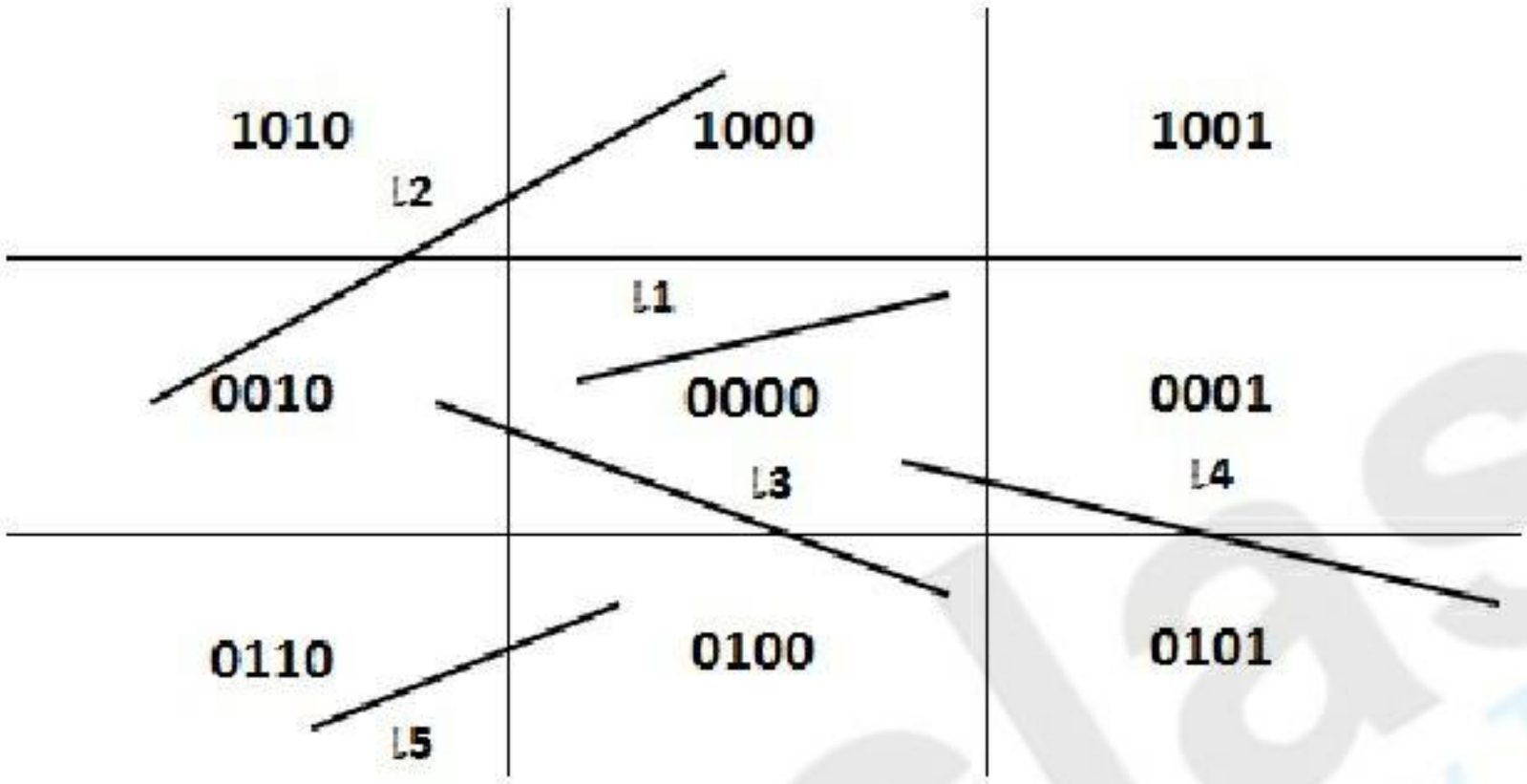
Q.1 (a) Explain the Cohen Sutherland line clipping algorithm with the help of an example.

Ans. The Cohen–Sutherland algorithm is a computer graphics algorithm used for line clipping. The algorithm divides a two-dimensional space into 9 regions and then efficiently determines the lines and portions of lines that are visible in the viewport. Cohen – Sutherland algorithm assigns 4-bit region codes to the end-points of a line and then AND operation is carried out to determine totally visible lines and totally invisible lines. For partially visible lines, the algorithm breaks the line segments into smaller sub-segments by finding intersection with appropriate window edge.

Consider the following example,



For any endpoint (x, y) of a line, the code can be determined that identifies which region the endpoint lies. The code's bits are set according to the following conditions:

1st bit ⇒ Up(U)

2nd bit ⇒ Down(D)

3rd bit ⇒ Left(L)

4th bit ⇒ Right(R)

The sequence for reading the codes' bits is UDLR (Up, Down, Left, Right)

E.g. As window is neither Up nor Down, neither Left nor Right, so, the respective bits UDLR are 0000

Case 1: l1 -> Completely visible i.e. trivial acceptance (both points lie inside the window)

If the UDLR bit codes of the end points (x, y) of a given line is 0000 then line is completely visible. Here this is the case as the end points of line l1 are (0000) and (0000). If this trival acceptance test is failed then, the line segment is passed onto Case 2.

Case 2: l2 & l5 -> Completely invisible i.e. trivial acceptance rejection

If the logical intersection (AND) of the bit codes of the end points of the line segment is ≠ 0000 then line segment is not visible or is rejected.

The end points of line l5 are (0110) and (0100). The logical AND operation of the points give 0100. So line l5 is invisible.

