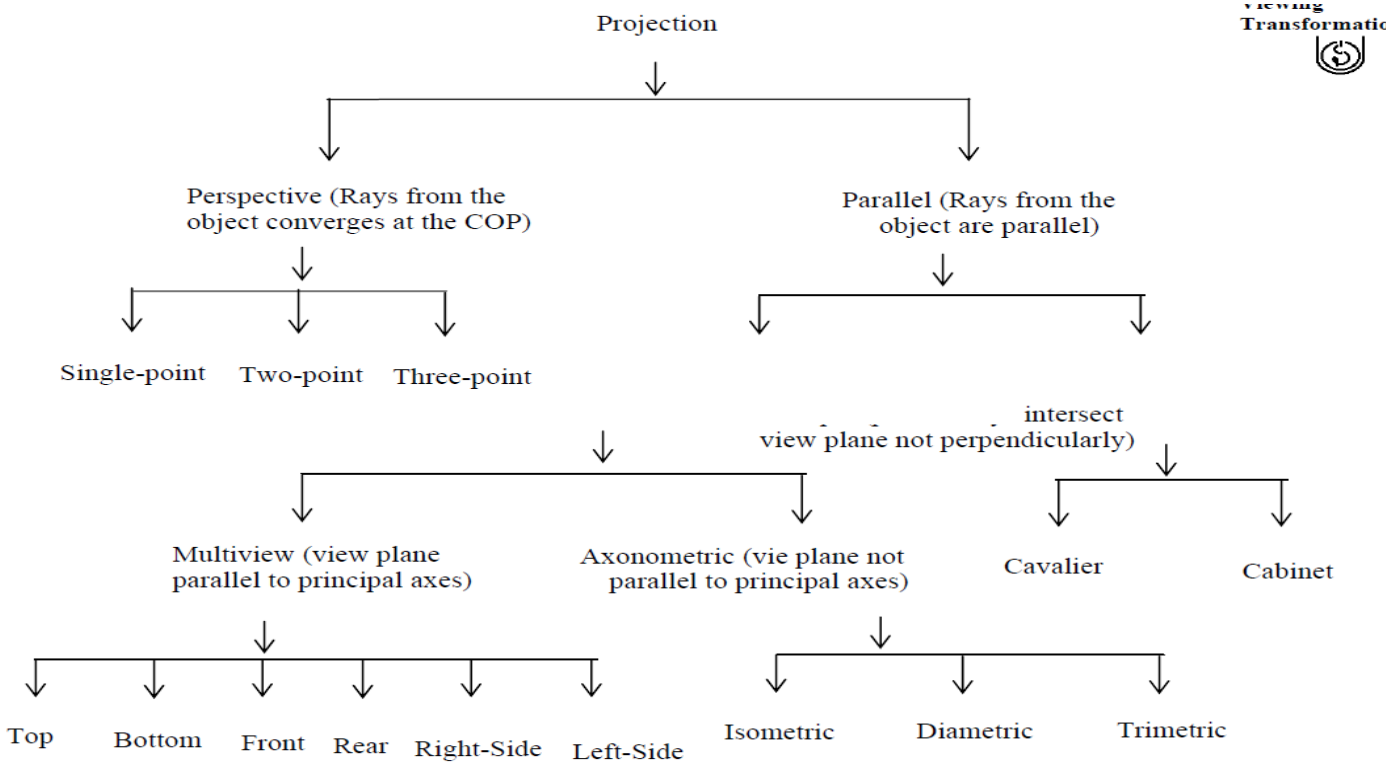


Figure 3: Taxonomy of projection

Parallel Projection: Parallel projection methods are used by engineers to create working drawings of an object which preserves its true shape. In the case of parallel projection the rays from an object converge at infinity, unlike the perspective projection where the rays from an object converse at a finite distance (called COP).

