Computer Graphics

Statistical Analysis

Chapter No.	May 2012	Dec. 2012	May 2013	Dec. 2013
Chapter 1	05 Marks	15 Marks	10 Marks	10 Marks
Chapter 2	15 Marks	10 Marks	20 Marks	20 Marks
Chapter 3	10 Marks	25 Marks	10 Marks	15 Marks
Chapter 4	15 Marks	10 Marks	15 Marks	20 Marks
Chapter 5	22 Marks	10 Marks	25 Marks	10 Marks
Chapter 6	20 Marks	25 Marks	20 Marks	25 Marks
Chapter 7	10 Marks	10 Marks	10 Marks	15 Marks
Chapter 8	33 Marks	20 Marks	20 Marks	05 Marks
Chapter 9	10 Marks	15 Marks	10 Marks	15 Marks
Repeated Questions	–	40 Marks	69 Marks	90 Marks

May 2012

Chapter 1 : Introduction to Computer Graphics [Total Marks - 05]

Q. 1(a) What are the applications of Computer Graphics. **(5 Marks)**

Ans. :

The applications of computer graphics are in a variety of field such as,

(1) Engineering/Scientific Software, Business Software.

(2) T.V. channels, space simulation training.

(3) PCB designing, map preparation.

(4) User Interface, Animation.

(5) Making Charts, Image Processing.

(6) Office Automation.

(7) Desktop Publishing.

(8) CAD/CAM.

(9) Art & Commerce.

(10) Process Controlling.

(11) 'Visual Effects' in Movies and Computer Games.

Chapter 2 : Output Primitives [Total Marks - 15]

Q. 1(d) Explain Antialiasing Techniques. **(5 Marks)**

Ans. : Antialiasing :

Most aliasing effects, occurs in static images at a moderate resolution, are often tolerable or negligible. But when the image is moving or in animation this drawback becomes bold. We can come out of this drawback by straight way increasing the image resolution. But for that we have to pay extra money and still the results are not always satisfactorily. There are techniques that can greatly reduce this effect and improve the images. Such techniques are collectively known as antialiasing.

When we have displays that allow more than two colors, we can use the following techniques to soften the jaggies.

1. **Prefiltering :** This is a technique that determines pixel intensity based on the amount of that particular pixels coverage by the object in the scene i.e. it computes pixel colors depending on an objects coverage. It means how much part or fraction of that pixel is covered by the object and depending on that, it sets the value of that pixel. It requires large number of calculations and approximations. Prefiltering generates more accurate antialiasing effect. But due to its high complexity of calculations it is not used.

2. **Supersampling :** Supersampling tries to reduce the aliasing effect by sampling or taking more than one sample per pixel. It subdivides each pixel into 9 sub pixels of equal size which is often called as sampling points. See Fig. 1. Some of these 9 sub pixels may get color of background i.e. a line may not pass through them and rest of sub pixels may get color of object. Suppose 3 samples or sub pixels get background color and 6 gets foreground color then the color of the pixel will be the sum of $\left(\frac{1}{3}\right)$ of background color and $\left(\frac{2}{3}\right)$ of object color.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Fig. 1

So ultimately the pixel value becomes the average of several samples.

3. **Postfiltering :**

1/8	1/8	1/8
1/8	50%	1/8
1/8	1/8	1/8

Fig. 2

Postfiltering and supersampling are almost same. If we consider 9 sampling points as in supersampling, we may give a lot more weightage to center point. We could give the center sample a weig htage of 50% or 1/2; and remaining 50% is distributed among the rest of 8 samples. See Fig. 2. So we can treat supersampling as the special case of postfiltering in which each sampling point has an equal weightage $\left(\frac{1}{9}\right)$.

4. **Pixel phasing :** Pixel phasing is a hardware based anti-aliasing technique. The graphics system in this case is capable of shifting individual pixels from their normal positions in the pixel grid by

a fraction (typically $\frac{1}{4}$ or $\frac{1}{2}$) of unit distance between pixels. By moving pixels closer to the true line, this technique reduces the aliasing effect.

5. **Gray level :** Many displays allow only two pixel states, ON and OFF. In that case we observe stair step aliasing effect. To reduce this effect we can make use of gray level technique. In this technique display allow setting pixels to gray levels between black and white to reduce the aliasing effect. See Fig. 3. It uses the gray levels to gradually turn off the pixels in one row as it gradually turns on the pixels in the next.

Fig. 3

Q. 2(a) Write Bresenham's line drawing algorithm. Also write mathematical derivations for the same. **(10 Marks)**

Ans. : Bresenham's Line Generation Algorithm :

There is one more algorithm to generate lines which is called as Bresenham's line algorithm.

The distance between the actual line and the nearest grid location is called error and it is denoted by G. Suppose line is as shown in Fig. 4. After displaying first point (0, 0) we have to select next point. Now there are two candidate pixels (1, 0) and (1, 1). Out of these two pixels we have to select one pixel. This selection of pixel will depend on the slope of the line. If the slope of line is greater than 0.5 then we are selecting upper pixel i.e. (1, 1) and if slope is less than 0.5 then we are selecting next pixel as (1, 0).

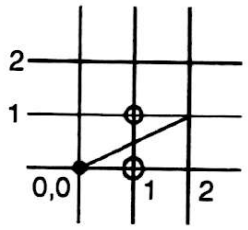


Fig. 4

As we are interested in only checking whether the point is below 0.5 or above 0.5, we will put condition as if the slope of line is greater than or less than 0.5.

Example :

Suppose the slope of line shown in Fig. 5 is 0.4. After displaying 1st point as (0, 0) we will add slope of line to error factor G, so that G will become 0.4. As we are considering gentle slope case we are moving along x-axis so next pixel will be x = 1 and at that time y will be decided by G. If G > 0.5 then y = 1 else y = 0. But here as G = 0.4 so we have to select next point as (1, 0).

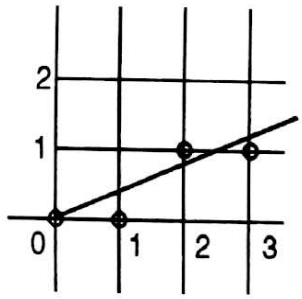


Fig. 5

Again for next point x is increased by 1 and it becomes x = 2, for that G will become G = G + m i.e. 0.8. Now this time G > 0.5 so upper pixel is more near to desired one. That is why we have to select upper point i.e. (2, 1). The beauty of Bresenham's algorithm is it is implemented entirely with integer numbers and the integer numbers are much more faster than floating-point arithmetic. But in previous section we have seen that we are checking slope with 0.5 i.e. floating point variable. So how we can make use of integers for this test ?

For Gentle slope case

Consider P as height of pixel or error. Our condition is whether P > 0.5 or not.

$$\text{If } P > 0.5 \qquad \ldots(1)$$

But here 0.5 is floating point so we have to convert floating point to integer. For that we will multiply both sides by 2.

$$\therefore 2P - 1 > 0 \qquad \ldots(2)$$

In this equation we have removed fractional part 0.5. But still it may contain fractional part from the denominator of slope. Because every time we are updating P by,

$P = P + \text{slope}$ or $P = P + \text{slope} - 1$

i.e. here we are adding slope to P

But slope is nothing but $\frac{Dy}{Dx}$.

So the fractional part may come due to Dx. To eliminate this we will multiply the Equation (2) by Dx.

$$\therefore 2PDx - Dx > 0 \quad \ldots(3)$$

Now we will define $G = 2PDx - Dx$

$\therefore$ Our test will become as $G > 0$

Now we will see how this G can be used to decide which row or column to select.

$$\text{As } G = 2PDx - Dx$$

$$\therefore \quad G + Dx = 2PDx$$

$$\therefore \quad \frac{G + Dx}{2Dx} = P$$

Now if we consider $P = P + \text{slope}$, then

$$\text{We will get } \frac{G + Dx}{2Dx} = \frac{G + Dx}{2Dx} + \frac{Dy}{Dx}$$

If we further solve this equation then we will get

$$G = G + 2Dy \quad \ldots(4)$$

Similarly if we consider $P = P + \text{slope} - 1$, then

$$\text{We will get } G = G + 2Dy - 2Dx \quad \ldots(5)$$

To calculate this G we need only addition and subtraction and no multiplication or division. Another thing is now, this G is not containing any fractional part also.

So, we can use test $G > 0$ to determine when a row boundary is crossed by a line instead of $P > 0.5$. As the initial value of P is slope, so we have to find initial value of G also. We know from Equation (3) that

$$G = 2DxP - Dx$$

Put initial value of P as Dy/Dx to find initial value of G.

$$\therefore \quad G = 2Dx\left(\frac{Dy}{Dx}\right) - Dx$$

$$\therefore \quad G = 2Dy - Dx$$

For each column we check G. If it is positive we move to the next row and add (2 Dy – 2 Dx) to G, because it is equivalent to $P = P + \text{slope} - 1$, otherwise we will keep the same row and add (2 Dy) to G.

Steps for Bresenham's line drawing algorithm for Gentle slope (m < 1) :

1) Accept two endpoints from user and store the left endpoint in (x_0, y_0) as starting point.
2) Plot the point (x_0, y_0)
3) Calculate all constants from two endpoints such as Dx, Dy, 2Dy, 2Dy – 2Dx and find the starting value for the G as G = 2 Dy – Dx.
4) For each column increment x and decide y-value by checking G > 0 condition. If it is true then increment y-value and add (2Dy – 2Dx) to current value of G otherwise add (2Dy) to G and don't increment y-value. Plot next point. See Fig. 6.
5) Repeat step 4 till Dx times.

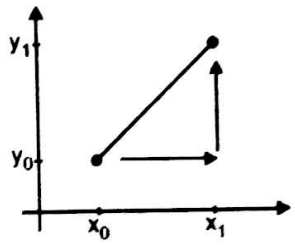


Fig. 6

For steep slope cases just interchange the rolls of x and y i.e. we step along y-direction in unit steps and calculate successive x-values nearest the line path. If the initial position for a line with positive slope is right endpoint, as shown in Fig. 7, then we have to decrease both x and y as we step from right to left.

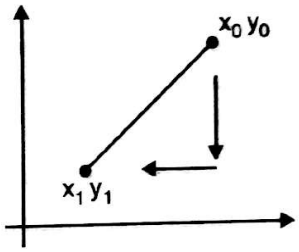


Fig. 7

Chapter 3 : Filled Area Primitives [Total Marks - 10]

Q. 2(b) Explain 4-connected and 8-connected methods. Also explain flood fill and boundary fill algorithms. **(10 Marks)**

Ans. :

Polygon Filling :

Filling is the process of "coloring in" a fixed area or region. Regions may be defined at pixel level or geometric levels. When the regions are defined at pixel level, we are having different algorithms like :

(i) Boundary fill (ii) Flood fill (iii) Edge fill (iv) Fence fill

In case of geometric level we are having scan line fill algorithm. We will first see pixel level polygon filling. But before going to actual algorithm, we will see 4-connected and 8-connected pixel concept. These two techniques are the ways in which pixels are considered as connected to each other.

In 4-connected method the pixels may have upto 4 neighbouring pixels. (See Fig. 8.). Let's assume that the current pixel is (x, y). From this pixel we can find four neighboring pixels as right, above, left and below of the current pixel.

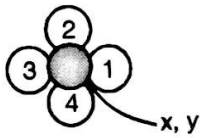


Fig. 8

Similarly in 8-connected method the pixels may have upto 8 neighbouring pixels. (See Fig. 9.) Here from one pixel location we are finding the neighbouring 8 pixels position.

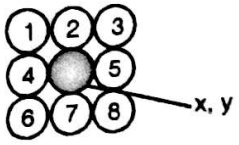


Fig. 9

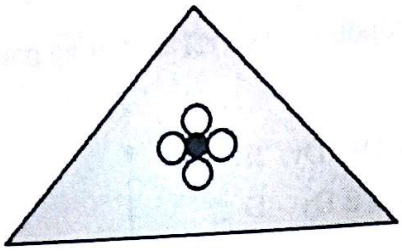


Fig. 10

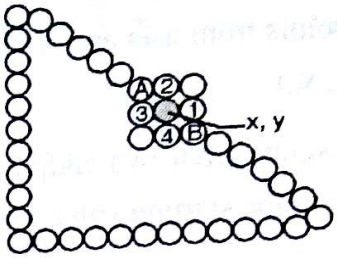


Fig. 11

By using any one of these technique we can fill the interior of the polygon. (See Fig. 10.) But there are some cases where the use of 4-connected method is not efficient. Suppose we want to fill a triangle, so we are using 4-connected method. It will work fine but at boundary this method is not efficient. (See Fig. 11). Suppose we have filled point (x, y). Now from that point we have to select either point A or B which is not possible by using 4-connected method because in 4-connected method we can go from (x, y) to either 1, 2, 3 or 4 and not A and B. So in this case we have to use 8-connected method where we can choose any one of the neighbouring pixel.

Boundary Fill Algorithm :

This algorithm is very simple. It needs one point which is surely inside the polygon. This point is called as "Seed point" which is nothing but a point from which we are starting the filling process. This is a recursive method. The algorithm checks to see if the seed pixel has a boundary pixels color or not. If the answer is no, then fill that pixel with color of boundary and make recursive call to itself using each of its neighbouring pixels as new seed. If the pixel color is same as boundary color then return to its caller. (See Fig. 12)

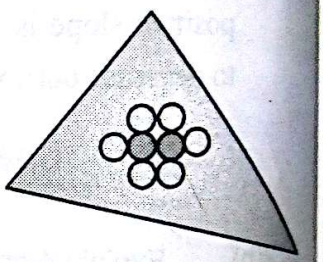


Fig. 12

This algorithm works for any shaped polygon and fills that polygon with boundary color. But as it uses high number of recursive calls it takes more time and memory. The following procedure illustrates a recursive method for boundary fill by using 4-conneced method.

```
bfill (x, y, newcolor)
{
    current = getpixel (x, y) ;
    if (current != newcolor) && (current != boundary color)
    {
        putpixel (x, y, newcolor);
        bfill (x + 1, y, newcolor);
        bfill (x – 1, y, newcolor);
        bfill(x, y + 1, newcolor);
        bfill(x, y – 1, newcolor);
    }
}
```

As we are using recursive function there is a chance that system stack may become full. So we have to develop our own logic to solve the stack problem. For this there are many solutions. One solution is instead of using system stack we can use our own stack and write our own PUSH and POP

functions for that stack. Second solution could be use of link list i.e. dynamic memory allocation, for implementation of stack. Another solution could be develop some logic while pushing the pixels on stack i.e. before pushing the pixel on stack check whether that pixel is already visited or not. If it is already visited there is no point to store that pixel again. Another solution could be instead of using stack we can make use of queue also. But the boundary fill algorithm is having limitation. It fills the polygon having unique boundary color. If the polygon is having boundaries with different colours then this algorithm fails.

Flood Fill / Seed Fill Algorithm :

The limitations of boundary fill algorithms are overcome in flood fill algorithm. Like boundary fill algorithm this algorithm also begins with seed point which must be surely inside the polygon. Now instead of checking the boundary color this algorithm checks whether the pixel is having the polygon's original color i.e. previous or old color. If yes, then fill that pixel with new color and uses each of the pixels neighbouring pixel as a new seed in a recursive call. If the answer is no i.e. the color of pixel is already changed then return to its caller. Sometimes we want to fill an area that is not defined within a single color boundary. (See Fig. 13).

Here edge AB, BC, CD and DA are having red, blue, green and pink color, respectively. This figure shows an area bordered by several color regions. We can paint such areas by replacing a specified interior color instead of searching for a boundary color value. Here we are setting empty pixel with new color till we get any colored pixel. Flood fill and boundary fill algorithms are somewhat similar. A flood fill algorithm is particularly useful when the region or polygon has no uniformed colored boundaries.

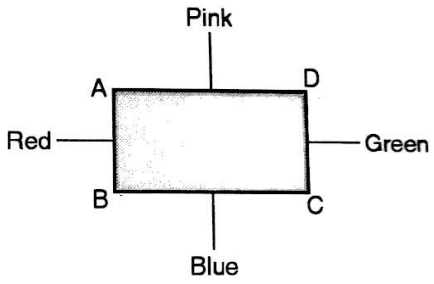


Fig. 13

The flood fill algorithm is sometimes also called as seed fill algorithm or Forest fire fill algorithm. Because it spreads from a single point i.e. seed point in all the direction. The following procedure illustrates a recursive method for flood fill by using 4-connected method.

```
f-fill (x, y, newcolor)
{   current = getpixel (x, y);
    if (current != newcolor)
    {       putpixel (x, y, newcolor);
            f-fill (x – 1, y, newcolor);
            f-fill ( x + 1, y, newcolor);
            f-fill (x, y – 1, newcolor);
            f-fill (x, y + 1, newcolor);
    }
}
```

The efficiency of this algorithm can be increased in the same way as discussed in boundary fill algorithm.

Chapter 4 : 2D Geometric Transformations [Total Marks - 15]

(5 Marks)

Q. 1(b) Explain Inverse Transformation.

Ans. : Inverse Transformation : The inverse transformation is very similar to matrix inversion. When any normal matrix (A) is multiplied with the inverse of same matrix (A^{-1}) then resultant will be identity matrix.

$$A * A^{-1} = I$$

Similarly when we multiply any transformation matrix with its inverse, we will get identity matrix. Now let us revise how to find the inverse of a matrix. First step for this is to find the determinant of the matrix. Suppose we want to find inverse of matrix A.

$$\det(A) = \det \begin{vmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{vmatrix}$$

$$= A_{11}(A_{22} \cdot A_{33} - A_{23} A_{32}) - A_{12}(A_{21} A_{33} - A_{23} \cdot A_{31}) + A_{13}(A_{21} \cdot A_{32} - A_{22} \cdot A_{31})$$

Then step two will be finding the cofactor of the matrix.

To find cofactor of

$$\begin{vmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{vmatrix}$$

We have to solve

$$B_{11} = \begin{vmatrix} A_{22} & A_{23} \\ A_{32} & A_{33} \end{vmatrix} \quad B_{12} = \begin{vmatrix} A_{21} & A_{23} \\ A_{31} & A_{33} \end{vmatrix} \quad B_{13} = \begin{vmatrix} A_{21} & A_{22} \\ A_{31} & A_{32} \end{vmatrix}$$

$$B_{21} = \begin{vmatrix} A_{12} & A_{13} \\ A_{32} & A_{33} \end{vmatrix} \quad B_{22} = \begin{vmatrix} A_{11} & A_{13} \\ A_{31} & A_{33} \end{vmatrix} \quad B_{23} = \begin{vmatrix} A_{11} & A_{12} \\ A_{31} & A_{32} \end{vmatrix}$$

$$B_{31} = \begin{vmatrix} A_{12} & A_{13} \\ A_{22} & A_{23} \end{vmatrix} \quad B_{32} = \begin{vmatrix} A_{11} & A_{13} \\ A_{21} & A_{23} \end{vmatrix} \quad B_{33} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix}$$

Then cofactor (A) will be

$$\begin{vmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{vmatrix}$$

After finding co-factor, next step is to find the adjoint of matrix A. To find this adjoint we have to take transpose of co-factor.

$$\text{i.e. Adj (A)} = \begin{vmatrix} B_{11} & B_{21} & B_{31} \\ B_{12} & B_{22} & B_{32} \\ B_{13} & B_{23} & B_{33} \end{vmatrix}$$

$$\text{Now inverse} = \frac{\text{Adj (A)}}{\text{determinant (A)}}$$

But in case of transformations we are having direct formula to find the inverse of transformations. For this we have to use homogeneous co-ordinates and we have to arrange the matrix in specific order.

$$\text{Inverse} \begin{vmatrix} a & d & 0 \\ b & e & 0 \\ c & f & 1 \end{vmatrix} = \frac{1}{(ae - bd)} \begin{vmatrix} e & -d & 0 \\ -b & a & 0 \\ (bf - ce) & (cd - af) & (ae - bd) \end{vmatrix}$$

Let us take an example so that the things will get more clear.

Q. 5(a) Explain the rotation of an object about an arbitrary point. Derive composite matrix for the same. **(10 Marks)**

Ans. : **Rotation about Arbitrary Point :**

We have derived rotation matrices with respect to origin. But suppose the reference point of rotation is other than origin, then in that case we have to follow series of transformation. Such transformation is also called as composite transformation.

Consider Fig. 14.

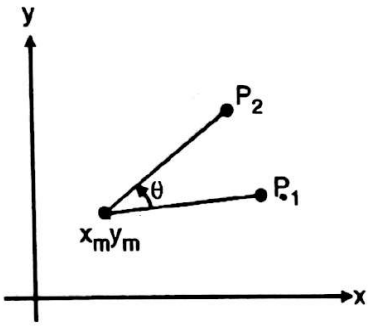


Fig. 14

Assume that we have to rotate a point P_1 with respect to (x_m , y_m) then we have to perform three steps. First we have to translate the (x_m , y_m) to origin as shown in Fig. 15. So our translation matrix (T_1) will become

$$T_1 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_m & -y_m & 1 \end{vmatrix} \Rightarrow$$

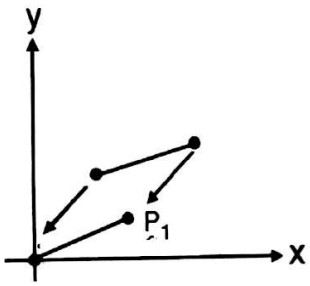


Fig. 15

Here $t_x = -x_m$ and $t_y = -y_m$

Then we have to follow second step i.e. rotate it in clockwise or anticlockwise. See Fig. 16. Let's assume as anticlockwise rotation by angle θ. So our rotation matrix will be

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

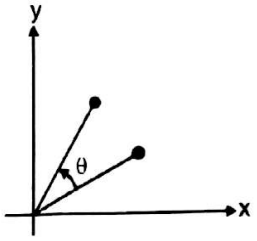


Fig. 16

Now follow 3rd step as shown in Fig. 17 i.e. translate back to original position. So the translation matrix (T_2) will become

$$T_2 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_m & y_m & 1 \end{vmatrix} \Rightarrow$$

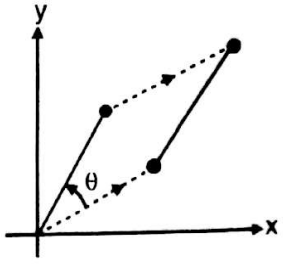


Fig. 17

Now let us form a combined matrix.

$$= \text{Translation} * \text{Rotation} * \text{Translation}$$

$$= T_1 * R * T_2$$

$$= \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_m & -y_m & 1 \end{vmatrix} * \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_m & y_m & 1 \end{vmatrix}$$

$$= \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_m & -y_m & 1 \end{vmatrix} * \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ x_m & y_m & 1 \end{vmatrix}$$

$$= \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -x_m * \cos\theta + y_m * \sin\theta + x_m & -x_m * \sin\theta - y_m \cos\theta + y_m & 1 \end{vmatrix}$$

This transformation matrix is the overall transformation matrix for rotation about arbitrary point (x_m, y_m) by an angle θ in anticlockwise direction.

Chapter 5 : 2D Viewing [Total Marks - 22]

Q. 3(b) Explain Liang Barsky line clipping algorithm. **(10 Marks)**

Ans. : Liang Barsky Line Clipping Algorithm :

The Liang-Barsky algorithm uses the parametric equation of a line and inequalities describing the range of the clipping box to determine the intersections between the line and the clipping box. With these intersections it knows which portion of the line should be drawn. This algorithm is significantly more efficient than Cohen-Sutherland. The ideas for clipping line of Liang-Barsky and Cyrus-Beck are the same. The only difference is Liang-Barsky algorithm has been optimized for an upright rectangular clip window. So we will study only the idea of **Liang-Barsky.**

Liang and Barsky have created an algorithm that uses floating-point arithmetic but finds the appropriate end points with at most four computations. This algorithm uses the parametric equations for a line and solves four inequalities to find the range of the parameter for which the line is in the viewport.

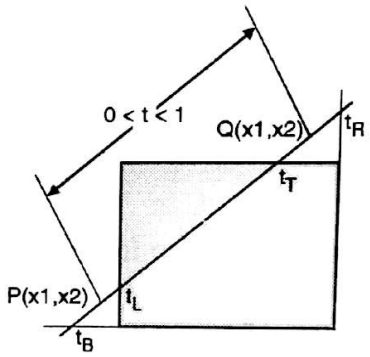


Fig. 18

Let P(X1,Y1), Q(X2,Y2) be the line which we want to study. Refer Fig. 18. The parametric equation of the line segment from gives x-values and y-values for every point in terms of a parameter tthat ranges from 0 to 1. The equations are

$$X = X1 + (X2 - X1)*t = Y1 + dY*t$$

And $$Y = Y1 + (Y2 - Y1)*t = X1 + dX*t$$

We can see that when t = 0, the point computed is P(x1, y1); and when t = 1, the point computed is Q(x2,y2).

Algorithm :

1. Set tmin=0 and tmax=1
2. Calculate the values of tL, tR, tT, and tB (tvalues).

   if t<tmin or t>tmax then ignore it and go to the next edge

   otherwise classify the tvalue as entering or exiting value (using inner product to classify)

   if t is entering value set tmin = t; if t is exiting value set tmax = t

3. If tmin<tmax then draw a line from

(x1 + dx*tmin, y1 + dy*tmin) to (x1 + dx*tmax, y1 + dy*tmax) If the line crosses over the window, you will see (x1 + dx*tmin, y1 + dy*tmin) and (x1 + dx*tmax, y1 + dy*tmax) are intersection between line and edge.

Example 1 :

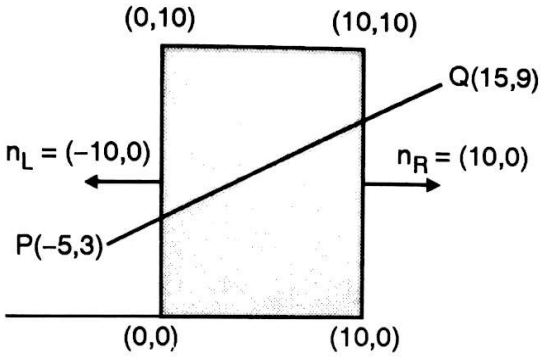


Fig. 19

Let us consider the case of Line Passing through Window in Fig. 19.

tB = (0 – 3) / (9 – 3) = – 1/2 t < 0 then ignore

tT = (10 – 3) / (9 – 3) = 7/6 t > 1 then ignore

tL = (0 – (– 5)) / (15 – (– 5)) = 1/4

tR = (10 – (– 5)) / (15 – (– 5)) = 3/4

The next step we consider if tvalue is entering or exiting by using inner product.