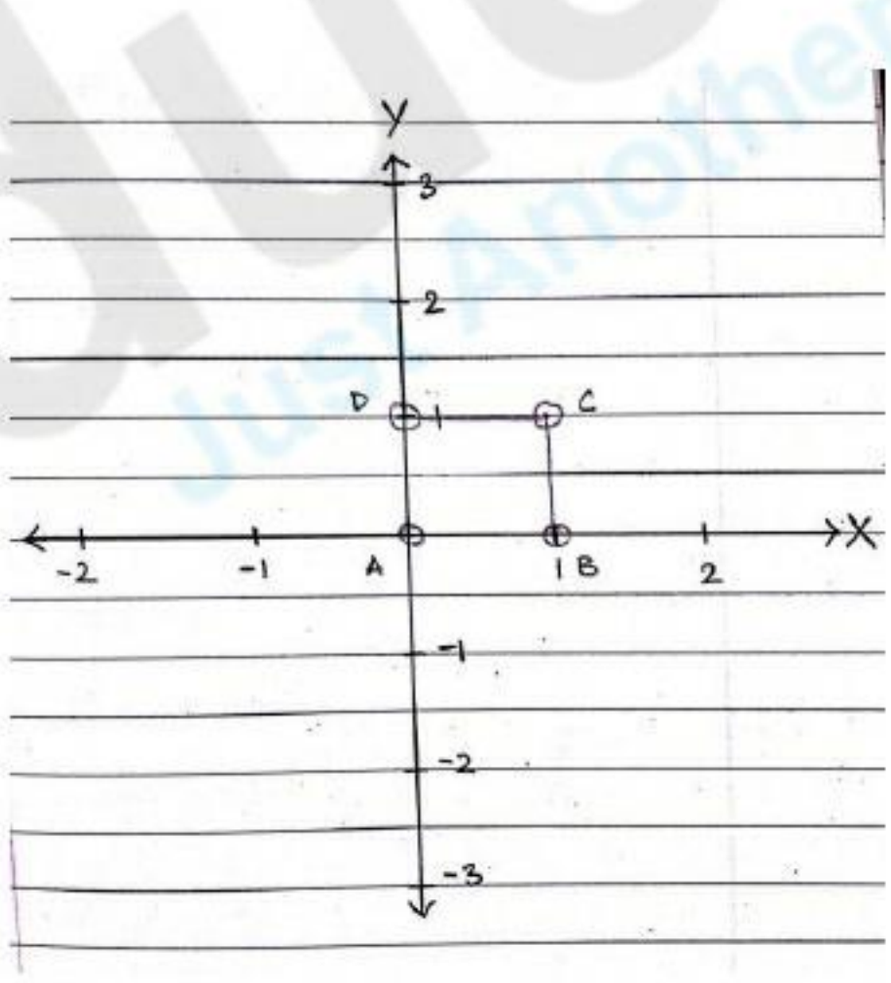
Case 3: l3 & l4 -> If the logical intersection (AND) of the bit codes of the end points of the line segment is ≠ 0000 then line segment is not visible or is rejected. But if one of the end points is 0000 then the line is partially visible. We have to clip the line, for that we have to know the point of intersection of the line with the window

Consider l3, the end points for l3 are 0010 and 0100. The line is partially visible. End point 0010 has '1' in the L-position. Therefore l3 intersects the left border of the window. End point 0100 has '1' in the D-position. Therefor l3 intersects the bottom window of the window too. l3 has to be clipped from both sides. The points of intersection with the window will be (a, y_{min}) and (b, y_{max}).

Q.1 (b) Apply the shearing transformation to square with A(0,0), B(1,0), C(1,1), D(0,1) as given below:

1. **Shear parameter value of 0.5 relative to the line Y_{ref} = -1**
2. **Shear parameter value of 0.5 relative to the line X_{ref} = -1**

Ans.



1. Shear parameter value of 0.5 relative to the line Y_{ref} = -1

The shear parameter value is 0.5 and it is relative to Y_{ref} = -1.

Therefore shear parameter value = -0.5

Transformation matrix for shear transformations is :

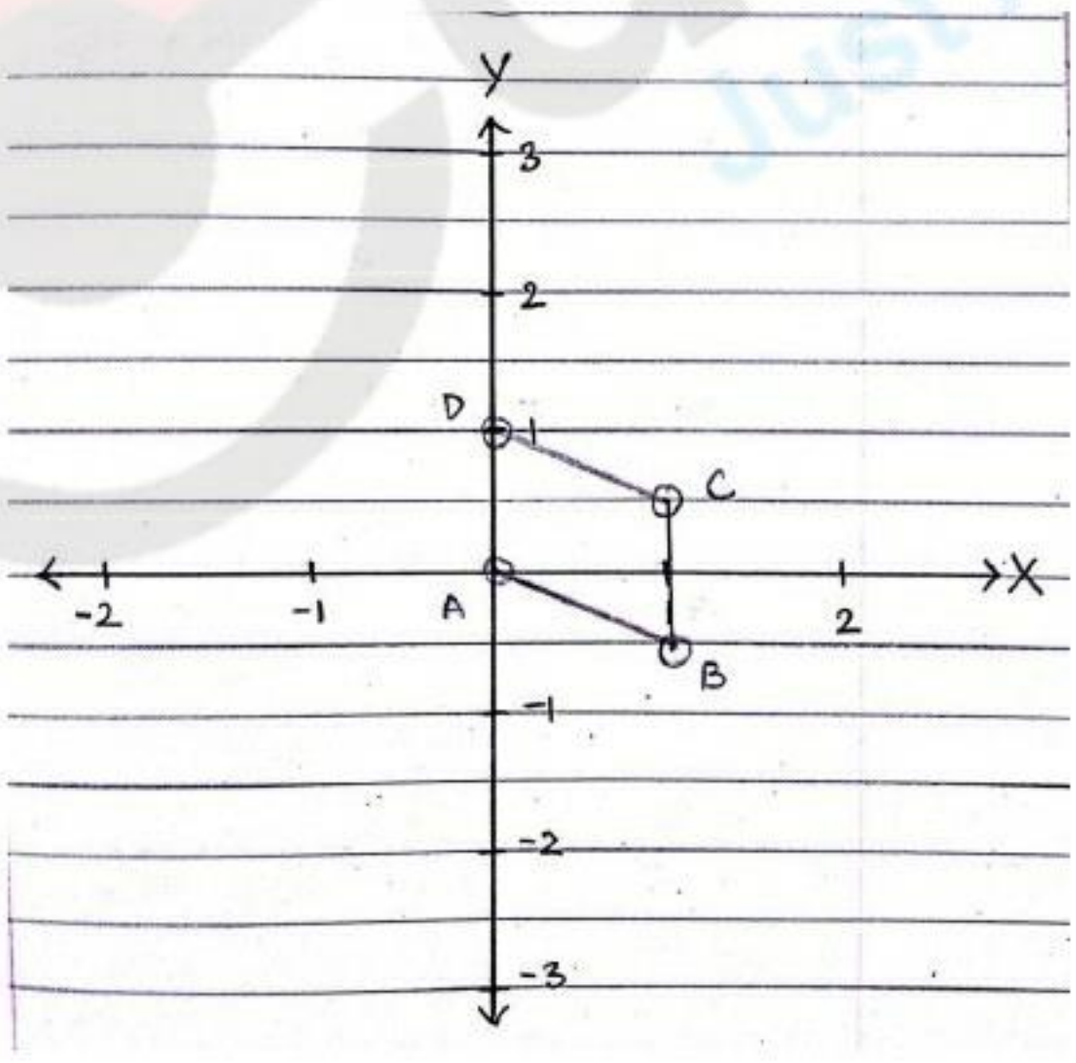
$$(x', y', 1) = (x, y, 1)\begin{pmatrix} 1 & b_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Therefore from the given information:

$$(x', y', 1) = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & -0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$(x', y', 1) = \begin{pmatrix} 0 & 0 & 1 \\ 1 & -0.5 & 1 \\ 1 & 0.5 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

Co-ordinates after transformation are A(0,0), B(1, -0.5), C(1, 0.5) and D(0,1)



2. Shear parameter value of 0.5 relative to the line $X_{ref} = -1$

The shear parameter value is 0.5 and it is relative to $X_{ref} = -1$.