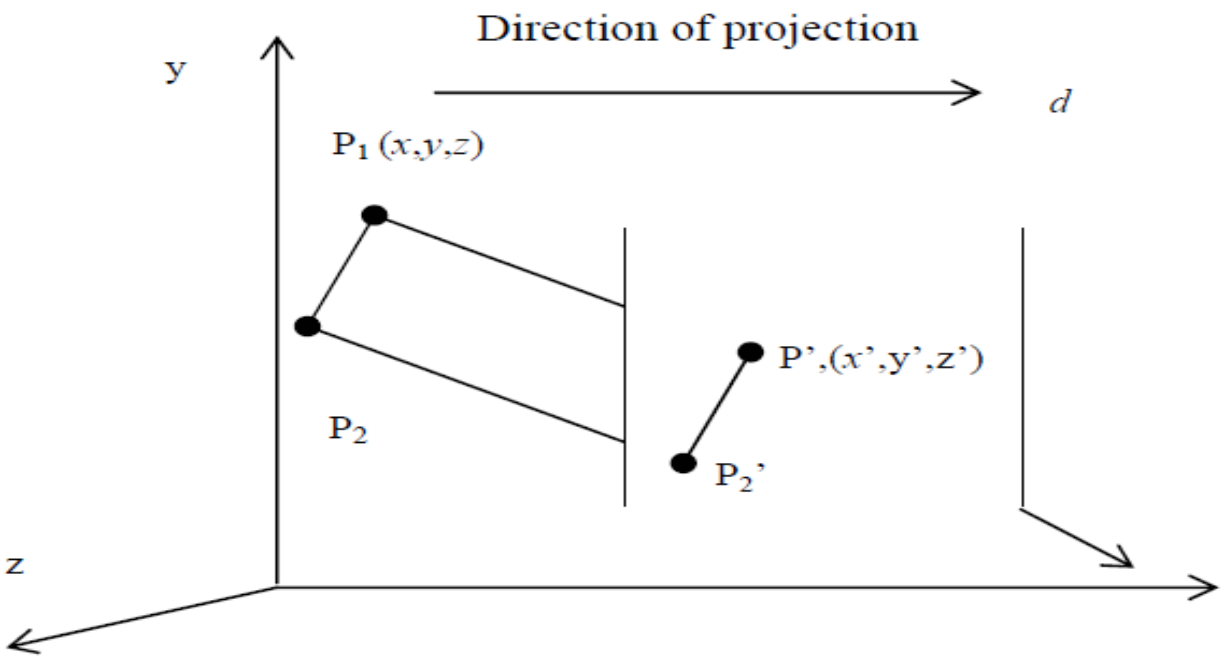


Figure 5: Parallel projection

Orthographic and Oblique Projections: Orthographic projection is the simplest form of parallel projection, which is commonly used for engineering drawings. They actually show the 'true' size and shape of a single plane face of a given object.