(Q – P) = (15 + 5, 9 – 3) = (20, 6)

At left edge

(Q – P)nL = (20, 6) (– 10,0) = – 200 < 0 entering so we set tmin = 1/4

Q. 4(a) Explain Sutherland-Hodgeman polygon clipping algorithm. HowWelier Athorton algorithm solves the problem of concave polygon clipping ? **(12 Marks)**

Ans. :

Sutherland-Hodgeman Polygon Clipping Algorithm :

We can clip a polygon by considering whole polygon against each boundary edge of the window. We know that to represent a polygon we need a set of vertices. We will pass this set of vertices or a polygon to a procedure which will clip the polygon against left edge of the window. This left clip procedure generates new set of vertices which indicates left clipped polygon. Again this new set of vertices is passed to the right boundary clipper procedure. Again we will get new set of vertices. Then we will pass this new set of vertices to bottom boundary clipper and lastly to top boundary clipper procedure.

---

easy solution

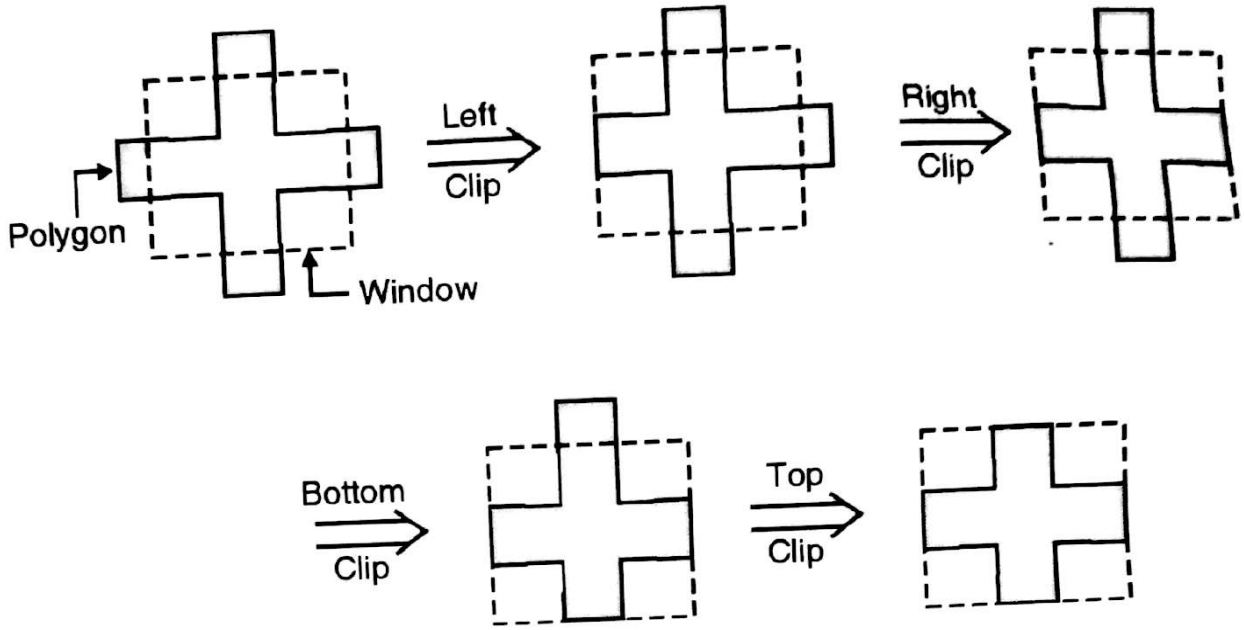


Fig. 20

Fig. 20 shows four different stages which are required to clip a polygon. At the end of every clipping stage a new set of vertices is generated and this new set or modified polygon is passed to the next clipping stage. After clipping a polygon with respect to all the four boundaries we will get final clipped polygon. When we clip a polygon with respect to any particular edge of the window at that time we have to consider following four different cases, as each pair of adjacent polygon vertices is passed to a window boundary clipper.

Case 1 :

If the first vertex is outside the window boundary and the second vertex is inside the window, then the intersection point of polygon with boundary edge of window and the vertex which is inside the window is stored in a output vertex list. (i.e. it will act as new vertex points). Refer Fig. 21.

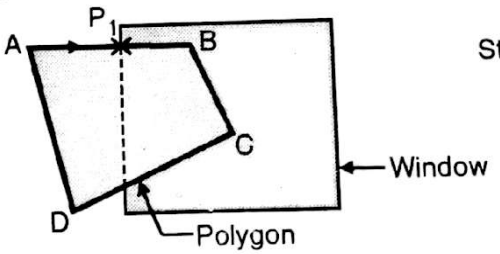


Fig. 21

For edge AB instead of storing vertex A and B we are storing P_1 and B in output vertex list.

Case 2 :

If both, first and second vertices of a polygon are lying inside the window, then we have to store the second vertex only in output vertex list. Refer Fig. 22.

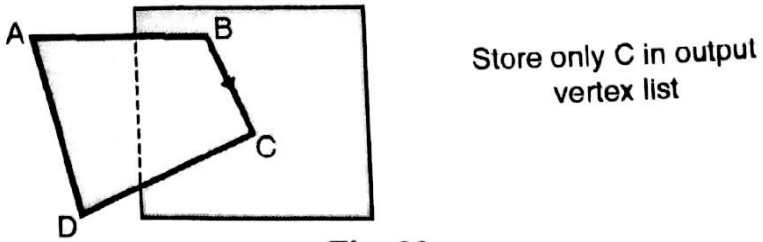
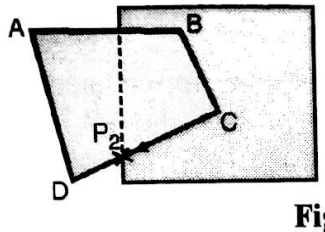


Fig. 22

Case 3 :

If the first vertex is inside the window and second vertex is outside the window i.e. opposite to case 1, then we have to store only intersection point of that edge of polygon with window in output vertex list. Refer Fig. 23.

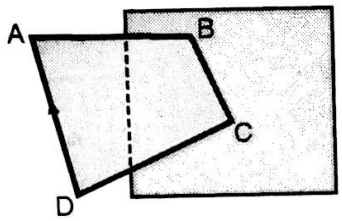


Store P_2 only in output vertex list

Fig. 23

Case 4 :

If both first and second vertices of a polygon are lying outside the window then no vertex is stored in output vertex list. Refer Fig. 24.



Nothing is stored in output vertex list

Fig. 24

Once all vertices have been considered for one clip window boundary, the output list of vertices is clipped against the next window boundary. So the block diagram for this algorithm will be as shown in Fig. 25. This block diagram is a self explanatory.

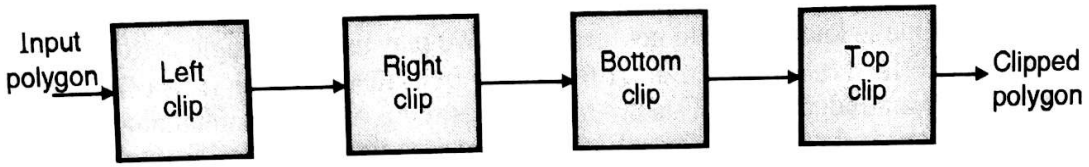


Fig. 25

Weiler-Atherton Polygon Clipping :

This algorithm is one of the generalized polygon clipping algorithm. In Sutherland-Hodgeman algorithm, we always follow the path of polygon. But in this case sometime we are selecting polygon path and sometime window path. When to follow which path, that will depend on polygon processing direction, and whether a pair of polygon vertices currently being processed represents an outside to inside pair or an inside to outside pair. We can follow polygon processing path either clockwise or anticlockwise.

When we are following a path of polygon in clockwise direction at that time we have to use following rules. We have to follow the polygon boundary, same as that of Sutherland-Hodgeman algorithm, if the vertex pair is outside to inside. We have to follow the window boundary in clockwise direction, if the vertex pair is inside to outside. Consider the concave polygon as shown in Fig. 26. Let us process the polygon in clockwise path. Here our vertex list will be the set of all vertices i.e. {A, B, C, D}.

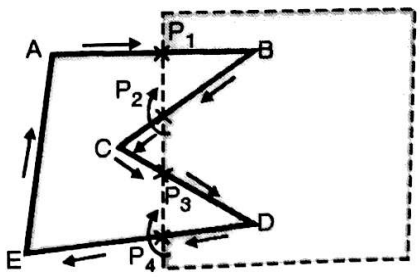


Fig. 26

For edge AB, as we are moving from outside to inside so store intersection point P_1 and second point i.e. B. As we are following polygon in clockwise direction, so we have to consider next edge as BC. Here we are moving from inside to outside. So we are storing only intersection point P_2. Upto this point it is similar to normal polygon clipping algorithm. But, as we are moving from inside to outside so we have to follow window boundary in clockwise direction. And the next point on this path will be P_1. So we have to store that. It means we are storing P_2 and P_1. For next edge i.e. CD, again we are storing intersection point P_3 and next point D. For edge DE we are storing intersection point P_4 and following window boundary in clockwise direction i.e. storing P_3. For edge EA, as both points are outside, no vertex is stored. See Fig. 27.

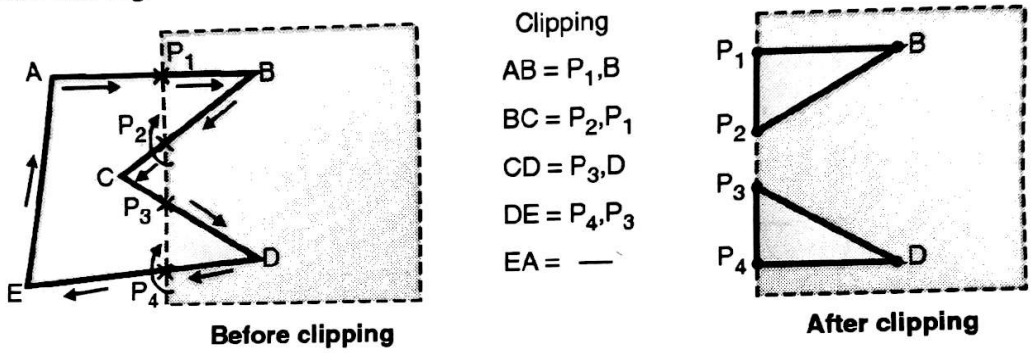


Fig. 27

Here in Weiler-Atherton algorithm we are maintaining as such two different vertex list in single list. Here after clipping, the vertex list will be $\{P_1, B, P_2, P_1, P_3, D, P_4, P_3\}$. We have to draw edges by joining vertices as P_1 to B, B to P_2, P_2 to P_1. Now there is no need to form an edge between P_1 to P_3. We have to continue with edge P_3 to D, D to P_4 and P_4 to P_3. Again here there is no need to close the figure by drawing edge between P_3 to P_1 because already the figure is closed. Same thing is for edge P_1 to P_3. Here how we will come to know when to not form edge. We may use some logic here such as, when any vertex is stored twice in vertex list then don't form edge from that vertex to next vertex. Now we will not have edge P_1 P_3 and edge P_3 P_1. That's why we are saying, we are maintaining two sub array in single array of vertex.

Chapter 6 : 3D Geometric Transformations and Viewing [Total Marks - 20]

Q. 3(a) Give and Explain types of perspective projection. **(10 Marks)**

Ans. : Types of Perspective Projections :

Vanishing point on the view plane is referred as a point where all the lines which are not parallel to view plane are appeared to meet. There are different types of perspective projections based on the number of vanishing points. We can have one point, two point and three point perspectives.

1. One point perspective :

One point perspective occurs when the projection plane is parallel to two principal axes. Conversely, when the projection plane is perpendicular to one of the principal axis, one point perspective occurs. Receding lines along one of the principal axis converge to a vanishing point. Refer Fig. 28.

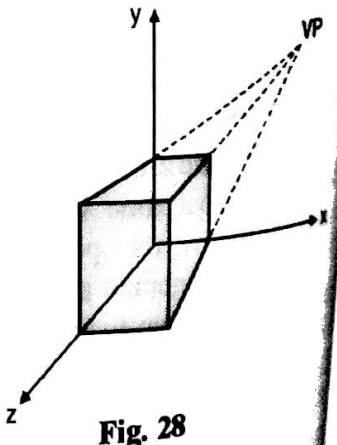


Fig. 28

2. Two point perspective :

If the projection plane is parallel to one of the principal axes or if the projection plane intersects exactly two principal axes, a two-point perspective projection occurs. Refer Fig. 29.

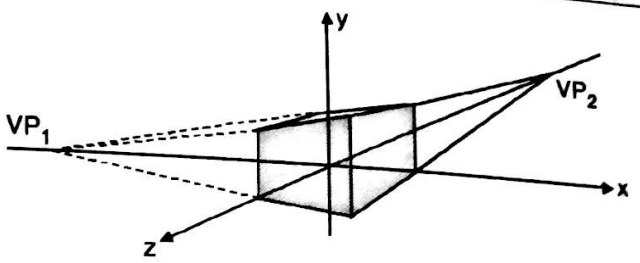


Fig. 29

3. Three Point Perspective :

If the projection plane is not parallel to any principal axis, a three-point projection occurs. Refer Fig. 30.

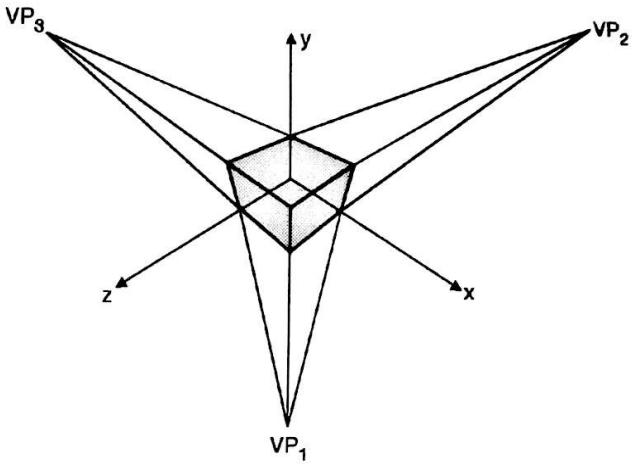


Fig. 30

Q. 5(b) Describe 3D clipping of an object. **(10 Marks)**

Ans. : 3D Clipping :

3D clipping is conceptually same as 2D clipping. In 2D we are clipping the lines or objects by using clipping window. But in 3D we are using Clipping Volume or View Volume. In simple terms we can say it is a 3D box. The objects within this 3D box or view volume may be seen, while those outside are not displayed. It means we are clipping the portion of object which lie outside the 3D box. As view volume is nothing but a box, we are clipping the object with respect to six sides. i.e. in addition to left clip, right clip, bottom clip and top clip we have to use front clip and back clip also.

In 3D clipping, left and right clipping deals with x co-ordinates; top and bottom clipping deals with y co-ordinates and front and back clipping deals with z co-ordinates. Here we require the front plane to be parallel to the view plane, which normally coincides with the screen. This view volumes contents will be projected onto the screen, which acts as a 2D window.

Different types of projections create different shapes of view volumes. In case of orthographic parallel projections, the view volume is a rectangular box whose front plane is parallel to the view plane. Fig. 31 shows orthographic view volume.

Fig. 31 shows view volume and its projections on XY plane. Here meaning of window is same as that of 2D. Remember that this window is on XY plane. So the portion of object which lies outside the window should be discarded.

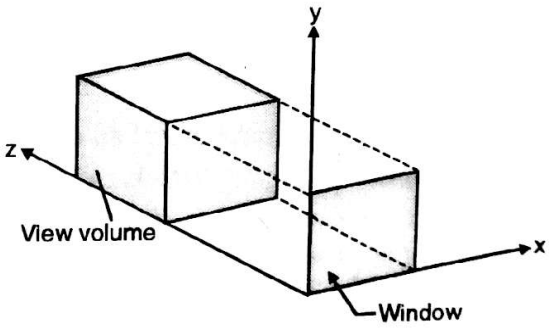


Fig. 31

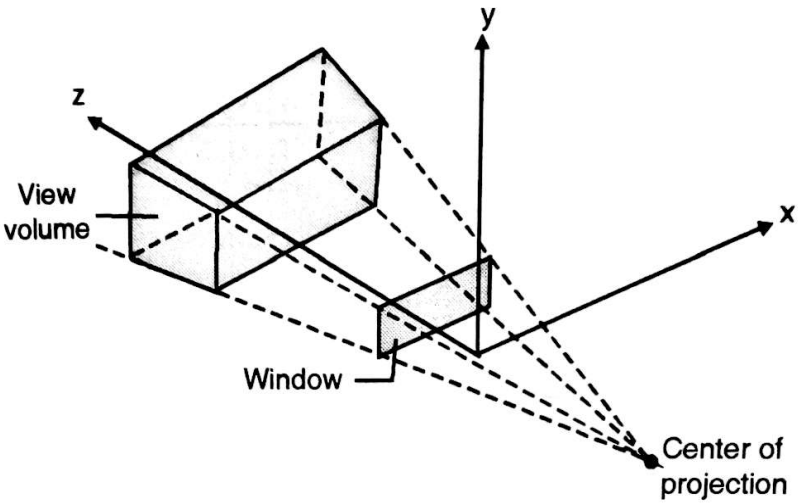


Fig. 32

Similarly in case of perspective projections, the shape of view volume will be truncated pyramid. The pyramid's apex will be the center of projection. Fig. 32 shows perspective view volume. The front plane is parallel to the view plane, but it must lie in front of the center of projection. The objects which are inside the truncated pyramid are visible.

The concept of 2D clipping is extended to achieve 3D clipping. IN 3D, depth clipping is the most important factor. Here we are having two options. First option will be, clip a 3D object in space only and then project that clipped object on XY plane, but clipping in space is conceptually more difficult and it require lot of computations so we will select second option. Here we are clipping only depth of the object in space. Then we will project that depth clipped object on view plane and apply our normal 2D clipping algorithm.

Parallel projections :

For parallel projections we are forming a rectangular box as view volume. This box will be in space. See Fig. 33.

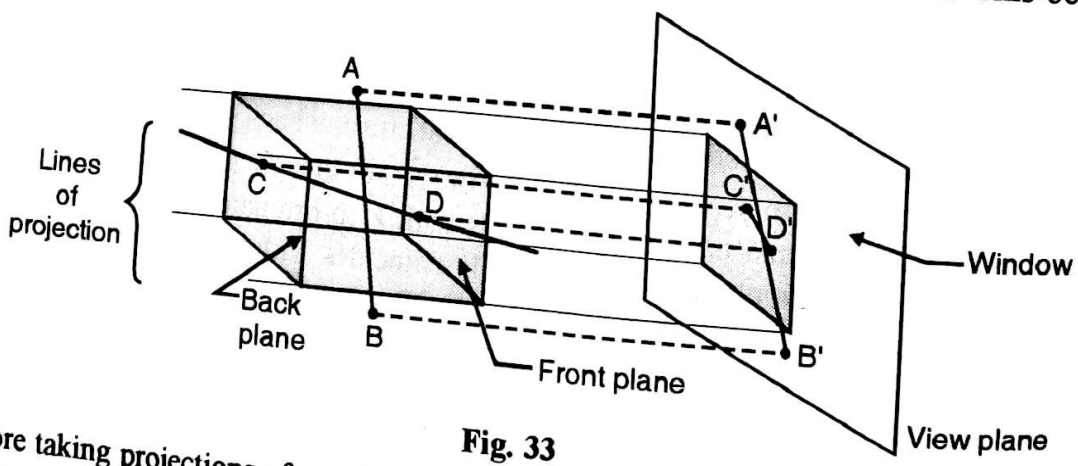


Fig. 33

Before taking projections of an object first we are going to clip the depth i.e. we are comparing all z co-ordinates of object with back plane and front plane. The points or part of object which lies beyond back plane and in front of the front plane are ignored. The portion of object which is in between back and front plane will be considered for further processing. Remember that we are not clipping object with respect to top, bottom right or left boundary. If we keep the window parallel to z-axis with $z = 0$, then we will get image on XY plane, which is similar to 2D clipping.

Refer Fig. 33, where we are forming a rectangular box and we are having two lines, AB and CD. When we are considering depth clipping, we are checking depth of end points of lines with back and front plane. As point C is beyond back plane and point D is in front of front plane, so we are clipping line CD with respect to back and front plane. Similarly we need to do it for line AB also. But AB is not beyond back plane and not in front of front plane, so there is no question of clipping line AB (Even though line AB is not completely inside view volume). Then we are taking projections of clipped line CD and line AB on view plane; at the same time we are taking projection of 3D box also on view plane which will act as window.

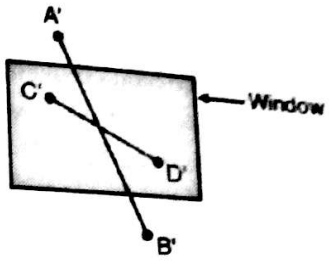


Fig. 34

Now we can apply our normal 2D clipping algorithm to clip the objects four sides. After taking projection of lines AB and CD on view plane, we will get Fig. 34.

Now we can apply our normal Cohen-Sutherland line clipping algorithm. It means the whole clipping process will be as shown in Fig. 35.

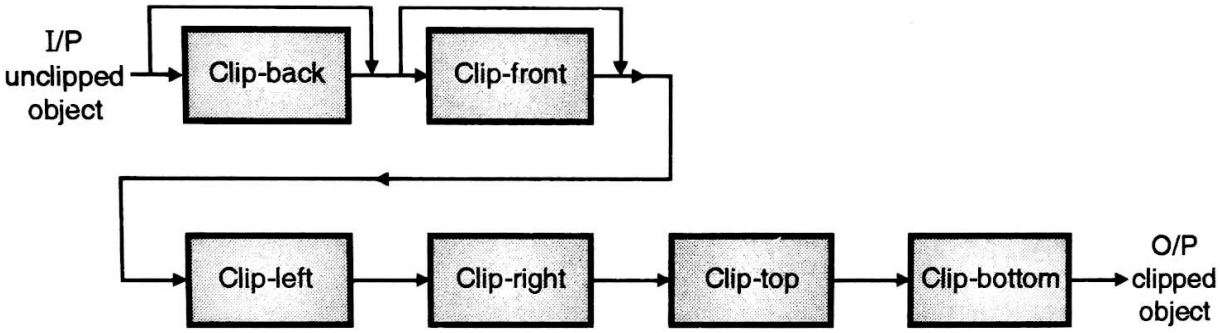


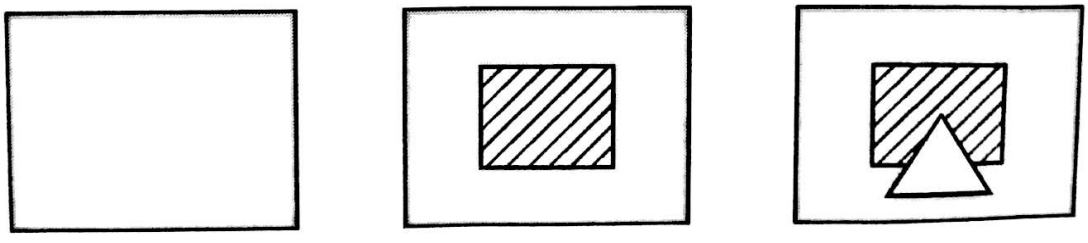
Fig. 35

Chapter 7 : Hidden Surfaces [Total Marks - 10]

Q. 6(a) Explain Painter's Algorithm. **(10 Marks)**

Ans. : Painter's Algorithm :

The object surfaces that are away from the viewers are called as back faces. So we have to identify and remove those faces. For this there are many algorithms. Painter's algorithm is one of them. The painter's algorithm is also called as depth sorting algorithm or priority algorithm. The painter's algorithm processes polygons as if they are painted onto the view plane, in order of their distance from the viewer. It is similar to a work of painter. Painter first paints whole canvas by some color, which will be background color of picture. Then on top of this he paints with other colors or objects. This will be his foreground picture. See Fig. 36.



(a) (b) (c)

Fig. 36

In Fig. 36(a) whole canvas is painted first then Fig. 36(b) shows on top of this canvas a rectangle is painted and Fig. 36(c) shows on top of this a triangle is painted. It gives appearance as the triangle is nearer to viewer than rectangle. It means more distance polygons are painted first and nearer distance polygons are painted over more distance polygons, partially or totally hiding them from view.

The main factor over here is to decide the priority of the polygons to determine which polygons are to be painted first. It means when two polygons overlap, how to decide which polygon hides the other. For painters algorithm uses a technique called as **minimax test** or boxing test. From this test we are immediately sorting or isolating polygons which are overlapping (i.e. hiding by other polygons) and which polygons are not overlapping. The minimax test says that, if we place both the polygons in two different boxes and if these boxes are not overlapping, then it means the polygons within these boxes are not overlapping. To make efficient use of this test the bounding box must be as small as possible and still contain the polygon. It should not be as shown in Fig. 37(a). But should be as shown in Fig. 37(b).

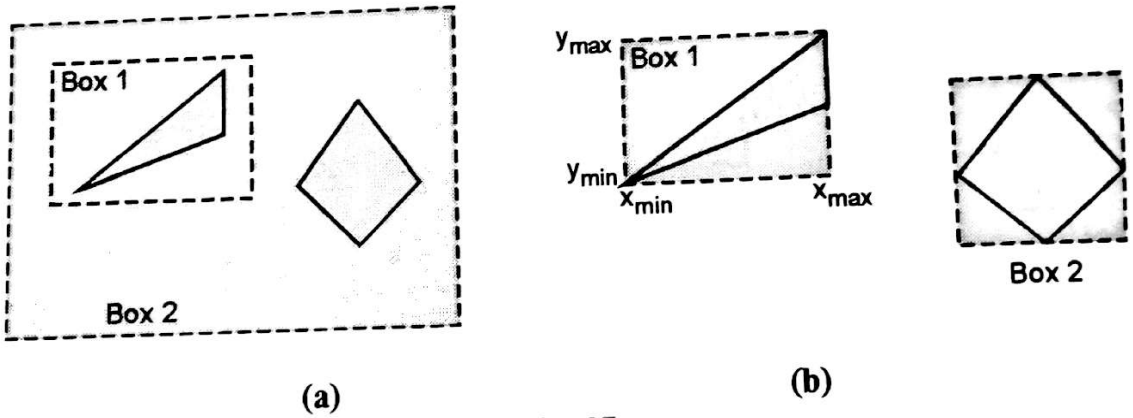


Fig. 37

It means we have to form a box with boundaries as x_{max}, x_{min}, y_{max} and y_{min} of that polygon. See box 1 of Fig. 37(b). Now we will apply the test. If the two boxes are one over the another with same x_{min} and x_{max} values but different y_{min} and y_{max} values, as shown in Fig. 38, then two polygons which are inside these boxes are not overlapping.

Here, x_{min} and x_{max} are same for both boxes.

But $y_{min\,(1)} > y_{max\,(2)}$

It means both polygons are visible, as there is no overlapping.

Similarly, if the two boxes are one next to another, with same y_{min} and y_{max} values but different x_{min} and x_{max} values, as shown in Fig. 38, then both polygons are visible.