Therefore shear parameter value = -0.5

Transformation matrix for shear transformations is :

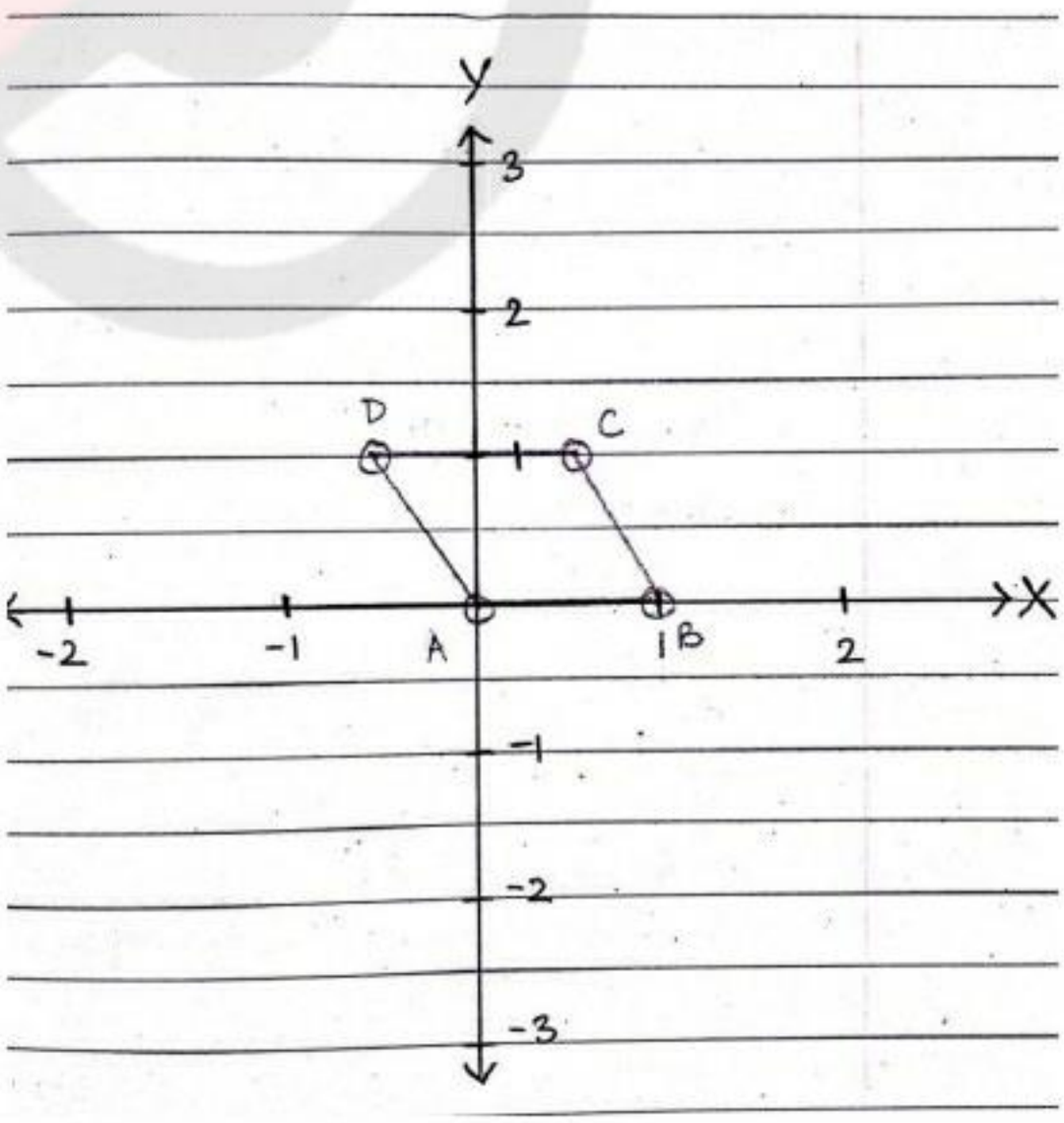
$$(x', y', 1) = (x, y, 1) \begin{pmatrix} 1 & 0 & 0 \\ a_x & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Therefore from the given information:

$$(x', y', 1) = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0.5 & 1 & 1 \\ -0.5 & 1 & 1 \end{pmatrix}$$

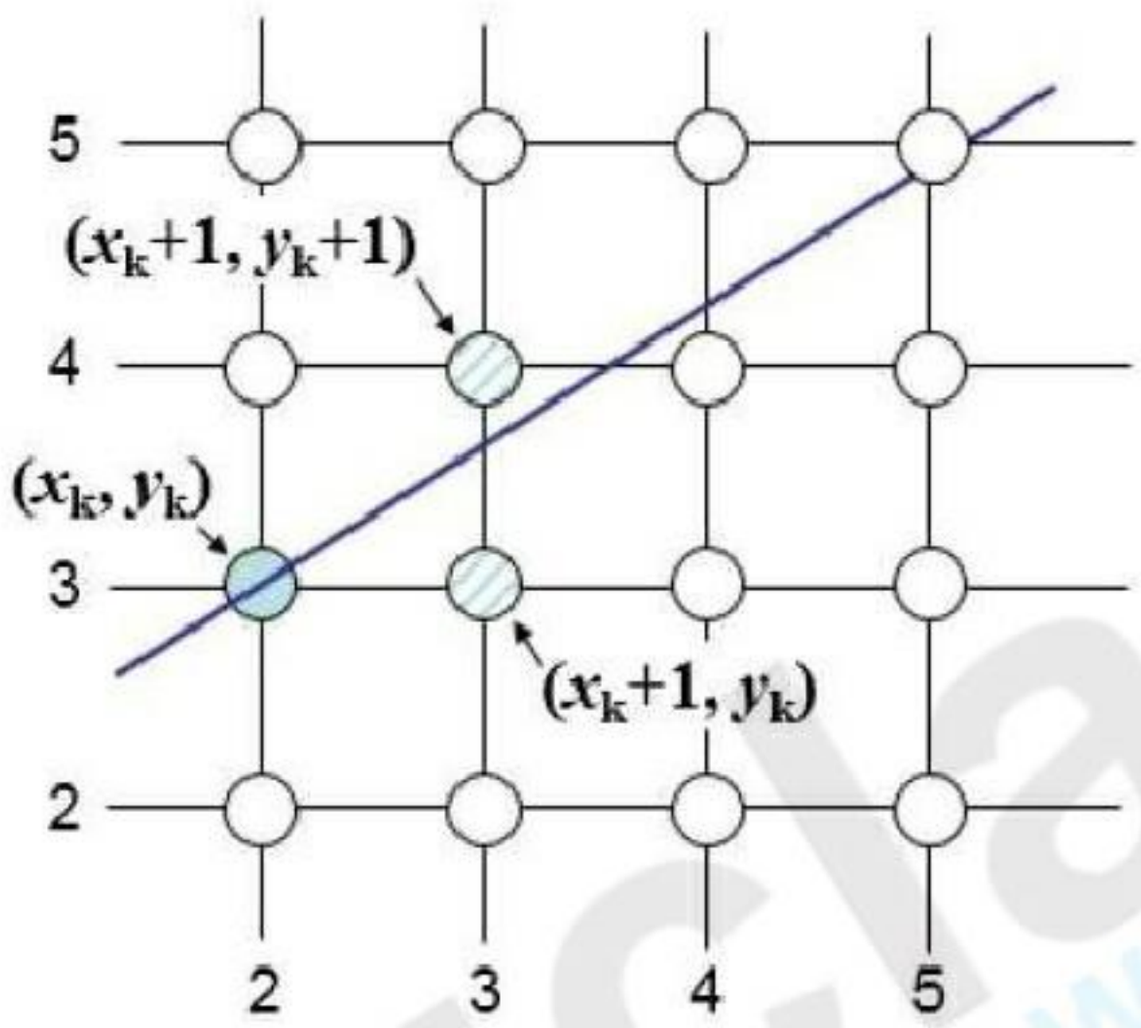
Co-ordinates after transformation are A(0,0), B(1,0), C(0.5,1) and D(-0.5,1)