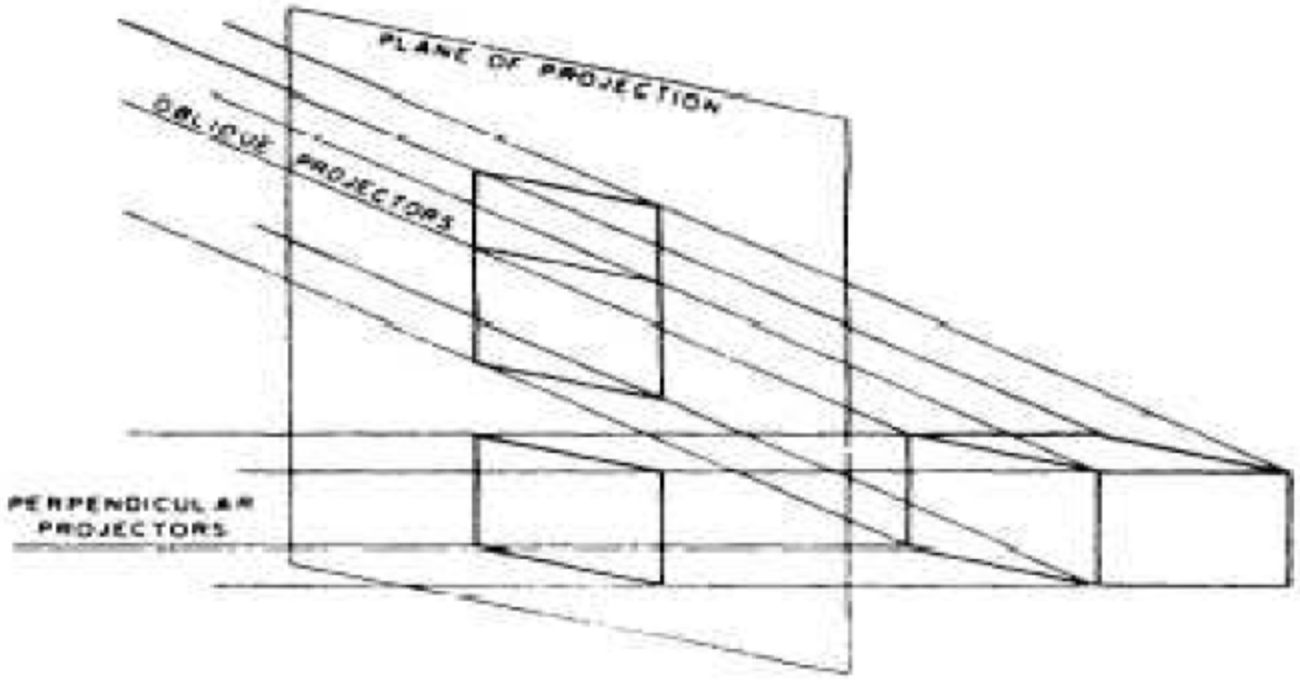


Figure 6: Orthographic and oblique projection

Example1: Derive the general transformation of parallel projection onto the xy-plane in the direction of projection d=aI+bJ+cK.

Solution: The general transformation of parallel projection onto the xy-plane in the direction of projection d=aI+bJ+cK, is derived as follows(see Figure a):

Let P(x,y,z) be an object point, projected to P'(x',y',z') onto the z'=0 plane. From Figure (a) we see that the vectors d and PP' have the same direction. This means that

PP'=k.d , comparing components, we have:

x'-x=k.a

y'-y=k.b

z'-z=k.c

Since z'=0 on the projection plane, we get k=-z/c.

Thus,

x'=x-a.z/c

y'=y-b.z/c

z'=0

Q.8. Compare and contrast the following:

***i)* Interlaced and progressive scan**

ii) Compression and decompression in digital video

iii) Hypermedia and hypertext

iv) Types of Bitmap and Vector graphics

v) Ray tracing and Ray casting

Ans:-

I) Interlaced and progressive scan

Interlaced Scan:

Interlace is a technique of improving the picture quality of a video transmission without consuming any extra bandwidth. It was invented by the RCA engineer **Randall Ballard** in the late 1920s . It was universally used in television until the 1970s, when the needs of computer monitors resulted in the reintroduction of progressive scan. While interlace can improve the resolution of still images, on the downside, it causes flicker and various kinds of distortion. Interlace is still used for all standard definition TVs, and the 1080i HDTV broadcast standard, but not for LCD, micromirror (DLP, or plasma displays).

These devices require some form of deinterlacing which can add to the cost of the set. 56 With progressive scan, an image is captured, transmitted and displayed in a path similar to the text on a page: line by line, from top to bottom. The interlaced scan pattern in a CRT (cathode ray tube) display would complete such a scan too, but only for every second line and then the next set of video scan lines would be drawn within the gaps between the lines of the previous scan. Such scan of every second line is called a field.

Progressive Scan

Progressive scan is used for most CRT computer monitors. (Other CRT-type displays, such as televisions, typically use interlacing.) It is also becoming increasingly common in high-end television equipment. Advantages of progressive scan include:

- Increased vertical resolution. The perceived vertical resolution of an interlaced image is usually equivalent to multiplying the active lines by about 1.6
- No flickering of narrow horizontal patterns
- Simpler video processing equipment

- Easier compression

ii) Compression and decompression in digital video

Compression

Compression is a reversible conversion of data to a format that requires fewer bits, usually performed so that the data can be stored or transmitted more efficiently. The size of the data in compressed form (C) relative to the original size (O) is known as the compression ratio (R=C/O). If the inverse of the process, decompression, produces an exact replica of the original data then the compression is lossless.