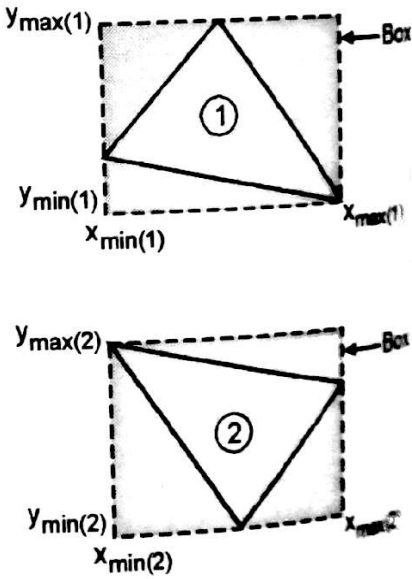


Fig. 38

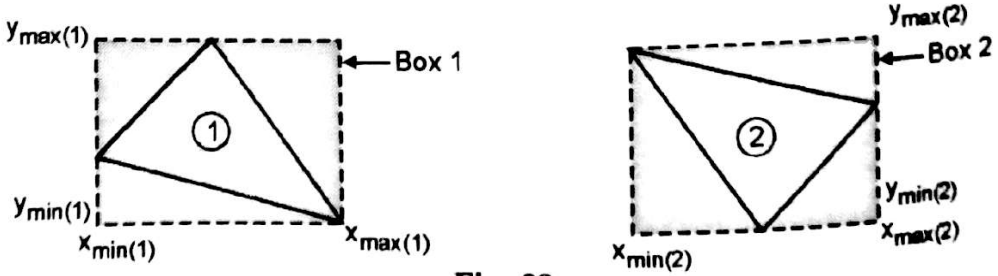


Fig. 39

Here, y_{min} and y_{max} are same for both boxes.

But $x_{min\,(2)} > x_{max\,(1)}$

It means both polygons are not overlapping. So both are visible.

Chapter 8 : Illumination Models and Surface Rendering [Total Marks - 33]

Q. 1(c) Explain Ray Tracing. **(5 Marks)**

Ans. : If we consider line of sight from a pixel position on the view plane through a scene, as in Fig. 40, we can determine which objects in the scene (if any) intersect this line. From the intersection points with different objects, we can identify the visible surface as the one whose intersection points is closest to the pixel. 'Ray Tracing' is an extension of this basic idea. Here, instead of identifying for the visible surface for each pixel, we continue to bounce the ray around the picture. This is illustrated in Fig. 41 when the ray is bouncing from one surface to another surface it contributes the intensity for that surfaces. This is simple and powerful rendering technique for obtaining global reflection and transmission effects.

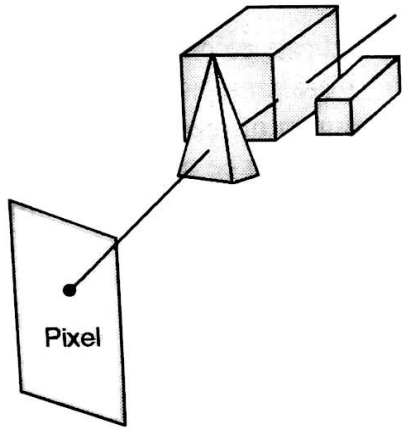


Fig. 41 : Bouncing of ray around the scene

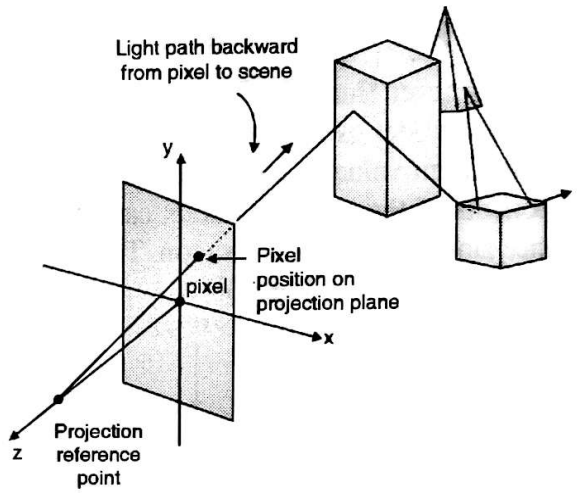


Fig. 40 : A ray along the line of sight from a pixel position through a scene

As shown in Fig. 41, usually pixel positions are designated in the xy plane and projection reference point lie on the z axis, i.e. the pixel screen area is centered on viewing co-ordinate origin. With this co-ordinate system the contributions to a pixel is determined by tracing a light path backward from the pixel to the picture. For each pixel ray, each surface is tested in the picture to determine if it is intersected by the ray. If surface is intersected, the distance from the pixel to the surface intersection point is calculated. The smallest calculated intersection distance identifies the visible surface for that pixel. Once the visible surface is identified the ray is reflected off the visible surface along a specular path where the angle reflection equals angle of incidence. If the surface is transparent, the ray is passed through the surface in the refraction direction.

The ray reflected from the visible surface or passed through the transparent surface in the refraction direction is called 'secondary ray', The ray after reflection or reflection strikes another visible surface. This process is repeated recursively to produce the next generations of reflection and refraction paths. These paths are represented by 'ray tracing tree' as shown in Fig. 41. As shown in Fig. 42, the left branches in the binary ray tracing tree are used to represent reflection paths, and right branches are used to represent transmission path. The recursion depth for ray tracing tree is determined by the amount of storage available, or by the user. The ray path is terminated when predetermined depth is reached or if ray strikes a light source. As we go from top to bottom of the tree, surface intensities are attenuated by

easy solution

distance from the parent surface. The surface intensities of all the nodes are added traversing the ray tre from bottom to top to determine the intensity of the pixel.

If pixel ray does not intersect to any surface then the intensity value of the background is assigned the intensity of the source, although light the pixel. If a pixel ray intersects a non reflecting light source, the pixel can be assigned the intensity of the source, although light sources are usually placed beyond the path of the initial rays.

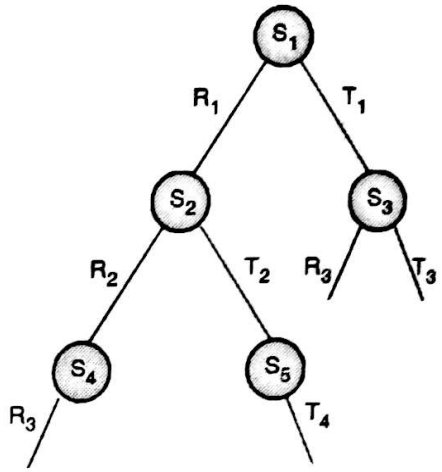
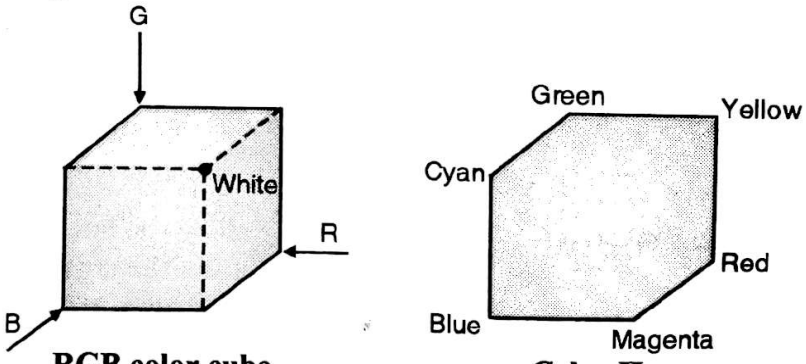


Fig. 42 : Binary ray tracing tree

Q. 4(b) Describe HSV and RGB color model. **(8 Marks)**

Ans. : HSV and HLS Color Models :

Whenever Hue (H), Saturation (S) and Value (V) are used to indicate colors at that time we are calling that model as HSV color model. Here the meaning of H and S are same as that of HSI model. Here new term value is introduced. Value is the intensity of the maximum of the red, blue and green components of the color. Value is easier for calculation. For that the hue can be approximated by the distance along the edge of hexagon. The HSV color model representation is derived from the RGB color model and is shown in Fig. 43.



RGB color cube **Color Hexagon**

Fig. 43

View the cube along its sides, giving as a shape of hexagon. This boundary represents the various hues and used as a top of HSV hexagon. See Fig. 44. In the Hexagon the saturation is measured along the horizontal axis and value is along the vertical axis through the centre of hexagon. Hue is represented as an angle about the vertical axis from 0° at red through 360°. As it is hexagon the vertices are separated at 60° intervals. Saturation (S) varies from 0 to 1. S for this model is the ratio of purity of a selected hue to its maximum purity at 1. The value (V) also varies from 0 to 1, when v = 0 it indicates black and white when, V = 1. At the top of hexagon, the colors have the maximum intensity.

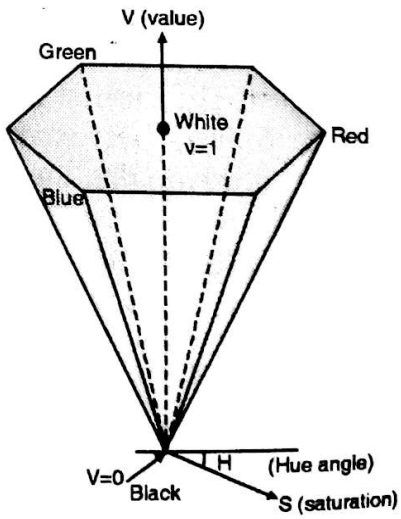


Fig. 44

Hue selection is done by hue angle H with V = S = 1. The user describing the color desired in terms of adding either white or black to the pure hue.

RGB Color Model :

The Red-Green-Blue (RGB) model is generally used in computer graphics. It corresponds to Red, Green and Blue intensity settings of a color monitor. We can represent this model with the unit cube defined on R, G and B axes. The origin represents black and the vertex with co-ordinates (1, 1, 1) is white. See Fig. 45.

RGB color model is just an extension of XYZ model. Other colors are generated by adding intensities of primary colors, such as yellow (1,1,0) is a combination of Red and Green. Each color is represented by a triple (R, G, B). Chromacity co-ordinates for standard color television i.e. NTSC standard and CIE RGB color model are represented in tabular form.

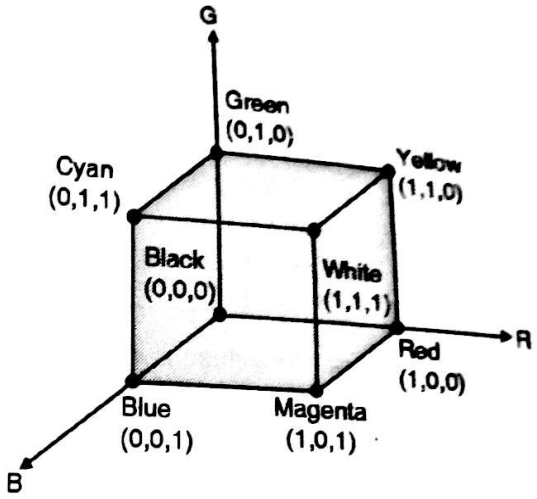


Fig. 45

Q. 7(a) Explain and compare Phong Shading and Gaurard Shading. **(10 Marks)**

Ans. :

Phong shading :

This method involves approximation of the surface normal at each point along a scan line, then calculating the intensity using the approximated normal vector at that point, displaying more realistic highlights on a surface. Phong shading method performs following steps to render a polygon surface.

1. At each vertex of polygon, determine average unit normal vector.
2. Linearly interpolate the vertex normal.
3. Calculate the pixel intensity of each scan line.

Gaurard shading :

Gouraud shading is an example of intensity interpolation of method. Here polygon surfaces are rendered by linearly interpolating intensity values. This method removes intensity discontinuities between adjacent planes of a surface representation by linearly varying the intensity over each plane. Here intensity values are matched at plane boundaries.

In Gouraud shading all polygon surfaces are accessed by performing following calculations :

1. At each vertex of polygon, determine average unit normal vector.
2. Calculate vertex intensity.
3. Linearly interpolate that vertex intensity over the surface of polygon.

This method interpolates the normal vectors at the bounding points along a scan line.

Sr. no.	Phong shading	Gouraud shading
1.	It displays more realistic highlights on a surface.	It removes the intensity discontinuities exist in constant shading model.
2.	It greatly reduces the Mach - band effect.	It can be combined with a hidden surface algorithm to fill in the visible polygons along each scan line.

Sr. no.	Phong shading	Gouraud shading
3.	It gives more accurate results	Highlights on the surface are sometimes displayed with anomalous shapes.
4.	It requires more calculations and greatly increases the cost of shading steeply	The linear intensity interpolation can result bright or dark intensity streaks to appear on the surface. These bright or dark intensity streaks are called 'Mach bands.' The Mach band effect can be reduced by breaking the surface into a greater number of smaller polygons.

Q. 7(b) Describe diffuse illumination and point-source illumination. **(10 Marks)**

Ans. :

Diffuse Illumination :

When a light falls on any surface, it can be absorbed by the surface, while the rest will be reflected or retransmitted. In diffuse illumination, incoming light is not reflected in a single direction but is scattered almost in all directions. The part of incoming light will be absorbed by the surface. The light which is not absorbed will be reflected randomly in all directions. See Fig. 46.

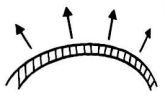


Fig. 46

The ratio of light reflected from the surface to the total incoming light is called the coefficient of reflection or reflectivity (Kd). The white surface is having its reflectivity close to 1, hence it reflects almost all incoming light. Whereas the black surface absorbs most of the incoming light since its reflectivity is near to 0. So the value of reflectivity (K_d) is assigned in the range of 0 to 1, depending on the nature of surface. Intensity of the diffuse reflection at any point when surface is exposed to ambient light is given as,

$$I = K_d \cdot I_a$$

Where I_a = Intensity of ambient light

K_d = Coefficient of reflection.

Shading model calculating intensities according to above equation will shade all visible surfaces of an object with same intensity. Realistic shading is obtained by including the effect of point sources in the shading model.

Point - Source Illumination :

Light from point source differs from ambient light. The light originate from a fixed place, and it comes from a particular direction. Here the position and orientation of the objects surface relative to the light source will determine how much light the surface will receive. Surfaces which are near the light source and facing towards light will receive more light. The illumination is decreased by a factor of cos θ, where θ is the angle between the direction of light (L) and the direction normal (N) to the plane. This angle θ is called as angle of incidence which is shown in Fig. 47.

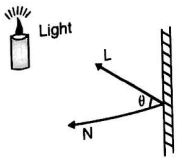
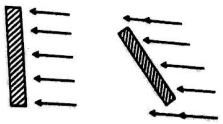


Fig. 47

As the angle between direction of light and direction normal i.e. incidence angle increases, the incident light which falls on the surface will be less. Fig. 48 shows the difference between more illuminated and less illuminated objects because of angle of incidence.



(a) More illuminated **(b) Less illuminated**

Fig. 48

If we denote angle of incidence as θ, the amount of illumination depends on cos θ. If I_p is the intensity of the point source light, then the diffuse reflection will be.

$$I = K_d \cdot I_p \cdot \cos\theta$$

If the angle of incidence i.e. θ is in the range of 0 to 1 then only the surface will be illuminated by the point source. But if we place the value of cos θ in above equation then we will get,

$$\cos\theta = N \cdot L \quad \text{(from Fig. 47)}$$

$$\therefore I = K_d \cdot I_p \cdot (N \cdot L)$$

Now we can combine the equations of ambient (diffuse illumination) and point source to obtain an expression for total diffuse reflection.

$$\therefore I = K_a \cdot I_a + K_d \cdot I_p (N \cdot L)$$

where both K_a and K_d will depend on surface material and will be in the range of 0 to 1.

Chapter 9 : Curves and Fractals [Total Marks - 10]

Q. 6(b) Explain the bezier curve and write the properties of bezier curve. **(10 Marks)**

Ans. : Bezier Curve :

It is a different way of specifying a curve, rather same shapes can be represented by B-spline and Bezier curves. The cubic Bezier curve require four sample points, these points completely specify the curve. Fig. 49 shows sample Bezier curves.

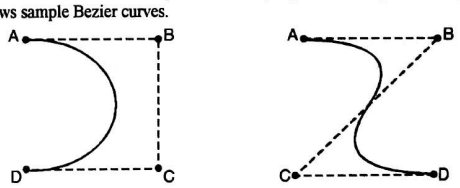


Fig. 49

The curve begins at the first sample point and ends at fourth point. If we need another Bezier curve then we need another four sample points. But if we need two Bezier curves connected to each other, then with six sample points we can achieve it. For this, the third and fourth point of first curve should be made same as first and second point of second curve.

The equations for the Bezier curve arc as follows :

$$x = x_4 a^3 + 3x_3 a^2 (1-a) + 3x_2 a (1-a)^2 + x_1 (1-a)^3$$

$$y = y_4 a^3 + 3y_3 a^2 (1-a) + 3y_2 a (1-a)^2 + y_1 (1-a)^3$$

$$z = z_4 a^3 + 3z_3 a^2 (1-a) + 3z_2 a (1-a)^2 + z_1 (1-a)^3$$

Here as the value of 'a' moves from 0 to 1, the curve travels from the first to fourth sample point. But we can construct a Bezier curve without referencing to the above expression. It is constructed by simply taking midpoints. See Fig. 50.

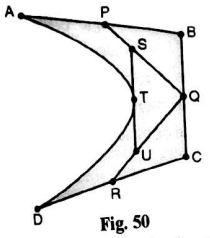


Fig. 50

The points A, B, C, D are the original Bezier curve control points. Here we are having three lines, AB, BC and CD, then we have to find the midpoints of these lines as 'P', 'Q' and 'R' respectively. After that we have to join PQ and QR. Then again find the midpoint of these newly generated lines as 'S' and 'U'. Then form a line segment between 'S' and 'U'. And find midpoint of this line as 'T'. Now point 'T' will be on Bezier curve. This point 'T' divides the curve into two section, one is (A, P, S and T) and second will be (D, R, U and T). Thus by taking midpoints, we can find a point on the curve and also split the curve into two sections. We can continue to split the curve into smaller sections, until we have sections so short that they cannot be replaced by straight lines or till the size of section is not greater than the size of pixel.

Properties of Bezier Curve :

1. The basic functions are real in nature.
2. Bezier curve always passes through the first and last control points i.e. curve has same end points as the guiding polygon.
3. The degree of the polynomial defining the curve segment is one less than the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is three, i.e. cubic polynomial.
4. The curve generally follows the shape of the defining polygon.
5. The direction of the tangent vector at the end points is the same as that of the vector determined by first and last segments.
6. The curve lies entirely within the convex hull formed by four control points.
7. The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
8. The curve exhibits the variation diminishing property. This means that the curve does not oscillate about any straight line more than the defining polygon.
9. The curve is invariant under an affine transformation.

□□□

Dec. 2012

Chapter 1 : Introduction to Computer Graphics [Total Marks - 15]

Q. 2(a) What is a display file structure? Also explain the need for display file interpreter. **(5 Marks)**

Ans. : **Display File and its Structure :**

In the raster grasphics displays pictures are stored in frame buffer which holds information about all pixels whereas in vector refresh displays we use commands to store the picture which we want to display. So in vector refresh displays we are storing commands i.e. input instead of output. The file in which we are storing these commands to draw picture is called Display file.

In this display file we are going to store command by proper way. Each command in display file, we are breaking it into two parts : opcode and operand.

Opcode is nothing but a command, which will be like Move or Line etc. and operand will be parameters.

e.g. move (10, 10)

Here we will break this command in opcode and operand as,

Opcode	Operand
Move	10, 10

Uptill now we have seen only two primitive command Move and Line. So initialize opcode 1 for Move command and opcode 2 for Line command.

Opcode	Command
1	Move
2	Line

Display File Interpreter :

The display file will contain the information necessary to construct the picture. Saving the instructions usually takes much less storage than saving the picture itself. These instructions are nothing but a sample programs. Each instruction like move or line indicates some action for display device. So we need something to convert these instructions into actual images. That conversion is done by display file interpreter. It interprets each command from display file into some action on display. (See Fig. 1. Interpreter may be a part of machine which executes these instructions. The result of execution is image. In some graphics systems there is separate computer which handles this job which is called **display processor**.

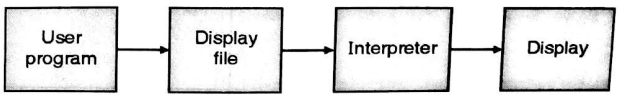


Fig. 1

Display file interpreter serves as an interface between our graphics program and the display device.

Q. 3(c) Write different applications of computer Graphics. **(5 Marks)**

Ans. : **Please refer Q. 1(a) of May 2012.**

Q. 7(e) Write short notes on : Raster techniques. **(5 Marks)**

Ans. :

There are two different ways of scanning video displays. These are called as raster and random scans. In raster scan the electron beam follows a fixed path. The electron beam starts at top left corner of the screen and moves horizontally to the right. This defines a scan line. During the scan the intensity of the beam is modulated according to the pattern of the desired image along the line. At the right corner of the screen, the beam becomes off and moved back to the left edge of the screen at the starting point on next line. This is shown as dotted line, which is called as horizontal retrace. This way of scanning is continued till the bottom right corner is reached. At this point, one scan is completed. The beam is then repositioned at the top left corner of the screen for starting another scan. This movement of beam from bottom right corner to top left corner is called as vertical retrace.

Raster graphics can be used in animation in raster scan cathode ray tubes (CRT) are used. Since the cost of devices used for random scan is much higher than the cost of devices for raster scan, people are using raster scan devices heavily.

Chapter 2 : Output Primitives [Total Marks - 10]

Q. 3(b) Derive Bresenham's line drawing algorithm for lines with slop < 1. **(5 Marks)**

Ans. : **Please refer Q. 2(a) of May 2012.**

Q. 7(a) Write short notes on : Thick line generation. **(5 Marks)**

Ans. : Thick Line Generation :

We can produce thick lines also, whose thickness is greater than one pixel. To produce a thick line segment, we can run two line drawing algorithms in parallel to find the pixels along the line edges. As we are moving to next pixel of line 1 and line 2, we must also turn on all the pixels which lie between the boundaries. See Fig. 2.

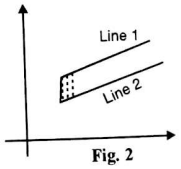


Fig. 2

To be more specific, if we want to draw a gentle slope line from (x_a, y_a) to (x_b, y_b) with thickness w, as shown in Fig. 3., then the corner points of thick line become.

$(x_a, y_a + w_y)$, $(x_b, y_b + w_y)$, $(x_a, y_a - w_y)$, $(x_b, y_b - w_y)$

Where w_y can be calculated as,

$$w_y = \frac{w-1}{2}\left(\frac{((x_b - x_a)^2 + (y_b - y_a)^2)^{1/2}}{|x_b - x_a|}\right)$$

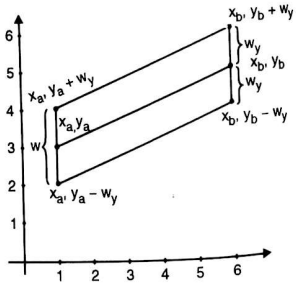


Fig. 3

W_y is the amount by which the boundary line is moved from the center. (w – 1) factor is the desired width minus one pixel thickness, we automatically receive from drawing the boundary. We divide this by 2 because half the thickness will be used to offset the top boundary and other half to move the bottom boundary.

The factor containing x and y-values is needed to find the amount to shift up and down in order to achieve proper width w. The above mentioned is one of the general method. For special thing we can have other alternatives also. Such as if thickness required is 2 and slope of line is < 1 i.e. line in gentle slope category. See Fig. 4. Then we can simply plot pixel or draw another line next to actual line.

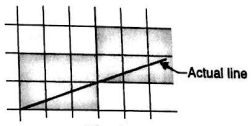


Fig. 4

If thickness is ≥ 3 then alternately plot the pixels above and below the actual line. If slope is > 1 i.e. line is steep slope category then pick the pixels which are at right and left of the line. i.e. we are plotting 2 pixels on right side and one pixel on left of line. See Fig. 5.

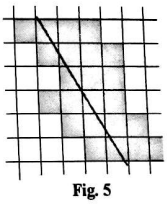


Fig. 5

Chapter 3 : Filled Area Primitives [Total Marks - 25]

Q. 1(a) Illustrate inside outside test. **(5 Marks)**

Ans. : Inside - Outside Test of Polygons :

Can we determine whether a point is inside the polygon or outside ? Yes, to determine this we are generally using two methods :

Even- odd Method :

One method of doing this is to construct a line segment between a point in question i.e. point to test whether inside or outside, and a point which is surely outside the polygon. But how we are going to find out a point which is surely outside the polygon ? It is very easy to find out that point.

For example, pick a point with an x co-ordinate smaller than the smallest x co-ordinate of the polygons vertices and the y co-ordinate will be any y value, or for simplicity we will take y value same as the y value of point in question. Refer Fig. 6.

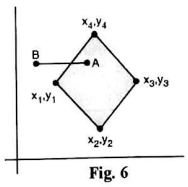


Fig. 6

In this case point A is one, which we want to check i.e. whether point A is inside polygon or not. As we are using arrays to store vertices of polygon we can easily come to know the vertex which is having lowest value of x and that is x_1. So we have to select a point smaller than x_1 and generally we select same value of y as that of point A, in order to draw straight line. But even if you select any y value for outside point, it will not make any difference.

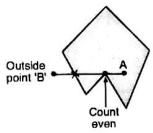


Fig. 7

Then count how many intersections are occurring with polygon boundary by this line till the point in question i.e. 'A', is reached. If there are an odd number of intersections then the point in question is inside. If even number of intersections then the point is outside the polygon. This is called Even-odd method to determine interior points of polygon.

But this even-odd test fails for one case i.e. when the intersection point is a vertex. See Fig. 7. To handle this case we have to make few modifications. We must look at other end points of the two segments of a polygon which meet at this vertex. If these points lie on the same side of the constructed line AB, then the intersection point counts as an even number of intersection i.e. (2, 4, 6 ... etc.). See Fig. 7.

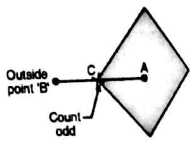


Fig. 8

But if they lie on opposite sides of the constructed line AB, as shown in Fig. 8, then the intersection point is counted as a single intersection. Let's take an example which is shown in Fig. 9(a). In this case the constructed line AB intersects polygon in a vertex 'C'. Here the other end points of both the line segments which meets at vertex 'C' are lieing in opposite sides of this constructed line AB. So the intersection at vertex 'C' is counted as odd count i.e. 1. The total number of intersections made by this new line with polygon is 1 i.e. odd and as it is odd therefore the point 'A' is inside.

Similarly for case II refer Fig. 9(b).

Here we want to check point A. So we are taking one point B which is surely outside and drawing a line between A to B. Here this new line intersects in two points i.e. C and D. Point C is not a vertex point so it is counted as normal intersection count. But point D is a vertex. So we have to check the other end points of both the line segments which meets at vertex D. These end points are on one side of constructed line AB. So we have to count this intersection point as even count, say as 2.