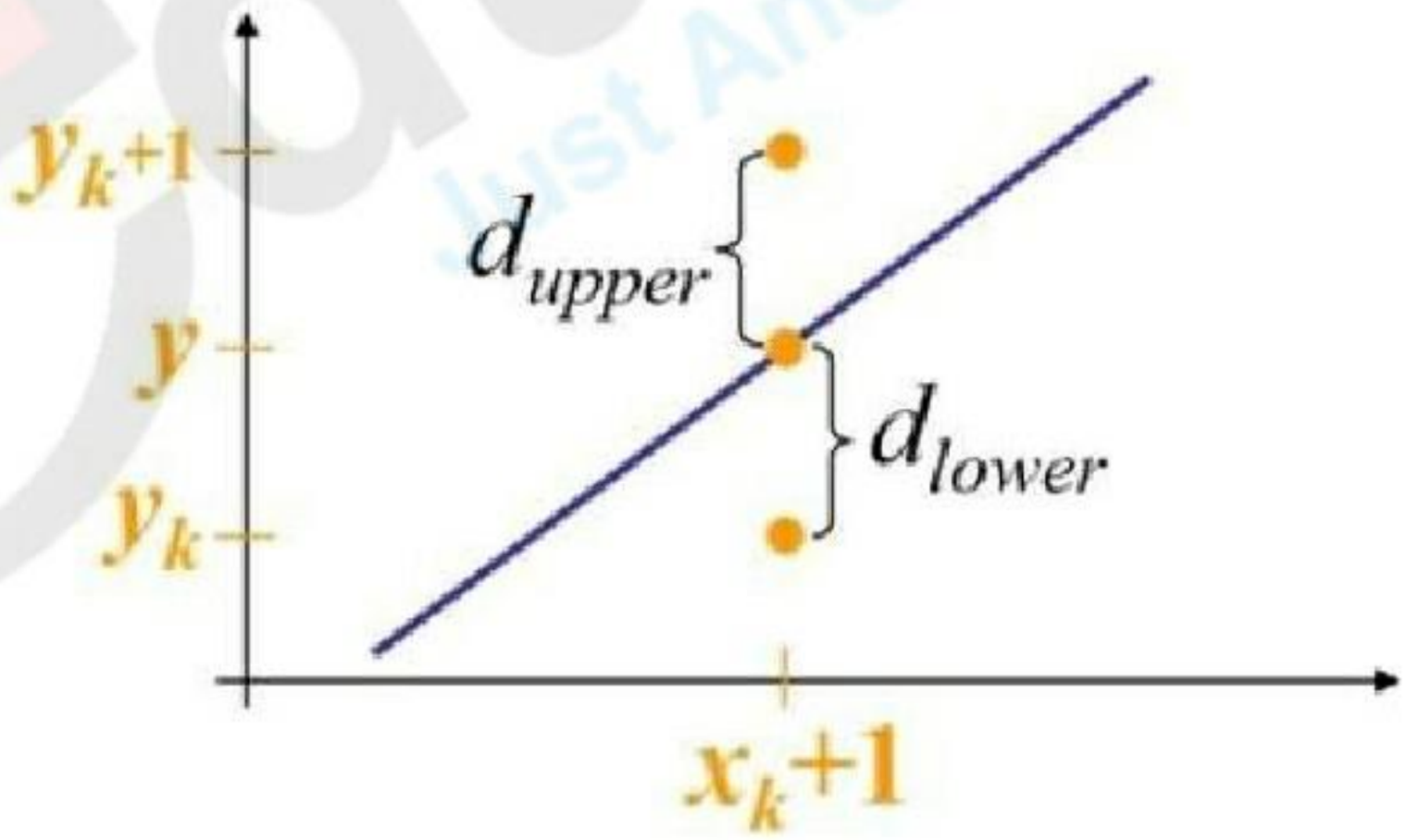
Q.2 (a) Derive and write the Generalized Bresenham's line drawing algorithm.

Ans. The Bresenham algorithm is another incremental scan conversion algorithm. The big advantage of this algorithm is that, it uses only integer calculations. Moving across the x axis in unit intervals and at each step choose between two different y coordinates.

For example, as shown in the following illustration, from position (2, 3) you need to choose between (3, 3) and (3, 4). You would like the point that is closer to the original line.



At sample position Xk+1,Xk+1, the vertical separations from the mathematical line are labelled as dupper and dlower



From the above illustration, the y coordinate on the mathematical line at xk+1xk+1 is −

Y = m(XkXk+1) + b

So, dupperdupper and dlowerdlower are given as follows −

$$dlower=y-yk$$

$$=m(Xk+1)+b-Yk=m(Xk+1)+b-Yk$$

and

$$dupper=(yk+1)-y$$

=Yk+1−m(Xk+1)−b=Yk+1−m(Xk+1)−b

You can use these to make a simple decision about which pixel is closer to the mathematical line. This simple decision is based on the difference between the two pixel positions.

$$dlower-dupper=2m(xk+1)-2yk+2b-1dlower-dupper=2m(xk+1)-2yk+2b-1$$

Let us substitute *m* with *dy/dx* where *dx* and *dy* are the differences between the end-points.

$$dx(dlower-dupper)=dx(2dydx(xk+1)-2yk+2b-1)dx(dlower-dupper)=dx(2dydx(xk+1)-2yk+2b-1)$$

$$=2dy.xk-2dx.yk+2dy+2dx(2b-1)=2dy.xk-2dx.yk+2dy+2dx(2b-1)$$

$$=2dy.xk-2dx.yk+C=2dy.xk-2dx.yk+C$$

So, a decision parameter PkPk for the *k*th step along a line is given by −

$$pk=dx(dlower-dupper)pk=dx(dlower-dupper)$$

$$=2dy.xk-2dx.yk+C=2dy.xk-2dx.yk+C$$

The sign of the decision parameter PkPk is the same as that of dlower−dupperdlower−dupper. If pkpk is negative, then choose the lower pixel, otherwise choose the upper pixel.