Lossy compression, usually applied to image data, does not allow reproduction of an exact replica of the original image, but has a higher compression ratio. Thus, lossy compression allows only an approximation of the original to be generated.

Compression is analogous to folding a letter before placing it in a small envelope so that it can be transported more easily and cheaply (as shown in the figure). Compressed data, like the folded letter, is not easily read and must first be decompressed, or unfolded, to restore it to its original form. The success of data compression depends largely on the data itself and some data types are inherently more compressible than others. Generally some elements within the data are more common than others and most compression algorithms exploit this property, known as redundancy. The greater the redundancy within the data, the more successful the compression of the data is likely to be. Fortunately, digital video contains a great deal of redundancy and thus, is very suitable for compression.

Decompression

A device (software or hardware) that compresses data is often know as an encoder or coder, whereas a device that decompresses data is known as a decoder. A device that acts as both a coder and decoder is known as a codec. Compression techniques used for digital video can be categorised into three main groups:

- General purpose compression techniques can be used for any kind of data.
- Intra-frame compression techniques work on images. Intra-frame compression is compression applied to still images, such as photographs and diagrams, and exploits the redundancy within the image, known as spatial redundancy. Intraframe compression techniques can be applied to individual frames of a video sequence.

Inter-frame compression techniques work on image sequences rather than individual images. In general, relatively little changes from one video frame to the next. Interframe compression exploits the similarities between successive frames, known as temporal redundancy, to reduce the volume of data required to describe the sequence.

iii) Hypermedia and hypertext

Hypertext - Hypertext is conceptually the same as a regular text - it can be stored, read, searched, or edited - with an important difference: hypertext is text with pointers to other text. The browsers let you deal with the pointers in a transparent way -- select the pointer, and you are presented with the text that is pointed to.

A way of presenting information online with connections between one piece of information and another. These connections are called hypertext links. Thousands of these hypertext links enable you to explore additional or related information throughout the online documentation. See also hypertext link.

In computing, hypertext is a user interface paradigm for displaying documents which, according to an early definition (Nelson 1970), “branch or perform on request.” The most frequently discussed form of hypertext document contains automated cross-references to other documents called hyperlinks. Selecting a hyperlink causes the computer to display the linked document within a very short period of time.

Hypermedia - Hypermedia is a superset of hypertext. Hypermedia documents contain links not only to other pieces of text, but also to other forms of media - sounds, images, and movies. Images themselves can be selected to link to sounds or documents. Hypermedia simply combines hypertext and multimedia.

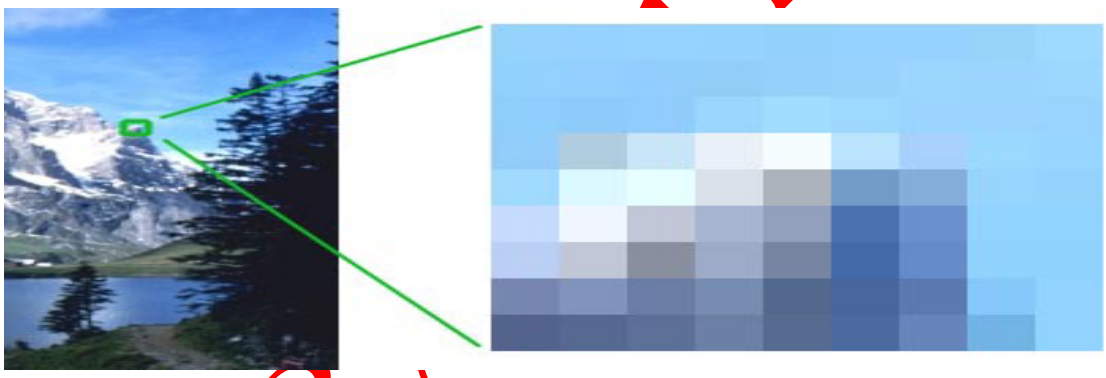
Hypermedia is a term created by **Ted Nelson** in 1970. It used as a logical extension of the term hypertext, in which graphics, audio, video, plain text and hyperlinks intertwine to create a generally non-linear medium of information. This contrasts with multimedia, which, although often capable of random access in terms of the physical medium, is essentially linear in nature. The difference should also be noted with hypergraphics or super-writing which is a Lettrist form from the 1950s which systemises creativity across disciplines.