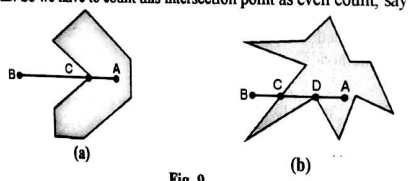


Fig. 9

Here as the constructed line intersects the polygon in two points C and D. So we have to make sum of this i.e. for C we are having count as 1 and for D we are having even count i.e. 2. So the sum of this count will be 1 + 2 i.e. 3, which is odd. So the point A is inside. If this sum is even then the point will be outside.

Q. 2(b) Write a pseudo-code to implement boundary fill and flood fill algorithm using 4 connected method. **(10 Marks)**

Ans. : **Please refer Q. 2(b) of May 2012.**

Q. 3(a) Write an algorithm to fill polygon in Fig. 10(a) with pattern given in Fig. 10(b) of size m x n. **(10 Marks)**

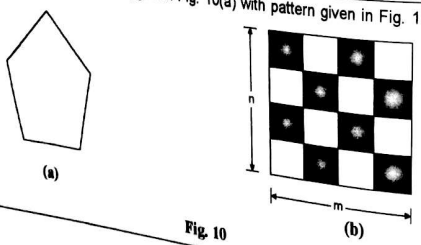


Fig. 10

Ans. :

Pattern Fill Algorithm :

In this section we consider, filling with pattern, which we do by adding extra control to the part of the scan conversion algorithm that actually writes each pixel. The main issue for filling with pattern is the relation of the area of the pattern to that of the primitive. In other words, we need to decide where the pattern is anchored so that we know which pixel in the pattern corresponds to the current pixel of the primitive. One technique is to anchor the pattern at a vertex of a polygon by placing the leftmost pixel in the patterns first row. This choice allows the pattern to move when the primitive is moved, a visual effect that would be expected for patterns with a strong geometric organization, such as the cross hatches often used in drafting applications. But there is no distinguished point on a polygon that is obviously right for such a relative anchor, and no distinguished points at all on smoothly varying primitives such as circles and ellipses. Therefore, the programmer must specify the anchor point as appoint on or within the primitive. In some systems, the anchor point may even be applied to a group of primitives.

Another technique is to consider the entire screen as being tiled with the pattern and to think of the primitive as consisting of an outline or filled area of transparent bits that let the pattern show through. To apply the pattern to the primitive, we index it with the current pixels (x, y) coordinates. Since patterns are defined as small P by Q bitmaps, we use modular arithmetic to make the pattern repeat. The pattern[0,0] pixel is considered coincident with the screen origin, and we can write, a bitmap pattern in transparent mode with the statement

```
If(pattern[x%P][y%Q])
        Writepixel (x,y,value);
```

If we are filling an entire span in replace write mode, we can copy a whole of the pattern at once, assuming a low-level version of a copy pixel facility is available to write multiple pixels. Another technique is to scan convert a primitive first into a rectangular work area, and then to write each pixel from that bitmap to the appropriate place in the canvas. This rectangle write to the canvas is simply a nested loop in which a 1 writes the current color and 0 writes nothing or background color. This two step process is twice as much work as filling during scan conversion and therefore is not worthwhile for primitives that are encountered and scan converted only once. The advantage of a pre-scan converted bitmap lies in the fact that it is clearly faster to write each pixel in a rectangular region, without having to do any clipping or span arithmetic, than to scan convert the primitive each time from scratch while doing such clipping.

For bi-level displays, writing current color 1, copy pixel works fine : For transparent mode, we use or write mode; for opaque mode, we use replace write mode. For multilevel displays, we cannot write the bitmap directly with a single bit per pixel, but must convert each bit to a full n-bit color value that is then written.

Chapter 4 : 2D Geometric Transformations [Total Marks - 10]

Q. 4(a) What do you mean by shear Transformation in 2D? Also explain its types. **(10 Marks)**

Ans. : Shear Transformations in 2D :

The shear transformation means slanting the image. This shear transformation is of two types :

Y-shear :

In y-shear we are keeping x co-ordinate values as it is and shifting the y co-ordinate values only, which results in tilting of horizontal lines. See Fig. 11.

$$P_1 * T = P_2$$

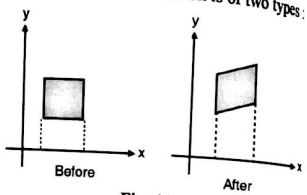


Fig. 11

If P_1 is (x, y), then x co-ordinate of P_2 must be same as x co-ordinate of P_1 but y co-ordinate of P_2 should change.

$\therefore$ Transformation matrix for y-shear will be $\begin{vmatrix} 1 & a \\ 0 & 1 \end{vmatrix}$ where a is constant factor which will decide, how much to tilt.

Now,

$$P_1 * T = P_2$$

$$|x, y| * \begin{vmatrix} 1 & a \\ 0 & 1 \end{vmatrix} = |x, x \cdot a + y|$$

So x co-ordinate of P_2 are not changed but y co-ordinate of P_2 will be the combination of constant value 'a' and 'x' and y values of P_1.

$$P_2 = (x, xa + y)$$

So here we can say value of y will depend on value of x.

X-shear :

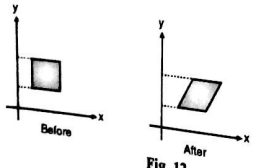


Fig. 12

x-shear preserves the y co-ordinate values and shifts x co-ordinate values causing vertical lines to tilt. See Fig. 12. If point P_1 is (x, y) then y co-ordinate of P_2 must remain unchanged but x co-ordinate of P_2 should change.

$\therefore$ Transformation matrix for x-shear will be $\begin{vmatrix} 1 & 0 \\ b & 1 \end{vmatrix}$ where b is constant factor, which will decide, how much to tilt.

Now, $P_1 * T = P_2$

$$|x, y| * \begin{vmatrix} 1 & 0 \\ b & 1 \end{vmatrix} = |x + yb, y|$$

Here we have to say, value of x will depend on value of y.

Chapter 5 : 2D Viewing [Total Marks - 10]

Q. 5(a) Find the clipping Co-ordinates to clip the line segment P1 P2 against the window ABCD using cohen Sutherland line cliping algorithm. **(10 Marks)**

P1 (10, 30) P2 (80, 90)

window ABCD A (20,20) B (90, 20)

C (90, 70) D (20, 70)

Ans. : Let's first draw the window and lines p_1p_2 and q_1q_2 from the given data,

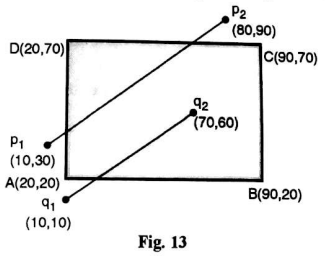


Fig. 13

From the above Fig. 13 lets 1st derive the region codes of p_1p_2 and q_1q_2.

Region code of p_1 = ABRL

= 0001

Region code of p_2 = ABRL

= 1000

Similarly region code of q_1 = ABRL = 0101

Region code of q_2 = ABRL = 0000

Now let's use Cohen-Sutherland algorithm to clip these lines. First consider line p_1p_2.

Since both region codes are not 0000 we have to take logical AND of these region codes.

p_1 = 0001

p_2 = 1000

AND = 0000

Since logical AND result is 0000 line p_1p_2 may be partially visible.

Let's consider point p_1 whose region code is 0001. Here last bit which signifies left boundary of window is set to 1. It means point p_1 is on left side of the window boundary i.e. point p_1 is surely lying outside the window.

So we will find the intersection of line p_1p_2 with left edge of window.

Since left edge of window is having x coordinate as 20. The intersection point x-coordinate will also be 20 only. Now we have to find y co-ordinate of intersection point. For this we will use slope intercept form of line.

i.e. let's find slope of line p_1p_2.

$$\therefore \quad m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{90 - 30}{80 - 10} = \frac{60}{70} = 0.8$$

So, slope of line p_1p_2 is 0.8.

We know that the slope between two endpoints of a line is same as that of slope between one endpoint and any other point which is lying on the line.

$\therefore$ Slope between point p_1 and intersection point will be

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

where x_1, y_1 are related with p_1, x_2 is known i.e. 20, y_2 we have to find.

$$\therefore \quad m(x_2 - x_1) = y_2 - y_1$$

$$\therefore \quad y_2 = m(x_2 - x_1) + y_1$$

$$\therefore \quad y_2 = 0.8\,(20 - 10) + 30 = 0.8\,(10) + 30 = 8 + 30$$

$$y_2 = 38$$

Therefore the intersection point is (20, 38). Let's call this point as I_1. Since I_1 point is exactly at the boundary. Its region code will be (0, 0, 0, 0). Now we have to discard the line segment $p_1 I_1$ Since p_1 point is outside window. Similarly we have to find another intersection point I_2 with respect to top boundary. Now I_2 points region. Code will be 0000. So we will discard I_2p_2 since p_2 point is above the window. Now our line segment is I_1 I_2. Both I_1 and I_2's region codes are 0000. So the line I_1I_2 is completely visible and we have to display I_1I_2 line.

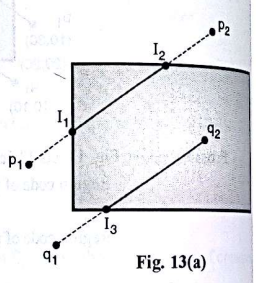


Fig. 13(a)

Similarly for line q_1q_2, one point q_2 is inside the window, whereas point q_1 is outside the window. So here also we have to find the intersection point with respect to the window boundary. Let's call it as point I_3 and then discard the line q_1I_3 since q_1's region code is nonzero, which means point q_1 is surely laying outside the window. As intersection point I_3's region code is 0000 and another endpoint q_2 region code is also 0000 this line segment I_3 q_2 is visible.

Chapter 6 : 3D Geometric Transformations and Viewing [Total Marks - 25]

Q. 1(b) Explain 3D display methods. **(5 Marks)**

Ans. :

3D display Methods :

Many real-world objects are inherently smooth, therefore need infinitely many points to model it. This is not feasible for a computer with finite storage. More often we merely approximate the object with pieces of planes, spheres, or other shapes that are easy to describe mathematically.

When we are representing a 3D object in computer graphics, the graphics scenes can contain many different kinds of objects. Not a single method is in a position to describe all these methods. If we are using accurate models they produce a realistic display of scenes. We can make use of following methods also

(1) Polygon and quadric surfaces (2) Spline surfaces and construction techniques

(3) Procedural methods (4) Physically-based modeling methods

(5) Octree encodings

Polygon Surfaces :

It is a set of surface polygons that enclose interiors of the object. It is a basic form of a 3D object representation. In some systems, all objects are described as polygon surfaces. Polygons are easy to process, so rendering and display of objects is speed up. Thus, some systems allow objects to be described in other ways, such as splines, but reduce all objects to polygons for processing. For a polygon, the representation is exact. For other surfaces, polygons are tiled in a polygon-mesh which approximates the object. Usually, the basic polygon is a triangle.

Polygon surfaces are commonly used for boundary representation. Here all surfaces are described with linear equations, which simplify and speed up surface rendering and display of objects. By using this we can precisely define a polyhedron. For non-polyhedron objects, the wireframe outline can be display quickly. Polygon surfaces are commonly used in design and solid modeling.

Q. 4(b) Derive the steps required to perform 3D rotation about arbitrary axis. **(10 Marks)**

Ans. : **Rotation about an Arbitrary Axis :**

Until now we have seen how to rotate a point or object with respect to co-ordinate axes, such as x, y and z. But suppose we want to rotate a point or object at angle θ about an arbitrary line which does not coincide with any of the co-ordinate axes, then we have to set up a composite transformation involving combinations of translations and the co-ordinate axes rotations.

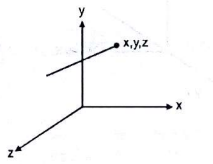


Fig. 14

We obtain the required composite matrix, for rotation about an arbitrary axis, by first setting up the transformation sequence that moves the selected rotation axis onto one of the co-ordinate axes. Then we set up the rotation matrix about that co-ordinate axis for the specified rotation angle. The last step is to obtain the inverse transformation sequence that returns the rotation axis to its original position.

If we want to rotate a point (x, y, z), by an angle θ, which is in space and lying on an arbitrary axis as shown in Fig. 14. Then we have to follow some steps in sequence.

Step 1 : Translate the arbitrary axis so that it will pass through the origin. When we are translating axis, a point also gets translated. See Fig. 15.

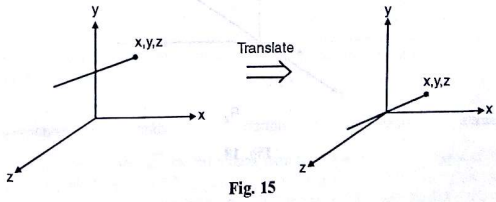


Fig. 15

Step 2 : Perform rotation about x-axis until the arbitrary axis completely lies in XZ plane. See Fig. 16.

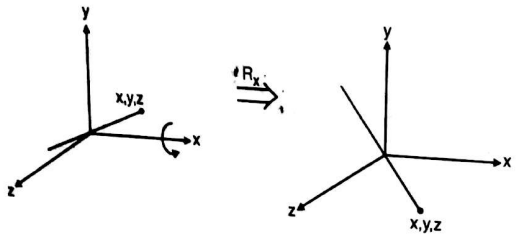


Fig. 16

Step 3 : Now perform rotation about y-axis until the arbitrary axis coincides with z-axis as shown in Fig. 17. We can also perform rotation about y-axis until arbitrary axis coincides with x-axis. Here we are matching the arbitrary axis to any one co-ordinate axis, so that we can make use of standard rotation matrices.

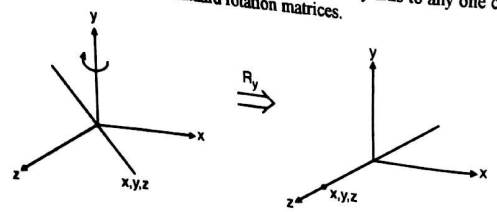
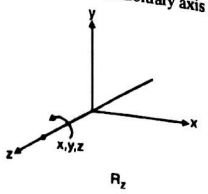


Fig. 17

Step 4 : Once the arbitrary axis coincides with any one co-ordinate axis then the rotation of arbitrary axis will be same as that of matched co-ordinate axis. Here we are performing rotation about z-axis as we have matched arbitrary axis to z-axis. See Fig. 18



R_z

Fig. 18

Step 5 : After performing rotation about z-axis (i.e. rotation about arbitrary axis) we have to perform reverse transformations. Here we should not perform reverse rotation about z-axis, because it is rotation about arbitrary axis. So we have to perform reverse rotation about y-axis. i.e. if earlier we rotated y-axis, in anticlockwise direction, now we have to rotate it in clockwise direction. See Fig. 19.

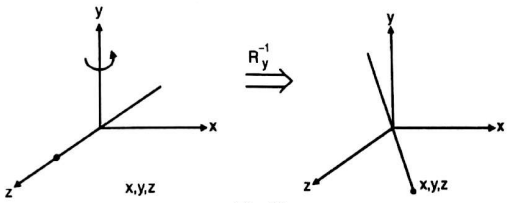


Fig. 19

Step 6 : Again we have to perform reverse rotation about x-axis. See Fig. 20.

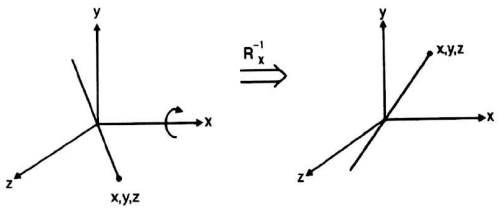


Fig.20

Step 7 : Lastly we have to apply inverse translation to bring the arbitrary axis back to its origina position. See Fig. 21.

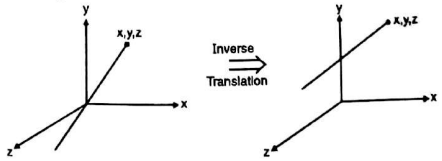


Fig. 21

Q. 5(b) What do you understand by terms parallel and prespective projection ? Also explain dept cueing. **(10 Marks**

Ans. :

Parallel Projection :

A parallel projection is formed by extending parallel lines from each vertex of the object unt they intersect the plane of the screen. The point of intersection is the projection of the vertex. Then w connect the projected vertices by line segments which correspond to connections on the original objec See Fig. 22.

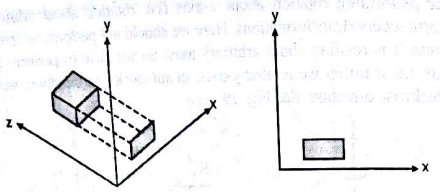


Fig. 22

Here we are taking projection of each vertex of the object till it intersects to the plane of screen. The plane on which we are taking projection is called as view plane. If we want to represent a 3D object on 2D plane, the most simple way is to discard the z co-ordinate. We are discarding the z co-ordinates only when the screen or viewing surface is parallel to the XY plane and the lines of projection are parallel to the z-axis.

A parallel projection preserves relative properties of objects, in x and y direction. Accurate views of the various sides of an object are obtained with a parallel projection, but this does not give us a realistic representation of the appearance of a 3D object.

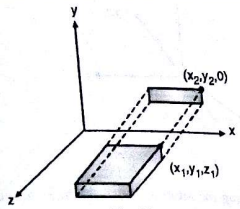


Fig. 23

Let us take an example. Suppose there is a 3D cube in space and we want to draw it on screen i.e. on XY plane. See Fig. 23.

We obtain the parallel projection of a point (x_1, y_1, z_1) by drawing a line through this point with a certain direction (x_p, y_p, z_p) in space. This line is called as projection vector. The point where this line intersects the plane $z = 0$ is the parallel projection of the given point; it has the coordinate $(x_2, y_2, 0)$

Perspective Projection :

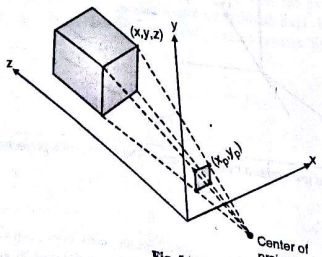


Fig. 24

In real world, the objects which are away from the viewer, appear smaller. Perspective projection preserves this property. In perspective projection, the lines of projection are not parallel. Here all

projection lines are ending at a single point called center of projection. In real world this center of projection is human eye. For perspective projection see Fig. 24.

To draw the image of object on 2D plane, assume that the object is in space having some x, y, and z values. We have to draw the image on XY plane, so the center of projection should be beyond the XY plane. All projections which are originating from different vertices of object meet at center of projection point, intersecting XY plane. Assuming that the plane XY or screen is placed at z = 0.

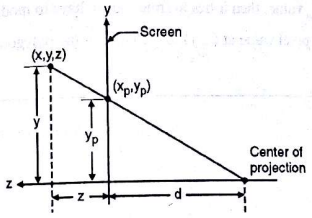


Fig. 25

So the distance of the center of projection from the XY plane, i.e. plane whose z = 0 is called 'd'. With this assumption computations become simple. The projection of a point is the intersection of the straight line from that point to the center with the plane z = 0. Fig. 25, shows side view of the projection of the point (x, y, z) which gives projected point (x_p, y_p).

To draw the image of an object we have to find the values of (x_p, y_p), as (x_p, y_p) is intersection point of projection with screen. To obtain y_p, we have to say

$$\frac{y}{(z+d)} = \frac{y_p}{d} \qquad \therefore \quad y_p = \left(\frac{y}{(z+d)}\right) \cdot d$$

Depth cueing :

To create realistic image, the depth information is important so that we can easily identify, for a particular viewing direction, which is the front and which is the back of displayed objects. The depth of an object can be represented by the intensity of the image. The parts of the objects closest to the viewing position are displayed with the highest intensities, and objects farther away are displayed with decreasing intensities. The effect is known as 'depth cueing'.

Chapter 7 : Hidden Surfaces [Total Marks - 10]

Q. 1(c) What is z-buffer algorithm? **(5 Marks)**

Ans. : z Buffer :

Another way to handle hidden lines and surfaces is z buffers. Here we are sorting the polygons according to their position in space. And then in frame buffer itself. We are storing polygons which are closer to viewer. We know that frame buffer is used to store the images which we want to display on monitor. Here for visibility test we are making use of z buffer along with frame buffer.

The z buffer is a large array to hold all the pixels of display. z buffer is some what similar to frame buffer. In frame buffer we are having arrays to store x and y co-ordinates of an image. Similarly z buffer contains z co-ordinates of pixels which we want to display. It helps to sort the polygons by comparing their z position. In simple terms it keeps track of nearest z coordinate of those points which are seen from the pixel (x, y).

When there is nothing to display on monitor i.e. frame buffer is empty, at that time we have to initialize z buffer elements to a very large negative value. A large negative value on z axis represents a point beyond which there is nothing i.e. setting background color.

$\therefore z_{buffer}(x, y) = z_{initial}$

Polygons will be entered one by one into frame buffer by the display file interpreter. During this process, for each pixel (x, y) that lies inside the polygon we have to calculate z (x, y) for that pixel (x, y). Then we have to compare the z position of the polygon point with the z_{buffer} (x, y) value and decide whether the new surface is in front of or behind the current contents of frame buffer.

If the new surface has z value greater than z_{buffer} value, then it lies in front. So we have to modify the contents of z_{buffer} (x, y) by new z value and set the pixel value at (x, y) to the color of the polygon at (x, y).

```
i.e. if (z (x, y) > z_buffer (x, y))
    {
        z_buffer (x, y) = z (x, y)
        put pixel (x, y, polygon-color)
    }
```

If the z value of new surface is smaller than z_{buffer} (x, y), then it lies behind some polygon which was previously entered. So the new surface should be hidden and should not be displayed. The frame buffer and z_{buffer} should not be modified here. Here the comparison should be carried out by using pixel by pixel method.

Q. 7(c) Write short notes on : Warnock's algorithm **(5 Marks)**

Ans. : Warnock's algorithm :

Area subdivision method is also called as area based method or Warnock's algorithm. In painter's algorithm, we have seen procedure to remove hidden-surfaces. It means painters algorithm deals with the procedure. But area subdivision method does not deal with the procedure to remove hidden surfaces. It just tries to display the final picture on screen. The basic idea is as the resolution of the display increases the size of the picture also increases and hence its correctness also increases.

The algorithm is a recursive procedure based on a 2-step strategy.

Step 1 : Decide which polygons overlap the given area on the screen.

Step 2 : Which polygons are visible in that area.

So, the area subdivision method checks whether the polygon is partially or fully visible in the given area. Let's call this area as 'A'. Then, each area is further subdivided into sample sub-areas and again the test is carried out for the each sub-area. It means whether the polygon is visible in that area or not till the visibility decision is made or until the screen area becomes a pixel. The algorithm starts with initial area as 'A' and divides that area in four sub-areas. See Fig. 26.

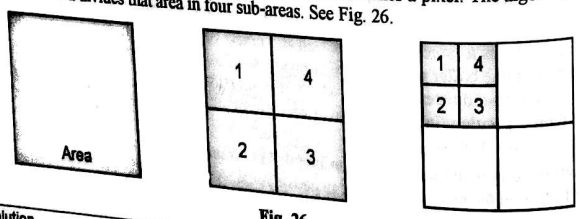


Fig. 26

Here according to the screen area, we are classifying the polygons into four basic categories. These categories are as follows :

1) **Surrounding polygon :**

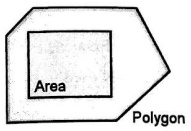


Fig. 27

In this case a polygon completely covers the given screen area as shown in Fig. 27.

2) **Intersecting polygon :**

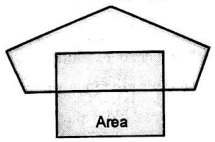


Fig. 28

In this case a polygon is getting intersected with the screen area i.e. the polygon is partially inside the screen area as shown in Fig. 28.

3) **Contained polygon :**

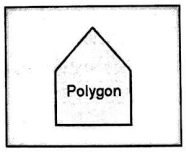


Fig. 29

In this case a polygon is lying completely inside the given screen area as shown in Fig. 29.

4) **Disjoint polygon :**

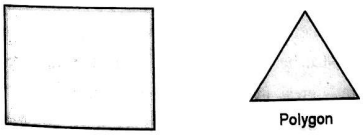


Fig. 30

Whenever the polygon is lying completely outside the given screen area, it is called disjoint polygon. See Fig. 30.

Chapter 8 : Illumination Models and Surface Rendering [Total Marks - 20]

Q. 6(a) Explain Gouroud and Phong shading. **(10 Marks)**

Ans. : Please refer Q. 7(a) of May 2012

Q. 7(b) Write short notes on : Dithering techniques **(5 Marks)**

Ans. :

Dithering Techniques :

Dither is an intentionally applied form of noise, used to randomize quantization error, thereby preventing large-scale patterns such as "banding" in images, or noise at discrete frequencies in an audio recording, that are more objectionable than uncorrelated noise. Dither is routinely used in processing of both digital audio and digital video data, and is often one of the last stages of audio production to CD.

Ordered dithering is an image dithering algorithm. It is commonly used by programs that need to provide continuous image of higher colors on a display of less color depth. For example, Microsoft Windows uses it in 16-color graphics modes. It is easily distinguished by its noticeable crosshatch patterns.

The ordered dither technique for bi-level displays also increases the visual resolution without reducing the spatial resolution, by introducing a random error into the image. This random error is added to the image intensity of each pixel before comparison with the selected threshold value. Adding a completely random error does not yield an optimum result. Because the error pattern is small in size than image it is tiled across the image. This technique is a form of dispersed dot-ordered dither.

Dither should be added to any low-amplitude or highly-periodic signal before any quantization or re-quantization process, in order to de-correlate the quantization noise with the input signal and to prevent distortion; the lesser the bit depth, the greater the dither must be. The results of the process still yield distortion, but the distortion is of a random nature so its result is effectively noise. Any bit-reduction process should add dither to the waveform before the reduction is performed.

There are different types of dither which are as follows :

RPDF stands for "Rectangular Probability Density Function," equivalent to a roll of a dice. Any number has the same random probability of surfacing. TPDF stands for "Triangular Probability Density Function," equivalent to a roll of two dice (the sum of two independent samples of RPDF). Gaussian PDF is equivalent to a roll of a large number of dice. The relationship of probability of results follows a bell-shaped, or Gaussian curve, typical of dither generated by analog sources such as microphone preamplifiers. If the bit depth of a recording is sufficiently great, that noise will be sufficient to dither the recording. Colored Dither is sometimes mentioned as dither that has been filtered to be different from white noise. Some dither algorithms use noise that has more energy in the higher frequencies so as to lower the energy in the critical audio band.

Dithering is the attempt by a computer program to approximate a color from a mixture of other colors when the required color is not available. For example, dithering occurs when a color is specified for a Web page that a browser on a particular operating system can't support. The browser will then attempt to replace the requested color with an approximation composed of two or more other colors it

can produce. The result may or may not be acceptable to the graphic designer. It may also appear somewhat grainy since it's composed of different pixel intensities rather than a single intensity over the colored space. Dithering also occurs when a display monitor attempts to display images specified with more colors than the monitor is equipped to handle.