Remember, the coordinate changes occur along the x axis in unit steps, so you can do everything with integer calculations. At step k+1, the decision parameter is given as −

$$pk+1=2dy.xk+1-2dx.yk+1+Cpk+1=2dy.xk+1-2dx.yk+1+C$$

Subtracting pkpk from this we get −

$$pk+1-pk=2dy(xk+1-xk)-2dx(yk+1-yk)pk+1-pk=2dy(xk+1-xk)-2dx(yk+1-yk)$$

But, xk+1xk+1 is the same as (xk)+1(xk)+1. So −

$$pk+1=pk+2dy-2dx(yk+1-yk)pk+1=pk+2dy-2dx(yk+1-yk)$$

Where, Yk+1−YkYk+1−Yk is either 0 or 1 depending on the sign of PkPk.

The first decision parameter p0p0 is evaluated at (x0,y0)(x0,y0) is given as −

$$p0=2dy-dxp0=2dy-dx$$

Now, keeping in mind all the above points and calculations, here is the Bresenham algorithm for slope m < 1 −

Step 1 − Input the two end-points of line, storing the left end-point in (x0,y0)(x0,y0).

Step 2 − Plot the point (x0,y0)(x0,y0).

Step 3 − Calculate the constants dx, dy, 2dy, and (2dy – 2dx) and get the first value for the decision parameter as −

$$p0=2dy-dxp0=2dy-dx$$

Step 4 − At each XkXk along the line, starting at k = 0, perform the following test −

If pkpk < 0, the next point to plot is (xk+1,yk)(xk+1,yk) and

$$pk+1=pk+2dypk+1=pk+2dy$$

Otherwise,

$$(xk,yk+1)(xk,yk+1)$$

$$pk+1=pk+2dy-2dxpk+1=pk+2dy-2dx$$

Step 5 – Repeat step 4 (dx – 1) times.

For m > 1, find out whether you need to increment x while incrementing y each time.

After solving, the equation for decision parameter PkPk will be very similar, just the x and y in the equation gets interchanged.

Q.2 (b) Rasterise the ellipse having $r_x 8$ = and r_y = 6 in first quadrant.

Ans. Given input ellipse parameters rx= 8 and ry= 6, we illustrate the steps in themidpoint ellipse algorithm by determining raster positions along the ellipse path in the first quadrant. Initial values and increments for the decision parameter calculations are

$2r^2_y x = 0$ (with increment $2r^2_y = 72$)

$2r^2_x y = 2r^2_x r_y$ (withincrement $-2r^2_x = -128$)

For region 1: The initial point for the ellipse centered on the origin is (x_0, y_0) = **(0,6)**,and the initial decision parameter value is

$p1_0 = r^2_y - r^2_x r_y + (1/4) r^2_x = -332$

Successive decision parameter values and positions along the ellipse path are calculated using the midpoint method as

k	$p1_k$	(x_{k+1}, y_{k+1})	$2r^2_y x_{k+1}$	$2r^2_x y_{k+1}$
0	-332	(1,6)	72	768
1	-224	(2,6)	144	768
2	-44	(3,6)	216	768
3	208	(4,5)	288	640
4	-108	(5,5)	360	640
5	288	(6,4)	432	512
6	244	(7,3)	504	384

We now **move** out of region 1, since $2r^2_y x > 2r^2_x y$

For region 2, the initial point is (x_0, y_0) = **(7,3)** and the initial decision parameter is

$p2_0 = f(7+(1/2),2) = -151$

The remaining positions along the ellipse path in the first quadrant are then calculated as

k	$P2_k$	(x_{k+1}, y_{k+1})	$2r^2_y x_{k+1}$	$2r^2_x y_{k+1}$
0	-151	(8,2)	576	256
1	233	(8,1)	576	128
2	745	(8,0)	-	-

Q.3 (a) Explain and write the Z buffer algorithm along with its advantages and disadvantages.

Ans. This method is developed by Cutmull. It is an image-space approach. The basic idea is to test the Z-depth of each surface to determine the closest (visible) surface. In this method each surface is processed separately one pixel position at a time across the surface. The depth values for a pixel are compared and the closest (smallest z) surface determines the color to be displayed in the frame buffer. It is applied very efficiently on surfaces of polygon. Surfaces can be processed in any order. To override the closer polygons from the far ones, two buffers named frame buffer and depth buffer, are used.

Depth buffer is used to store depth values for (x, y) position, as surfaces are processed ($0 \leq$ depth ≤ 1). The frame buffer is used to store the intensity value of color value at each position (x, y). The z-coordinates are usually normalized to the range [0, 1]. The 0 value for z-coordinate indicates back clipping pane and 1 value for z-coordinates indicates front clipping pane.

Algorithm:

Step-1 – Set the buffer values –

Depthbuffer (x, y) = 0

Framebuffer (x, y) = background color

Step-2 – Process each polygon (One at a time)

For each projected (x, y) pixel position of a polygon, calculate depth z.

If Z > depthbuffer (x, y)

Compute surface color,

set depthbuffer (x, y) = z,

framebuffer (x, y) = surfacecolor (x, y)

Advantages

- It is easy to implement.
- It reduces the speed problem if implemented in hardware.
- Can be executed quickly, even with many polygons
- It processes one object at a time.

Disadvantages

- It requires large memory.

- It is time consuming process.
- Can't do transparent surfaces without additional code

Q.3 (b) Find out the final co-ordinates of a figure bounded by the co-ordinates (1, 1), (3, 4), (5, 7), (10, 3) when rotated about a point (8, 8) by 30^0 in clockwise direction and scaled by two units in x-direction and three units in y-direction.