A classic example of hypermedia is World Wide Web, whereas, a movie on a CD or DVD is an example of standard multimedia. The difference between the two can (and often do) blur depending on how a particular technological medium is implemented. The first hypermedia system was the Aspen Movie Map

iv) Types of Bitmap and Vector graphics

Bitmap Graphics: The information below describes bitmap data. Bitmap images are a collection of bits that form an image. The image consists of a matrix of individual dots (or pixels) that have their own colour described using bits.

Lets take a look at a typical bitmap image to demonstrate the principle:

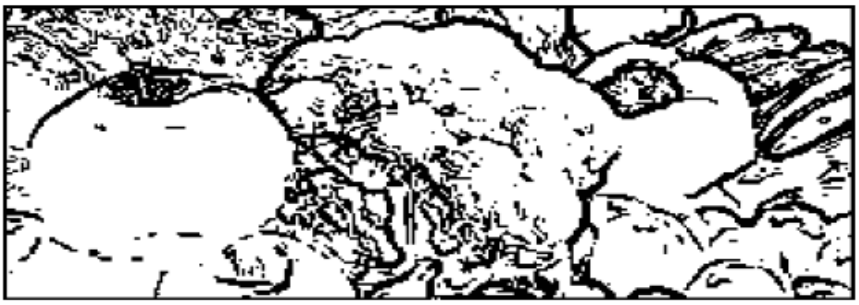


To the left you see an image and to the right a 250 percent enlargement of the top of one of the mountains. As you can see, the image consists of hundreds of rows and columns of small elements that all have their own colour. One such element is called a pixel. The human eye is not capable of seeing each individual pixel so we perceive a picture with smooth gradations. Application of the image decides the number of pixels you need to get a realistic looking image.

Types of Bitmap Images

Bitmap images can contain any number of colours but we distinguish between four main categories:

1) Line-art: These are images that contain only two colours, usually black and white.



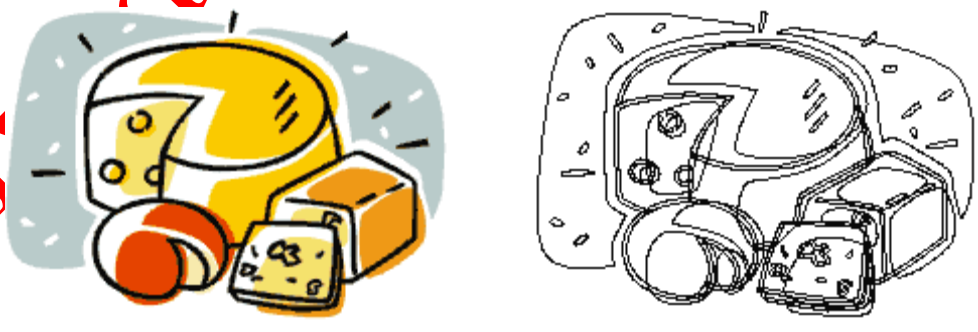
2) Grayscale images, which contain various shades of grey as well as pure black and white.

3) Multitones: Such images contain shades of two or more colours.