Q. 7(d) Write short notes on : Half toning **(5 Marks)**

Ans. :

Halftone :

It is the reprographic technique that simulates continuous tone imagery through the use of dots, varying either in size or in spacing. 'Halftone' can also be used to refer specifically to the image that is produced by this process.

Where continuous tone imagery contains an infinite range of colors or grays, the halftone process reduces visual reproductions to a binary image that is printed with only one color of ink. This binary reproduction relies on a basic optical illusion - that these tiny halftone dots are blended into smooth tones by the human eye. At a microscopic level, developed black and white photographic film also consists of only two colors, and not an infinite range of continuous tones. Halftoning is the process of turning continuous tone grayscale or color images into a series of dots for printing that fool the eye. Each intensity position in the original scene is replaced with a rectangular pixel grid. This method is used for printing to reproduce photographs in magazines, books or newspaper.

For bi-level systems one can represent shading intensities with a halftoning method that converts the intensity of each point on a surface into a regular pixel grid that can display a number of intensity level. The number of intensity level with this method depends on how many pixels we include on the grid. A graphics package employing a halftone technique displays a scene by replacing each position in the original scene with an (n x n) grid of pixels. Intensity level of the corresponding position in the scene determines number of pixels that are turned on. Such a technique of turns on two level systems into one with the five possible intensities is shown in Fig. 31. These levels are labeled 0 through 4.

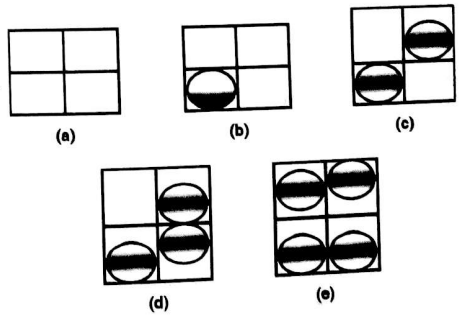


Fig. 31

With this technique one can obtain n2 intensity levels above zero for any (n x n) grid. To avoid unwanted pattern introduction into the original possible, symmetrical pixel arrangements have to be avoided whenever possible. A symmetrical pattern would produce either vertical and horizontal or diagonal streaks in a halftoning representation. To avoid such patterns, select different pixel arrangements for the representation of an intensity level. Fig. 'c' can be selected to represent the second intensity level above zero. Another method is to form successive grid patterns with the same pixels turned on i.e. if the pixel is on for one grid level, it is on for all higher levels.

Chapter 9 : Curves and Fractals [Total Marks - 15]

Q. 1(d) Write a note on Fractars. **(5 Marks)**

Ans. :

Fractals :

The objects which are having smooth surfaces and regular shapes are generally described by using equations. But natural objects such as mountains, trees, ocean waves and clouds have irregular shapes. It will be very difficult to draw these shapes by using normal equations. There are many methods of modeling these natural objects, but one of the most interesting from a mathematical perspective is that of fractals. So we can describe natural objects by using fractals, where procedures rather than equations are used to model the objects. Procedurally defined objects have characteristics quite different from objects described with equations.

One of the basic properties that characterize fractals is self-similarity. The self similarity property of an object can take different forms, depending on the choice of fractal representation. Self-similarity means if we zoom into a piece of a fractal we will keep seeing the same structures repeated over and over. From a certain distance we will see a coastline as a simple, quite smooth line. See Fig. 32. But as we go near to that line, it will appear more rough. If we take more closer view we will see the whole line as jaggy and rough. There is no limit how many times we can zoom in.

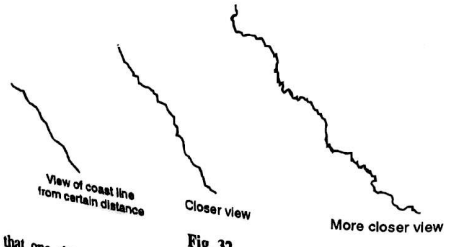


Fig. 32

Imagine that one object is made up of clay. If we broke that object in terms of line or line segments, then we will give the dimension $D_t = 1$. If the object is broken into a plane, then we will give

the dimension as $D_t = 2$. And if the object is broken into 3D objects like cube, sphere etc. then we will given the dimension as $D_t = 3$. Here the variable D_t is called as topological dimension.

Now imagine that we have a thread of say 5 cm. If we cut this in five equal parts i.e. each sub thread's length will be 1 cm. It means we have scaled the thread i.e. line, by 1/5. Now if we want to build original object from the scaled one, then how many scaled objects will be require to form the original object ? Obviously 5. Fig. 33 explains this concept in more detail.

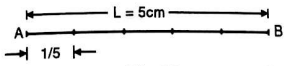


Fig. 33

Fig. 33 shows a line AB having length (L) = 5 cm. If we break this line in five equal parts, i.e. Number of pieces (N) = 5, then each parts length (l) will be 1 cm.

$$\therefore \quad l = \frac{L}{N} = \frac{5}{5} = 1 \text{ cm}$$

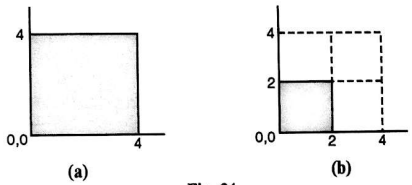
Here we can define a scaling factor as, $= \frac{1}{S}$

Where S = Number of scaled objects

In this cause $S = 5$

$\therefore$ We can say $N = S^1$ ...(1)

Now consider a square, as shown in Fig. 34(a) and if we want to scale it by 1/2, then how many small squares will get generated? The answer is four. See Fig. 34(b).



(a) **(b)**

Fig. 34

Now if we want to build the original square again, then we need N number of scaled objects i.e. 4.

So here S = 2 and N = 4. $\therefore \quad N = S^2$ ...(2)

Similarly if we have a 3D object, like cube, and if we scale it as half, as shown in Fig. 34, then we need 8 scaled objects to form original object.

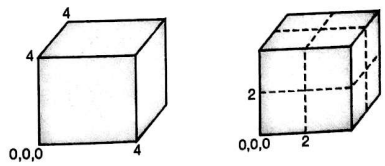


Fig. 35

Here S = 2 and N = 8.

$$\therefore N = S^3$$

If we observe the Equations (1), (2) and (3) we will come to know that there is some relation i.e. ...(3)

the exponent goes on increasing. So we can generalize these equations as $N = S^{D_f}$

It means if we scale an object by 'S' and must assemble 'N' of scaled objects to reconstruct the full sized/original object, then the dimension 'D_f' of the object is given by,

$$D_f = \frac{\log N}{\log S}$$

where 'D_f' is called as fractal dimension.

Q. 6(b) Explain Bezier curves and Bezier surfaces with equations. Explain properties of Bezier curve. **(10 Marks)**

Ans. : **Please refer Q. 6(b) of May 2012.**

❑❑❑

May 2013

Chapter 1 : Introduction to Computer Graphics [Total Marks - 10]

Q. 4(b) Explain the different raster techniques and the transformation associated with it. **(10 Marks)**

Ans. : **Please refer Q. 7(e) of Dec. 2012.**

Chapter 2 : Output Primitives [Total Marks - 20]

Q. 1(a) Explain the method to draw a thick line using Bresenham's algorithm. **(5 Marks)**

Ans. : **Please refer Q. 7(a) of Dec. 2012.**

Q. 1(c) What is aliasing ? Explain some antialiasing techniques. **(5 Marks)**

Ans. : **Aliasing :**

The various forms of distortion that results from the scan conversion operations are collectively called as **aliasing**.

Aliasing techniques : Please refer Q. 1(d) of May 2012.

Q. 3(b) Derive the midpoint algorithm for ellipse generation. **(10 Marks)**

Ans. :

The midpoint ellipse method is applied throughout the first quadrant in two parts i.e. region-1 and region-2. We are forming this regions by considering the slope of the curve. If the slope of the curve is less than –1 then we are in region-1 and when the slope becomes greater than – 1 then in region-2. See Fig. 1.

i.e. At the boundary between region-1 and 2

$$\frac{dy}{dx} = -1.$$

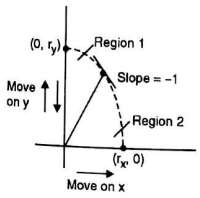


Fig. 1

The slope of the ellipse is calculated as

$$\frac{dy}{dx} = -\frac{2r_y^2 x}{2r_x^2 y}$$

At the boundary between region-1 & 2,

$$\frac{dy}{dx} = -1 \quad \text{and} \quad 2r_y^2 x = 2r_x^2 y$$

So, we move out of region-1 when $2r_y^2 x \geq 2r_x^2 y$

For region-1, we can start at (0, r_y) and step clockwise along the elliptical path in the 1st quadrant shifting from unit steps x to unit steps in y, till the slope becomes less than – 1 OR we can start at (r_x, 0) and select points in anticlockwise shifting from unit steps in y to unit steps in x, till the slope becomes greater than – 1.

Equation of ellipse is,

$$f_{ellipse}(x, y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2$$

Here we are making use of this function to decide whether the point is inside or outside the elliptical curve same as in case of circle.

If $f_{ellipse}(x, y) < 0$ then (x, y) is inside the ellipse boundary.

$= 0$ then (x, y) is on boundary.

> 0 then (x, y) is outside the ellipse boundary

Thus the $f_{ellipse}(x, y)$ acts as a decision parameter. So, we are going to select the next pixel along the path of ellipse, according to the sign of the function.

We are starting at position $(0, r_y)$ and take unit steps in the x-direction until we reach the boundary between region-1 and 2. Once we reach to the boundary, we switch to unit steps in the y-direction for the remainder of the curve.

Here we have to calculate the slope at each step.

$$\frac{dy}{dx} = -\frac{2 r_y^2 x}{2 r_x^2 y}$$

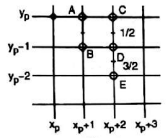
We move to region-2 from region-1 whenever

$$2 r_y^2 x \geq 2 r_x^2 y$$

In region-1 if the current pixel is located at (x_p, y_p). Then next candidate pixels are A & B, So the decision variable for region-1 will be $(x_p + 1, y_p - 1/2)$, as we are using midpoint algorithm. See Fig. 2.

$$\therefore \quad f_{ellipse}(x, y) = f_{ellipse}(x_p + 1, y_p - 1/2)$$

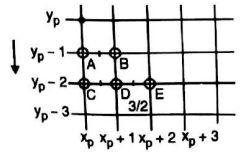
$$\text{dp-1-old} = r_y^2 (x_p + 1)^2 + r_x^2 (y_p - 1/2)^2 - r_x^2 r_y^2$$



Changing x uniformly and deciding which row is nearer

Fig. 2

Similarly, in region-2, if current pixel is at (x_p, y_p) then candidate pixels are A and B, so midpoint will be $(x_p + 1/2, y_p - 1)$. See Fig. 3.



Changing y uniformly and deciding which column is closer

Fig. 3

$$\therefore \quad f_{ellipse}(x, y) = f_{ellipse}(x_p + 1/2, y_p - 1)$$

$$\therefore \quad \text{dp-2-old} = f_{ellipse}(x_p + 1/2, y_p - 1)$$

$$\therefore \quad \text{dp-2-old} = r_y^2 (x_P + 1/2)^2 + r_x^2 (y_P - 1)^2 - r_x^2 \cdot r_y^2$$

We must also compute the initial condition. There will be two initial conditions, one for region-1 and another for region-2. Let us first see for region-1.

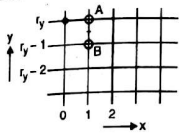


Fig. 4

Assuming ellipse starts at $(0, r_y)$. So candidate pixels are A and B. See Fig. 4.

$\therefore$ Midpoint will be $(1, r_y - 1/2)$ and the initial value of decision parameter will be $f_{ellipse}(1, r_y - 1/2)$. The variable dp-1-initial represents initial decision parameter for region-1.

$$\text{dp-1-initial} = r_y^2 + (r_y - 1/2)^2 \cdot r_x^2 - r_x^2 \cdot r_y^2 = r_y^2 + \left[r_y^2 - r_y + \frac{1}{4}\right] r_x^2 - r_x^2 \cdot r_y^2$$

$$= r_y^2 + r_x^2 \cdot r_y^2 - r_x^2 \cdot r_y + \frac{1}{4} r_x^2 - r_x^2 \cdot r_y^2 = r_y^2 + r_x^2\left(-r_y + \frac{1}{4}\right)$$

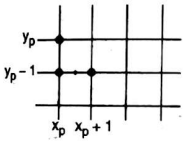


Fig. 5

At every iteration in region-1, we must not only test the decision variable but also see whether we should switch regions. When the midpoint crosses over into region-2 we change to travel along y-axis. That is, if the last pixel chosen in region-1 is located at (x_p, y_p) then the decision variable is initialized at $(x_p + 1/2, y_p - 1)$. We stop drawing pixels in region-2 when the y-value of the pixel is equal to zero. See Fig. 5.

The last point of region-1 will be acting as starting point of region-2.

Initial value of region-2 will be $f_{ellipse}(x_p + 1/2, y_p - 1)$.

$$\text{dp-2-initial} = r_y^2 (x_p + 1/2)^2 + r_x^2 (y_p - 1)^2 - r_x^2 \cdot r_y^2 = r_y^2 \left[x_p^2 + x_p + \frac{1}{4}\right] + r_x^2 [y_p^2 - 2 y_p + 1] - r_x^2 \cdot r_y^2$$

$$= r_y^2 x_p^2 + r_y^2 \cdot x_p + \frac{1}{4} r_y^2 + r_x^2 \cdot y_p^2 - 2 y_p \cdot r_x^2 + r_x^2 - r_x^2 \cdot r_y^2$$

$$\text{dp-2-initial} = r_y^2 \left[x_p^2 + x_p + \frac{1}{4}\right] + r_x^2 (1 - 2 y_p)$$

Steps for midpoint ellipse generation :

1) Accept r_x, r_y and ellipse center co-ordinates (x_c, y_c) and plot the first point on an ellipse centered on the origin as,

$$(x, y) = (0, r_y)$$

2) Calculate the initial value of decision parameter for region-1 as,

$$\text{dp-1} = r_y^2 + r_x^2\left(-r_y + \frac{1}{4}\right)$$

3) We have to check whether the slope of the curve is < -1, if yes, then we are in region-1 and perform following steps for region-1.

i) If dp-1 is less than 0 then

step along x-axis and modify decision parameter as,

$$dp\text{-}1 = dp\text{-}1 + r_y^2 (2x_p + 3)$$

Otherwise

Step along x-axis and decrease y by 1 and modify decision parameter as,

$$dp\text{-}1 = dp\text{-}1 + r_y^2 (2x_p + 3) + r_x^2 (-2y_p + 2)$$

ii) Draw the pixel (x, y) by using four point symmetry.

4) If the slope of the curve is > -1 then we have to change the region from 1 to 2. Now the last point of region-1 will be considered as first point of region-2.

5) Calculate the initial value of decision parameter for region-2 as,

$$dp\text{-}2 = r_y^2 \left(x_p + \frac{1}{2}\right)^2 + r_x^2 (y_p - 1)^2 - r_x^2 \cdot r_y^2$$

6) Perform following steps till $(y > 0)$

i) If $dp\text{-}2 > 0$ then

decrease y by 1 and update dp-2.

$$dp\text{-}2 = dp\text{-}2 + r_x^2 (-2y_p + 3)$$

otherwise

decrease y by 1 and increase x by 1 and update dp-2 as,

$$dp\text{-}2 = dp\text{-}2 + r_y^2 (2x_p + 2) + r_x^2 (3 - 2y_p)$$

ii) Draw the pixel (x, y) by using four point symmetry.

Chapter 3 : Filled Area Primitives [Total Marks - 10]

Q. 6(b) Explain scan line fill algorithm with some suitable examples. **(10 Marks)**

Ans. : Scan line fill algorithm :

In contrast to boundary fill and flood fill algorithm at pixel level, this algorithm is defined at geometric level i.e. co-ordinates, edges, vertices etc. This algorithm starts with first scan line and proceeds line by line toward the last scan line and checks whether every pixel on that scan line satisfies our inside point test or not. i.e. it checks which points on that scan line are inside the polygon. This method avoids the need for seed point.

Let us explain this algorithm by considering an example of convex polygon. (See Fig. 6). Here the algorithm begins with the first scan line that the polygon occupies i.e. y_{max} and proceeds line by line towards the last scan line i.e. y_{min}.

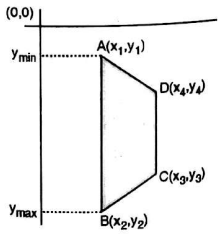


Fig. 6

Here we are considering first and last scan line of polygon and not individual edges. For each edge of the polygon we are storing five attributes. As we know that to draw any edge or line we need two points. So we are going to store x_{max}, x_{mn}, y_{max}, and y_{mn} of the edges of the polygon along with their slopes. For edge AB we are going to store

$x_{max} = x_2$ $\quad x_{min} = x_1$

$y_{max} = y_2$ $\quad y_{min} = y_1$

Here x_{max} and x_{mn} has as such no meaning, whereas y_{max} is the maximum y value for a particular edge and y_{mn} is the minimum y value for that edge. x_{max} and x_{min} are the corresponding x component of y_{max} and y_{mn}. Along with this we are going to store the slope of the edge also.

Let us tabularize this

Edge	y_{max}	x_{max}	y_{min}	x_{min}	Slope
AB	y_2	x_2	y_1	x_1	m_1
AD	y_4	x_4	y_1	x_1	m_2
CD	y_3	x_3	y_4	x_4	m_3
BC	y_2	x_2	y_3	x_3	m_4

For edge AB which is formed by (x_2,y_2) and (x_1,y_1), y_2 is greater than y_1. Therefore $y_{max} = y_2$ and $y_{min} = y_1$ and so $x_{max} = x_2$ and $x_{min} = x_1$.

As we are filling the polygon line by line, at any particular time we are not considering all the edges. So there is no point to find whether a particular scan line intersects with each edge or not. In fact we have to select only those edges which are getting intersected by the scan line. To decide which edges are getting intersected by scan line we are making use of y_{max} of particular edge. One thing is sure that we are storing all the attributes of all edges. Out of those attributes we are selecting y_{max} and then we are sorting this y_{max} array. If some swapping is required in this y_{max} then we are going to swap whole edge. After sorting y_{max} array we will get following attribute table.

Edge	y_{max}	x_{max}	y_{min}	x_{min}	Slope
AB	y_2	x_2	y_1	x_1	m_1
BC	y_2	x_2	y_3	x_3	m_4
CD	y_3	x_3	y_4	x_4	m_3
AD	y_4	x_4	y_1	x_1	m_2

Now the y_{max} array is sorted. So we can find out the intersection of scan line with first two edges from the sorted attribute table i.e. AB and BC. Every time we are decreasing scan line by 1, from y_{max} to y_{min} of the polygon. In this procedure it may happen that the edge which we have selected to find intersection point may get finished i.e. scan line goes below the y_{mn} of selected edge. In that case we have to discard that edge and select next edge from the sorted table and continue till scan line becomes equal to y_{min} of polygon.

Here important point is how to find out the intersection point of scan line with a particular edge. While at the time of storing attributes of edges we have stored the slope of that edge. We can make use of this slope to find intersection point of scan line with edge. (See Fig. 7). When we are moving from y_{max} to y_{min} every time we are decreasing y by 1, as the distance between two scan line is 1. So we know new points y value, which will be $y_{max} - 1$. Now we have to find, what will be its corresponding x value for the intersection point.

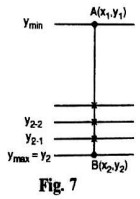


Fig. 7

We know that

$$\text{Slope (m)} = \frac{\Delta y}{\Delta x} = \frac{\text{change in y}}{\text{change in x}} = \frac{y_{new} - y_{old}}{x_{new} - x_{old}}$$

$$m = \frac{1}{x_{new} - x_{old}} \qquad x_{new} - x_{old} = \frac{1}{m}$$

$$\therefore x_{new} = x_{old} + \frac{1}{m}$$

Where m, we know from attribute table.

x_{old} will be x-coordinate of previous lines intersection point. Thus by knowing the slope of every edges we can find the intersection point by decrementing 1 every time from y_{max} to y_{min} and calculating corresponding x by, $x_{old} + \frac{1}{m}$

Chapter 4 : 2D Geometric Transformations [Total Marks - 15]

Q. 1(d) Derive the transformation matrix to magnify the triangle A (0, 0), B (1, 2) C (3, 2) 5 to twice its size so that the point C (3, 2) remain fixed. **(5 Marks)**

Ans. :

Magnify the triangle to twice means we need to scale the triangle with $S_x = 2$ and $S_y = 2$.

Let's represent the given triangle in matrix form as $\begin{vmatrix} 0 & 0 \\ 1 & 2 \\ 3 & 2 \end{vmatrix}$

Now we need scaling matrix with $S_x = 2$ and $S_y = 2$.

$$\begin{vmatrix} 0 & 0 \\ 1 & 2 \\ 3 & 2 \end{vmatrix} * \begin{vmatrix} 2 & 0 \\ 0 & 2 \end{vmatrix} = \begin{vmatrix} 0 & 0 \\ 2 & 4 \\ 6 & 4 \end{vmatrix}$$

Since the point C(3,2) should remain fixed we have to apply translation with $tx = -3$ and $ty = -2$. For this we have to use homogeneous coordinates

$$\begin{vmatrix} 0 & 0 & 1 \\ 2 & 4 & 1 \\ 6 & 4 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & -2 & 1 \end{vmatrix} = \begin{vmatrix} -3 & -2 & 1 \\ -1 & 2 & 1 \\ 3 & 2 & 1 \end{vmatrix}$$

$\therefore$ The new co-ordinates of triangle after magnifying it to twice and keeping point C(3, 2) fixed will be A (– 3, – 2), B (– 1, 2), C (3, 2).

Q. 3(a) Find out composite transformation matrix to reflect a triangle with vertices A(– 2, 1), B(– 1, 2) and C(– 2, 2) about line y = x + 2. Also find the coordinate of reflected object. **(10 Marks)**

Ans. :

Lets represent the given triangle and line.

Since the equation of line is y = x + 2 means slope m = 1 and y intercept = 2

As slope $= \frac{dy}{dx}$ $\quad \therefore m = \frac{dy}{dx}$

but $m = 1$ $\quad \therefore dy = dx$

It means a line is a slanting line making 45° angle. Since y-intercept is 2. One point is (0,2) which is drawn in the Fig. 8.

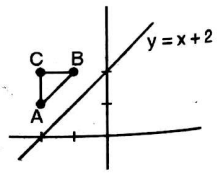


Fig. 8

Now we have to reflect the triangle along the given line. But for that we have to first shift or translate a line so that a y intercept point i.e. (0,2) will become (0,0). Then only we can apply our standard reflection matrix.

$$\therefore \begin{vmatrix} -2 & 1 & 1 \\ -1 & 2 & 1 \\ -2 & 2 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{vmatrix} = \begin{vmatrix} -2 & -1 & 1 \\ -1 & 0 & 1 \\ -2 & 0 & 1 \end{vmatrix}$$

After this translation the line and triangle will be as shown in Fig. 9.

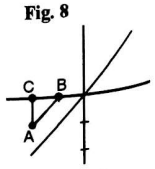


Fig. 9

Now we can perform reflection by using reflection at y = x.

$$\therefore \begin{vmatrix} -2 & -1 & 1 \\ -1 & 0 & 1 \\ -2 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} -1 & -2 & 1 \\ 0 & -1 & 1 \\ 0 & -2 & 1 \end{vmatrix}$$

Now we got the triangle as shown in Fig. 10.

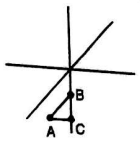


Fig. 10

But now we need to undo the translation which we have done at the start.

$$\therefore \begin{vmatrix} -1 & -2 & 1 \\ 0 & -1 & 1 \\ 0 & -2 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \end{vmatrix} = \begin{vmatrix} -1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{vmatrix}$$

$\therefore$ The final co-ordinates of a reflected triangle are A (–1, 0), B(0, 1), and C(0, 0) which is represented as shown in Fig. 11.

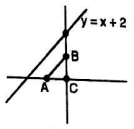


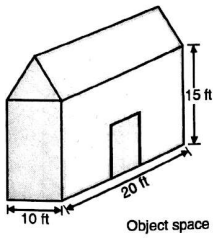
Fig. 11

Chapter 5 : 2D Viewing [Total Marks - 25]

Q. 1(b) Differentiate between Image space and Object space. **(5 Marks)**

Ans. :

Whenever we want to display any scene, we are considering two models of that scene. They are object model and image model. When we are saying object, we are referring a model of object which is stored in computer with the actual scale. It may be millimeter or in Kilometers. The space where object model reside is called object space. Image model is one which appears on display device. The image which we want to draw on display device must be measured in screen co-ordinates. See Fig. 12. There are different types of display devices with variations of screen co-ordinates. So to make screen coordinates device independent we have to use normalized co-ordinates. We must convert the object space units to image space co-ordinates.



Object space

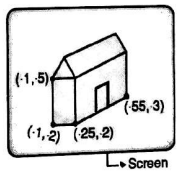


Image space

Fig. 12

For this we have to use scaling transformations.

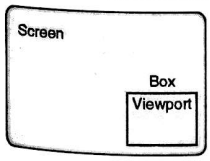


Fig. 13

Before proceeding with transformation, we will first see another term which is called as viewport. It may happen that we may not want to use entire screen for display. We just want some part of screen to display the image. So we will form a rectangular box on screen and in that box only we will display the image. This box is called as viewport. See Fig. 13.

The object space contains the dimensions as actual which is called as world co-ordinate system. Now let us revise all the three terms. We can say, a world co-ordinate area selected for display is called window. An area on a display device to which a window is mapped is called viewport. So the window defines what is to be viewed, clipping means what to omit and viewport defines where it is to be displayed on display device. See Fig. 14.

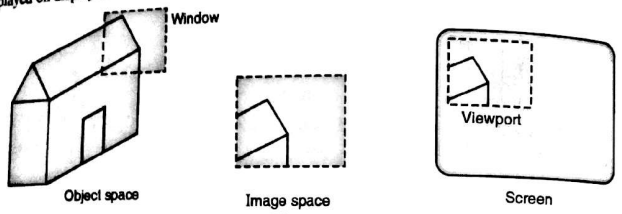


Fig. 14

If we are changing the position of window by keeping the viewport location constant, then the different part of the object is displayed at the same position on the display device. See Fig. 15. As the position of viewport is same, on screen only, the contents will get changed.