Ans. Translate the figure to origin

Translation Matrix = $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -tx & -ty & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -8 & -8 & 1 \end{pmatrix}$

Rotate the figure by 30^0

Rotation Matrix = $\begin{pmatrix} \cos\Theta & \sin\Theta & 0 \\ -\sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos30 & \sin30 & 0 \\ -\sin30 & \cos30 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Scale the figure two units in x-direction and three units in y-direction

Scaling Matrix = $\begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Translate the figure back to (8,8)

Translation Matrix = $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -tx & -ty & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 8 & 8 & 1 \end{pmatrix}$

Therefore the composite transformation matrix,

$$T_c = \begin{pmatrix} 1 & 1 & 1 \\ 3 & 4 & 1 \\ 5 & 7 & 1 \\ 10 & 3 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -8 & -8 & 1 \end{pmatrix} \begin{pmatrix} \cos 30 & \sin 30 & 0 \\ -\sin 30 & \cos 30 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 8 & 8 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 4.77 & -14.24 & 1 \\ 0.24 & 4.46 & 1 \\ -0.77 & 5.32 & 1 \\ 6.88 & -0.055 & 1 \end{pmatrix}$$

Q.4 (a) What are display files? Explain with examples, how are polygons and characters represented in display files.

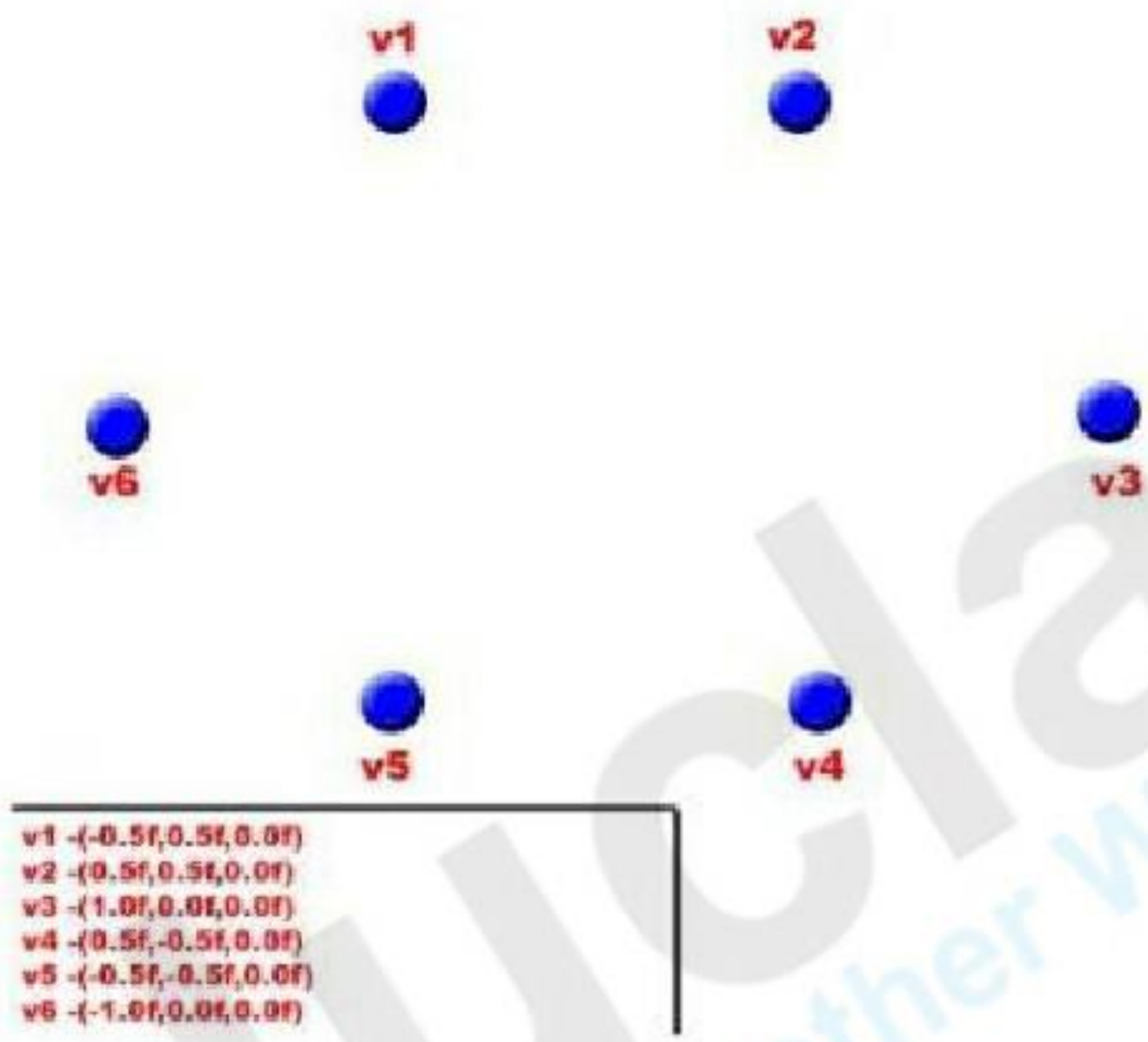
Ans. A display file is a series of graphics commands that define an output image. The image is created (rendered) by executing the commands to combine various primitives. This activity is most often performed by specialized display or processing hardware partly or completely independent of the system's CPU for the purpose of freeing the CPU from the overhead of maintaining the display, and may provide output features or speed beyond the CPU's capability.
A display list can represent both two- and three-dimensional scenes. Systems that make use of a display list to store the scene are called retained mode systems as opposed to immediate mode systems.

Using OpenGL to create display files. Open Graphics Library (OpenGL) is a crosslanguage, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering. The easiest way to do drawing in OpenGL is using the Immediate Mode. For this, you use the glBegin() function which takes as one parameter the “mode” or type of object you want to draw. Here is a list of the possible modes and what they mean:

GL_POINTS	Draws points on screen. Every vertex specified is a point.
GL_LINES	Draws lines on screen. Every two vertices specified compose a line.
GL_LINE_STRIP	Draws connected lines on screen. Every vertex specified after first two are connected.
GL_LINE_LOOP	Draws connected lines on screen. The last vertex specified is connected to first vertex.
GL_TRIANGLES	Draws triangles on screen. Every three vertices specified compose a triangle.
GL_TRIANGLE_STRIP	Draws connected triangles on screen. Every vertex specified after first three vertices creates a triangle.
GL_TRIANGLE_FAN	Draws connected triangles like GL_TRIANGLE_STRIP, except draws triangles in fan shape.
GL_QUADS	Draws quadrilaterals (4 – sided shapes) on screen. Every four vertices specified compose a quadrilateral.
GL_QUAD_STRIP	Draws connected quadrilaterals on screen. Every two vertices specified after first four compose a connected quadrilateral.
GL_POLYGON	Draws a polygon on screen. Polygon can be composed of as many sides as you want.