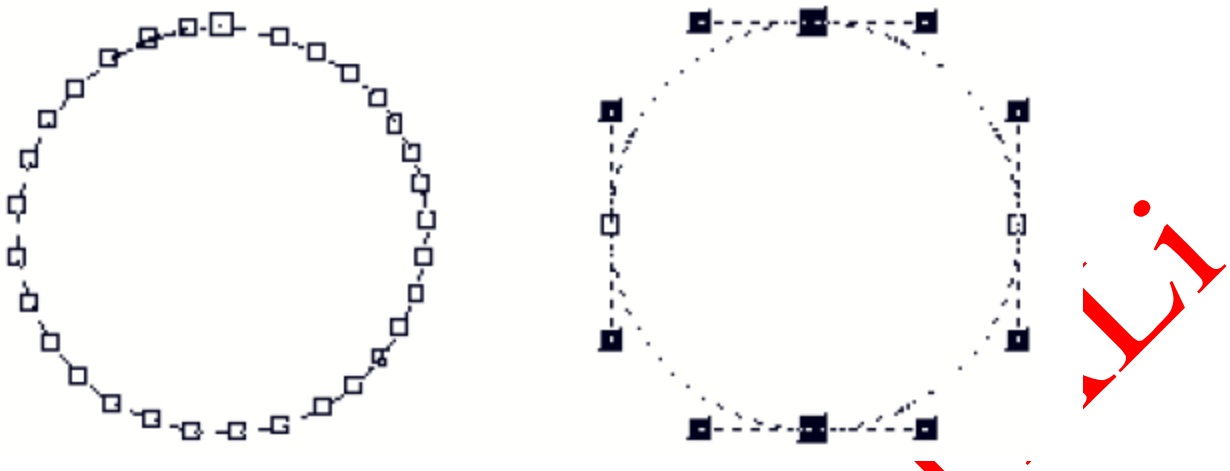
4) Full colour images: The colour information can be described using a number of colour spaces: RGB, CMYK for instance.

Vector Graphics

Vector graphics are images that may be entirely described using mathematical definitions. The image below shows the principle. To the left you see the image itself and to the right you see the actual lines that make up the drawing.



Each individual line is made up a large number of small lines that interconnect a large number of or, just a few control points that are connected using Bezier curves. It is this latter method that generates the best results and that is used by most drawing programs.



This drawing demonstrates the two principles. To the left a circle is formed by connecting a number of points using straight lines. To the right, you see the same circle that is now drawn using 4 points (nodes) only.

v) Ray tracing and Ray casting :

Ray tracing is a method of generating realistic images by computer, in which the paths of individual rays of light are followed from the viewer to their points of origin". Whereas ray casting only concerns itself with finding the visible surfaces of objects, ray tracing takes that a few steps further and actually tries to determine what each visible surface looks like.

Ray tracing: Ray tracing is an exercise performed to attain the realism in a scene. In simple terms Ray Tracing is a global illumination based rendering method used for producing views of a virtual 3-dimensional scene on a computer. Ray tracing is closely allied to, and is an extension of, ray-casting, a common hidden-surface removal method. It tries to mimic actual physical effects associated with the propagation of light. Ray tracing handles shadows, multiple specular reflections, and texture mapping in a very easy straight-forward manner. So, the crux is "Ray tracing is a method of generating realistic images by computer, in which the paths of individual rays of light are followed from the viewer to their points of origin". Any program that implements this method of ray tracing is ray tracer. One of the prime advantages of method of Ray tracing is, it makes use of the actual physics and mathematics behind light. Thus the images produced can be strikingly, life-like, or "photo-realistic".

Ray casting :Ray casting is a method in which the surfaces of objects visible to the camera are found by throwing (or casting) rays of light from the viewer into the scene. The idea behind ray casting is to shoot rays from the eye, one per pixel, and find the closest object blocking the path of that ray – think of an image as a screen-door, with each square in the screen being a pixel. This is then the object the eye normally sees through that pixel. Using the material properties and the effect of the lights in the scene, this algorithm can determine the shading of this object. The simplifying assumption is made that if a surface faces a light, the light will reach that surface and not be blocked or in shadow. The shading of the surface is computed using traditional 3D computer graphics shading models. Ray casting is not a synonym for ray tracing, but can be thought of as an abridged, and significantly faster, version of the ray tracing algorithm. Both are image order algorithms used in computer graphics to render three dimensional scenes to two dimensional screens by following rays of light

Methods from the eye of the observer to a light source. Although ray tracing is similar to ray casting, it may be better thought of as an extension of ray casting we will discuss this in the next topic under this section.