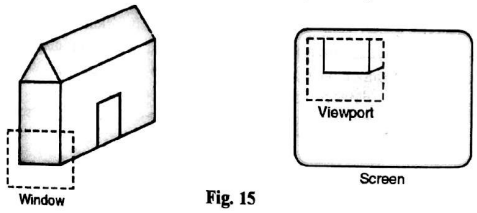


Fig. 15

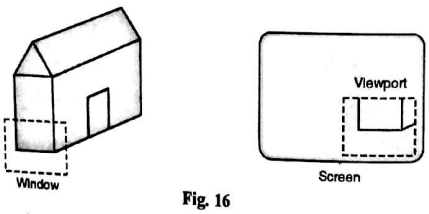


Fig. 16

Similarly if we change the location of viewport then we will see the same part of the object drawn at different places on screen See Fig. 16. In general the mapping of a part of world co-ordinate scene to device co-ordinates is referred as Window to viewport transformation. Sometimes it is also called as viewing transformation or windowing transformation.

Q. 2(a) Explain Liang-Barsky line clipping algorithm. Apply the algorithm to the line with 10 co-ordinates (30, 60) and (60, 25) against the window (xinin, ymin) = (10, 10) and (Xmax, Ymax) = (50, 50). **(10 Marks)**

Ans. : **Please refer Q. 3(b) of May 2012.**

Let's first draw the window and line as shown in Fig. 17.

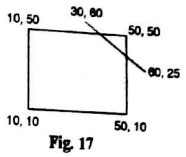


Fig. 17

Given things are XL = 10, YB = 10,

XR = 50, YT = 50.

Let's call a line as AB with its coordinates as X1 = 30,

Y1 = 60, X2 = 60, Y2 = 25

Now we have to find Dx and Dy as

$$Dx = X2 - X1 = 60 - 30 = 30$$

$$Dy = Y2 - Y1 = 25 - 60 = -35$$

Let's calculate the values of parameters P and Q as

$P1 = -Dx$ i.e. $= -30$ $\quad P2 = Dx$ i.e. $= 30$

$P3 = -Dy$ i.e. $= -(-35) = 35$ $\quad P4 = Dy$ i.e. $= -35$

Now $Q1 = X1 - XL = 30 - 10 = 20$ $\quad Q2 = XR - X1 = 50 - 30 = 20$

$Q3 = Y1 - YB = 60 - 10 = 50$ $\quad Q4 = YT - Y1 = 50 - 60 = -10$

Now let's find out values of P for i =1 to 4

$P1 = Q1 / P1 = 20/(-30) = (-2/3)$ $\quad P2 = Q2 / P2 = 20/30 = 2/3$

$P3 = Q3 / P3 = 50/35 = 10/7$ $\quad P4 = Q4 / P4 = (-10)/(-35) = 2/7$

Lets' initialize t1 = 0 and t2 = 1. With this we will find t1 and t2's new values

$t1 = \text{Max}(-2/3, 10/7, 0) = 10/7$ $\quad t2 = \text{Min}(2/3, 2/7, 1) = 2/7$

Now put these values in following expression to find our intersection point as

$X1' = X1 + Dx * t1 = 30 + 30 * (10/7) = 72.71$ $\quad Y1' = Y1 + Dy * t1 = 60 + (-35) * (10/7) = 10$

And with t2 it will be

$X2' = X1 + Dx * t2 = 30 + 30 * (2/7) = 38.57$ $\quad Y2' = Y1 + Dy * t2 = 60 + (-35) * (2/7) = 50$

From this we will come to know that a point (38.57, 50) is an intersection point with respect to top edge of the window boundary. So we need to discard the line from point (30,60) to (38.57,50) and consider the line (38.57,50) to (60,25). We need to repeat the same procedure for finding the intersection point with right edge of the window also and then we will get the line which is surely lying inside the window.

Q. 4(a) Explain Weiber-Atherton algorithm for polygon clippling. What are its advantages over other polygon clipping algorithms ? **(10 Marks)**

Ans. : **Please refer Q. 4(a) of May 2012.**

Chapter 6 : 3D Geometric Transformations and Viewing [Total Marks - 20]

Q. 2(b) Explain parallel and perspective projections. Perform a perspective projection of the unit cube when the centre of projection is at xe = 10 ye = 10 on to z = 0 plane. **(10 Marks)**

Ans. : **Please refer Q. 5(b) of Dec. 2012.**

For perspective projection, we are taking projections on xy plane i.e. with $z = 0$ plane. Given is center of projection as $x_c = 10$ and $y_c = 10$. But to take projection on $z = 0$ plane our center of projection must be behind the xy plane i.e. z_c must be negative. The value of z_c is not given in the problem definition. So we are solving this example using general form. Let's represent this information in pictorial form.

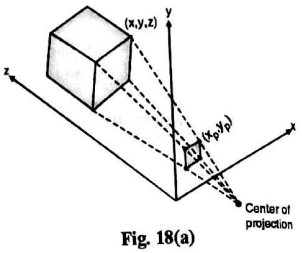


Fig. 18(a)

Now let's find out the x_p and x_p point on xy plane. So the distance of the center of projection from the xy plane, i.e. plane whose $z = 0$ is called 'd'. The projection of a point is the intersection of the straight line from that point to the center with the plane $z = 0$. Fig. 18(b), shows side view of the projection of the point (x, y, z) which gives projected point (x_p, y_p).

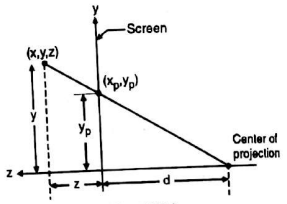


Fig. 18(b)

To draw the image of a unit cube object we have to find the values of (x_p, y_p), as (x_p, y_p) is intersection point of projection with screen. To obtain y_p, we have to say

$$\frac{y}{(z+d)} = \frac{y_p}{d}$$

$$\therefore \quad y_p = \left(\frac{y}{(z+d)}\right) \cdot d$$

Similarly to obtain x_p, we have to consider top view of Fig. 18(c). It will be as shown in Fig. 18(c).

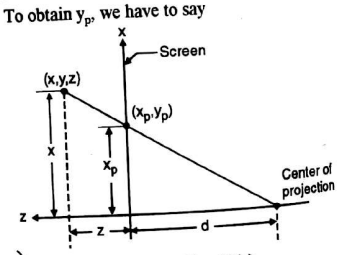


Fig. 18(c)

Here, to find x_p, we have to say

$$\frac{x}{(z+d)} = \frac{x_p}{d} \qquad \therefore x_p = \left(\frac{x}{(z+d)}\right) \cdot d$$

$\therefore$ The co-ordinates of projected point on screen are

$$x_p = \frac{(d * x)}{(d+z)} \quad \text{and} \quad y_p = \frac{(d * y)}{(d+z)} \quad z_p = 0$$

It is not compulsory to take $z = 0$ and center on z-axis. We can have center on any other axis also, but then we have to use translation also and due to this computations will increase. Here we are using z-axis as it generates simple formula to find projected point on screen.

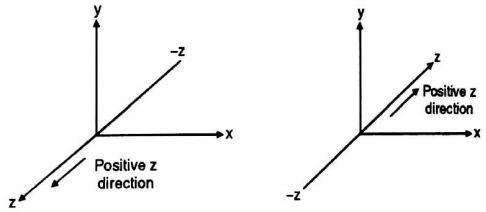
Q. 6(a) Explain 3D geometry in detail. **(10 Marks)**

Ans. : **3D Geometry :**

Until now we have seen two dimensional objects and the operations which we can perform on them. We know that to represent a 2D object we need x and y co-ordinates. It means to draw a 2D object

we need width and height of that object. But in nature objects are having three dimensions. So to represent such objects naturally we need three parameters. Out of these three parameters, two we know. They are width, i.e. x co-ordinate, and height, i.e. y co-ordinate. In 3D we need one additional parameter, which is called as depth and is represented by z co-ordinate. In 2D, only x and y-axis are there whereas in 3D, we are having 3 axes i.e. x, y and z. These three axes are arranged in such a way that they are normal to each other.

Now it is important to recognize that there are two different orientations for the z co-ordinate axis. A right handed system (RHS) has the z-axis pointing towards the viewer. If our right hand's four fingers (i.e. other than thumb) are curling from x to y direction, then the thumb of right hand indicates positive z direction. A left handed system (LHS) has the z-axis pointing away from the viewer. Fig. 19 shows both right handed and left handed system. If our left hand's four fingers (i.e. other than thumb) are curling from x to y direction, then the thumb of left hand points positive z direction. As most of geometry uses right handed system so we adopt the same for our use.



(a) Right handed system **(b) Left handed system**

Fig. 19

Now we can represent any point by a set of three values (x, y, z). Here x represents horizontal distance, i.e. width, along x-axis; y represents vertical distance i.e. height, along y-axis; and z represents depth along z-axis. Fig. 20 shows how to plot a 3D point. Let us plot a point A (3, 2, 1).

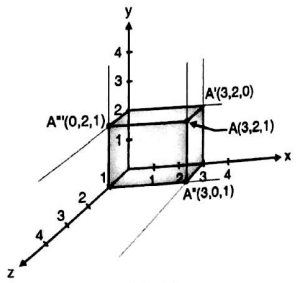


Fig. 20

To plot a 3D point we will follow some steps :

Step 1 : Plot a point A″(3, 0, 1).

To plot this point we have to draw one parallel line to z-axis at a distance of 3 units i.e. x = 3. Now draw parallel line to x-axis at a distance of 1 unit i.e. z = 1. The point where these two lines intersects will be a point having x = 3 and z = 1 but y = 0.

Step 2 : As we want a point as (3, 2, 1) we have to lift the point A″ (3, 0, 1) up by 2 units i.e. y = 2. So draw a parallel line to y-axis from A″ (3, 0, 1) point. Similarly, draw a point A′(3, 2, 0) and draw a parallel line from this point A′(3, 2, 0) to z-axis. Similarly, plot a point A‴ (0, 2, 1) and then draw a parallel line form this point to x-axis. The point at which these three lines intersect is our required point A (3, 2, 1). Fig. 21 shows different 3D points.

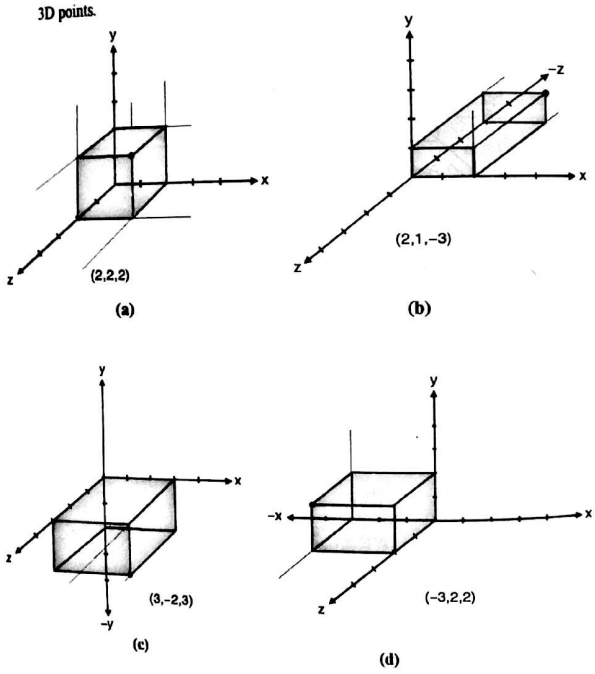


Fig. 21

Like this we can represent a line also. In 2D equation of a line is,

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

i.e. As x changes, y also changes accordingly.

Similarly in 3D as x changes both y and z will get changed accordingly. So to represent a line in 3D we need a pair of equations.

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

and

$$\frac{z - z_1}{x - x_1} = \frac{z_2 - z_1}{x_2 - x_1}$$

It requires the (x_1, y_1, z_1) and (x_2, y_2, z_2) i.e. the co-ordinates of two points. We can represent the equation of a line in parametric form also.

$$x = x_1 + (x_2 - x_1)\,u$$

$$y = y_1 + (y_2 - y_1)\,u$$

$$z = z_1 + (z_2 - z_1)\,u$$

Chapter 7 : Hidden Surfaces [Total Marks - 10]

Q. 5(a) Explain Painter's algorithm. **(10 Marks)**

Ans. : **Please refer Q. 6(a) of May 2012.**

Chapter 8 : Illumination Models and Surface Rendering [Total Marks - 20]

Q. 5(b) Explain Gouraud and Phong shading with their advantages and disadvantages. **(10 Marks)**

Ans. : **Please refer Q. 7(a) of May 2012.**

Advantag Gouraud shading : Removal of discontinuities associated with the constant shading model.

Disadvantage Gouraud shading : Highlighted surfaces are sometimes displayed with anomalous shape and the linear intensity interpolation can use bright or dark intensity strips. This effect can be reduced by using Phong shading method.

Advantage of Phong shading : It displays more realistic surfaces and gives more accurate results. The effect of reducing bright or dark intensity srips is done by Phong shading.

Disadvantage of Phong shading : It needs more calculations and increases the shading cost.

Q. 7(a) Explain RGB and CMY solar models. **(10 Marks)**

Ans. :

RGB : Please refer Q. 4(b) of May 2012.

CMY :

Whenever the primary colors are Cyan, Magenta and Yellow (CMY) in a color model, then it will be useful for describing color output on printer. These three colors are placed on white paper, when we want to draw any object. As we know that cyan can be formed by adding green and blue color. It means there is no red component, i.e. Red color is absorbed or subtracted by the ink. Similarly magenta absorbs green and yellow removes the blue colors. So by using this complementary relationship property between red, green, blue i.e. (RGB) and cyan, magenta, yellow i.e. (CMY) makes it easy to convert from one color model to the other.

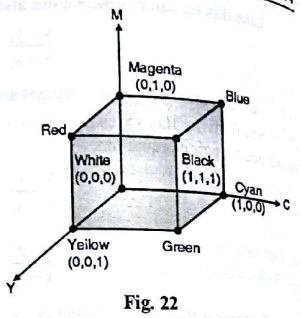


Fig. 22

Above Fig. 22 shows a unit cube representation for CMY model. Here white color is represented by origin and black is represented by (1, 1, 1). Here principle axes (x, y, z) are mapped with cyan, magenta and yellow colors. Other colors are generated, by adding intensities of these (CMY) primary colors, such as green will be the combination of cyan and yellow and is represented as (1, 0, 1). If we assume cyan, magenta and yellow colors as exact complements of red, green and blue then we can represent this relationship as

$$C = 1 - R, \quad M = 1 - G, \quad Y = 1 - B$$

In matrix form it will be

$$\begin{vmatrix} C \\ M \\ Y \end{vmatrix} = \begin{vmatrix} 1 \\ 1 \\ 1 \end{vmatrix} - \begin{vmatrix} R \\ G \\ B \end{vmatrix}$$

Usually printers with CMY model are using four colors i.e. Cyan, Magenta, Yellow and Black. Theoretically mixing all three i.e. cyan, magenta, and yellow should produce black, but it produces dark gray instead of black. So the fourth color is used separately as black.

Chapter 9 : Curves and Fractals [Total Marks - 10]

Q. 7(b) State the properties of Bezier curves. How can a Bezier surface be generated from a Bezier curve ? **(10 Marks)**

Ans. : **Please refer Q. 8(b) of May 2012.**

□□□

Dec. 2013

Chapter 1 : Introduction to Computer Graphics [Total Marks - 10]

Q. 7(b) Write short note on : Raster techniques

Ans. : **Please refer Q. 7(e) of Dec. 2012.** **(5 Marks)**

Q. 7(c) Write short note on : Display file interpreter

Ans. : **Please refer Q. 2(a) of Dec. 2012.** **(5 Marks)**

Chapter 2 : Output Primitives [Total Marks - 20]

Q. 1(a) Explain character generation methods. **(5 Marks)**

Ans. :

Usually characters are generated by hardware. But we can generate/prepare characters by software also. There are three primary methods for character generation. First is called as stroke method or vector character generation and the second is called as dot-matrix or bitmap method and third is called as starbust method.

(1) Stroke method/vector character generation method :

This method creates characters by using a set of line segments. We could build our own stroke method by vector generation algorithm or by using any line generation method. To produce a character we will give a sequence of commands that defines the start point and end points of the straight lines. By using this we can change the scale of the characters. We can make a character twice as large as its original size. Similarly we can get characters slanted also. By using this method we can change the style of characters also. Different character styles are shown in Fig. 1.

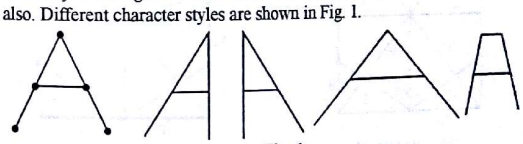


Fig. 1

(2) Dot-matrix or bit-map method :

In this scheme, characters are represented by an array of dots. The size of this array may vary. An array of 5 dots wide and 7 dots high is generally used, but 7 × 9 and 9 × 13/14 are also used. We can select any size of array. This array is like a small buffer, just big enough to hold a single character. The dots are the pixels of the small array. Placing the character on the screen then becomes a matter of copying pixel values from small character array into some portion of the screens frame buffer. See Fig. 2.

A bitmap font uses a rectangular pattern of pixels to define each character.

The entire font can be loaded into an area of memory called font cache displaying character means then copying characters image from font cache into the frame buffer at the desired position. Bitmap fonts require more space, because each variation (size or format) must be stored.

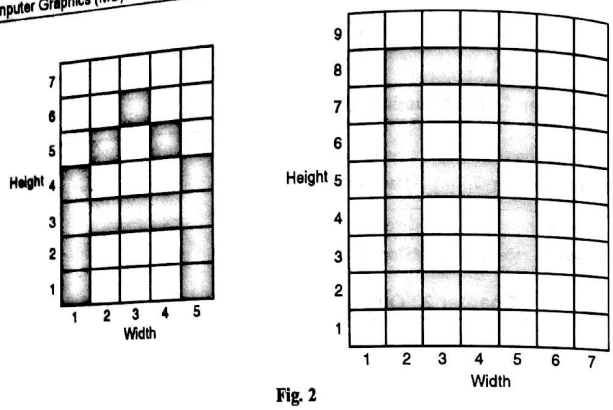


Fig. 2

(3) Starburst method : In this method a fix pattern of line segments are used to generate characters. There are 24 line segments and out of these 24 line segments, segments required to display for particular character are highlighted. This method of character generation is called Starburst method because of its characteristic appearance. The pattern for particular character is stored in the form of 24 bit code. Each bit representing one line segment. The bit is set to one to highlight the line segment. Otherwise it is set to zero.

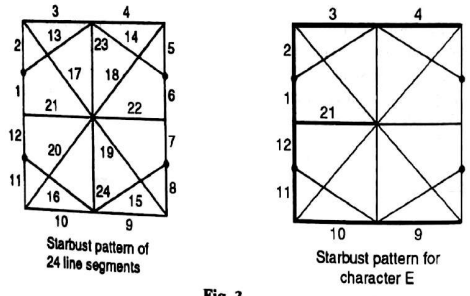


Fig. 3

24 bit code for character E is,

Bit number

24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1

But its main disadvantage is, it requires more memory and requires additional code conversion programs to display characters from the 24 bit code.

Q. 1(c) What is antialiasing, how can it be reduced. **(5 Marks)**

Ans. : **Please refer Q. 1(d) of May 2012.**

Q. 2(b) Derive Bresenham's line drawing algorithm. Plot a line by using Bresenham's line generation algorithm from (1, 1) to (5, 3). **(10 Marks)**

Ans. :

Derive Bresenham's line drawing algorithm : Please refer Q. 2(a) of May 2012.

Plot a line by using Bresenham's line generation algorithm :

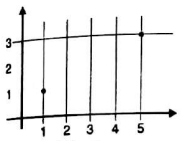


Fig. 4

$x_1 = 1, \quad y_1 = 1$

$x_2 = 5, \quad y_2 = 3$

$\therefore Dx = x_2 - x_1 = 4$

$Dy = y_2 - y_1 = 2$

$G = 2\,Dy - Dx = 2\,(2) - (4) = 0$

Plot 1st point (1, 1) here as $|Dx| > |Dy|$ that means line is Gentle slope, so here we have to move on x till x_2 i.e. 5. After ploting 1st point as (1, 1) increase x by 1

$\therefore x = 2$ Here $G = 0$

We have to increase y by 1 and update G as,

$G = G + 2\,(Dy - Dx) = 0 + 2\,(2 - 4) = -4$

So, plot next point as (2, 2) then again increase x by 1. Now it will become x = 3.

Here $G = -4$.

So, don't increase y just update G only as

$G = G + 2\,Dy = -4 + 2\,(2) = 0$

So, plot (3, 2) go on doing this till x reaches to x_2. We will get points as,

X	Y
1	1
2	2
3	2
4	3
5	3

So, the final points on line will be as shown in Fig. 5.

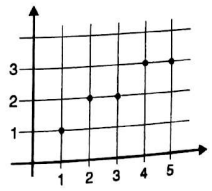


Fig. 5

Chapter 3 : Filled Area Primitives [Total Marks - 15]

Q. 1(b) Explain inside outside test used in filling algorithm. **(5 Marks)**

Ans. : **Please refer Q. 1(a) of Dec. 2012.**

Q. 2(a) Explain flood fill algorithm using 8-connected approach. Given its advantages and disadvantages. **(10 Marks)**

Ans. : **Please refer Q. 2(b) of May 2012.**

Chapter 4 : 2D Geometric Transformations [Total Marks - 20]

Q. 3(a) Translate the square ABCD whose co-ordinates are A(0, 0), B(3, 0), C(3, 3) and D(0, 3) by 2 units in y - direction. **(10 Marks)**

Ans. :

Here we have to divide the given problem into two parts. First part is translation and second will be scaling.

1st translation : The square ABCD will be represented at matrix in the following way.

$$\begin{vmatrix} 0 & 0 & 1 \\ 3 & 0 & 1 \\ 3 & 3 & 1 \\ 0 & 3 & 1 \end{vmatrix}$$

We have to translate this original square by 2 units in both directions. So here t_x and t_y will equal to 2. So the translation matrix will be

$$T = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 2 & 1 \end{vmatrix}$$

After performing translation we will get,

$$\begin{vmatrix} 0 & 0 & 1 \\ 3 & 0 & 1 \\ 3 & 3 & 1 \\ 0 & 3 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 2 & 1 \end{vmatrix} = \begin{vmatrix} 2 & 2 & 1 \\ 5 & 2 & 1 \\ 5 & 5 & 1 \\ 2 & 5 & 1 \end{vmatrix}$$

The pictorial representation of this translation step will be as shown in Fig. 6.

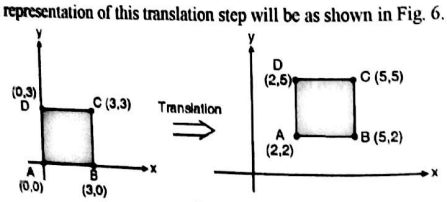


Fig. 6

Now we have to perform second step.

2nd scaling : Here we have to scale this translated image by 1.5 units in x-direction and by 0.5 units in y direction.

Scaling matrix will be

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

Here $s_x = 1.5$ and $s_y = 0.5$

∴ The scaling matrix will become

$$S = \begin{vmatrix} 1.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

∴ The translated image will become as

$$\begin{vmatrix} 2 & 2 & 1 \\ 5 & 2 & 1 \\ 5 & 5 & 1 \\ 2 & 5 & 1 \end{vmatrix} * \begin{vmatrix} 1.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 3 & 1 & 1 \\ 7.5 & 1 & 1 \\ 7.5 & 2.5 & 1 \\ 3 & 2.5 & 1 \end{vmatrix}$$

The pictorial representation of this step will be as shown in Fig. 7.

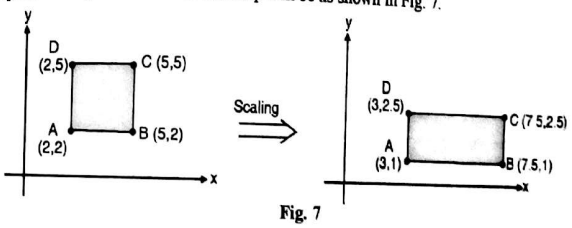


Fig. 7

Q. 3(b) Explain Homogeneous co-ordinates in detail. **(10 Marks)**

Ans. :

To rotate a point (x, y) with respect to a point other than the origin i.e. Q as shown in Fig. 8, then in that case we cannot use our rotation matrices, because they are considering reference point as origin. So to handle such situations we have to use series of transformations. i.e. we have to first translate that image until the center of rotation becomes origin. Now we can perform rotation with respect to origin by angle θ in clockwise or in anticlockwise direction. But it doesn't finish our job, we have to again translate this new image back to its position by shifting the image with the same amount which we have used when we shifted image to origin.

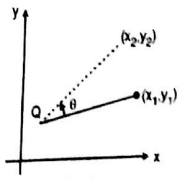


Fig. 8

So here we have to use, three operations or transformations namely translation, rotation and again translation. So instead of three transformations can we accomplish this effect in single transformation for the sake of simplicity? Yes, we can combine different transformations by using Homogeneous co-ordinates. Homogeneous coordinates are used in composite transformation.

In homogeneous co-ordinates we use 3 × 3 matrix instead of 2 × 2 and introduce additional dummy variable w. Generally, points are represented as (x, y); but here every point is specified by a set of 3 values (xw, yw, w). The first value of homogeneous co-ordinate will be product of x and w, the 2nd will be y and w and the third will be just w. But how we are going to represent this homogeneous point on screen because screen deals with only x and y. so we have to recover x and y from (xw, yw, w). The x and y co-ordinates can be easily recovered by dividing the 1st and 2nd number by the third. We are not really using the 3rd variable w. For simplicity we are keeping the value of w as 1.

In normal co-ordinates the scaling matrix is

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

But in homogeneous co-ordinate it becomes

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

In homogeneous co-ordinates as every point is represented as (xw, yw, w)

$$P_1 * T = P_2$$

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

Dividing first and second number by w, we will get,

$| x \cdot s_x, y \cdot s_y |$ which will be correctly scaled point.