Drawing polygons:

Polygons consist of at least three vertices that, when connected, make up a shape. In this example we are going to be using the GL_POLYGON mode to draw a six – sided shape. The GL_POLYGON mode allows you to draw a shape with any number of sides as you want. Since GL_POLYGON is a singular word (meaning no "S" at the end of the word), you can only draw one polygon between a glBegin() and glEnd() function call. Also, the last vertex you specify is automatically connected to the first vertex specified. And of course, since a polygon is a closed shape, the shape will be filled with your specified color (which for now is blue). Here is the diagram for our six – sided shape:



Here is the code to draw this polygon:

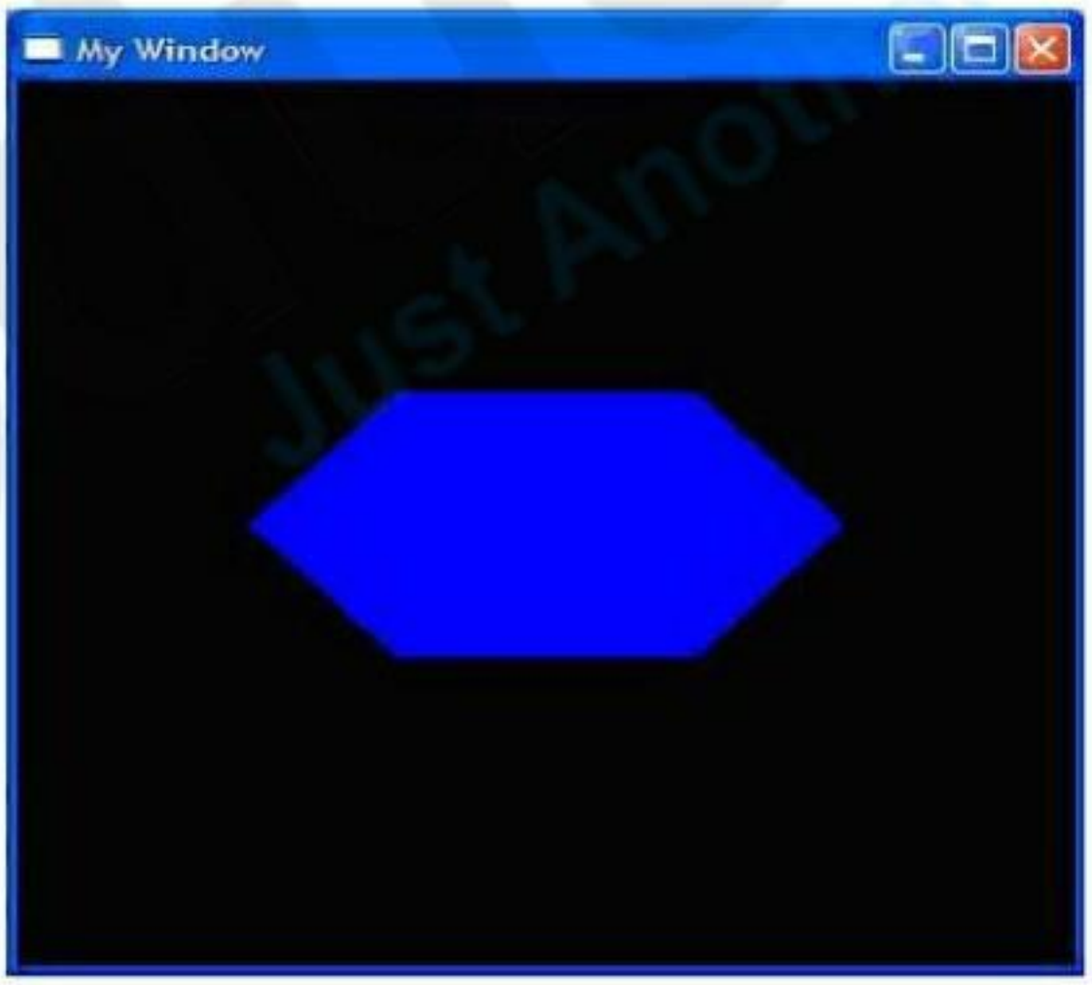
```
glBegin(GL_POLYGON); //begin drawing of polygon
        glVertex3f(-0.5f,0.5f,0.0f); //first vertex
        glVertex3f(0.5f,0.5f,0.0f); //second vertex
        glVertex3f(1.0f,0.0f,0.0f); //third vertex
        glVertex3f(0.5f,-0.5f,0.0f); //fourth vertex
        glVertex3f(-0.5f,-0.5f,0.0f); //fifth vertex
        glVertex3f(-1.0f,0.0f,0.0f); //sixth vertex
glEnd(); //end drawing of polygon
```

Following is the render() function followed by a sample output:

```
void render() {
//clear color and depth buffer
```

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity(); //load identity matrix
glTranslatef(0.0f,0.0f,-4.0f); //move forward 4 units
glColor3f(0.0f,0.0f,1.0f); //blue color