Similarly the anticlockwise rotation matrix can be represented in homogeneous co-ordinates as

$$\begin{vmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{vmatrix} \text{ becomes } \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Let us say we are applying anticlockwise rotation to point P_1 then

$$P_1 * T = P_2$$

$$| xw, yw, w | * \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} = | (xw \cdot \cos\theta - yw \cdot \sin\theta), (xw \cdot \sin\theta + yw \cdot \cos\theta), w |$$

After dividing by w we will get,

$$P_2 = (x \cos\theta - y \sin\theta, x \sin\theta + y \cos\theta)$$

We know that we cannot represent translation by using matrix. For translation we have to say

$$x_2 = x_1 + t_x \quad \text{and} \quad y_2 = y_1 + t_y$$

But by using homogeneous co-ordinates we can represent translation in the form of matrix. It will be

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

Let us see whether it works or not. $P_1 * T = P_2$

$$| xw, yw, w | * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{vmatrix} = | (xw + t_x w), (yw + t_y w), w |$$

After dividing by w we will get translated point P_2 as ; $| x + t_x, y + t_y |$

So, by using homogeneous co-ordinates we can represent translation in the form of matrix and also we can combine different transformations to form a single transformation matrix. Normalized device co-ordinate is a good example of transformation. In normalized device co-ordinates screen is treated as 1 × 1 size. See Fig. 9. We can convert screen co-ordinates to normalized co-ordinates by using

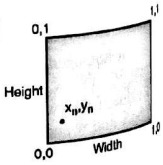


Fig. 9

$$x_n = \frac{x_s - \text{widthstart}}{\text{width}} \qquad y_n = \frac{y_s - \text{heightstart}}{\text{height}}$$

where x_n, y_n represents normalized co-ordinates and x_s, y_s represents screen co-ordinates. Similarly we can convert normalized co-ordinates to screen co-ordinates by using

$$x_s = \text{width} * x_n + \text{widthstart} \qquad y_s = \text{height} * y_n + \text{heightstart}$$

By using transformation we can form a single matrix which will convert normalized co-ordinates to screen co-ordinates. Here to convert normalized co-ordinate to screen co-ordinate i.e. x co-ordinate of screen, we are multiplying x_n with width and then adding widthstart. Similarly for y co-ordinate of screen we are multiplying y_n with height and then adding heightstart. That means in terms of transformation we have to say, first scale x_n or y_n by width or height and then translate it by width-start or height-start.

Let us create a full transformation matrix for this.

$$P_1 * T = P_2$$

But T is itself a combination of Scaling and Translation.

$$\therefore T = \begin{vmatrix} \text{width} & 0 & 0 \\ 0 & \text{height} & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \text{widthstart} & \text{heightstart} & 1 \end{vmatrix}$$

(Scaling) (Translation)

$$\therefore T = \begin{vmatrix} \text{width} & 0 & 0 \\ 0 & \text{height} & 0 \\ \text{widthstart} & \text{heightstart} & 1 \end{vmatrix}$$

Chapter 5 : 2D Viewing [Total Marks - 10]

Q. 6(b) Define the window, view port and derive window to viewport transformation. **(10 Marks)**

Ans. :

This question is very important from theory examinations point of view. Here we have to first explain the meaning of window and viewport. Then the necessity of window to viewport transformation and then we have to explain following points.

Translation of lower left corner of window to origin. Form a translation matrix for this

Then scale the window to viewport. Use scaling factor s_x and s_y.

$$S_x = \frac{\text{width of viewport}}{\text{width of window}} \qquad S_y = \frac{\text{height of viewport}}{\text{height of window}}$$

Form a scaling matrix for this. Then again translate the viewport to desired location on screen by using translation matrix. Combine all the three transformation matrices to a single matrix. For this question we can make use of diagrams.

Chapter 6 : 3D Geometric Transformations and Viewing [Total Marks - 25]

Q. 5(b) Explain parallel and perspective projection ? Derive the matrix for perspective projection. **(10 Marks)**

Ans. : **Please refer Q. 5(b) of Dec. 2012.**

Q. 6(a) Explain 3D object representation methods. **(10 Marks)**

Ans. : **Please refer Q. 1(b) of Dec. 2012.**

Q. 7(e) Write short note on : 3D clipping. (5 Marks)

Ans. : Please refer Q. 5(b) of May 2012.

Chapter 7 : Hidden Surfaces [Total Marks - 15]

Q. 1(d) Explain z-buffer algorithm for removing hidden surfaces. (5 Marks)

Ans. : Please refer Q. 1(c) of Dec. 2012.

Q. 4(b) Explain Warnock's algorithm. (10 Marks)

Ans. : Please refer Q. 7(e) of Dec. 2012.

Chapter 8 : Illumination Models and Surface Rendering [Total Marks - 05]

Q. 7(a) Write short note on : Colour models (5 Marks)

Ans. : Please refer Q. 4(b) of May 2012 and Q. 7(a) of May 2013.

Chapter 9 : Curves and Fractals [Total Marks - 15]

Q. 5(a) State important properties of Bezier curve. Compare Bezier curves and B-spline curve. (10 Marks)

Ans. : Please refer Q. 6(b) of May 2012.

Compare Bezier curves and B-spline curve :

Sr. No.	Bezier Curve Generation	B-Spline Curve Generation
1	Any shape of the curve is achieved by Bezier curve.	By using Spline, we can achieve smooth curves.
2	Depending on the sequence of control points the shape of the curve gets change.	Depending on the sequence of control points the shape of the curve is not changing.
3	Sharp corners are achieved by passing the curve from the sample points	We can achieve the sharp corners by using the blending function repeatedly for the same point.
4	The curve is always passing from minimum two points, i.e. starting and ending point.	In a special case the curve may not pass from any of the sample point.

Q. 7(d) Write short note on : Fractals (5 Marks)

Ans. : Please refer Q. 1(d) of Dec. 2012.

□□□

Computer Graphics

Statistical Analysis

Chapter No.	May 2014
Chapter 1	10 Marks
Chapter 2	20 Marks
Chapter 3	10 Marks
Chapter 4	10 Marks
Chapter 5	30 Marks
Chapter 6	20 Marks
Chapter 7	-
Chapter 8	20 Marks
Chapter 9	10 Marks

May 2014

Chapter 1 : Introduction to Computer Graphics [Total Marks - 10]

Q. 6(b) Write a short note : Raster scan. (10 Marks)

Ans. :

Raster Scan :

There are two different ways of scanning video displays. These are called as raster and random scans. In raster scan the electron beam follows a fixed path as shown in Fig. 1. The electron beam starts at top left corner of the screen and moves horizontally to the right. This defines a scan line. During the scan the intensity of the beam is modulated according to the pattern of the desired image along the line.

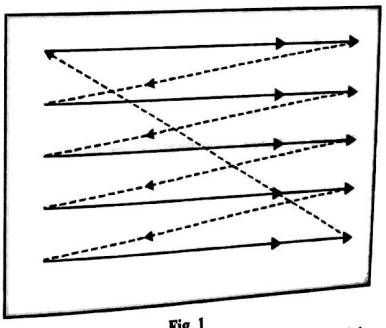


Fig. 1

At the right corner of the screen, the beam becomes off and moved back to the left edge of the screen at the starting point on next line. This is shown as dotted line, which is called as horizontal retrace. This way of scanning is continued till the bottom right corner is reached. At this point, one scan is completed.

The beam is then repositioned at the top left corner of the screen for starting another scan. This movement of beam from bottom right corner to top left corner is called as vertical retrace. Raster graphics can be used in animation. The cost of devices used for raster scan is much cheaper.

Chapter 2 : Output Primitives [Total Marks - 20]

Q. 1(a) Explain Bresenham's line drawing algorithm. Plot a line by using Bresenham's line generating algorithm from (1, 1) to (5, 3). **(10 Marks)**

Ans. :

Steps for Bresenham's line drawing algorithm for Gentle slope (m < 1) :

1) Accept two endpoints from user and store the left endpoint in (x_0, y_0) as starting point.
2) Plot the point (x_0, y_0)
3) Calculate all constants from two endpoints such as D_x, D_y, $2D_y$, $2D_y - 2D_x$ and find the starting value for the G as $G = 2D_y - D_x$.
4) For each column increment x and decide y-value by checking $G > 0$ condition. If it is true then increment y-value and add $(2D_y - 2D_x)$ to current value of G otherwise add $(2D_y)$ to G and don't increment y-value. Plot next point in Fig. 2.

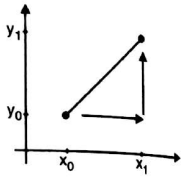


Fig. 2

5) Repeat step 4 till Dx times.

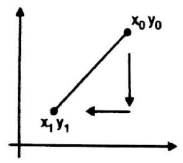


Fig. 3

For steep slope cases just interchange the rolls of x and y i.e. we step along y-direction in unit steps and calculate successive x-values nearest the line path.

If the initial position for a line with positive slope is right endpoint, as shown in Fig. 3, then we have to decrease both x and y as we step from right to left.

Plot a line by using Bresenham's line generating algorithm from point (1,1) to (5,3) :

Given :

$x_1 = 1, \quad y_1 = 1$

$x_2 = 5, \quad y_2 = 3$

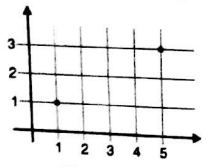


Fig. 4(a)

$\therefore D_x = x_2 - x_1 = 4$

$D_y = y_2 - y_1 = 2$

$G = 2D_y - D_x$

$= 2(2) - (4)$

$G = 0$

Plot 1st point (1, 1) here as $|D_x| > |D_y|$ that means line is Gentle slope, so here we have to move on x till x_2 i.e. 5

After plotting 1st point as (1, 1) increase x by 1

$\therefore x = 2$

Here $G = 0$

We have to increase y by 1 and update G as,

$$G = G + 2(D_y - D_x) = 0 + 2(2 - 4) = -4$$

So, plot next point as (2, 2) then again increase x by 1. Now it will become $x = 3$.

Here $G = -4$.

So, don't increase y just update G only as

$$G = G + 2D_y = -4 + 2(2) = 0$$

So, plot (3, 2) go on doing this till x reaches to x_2.

We will get points as,

X	Y
1	1
2	2
3	2
4	3
5	3

So, the final points on line will be as shown in Fig. 4(b).

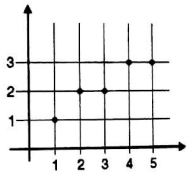


Fig. 4(b)

Q. 2(b) Specify midpoint circle algorithm. Using the same, plot the circle whose radius is 10 units. **(10 Marks)**

Ans. : Midpoint circle generation algorithm :

1) Accept radius 'r' and center of circle (x_c, y_c) from user and plot 1st point on circumference of circle.

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

2) Calculate the initial value of the decision parameter.

$$P_0 = 1 - r$$

3) If we are using octant symmetry property to plot the pixels, then until $(x < y)$ we have to perform following steps.

If P_k is less than 0
modify P_k as $P_k = P_k + 2x + 1$ and
then increase x by 1

otherwise
modify P_k as $P_k = P_k + 2(x - y) + 1$ and increase x by 1 and decrease y by 1

4) Determine the symmetry points in other octants also.
5) Since it have derived all formulas by considering center point as origin. Move each calculated pixel position (x, y) on to the circular path centered on (x_c, y_c) and plot the co-ordinate values as

$x = x + x_c$ and $y = y + y_c$

But actually the center point may be anything, so we have to add that center co-ordinates to midpoint x and y to plot the point.

Example for midpoint circle algorithm having radius is 10 :

Given : Since center of the circle is not specified we will assume it as origin.
So the given things will be $r = 10, x_c = 0, y_c = 0$
Plot 1st point as (0, r) i.e. $y = r$
$\therefore$ Plot (0, 10)
Like that find out other points from (0, 10) by using symmetry property
So plot (10, 0)
plot (0, – 10)
plot (– 10, 0)
find $P = 1 - r$ $\therefore P = 1 - 10 = -9$
Till $(x < y)$ we have to perform following
If $(P < 0)$
In this case $P = -9$
So we have to increase x by 1 and
modify P as $P = P + 2x + 1$
i.e. $P = -9 + 2(0) + 1 = -9 + 1 = -8$

Here we are not updating y.

$\therefore$ Plot $(x + 1, y)$ i.e. (1, 10)

Continue this process till $(x = y)$ i.e. till angle 45, where x and y becomes equal. Then plot the co-ordinates of this set by using either octant symmetry property. The following table shows co-ordinates of the first octant.

x	y
0	10
1	10
2	10
3	9
4	9
5	8
6	8
7	7

Chapter 3 : Filled Area Primitives [Total Marks - 10]

Q. 3(b) Explain scan line fill algorithm with an example. **(10 Marks)**

Ans. :

Scan line fill algorithm :

In contrast to boundary fill and flood fill algorithm at pixel level, this algorithm is defined at geometric level i.e. co-ordinates, edges, vertices etc. This algorithm starts with first scan line and proceeds line by line toward the last scan line and checks whether every pixel on that scan line satisfies our inside point test or not. i.e. it checks which points on that scan line are inside the polygon.

This method avoids the need for seed point. Let us explain this algorithm by considering an example of convex polygon in Fig. 5. Here the algorithm begins with the first scan line that the polygon occupies i.e. y_{max} and proceeds line by line towards the last scan line i.e. y_{min}.

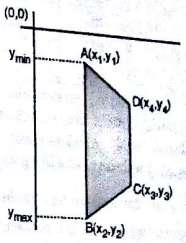


Fig. 5

Consider first and last scan line of polygon and not individual edges. For each edge of the polygon we are storing five attributes.

As we know that to draw any edge or line we need two points. So we are going to store x_{max}, x_{min}, y_{max} and y_{min} of the edges of the polygon along with their slopes. For edge AB we are going to store

$$x_{max} = x_2$$
$$x_{min} = x_1$$
$$y_{max} = y_2$$
$$y_{min} = y_1$$

Here x_{max} and x_{min} has as such no meaning, whereas y_{max} is the maximum y value for a particular edge and y_{min} is the minimum y value for that edge. x_{max} and x_{min} are the corresponding x component of y_{max} and y_{min}. Along with this we are going to store the slope of the edge also.

Let us tabularize this

Edge	y_{max}	x_{max}	y_{min}	x_{min}	Slope
AB	y_2	x_2	y_1	x_1	m_1
AD	y_4	x_4	y_1	x_1	m_2
CD	y_3	x_3	y_4	x_4	m_3
BC	y_2	x_2	y_3	x_3	m_4

For edge AB which is formed by (x_2, y_2) and (x_1, y_1), y_2 is greater than y_1. Therefore $y_{max} = y_2$ and $y_{min} = y_1$ and so $x_{max} = x_2$ and $x_{min} = x_1$. Filling the polygon line by line, at any particular time are not considering all the edges. So there is no point to find whether a particular scan line intersects with each edge or not. In fact we have to select only those edges which are getting intersected by the scan line.

To decide which edges are getting intersected by scan line we are making use of y_{max} of particular edge. One thing is sure that we are storing all the attributes of all edges. Out of those easy solution

attributes we are selecting y_{max} and then we are sorting this y_{max} array. If some swapping is required in this y_{max} then we are going to swap whole edge. After sorting y_{max} array we will get following attribute table.

Edge	y_{max}	x_{max}	y_{min}	x_{min}	Slope
AB	y_2	x_2	y_1	x_1	m_1
BC	y_2	x_2	y_3	x_3	m_4
CD	y_3	x_3	y_4	x_4	m_3
AD	y_4	x_4	y_1	x_1	m_2

Now the y_{max} array is sorted. So find out the intersection of scan line with first two edges from the sorted attribute table i.e. AB and BC. Every time decreasing scan line by 1, from y_{max} to y_{min} of the polygon. In this procedure it may happen that the edge which it have selected to find intersection point may get finished i.e. scan line goes below the y_{min} of selected edge.

In that case we have to discard that edge and select next edge from the sorted table and continue till scan line becomes equal to y_{min} of polygon. Here important point is how to find out the intersection point of scan line with a particular edge. While at the time of storing attributes of edges we have stored the slope of that edge. We can make use of this slope to find intersection point of scan line with edge in Fig. 6.

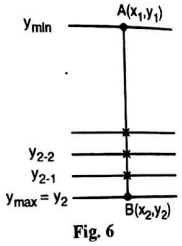


Fig. 6

Chapter 4 : 2D Geometric Transformations [Total Marks - 10]

Q. 5(a) Derive the matrix for 2D rotation about an arbitrary point. **(10 Marks)**

Ans. :

Rotation about an arbitrary point :

In Fig. 7 assume that we have to rotate a point P_1 with respect to (x_m, y_m) then we have to perform three steps. First we have to translate the (x_m, y_m) to origin as shown in Fig. 6. So our translation matrix (T_1) will become

$$T_1 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_m & -y_m & 1 \end{vmatrix} \Rightarrow$$

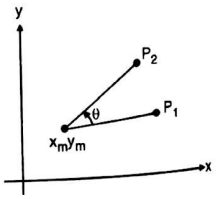


Fig. 7

Here $t_x = -x_m$

and $t_y = -y_m$

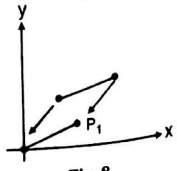


Fig. 8

Then we have to follow second step i.e. rotate it in clockwise or anticlockwise.

In Fig. 9 let's assume as anticlockwise rotation by angle θ. So our rotation matrix will be

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

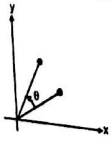


Fig. 9

Now follow 3rd step as shown in Fig. 10 i.e. translate back to original position. So the translation matrix (T_2) will become

$$T_2 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_m & y_m & 1 \end{vmatrix} \Rightarrow$$

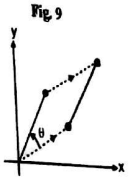


Fig. 10

Now let us form a combined matrix.

$$= \text{Translation} * \text{Rotation} * \text{Translation}$$

$$= T_1 * R * T_2$$

$$= \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_m & -y_m & 1 \end{vmatrix} * \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_m & y_m & 1 \end{vmatrix}$$

$$= \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_m & -y_m & 1 \end{vmatrix} * \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ x_m & y_m & 1 \end{vmatrix}$$

$$= \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -x_m * \cos\theta + y_m * \sin\theta + x_m & -x_m * \sin\theta - y_m \cos\theta + y_m & 1 \end{vmatrix}$$

This transformation matrix is the overall transformation matrix for rotation about arbitrary point (x_m, y_m) by an angle θ in anticlockwise direction.

Chapter 5 : 2D Viewing [Total Marks - 30]

Q. 1(b) Define window, view port and derive window to view port transformation. **(10 Marks)**

Ans. :

Window to viewport transformation :

Whenever we want to display any scene, we are considering two models of that scene. They are object model and image model. When we are saying object, we are referring a model of object which is stored in computer with the actual scale. It may be millimeter or in Kilometers. The space where object model reside is called object space. Image model is one which appears on display device. The image which we want to draw on display device must be measured in screen co-ordinates in Fig. 11.

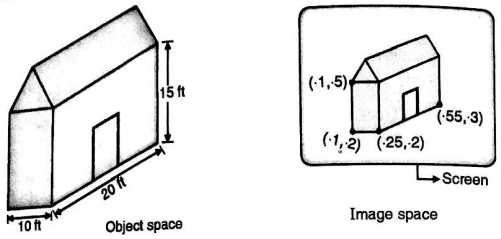


Fig. 11

There are different types of display devices with variations of screen co-ordinates. So to make screen coordinates device independent we have to use normalized co-ordinates. We must convert the object space units to image space co-ordinates. For this we have to use scaling transformations.

Before proceeding with transformation, we will first see another term which is called as viewport. It may happen that we may not want to use entire screen for display. We just want some part of screen to display the image. So we will form a rectangular box on screen and in that box only we will display the image. This box is called as **viewport** in Fig. 12.

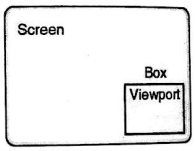


Fig. 12

The object space contains the dimensions as actual which is called as world co-ordinate system. Now let us revise all the three terms. We can say, a world co-ordinate area selected for display is called ***window***. An area on a display device to which a window is mapped is called ***viewport***. So the window defines what is to be viewed, clipping means what to omit and viewport defines where it is to be displayed on display device in Fig. 13.

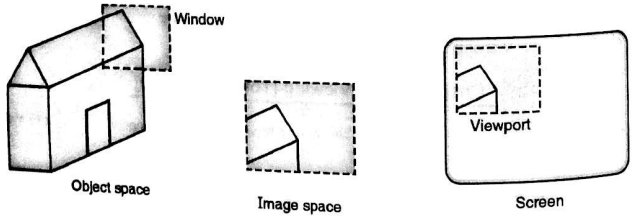


Fig. 13

If we are changing the position of window by keeping the viewport location constant, then the different part of the object is displayed at the same position on the display device in Fig. 14. As the position of viewport is same, on screen only, the contents will get changed.

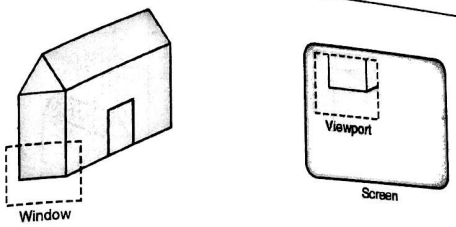


Fig. 14

Similarly if we change the location of viewport then we will see the same part of the object drawn at different places on screen in Fig. 15.

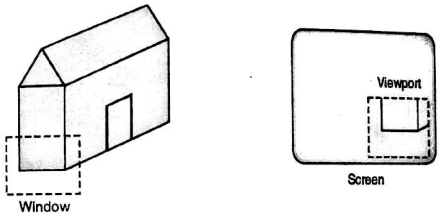


Fig. 15

In general the mapping of a part of world co-ordinate scene to device co-ordinates is referred as **Window to viewport transformation**. Sometimes it is also called as *viewing transformation* or *windowing transformation*. This viewing transformation is a process of three steps.

Step 1 : The object with its window is shifted or translated until the lower left corner of the window matches to the origin in Fig. 16.

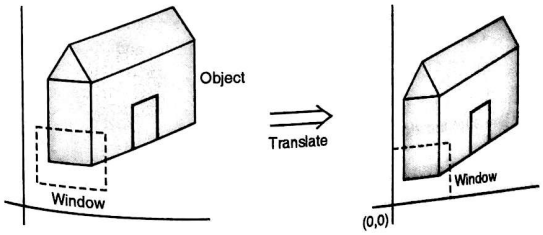


Fig. 16

Step 2 : The object and the window are now scaled until the window has the dimension same as viewport. In other words we are converting the object into image and window in viewport. For this we have to use scaling in Fig. 17.

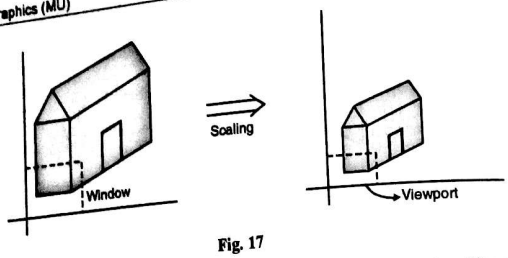


Fig. 17

Step 3 : Now perform another translation to move the viewport to its correct position on the screen in Fig. 18.

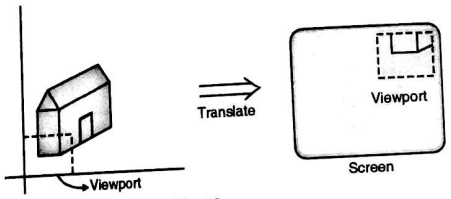


Fig. 18

So the viewing transformation performs three steps :

1. Translation 2. Scaling 3. Translation.

In reality we are using scaling to map window to viewport and translation to place viewport properly on screen.

Q. 4(a) Explain Liang Barsky line clipping algorithm. Apply this algorithm to the line with coordinates (30,60) and (60,25) against the window $(X_{min}, Y_{min}) = (10,10)$ and $(X_{max}, Y_{max}) = (50,50)$. **(10 Marks)**

Ans. :

Liang Barsky line clipping algorithm :

The Liang-Barsky algorithm uses the parametric equation of a line and inequalities describing the range of the clipping box to determine the intersections between the line and the clipping box. With these intersections it knows which portion of the line should be drawn. This algorithm is significantly more efficient than Cohen-Sutherland.

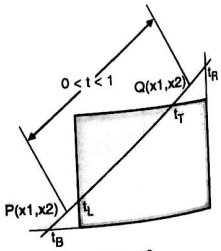


Fig. 19

The ideas for clipping line of Liang-Barsky and Cyrus-Beck are the same. The only difference is Liang-Barsky algorithm has been optimized for an upright rectangular clip window. So we will study only the idea of **Liang-Barsky.** Liang and Barsky have created an algorithm that uses floating-point arithmetic but finds the appropriate end points with at most four computations. This algorithm uses the parametric equations for a line and solves four inequalities to find the range of the parameter for which the line is in the viewport.

Let $P(X_1, Y_1)$, $Q(X_2, Y_2)$ be the line which we want to study in Fig. 19. The parametric equation of the line segment from gives x-values and y-values for every point in terms of a parameter that ranges from 0 to 1. The équations are

$$X = X_1 + (X_2 - X_1)*t = Y_1 + dY*t$$

and $$Y = Y_1 + (Y_2 - Y_1)*t = X_1 + dX*t$$

We can see that when t = 0, the point computed is $P(x_1, y_1)$; and when t = 1, the point computed is $Q(x_2, y_2)$.

Algorithm :

1. Set $t_{min} = 0$ and $t_{max} = 1$
2. Calculate the values of t_L, t_R, t_T, and t_B (t_{values}).
   if $t < t_{min}$ or $t > t_{max}$ then ignore it and go to the next edge
   otherwise classify the t_{value} as entering or exiting value (using inner product to classify)
   if t is entering value set $t_{min} = t$; if t is exiting value set $t_{max} = t$
3. If $t_{min} < t_{max}$ then draw a line from $(x_1 + dx*t_{min}, y_1 + dy*t_{min})$ to $(x_1 + dx*t_{max}, y_1 + dy*t_{max})$ If the line crosses over the window, you will see $(x_1 + dx*t_{min}, y_1 + dy*t_{min})$ and $(x_1 + dx*t_{max}, y_1 + dy*t_{max})$ are intersection between line and edge.

Applying this algorithm to the line with coordinates (30,60) and (60,25) against the window $(X_{min}, Y_{min}) = (10,10)$ and $(X_{max}, Y_{max}) = (50,50)$:

Let's first draw the window and line as shown in Fig. 20.

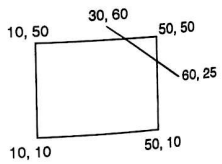


Fig. 20

Given things are $X_L = 10$, $Y_B = 10$, $X_R = 50$, $Y_T = 50$.

Let's call a line as AB with its coordinates as $X_1 = 30$, $Y_1 = 60$, $X_2 = 60$, $Y_2 = 25$

Now we have to find D_x and D_y as

$$D_x = X_2 - X_1 = 60 - 30 = 30$$

$$D_y = Y_2 - Y_1 = 25 - 60 = -35$$

Let's calculate the values of parameters P and Q as

$P_1 = -D_x$ i.e. $= -30$

$P_2 = D_x$ i.e. $= 30$

$P_3 = -D_y$ i.e. $= -(-35) = 35$

$P_4 = D_y$ i.e. $= -35$

Now $Q_1 = X_1 - X_L = 30 - 10 = 20$

$Q_2 = X_R - X_1 = 50 - 30 = 20$

$Q_3 = Y_1 - Y_B = 60 - 10 = 50$

$Q_4 = Y_T - Y_1 = 50 - 60 = -10$

Now let's find out values of P for i = 1 to 4

$P_1 = Q_1 / P_1 = 20/(-30) = (-2/3)$

$P_2 = Q_2 / P_2 = 20/30 = 2/3$

$P_3 = Q_3 / P_3 = 50/35 = 10/7$

$P_4 = Q_4 / P_4 = (-10)/(-35) = 2/7$

Lets' initialize $t_1 = 0$ and $t_2 = 1$. With this we will find t_1 and t_2's new values

$t_1 = \text{Max}(-2/3, 10/7, 0) = 10/7$

$t_2 = \text{Min}(2/3, 2/7, 1) = 2/7$

Now put these values in following expression to find our intersection point as

$X_1' = X_1 + D_x * t_1 \qquad Y_1' = Y_1 + D_y * t_1$

$= 30 + 30 * (10/7) = 60 + (-35) * (10/7)$

$= 72.71 = 10$

And with t_2 it will be

$X_2' = X_1 + D_x * t_2 \quad Y_2' = Y_1 + D_y * t_2$

$= 30 + 30 * (2/7) = 60 + (-35) * (2/7) = 38.57 = 50$

From this we will come to know that a point (38.57, 50) is an intersection point with respect to top edge of the window boundary. So we need to discard the line from point (30,60) to (38.57,50) and consider the line (38.57,50) to (60,25). We need to repeat the same procedure for finding the intersection point with right edge of the window also and then we will get the line which is surely lying inside the window.

Q. 4(b) Explain any one polygon clipping algorithm. **(10 Marks)**

Ans. :

Sutherland-Hodgeman polygon clipping algorithm :

Clip a polygon by considering whole polygon against each boundary edge of the window. We know that to represent a polygon we need a set of vertices. Pass this set of vertices or a polygon to a procedure which will clip the polygon against left edge of the window.

This left clip procedure generates new set of vertices which indicates left clipped polygon. Again this new set of vertices is passed to the right boundary clipper procedure. Again we will get new set of vertices. Then we will pass this new set of vertices to bottom boundary clipper and lastly to top boundary clipper procedure.

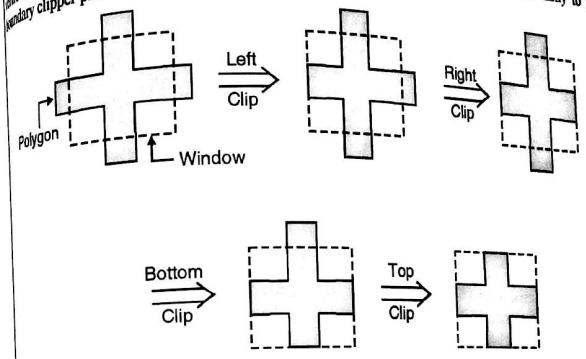


Fig. 21

In Fig. 21 shows four different stages which are required to clip a polygon. At the end of every clipping stage a new set of vertices is generated and this new set or modified polygon is passed to the next clipping stage. After clipping a polygon with respect to all the four boundaries we will get final clipped polygon.

When we clip a polygon with respect to any particular edge of the window at that time we have to consider following four different cases, as each pair of adjacent polygon vertices is passed to a window boundary clipper.

Case 1 :

If the first vertex is outside the window boundary and the second vertex is inside the window, then the intersection point of polygon with boundary edge of window and the vertex which is inside the window is stored in a output vertex list. (i.e. it will act as new vertex points) in Fig. 22.

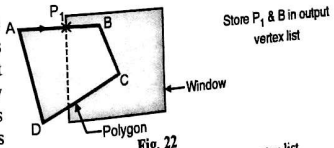


Fig. 22

For edge AB instead of storing vertex A and B we are storing P_1 and B in output vertex list.

Case 2 :

If both, first and second vertices of a polygon are lying inside the window, then we have to store the second vertex only in output vertex list in Fig. 23.

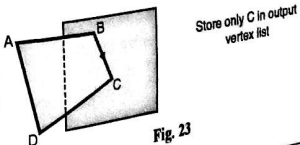
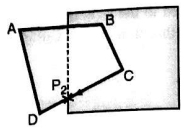


Fig. 23

Case 3 :

If the first vertex is inside the window and second vertex is outside the window i.e. opposite to case 1, then we have to store only intersection point of that edge of polygon with window in output vertex list in Fig. 24.

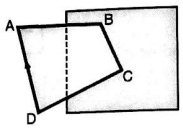


Store P_2 only in output vertex list

Fig. 24

Case 4 :

If both first and second vertices of a polygon are lying outside the window then no vertex is stored in output vertex list in Fig. 25.



Nothing is stored in output vertex list

Fig. 25

Once all vertices have been considered for one clip window boundary, the output list of vertices is clipped against the next window boundary. So the block diagram for this algorithm will be as shown in Fig. 26. This block diagram is a self explanatory.

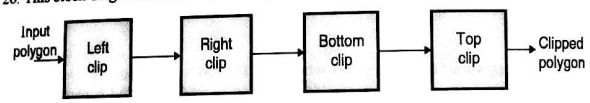


Fig. 26

Chapter 6 : 3D Geometric Transformations and Viewing [Total Marks - 20]

Q. 2(a) Explain parallel and perspective projections and derive the matrix for perspective projection. **(10 Marks)**

Ans. : Parallel Projection :

A parallel projection is formed by extending parallel lines from each vertex of the object until they intersect the plane of the screen. The point of intersection is the projection of the vertex. Then we connect the projected vertices by line segments which correspond to connections on the original object in Fig. 27.

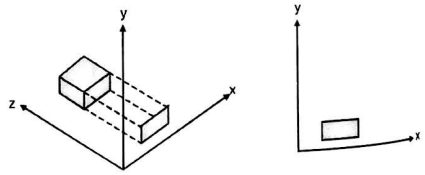


Fig. 27

Here we are taking projection of each vertex of the object till it intersects to the plane of screen. The plane on which we are taking projection is called as view plane. If we want to represent a 3D object on 2D plane, the most simple way is to discard the z co-ordinate. We are discarding the z co-ordinates only when the screen or viewing surface is parallel to the XY plane and the lines of projection are parallel to the z-axis.

A parallel projection preserves relative properties of objects, in x and y direction. Accurate views of the various sides of an object are obtained with a parallel projection, but this does not give us a realistic representation of the appearance of a 3D object. Let us take an example. Suppose there is a 3D cube in space and we want to draw it on screen i.e. on XY plane in Fig. 28.

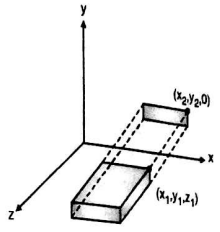


Fig. 28

We obtain the parallel projection of a point (x_1, y_1, z_1) by drawing a line through this point with a certain direction (x_p, y_p, z_p) in space. This line is called as projection vector. The point where this line intersects the plane $z = 0$ is the parallel projection of the given point; it has the coordinate $(x_2, y_2, 0)$.

We will write the equation for a line passing through the point (x_1, y_1, z_1) and the direction of projection.

$$x = x_1 + x_p \cdot u \quad , \quad y = y_1 + y_p \cdot u \quad , \quad z = z_1 + z_p \cdot u$$

where $u = 0$ to 1

$$x_p = (x_2 - x_1) \quad , \quad y_p = (y_2 - y_1) \quad , \quad z_p = (z_2 - z_1)$$

Now we have to find at which point this line intersects the XY plane. It means what will be x and y values at which z becomes zero. If we place $z = 0$ in above equation we will get,

$$z = z_1 + z_p \cdot u$$

put $z = 0$

$$0 = z_1 + z_p \cdot u \qquad \therefore \; u = \frac{-z_1}{z_p}$$

Substituting the value of 'u' in remaining two equations,

$$\therefore \; x = x_1 + \left(\frac{-z_1}{z_p}\right) \cdot u \; , \qquad y = y_1 + \left(\frac{-z_1}{z_p}\right) \cdot u, \qquad z = 0$$

When the projection lines are normal (Perpendicular) to the projection plane, the projection is called orthographic projection. We obtain this projection by setting z co-ordinate to zero.

This projection is a good approximation of the actual projections that the human eye makes in the real world. Orthographic projections do not change the length of line segments which are parallel to projection plane. Other lines are projected with reduced length. Generally we are using orthographic projections in engineering drawings to produce front, side and top view of an object. Fig. 29 shows different view of an object.

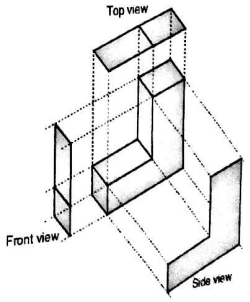


Fig. 29

Here by using orthographic projections we can draw any sides view of an object. But we can also form orthographic projections in such a way that more than one face of an object can be displayed. Such views are called as **axonometric orthographic projections.**

Perspective Projection :

In real world, the objects which are away from the viewer, appear smaller. Perspective projection preserves this property. In perspective projection, the lines of projection are not parallel. Here all projection lines are ending at a single point called center of projection. In real world this center of projection is human eye. For perspective projection in Fig. 30.

To draw the image of object on 2D plane, assume that the object is in space having some x, y, and z values. We have to draw the image on XY plane, so the center of projection should be beyond the XY plane. All projections which are originating from different vertices of object meet at center of projection point, intersecting XY plane. Assuming that the plane XY or screen is placed at z = 0.

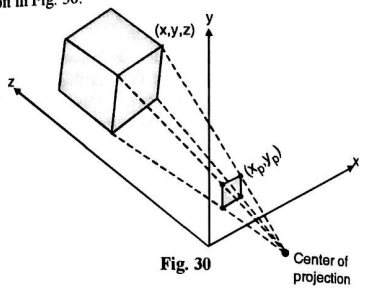


Fig. 30

So the distance of the center of projection from the XY plane, i.e. plane whose z = 0 is called 'd'. With this assumption computations become simple. The projection of a point is the intersection of the straight line from that point to the center with the plane z = 0 in Fig. 31, shows side view of the projection of the point (x, y, z) which gives projected point (x_p, y_p)

To draw the image of an object we have to find the values of (x_p, y_p), as (x_p, y_p) is intersection point of projection with screen. To obtain y_p, we have to say

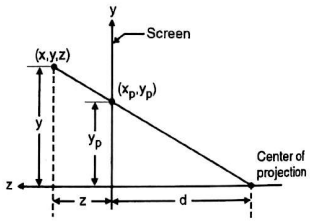


Fig. 31

$$\frac{y}{(z+d)} = \frac{y_p}{d}$$

$$\therefore \quad y_p = \left(\frac{y}{(z+d)}\right) \cdot d$$

Similarly to obtain x_p, we have to consider top view of Fig. 30. It will be as shown in Fig. 32.

Here, to find x_p, we have to say

$$\frac{x}{(z+d)} = \frac{x_p}{d}$$

$$\therefore \quad x_p = \left(\frac{x}{(z+d)}\right) \cdot d$$

$\therefore$ The co-ordinates of projected point on screen are

$$x_p = \frac{(d * x)}{(d+z)}$$

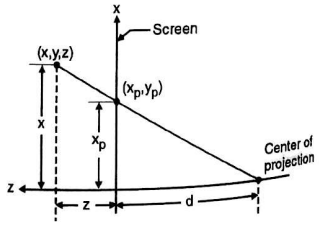


Fig. 32

and $y_p = \frac{(d * y)}{(d+z)}$ $z_p = 0$

It is not compulsory to take z = 0 and center on z-axis. We can have center on any other axis also, but then we have to use translation also and due to this computations will increase. Here we are using z-axis as it generates simple formula to find projected point on screen.

Q. 6(c) Describe the following 3-D representation methods : **(10 Marks)**

(i) Sweep representation (ii) B-REP (iii) CSG

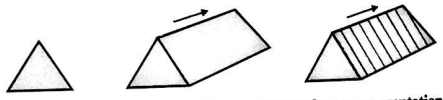
Ans. :

(i) Sweep representation :

Sweep representations are used to construct 3D object from 2D shape. There are two ways to achieve sweep.

Translation sweep :

In translation sweep, the 2D shape is swept along a linear path normal to the plane of the area to construct 3D object. To obtain the wireframe representation we have to replicate the 2D shape and draw a set of connecting lines in the direction of shape, as shown in Fig. 33.



(a) 2D shape (b) 3D object (c) Wireframe representation

Fig. 33

Rotational sweep :

In rotational sweep, the 2D shape is rotated about an axis of rotation specified in the plane of 2D shape to produce three dimensional objects.

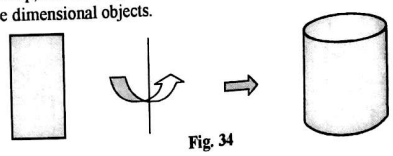


Fig. 34

Fig. 34 shows a rotational sweep creates a cylindrical object. Sweep representations are widely-used in computer vision. However, the generation of arbitrary objects becomes rather difficult using this technique.

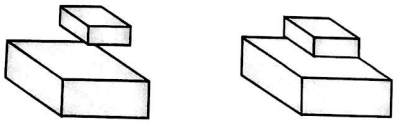
(ii) B-REP :

Boundary representation, B-rep in short, can be considered as an extension to the wireframe model. The merit of a B-rep is that a solid is bounded by its surface and has its interior and exterior. The surface of a solid consists of a set of well-organized faces, each of which is a piece of some surface. Faces may share vertices and edges that are curve segments. Therefore, B-rep is an extension to the wireframe model by adding face information to the latter. There are two types of information in a B-rep topological and geometric. Topological information provides the relationships among vertices, edges and faces similar to that used in a wireframe model.

In addition to connectivity, topological information also includes orientation of edges and faces. Geometric information are usually the set of equations of the edges and faces. The orientation of each face is important. Normally a face is surrounded by a set of vertices using the right-handed rule, the ordering of these vertices for describing a particular face must guarantee that the normal vector of that face is pointing to the exterior of the solid. Normally, the order is counter clockwise. If that face is given by an equation, the equation must be rewritten so that the normal vector at every point on the part that is being used as a face points to the exterior of the solid. Therefore, by inspecting normal vectors one can immediately tell the inside/outside of a solid under B-rep.

(iii) CSG :

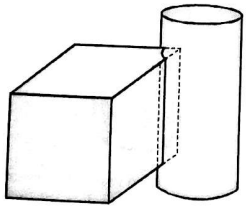
Another technique for solid modeling is to combine the volumes occupied by overlapping 3D objects using Boolean set operations. This modeling tech. is called constructive solid geometry (CSG). It creates a new volume by applying Boolean operators such as union, intersection, or difference to two specified objects. The Figs. 35, 36, 37 show the examples for forming new shapes using Boolean set operations. The Fig. 35(a) shows two rectangle blocks are place adjacent to each other. We can obtain object with the union operation as shown in Fig. 35(b).



(a) Objects **(b) Combined objects**

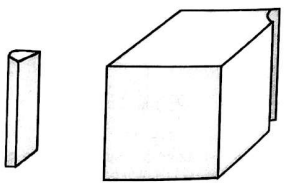
Fig. 35: Combined object by union operator

The Fig. 36 shows the result of intersection operation obtained by overlapping cylinder and cube. With the difference operation, we can obtain the resulting solid as shown in Fig. 37.



(a)

Fig. 36



(b)

Fig. 37

CSG representations are particularly useful for capturing design intent in the form of features corresponding to material addition or removal such as bosses, holes, pockets etc. Some of the important properties of CSG include conciseness, guaranteed validity of solids, computationally convenient Boolean algebraic properties, and natural control of a solid's shape in terms of high level parameters defining the solid's primitives and their positions and orientations. The CSG method uses three dimensional objects such as blocks, pyramids, cylinders, cones, spheres, and closed spline surfaces to generate other solid objects.

Chapter 8 : Illumination Models and Surface Rendering [Total Marks - 20]

Q. 3(a) Explain Gouraud and Phong shading along with their advantages and disadvantages. **(10 Marks)**

Ans. :

Gouraud Shading :

Gouraud shading is an example of intensity interpolation of method. Here polygon surfaces are rendered by linearly interpolating intensity values. This method removes intensity discontinuities between adjacent planes of a surface representation by linearly varying the intensity over each plane. Here intensity values are matched at plane boundaries.

In Gouraud shading all polygon surfaces are accessed by performing following calculations :

At each vertex of polygon, determine average unit normal vector. Calculate vertex intensity.

Linearly interpolate that vertex intensity over the surface of polygon. To determine a normal vector N at vertex V in Fig. 38, we use the average of the normal vectors of the polygons that meet at vertex V.

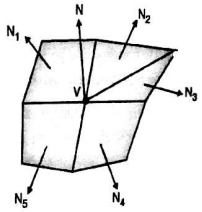


Fig. 38

$$N = N_1 + N_2 + N_3 + N_4 + N_5$$

where N is then normalized by dividing it by 5.

Fig. 39 demonstrates the interpolation scheme. This shading model determines intensity values at the vertices of each polygon. All other intensities for the surface are then calculated from these values in Fig. 39, a scan line is getting intersected with edge AB at point P. To find the intensity of this intersection point P we are interpolating between the intensities of point A and point B. Let's call intensity of point A as I_A and for B as I_B. Now the intensity of point P, which we call as I_p will be calculated as,

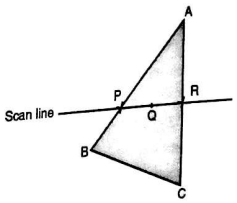


Fig. 39

$$I_p = \frac{y_P - y_B}{y_A - y_B} I_A + \frac{y_A - y_P}{y_A - y_B} I_B$$

By using same procedure we can find the intensity of point R, which will be interpolated from points A and C. Once we know the intensities of P and R point then we can easily find intensities of interior points of the line PR. Let's call intensity for point Q as I_Q.

$$I_Q = \frac{X_R - X_Q}{X_R - X_P} I_P + \frac{X_Q - X_P}{X_R - X_P} I_R$$

This process is repeated for each scan line passing through polygon. By combining this method with hidden surface algorithm we can fill the visible polygons along each scan line.

Advantage :

Removal of discontinuities associated with the constant shading model.

Disadvantage :

Highlighted surfaces are sometimes displayed with anomalous shape and the linear intensity interpolation can use bright or dark intensity strips. This effect can be reduced by using Phong shading method.

Phong Shading :

This method involves approximation of the surface normal at each point along a scan line, then calculating the intensity using the approximated normal vector at that point, displaying more realistic highlights on a surface. Phong shading method performs following steps to render a polygon surface. At each vertex of polygon, determine average unit normal vector. Linearly interpolate the vertex normal. Calculate the pixel intensity of each scan line.

This method interpolates the normal vectors at the bounding points along a scan line in Fig. 40.

In Fig. 40, to find the normal vector N_Q at point Q in a triangle ABC, we first interpolate N_1 and N_2 to get Np. Then we interpolate N_2 and N_3 to get N_R. And finally we interpolate Np and N_R to get NQ. This technique is relatively time-consuming since the illumination model is evaluated at every point using interpolated normal vectors. But it is very effective in dealing with specular highlights.

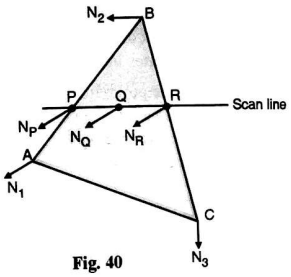


Fig. 40

Advantages :

1. It displays more realistic highlights on a surface.
2. It greatly reduces the Mach - band effect.
3. It gives more accurate results

Disadvantage :

It requires more calculations and greatly increases the cost of shading steeply.

Q. 6(a) Write a short note : Half tonning and dithering techniques. **(10 Marks)**

Ans. :

Halftone :

It is the reprographic technique that simulates continuous tone imagery through the use of dots, varying either in size or in spacing. 'Halftone' can also be used to refer specifically to the image that is produced by this process. Where continuous tone imagery contains an infinite range of colors or grays, the halftone process reduces visual reproductions to a binary image that is printed with only one color of ink. This binary reproduction relies on a basic optical illusion - that these tiny halftone dots are blended into smooth tones by the human eye.

At a microscopic level, developed black and white photographic film also consists of only two colors, and not an infinite range of continuous tones. Halftoning is the process of turning continuous

tone grayscale or color images into a series of dots for printing that fool the eye. Each intensity position in the original scene is replaced with a rectangular pixel grid. This method is used for printing to reproduce photographs in magazines, books or newspaper. For bi-level systems one can represent shading intensities with a half toning method that converts the intensity of each point on a surface into a regular pixel grid that can display a number of intensity level.

The number of intensity level with this method depends on how many pixels we include on the grid. A graphics package employing a halftone technique displays a scene by replacing each position in the original scene with an (n x n) grid of pixels. Intensity level of the corresponding position in the scene determines number of pixels that are turned on. Such a technique of turns on two level systems into one with the five possible intensities is shown in Fig. 41. These levels are labeled 0 through 4.

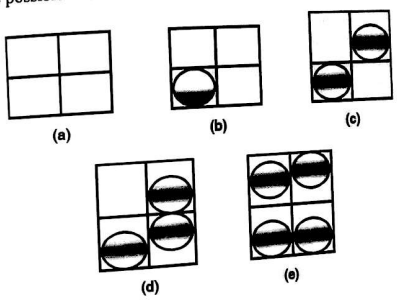


Fig. 41

With this technique one can obtain n2 intensity levels above zero for any (n x n) grid. To avoid unwanted pattern introduction into the original possible, symmetrical pixel arrangements have to be avoided whenever possible. A symmetrical pattern would produce either vertical and horizontal or diagonal streaks in a half toning representation.

To avoid such patterns, select different pixel arrangements for the representation of an intensity level. In Fig. 41(c) can be selected to represent the second intensity level above zero. Another method is to form successive grid patterns with the same pixels turned on i.e. if the pixel is on for one grid level, it is on for all higher levels.

Dithering Techniques :

Dither is an intentionally applied form of noise, used to randomize quantization error, thereby preventing large-scale patterns such as "banding" in images, or noise at discrete frequencies in an audio recording, that are more objectionable than uncorrelated noise. Dither is routinely used in processing of both digital audio and digital video data, and is often one of the last stages of audio production to CD.

Ordered dithering is an image dithering algorithm. It is commonly used by programs that need to provide continuous image of higher colors on a display of less color depth. For example, Microsoft Windows uses it in 16-color graphics modes. It is easily distinguished by its noticeable crosshatch patterns.

The ordered dither technique for bi-level displays also increases the visual resolution without reducing the spatial resolution, by introducing a random error into the image. This random error is added to the image intensity of each pixel before comparison with the selected threshold value. Adding a

completely random error does not yield an optimum result. Because the error pattern is small in size than image it is tiled across the image. This technique is a form of dispersed dot-ordered dither.

Dither should be added to any low-amplitude or highly-periodic signal before any quantization or re-quantization process, in order to de-correlate the quantization noise with the input signal and to prevent distortion; the lesser the bit depth, the greater the dither must be. The results of the process still yield distortion, but the distortion is of a random nature so its result is effectively noise. Any bit-reduction process should add dither to the waveform before the reduction is performed.

There are different types of dither which are as follows :

RPDF stands for "Rectangular Probability Density Function," equivalent to a roll of a dice. Any number has the same random probability of surfacing.

TPDF stands for "Triangular Probability Density Function," equivalent to a roll of two dice (the sum of two independent samples of RPDF).

Gaussian PDF is equivalent to a roll of a large number of dice. The relationship of probability of results follows a bell-shaped, or Gaussian curve, typical of dither generated by analog sources such as microphone preamplifiers. If the bit depth of a recording is sufficiently great, that noise will be sufficient to dither the recording.

Colored Dither is sometimes mentioned as dither that has been filtered to be different from white noise. Some dither algorithms use noise that has more energy in the higher frequencies so as to lower the energy in the critical audio band.

Dithering is the attempt by a computer program to approximate a color from a mixture of other colors when the required color is not available. For example, dithering occurs when a color is specified for a Web page that a browser on a particular operating system can't support.

The browser will then attempt to replace the requested color with an approximation composed of two or more other colors it can produce. The result may or may not be acceptable to the graphic designer. It may also appear somewhat grainy since it's composed of different pixel intensities rather than a single intensity over the colored space.

Dithering also occurs when a display monitor attempts to display images specified with more colors than the monitor is equipped to handle.

Chapter 9 : Curves and Fractals [Total Marks - 10]

Q. 5(b) Explain Bezier curve and also specify the properties of Bezier curve. **(10 Marks)**

Ans. : Bezier Curve :

It is a different way of specifying a curve, rather same shapes can be represented by B-spline and Bezier curves. The cubic Bezier curve require four sample points, these points completely specify the curve in Fig. 42 shows sample Bezier curves.

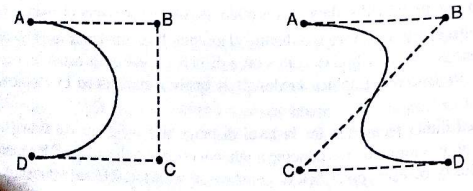


Fig. 42

The curve begins at the first sample point and ends at fourth point. If we need another Bezier curve then we need another four sample points. But if we need two Bezier curves connected to each other, then with six sample points we can achieve it. For this, the third and fourth point of first curve should be made same as first and second point of second curve.

The equations for the Bezier curve are as follows :

$$x = x_4 a^3 + 3x_3 a^2 (1-a) + 3x_2 a (1-a)^2 + x_1 (1-a)^3$$

$$y = y_4 a^3 + 3y_3 a^2 (1-a) + 3y_2 a (1-a)^2 + y_1 (1-a)^3$$

$$z = z_4 a^3 + 3z_3 a^2 (1-a) + 3z_2 a (1-a)^2 + z_1 (1-a)^3$$

Here as the value of 'a' moves from 0 to 1, the curve travels from the first to fourth sample point. But we can construct a Bezier curve without referencing to the above expression. It is constructed by simply taking midpoints in Fig. 43.

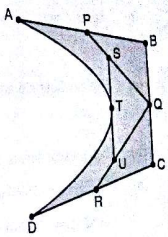


Fig. 43

The points A, B, C, D are the original Bezier curve control points. Here we are having three lines, AB, BC and CD, then we have to find the midpoints of these lines as 'P', 'Q' and 'R' respectively. After that we have to join PQ and QR. Then again find the midpoint of these newly generated lines as 'S' and 'U'.

Then form a line segment between 'S' and 'U'. And find midpoint of this line as 'T'. Now point 'T' will be on Bezier curve. This point 'T' divides the curve into two section, one is (A, P, S and T) and second will be (D, R, U and T).

Thus by taking midpoints, we can find a point on the curve and also split the curve into two sections. We can continue to split the curve into smaller sections, until we have sections so short that they cannot be replaced by straight lines or till the size of section is not greater than the size of pixel.

Properties of Bezier Curve :

1. The basic functions are real in nature.
2. Bezier curve always passes through the first and last control points i.e. curve has same end points as the guiding polygon.
3. The degree of the polynomial defining the curve segment is one less than the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is three, i.e. cubic polynomial.
4. The curve generally follows the shape of the defining polygon.
5. The direction of the tangent vector at the end points is the same as that of the vector determined by first and last segments.
6. The curve lies entirely within the convex hull formed by four control points.
7. The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
8. The curve exhibits the variation diminishing property. This means that the curve does not oscillate about any straight line more than the defining polygon.
9. The curve is invariant under an affine transformation.

□□□