glBegin(GL_POLYGON); //begin drawing of polygon
        glVertex3f(-0.5f,0.5f,0.0f); //first vertex
        glVertex3f(0.5f,0.5f,0.0f); //second vertex
        glVertex3f(1.0f,0.0f,0.0f); //third vertex
        glVertex3f(0.5f,-0.5f,0.0f); //fourth vertex
        glVertex3f(-0.5f,-0.5f,0.0f); //fifth vertex
        glVertex3f(-1.0f,0.0f,0.0f); //sixth vertex
glEnd(); //end drawing of polygon
}
```



Q.5 (a) What are projections? How are they useful? Explain different types of projections.

Ans. Projection in computer graphics means the transformation of a three-dimensional (3D) area into a two-dimensional (2D) area. The plane in the area into which we transform (project) objects is called the 'Table'

1. Parallel Projection:

Parallel projection discards z-coordinate and parallel lines from each vertex on the object are extended until they intersect the view plane. In parallel projection, we specify a direction of projection instead of center of projection. In parallel projection, the distance from the center of projection to project plane is infinite. In this type of projection, we connect the projected vertices by line segments which correspond to connections on the original object. Parallel projections are less realistic, but they are good for exact measurements. In this type of projections, parallel lines remain parallel and angles are not preserved.

2. Perspective Projection:

In perspective projection, the distance from the center of projection to project plane is finite and the size of the object varies inversely with distance which looks more realistic.The distance and angles are not preserved and parallel lines do not remain parallel. Instead, they all converge at a single point called center of projection or projection reference point. There are 3 types of perspective projections:

1. One point perspective projection is simple to draw.
2. Two point perspective projection gives better impression of depth.
3. Three point perspective projection is most difficult to draw.

3. Orthographic Projection

In orthographic projection the direction of projection is normal to the projection of the plane. There are three types of orthographic projections –

a. Front Projection
b. Top Projection
c. Side Projection

4. Oblique Projection

There are two types of oblique projections – Cavalier and Cabinet. The Cavalier projection makes 45° angle with the projection plane. The projection of a line perpendicular to the view plane has the same length as the line itself in Cavalier projection. In a cavalier projection, the foreshortening factors for all three principal directions are equal.

The Cabinet projection makes 63.4° angle with the projection plane. In Cabinet projection, lines perpendicular to the viewing surface are projected at ½ their actual length.

Q.5 (b) Compare and contrast B-Spline and Bezier curves

Ans In numerical analysis in mathematics and in drawing computer graphics, many types of curves are taken help of. Bezier Curve and B-Spline Curve are two of the popular models for such analysis. There are many similarities in these two types of curves and experts call B-Spline curve to be a variation of Bezier curve. However, there are many differences too

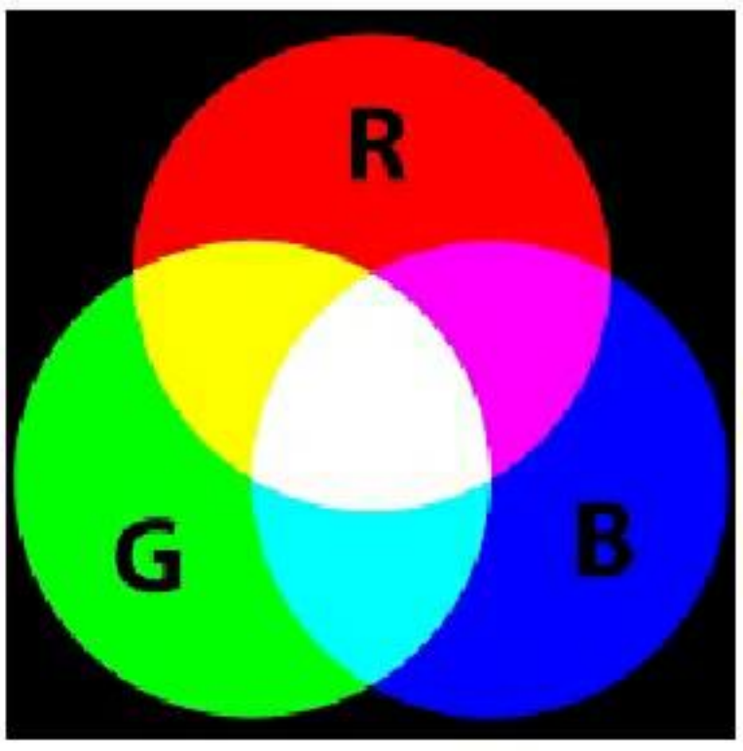
Bezier Curve	Bspline curve
Bezier curves are parametric curves used frequently in modelling smooth surfaces in computer graphics and many other related fields.	B-Spline curves are considered as a generalization of Bezier curves and as such share many similarities with it.
Bezier curve does not allow programmer a lot of control and flexibility	B-Spline offer more control and flexibility
Any Bezier curve of arbitrary degree can be converted in to a B-spline	Any B-spline can be converted in to one or more Bezier curves.
the Bezier curve automatically clamps its endpoints.	B-splines do not interpolate any of its control points,
Bezier curve is not as complicated as B-spline	B-spline curve requires more computation
It is not possible to use lower degree curves	It is possible to use lower degree curves and still maintain a large number of control points.

Q.6 (a) Discuss various colour models used in graphics system.

Ans A color model is an abstract mathematical model describing the way colors can be represented as tuples of numbers, typically as three or four values or color components. When this model is associated with a precise description of how the components are to be interpreted (viewing conditions, etc.), the resulting set of colors is called color space.

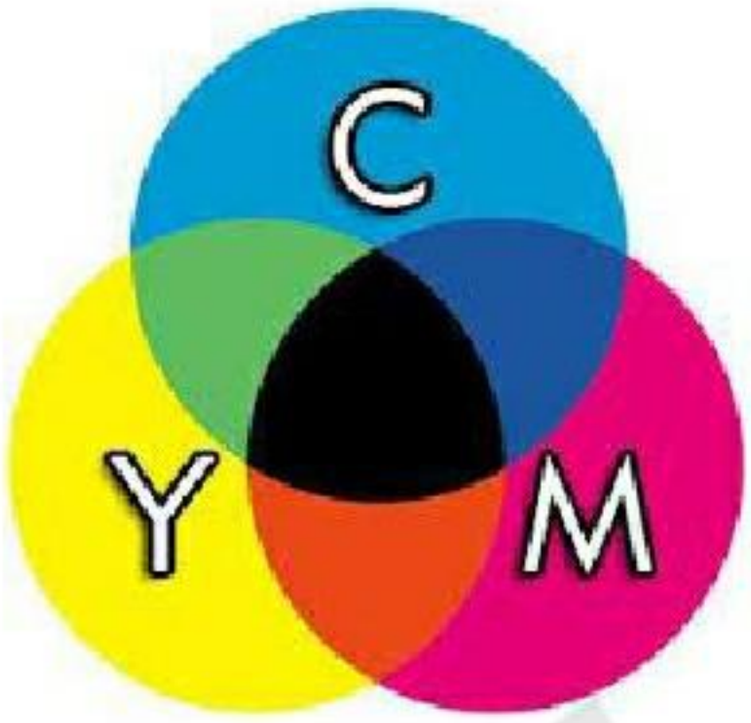
RGB:

The RGB color model is an additive color model. In this case red, green and blue light are added together in various combinations to reproduce a wide spectrum of colors. The primary purpose of the RGB color model is for the display of images in electronic systems, such as on television screens and computer monitors and it's also used in digital photography. Cathode ray tube, LCD, plasma and LED displays all utilize the RGB model.



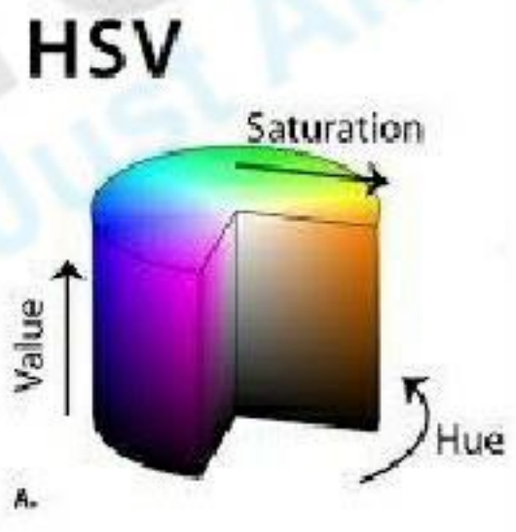
CYMK:

The CMYK color model (four-color process) is a subtractive color model. Primarily used in printing, CMYK works by partially or completely masking colors on a white background. The printed ink reduces the light that would otherwise be reflected. That's why this model is called subtractive because inks 'subtract' brightness from a white background from four colors: cyan, magenta, yellow and black.



HSV:

HSV, which stands for hue, saturation and value, depicts three-dimensional color. HSV seeks to depict relationships between colors, and improve upon the RGB color model. If you think about HSV as a wheel, the center axis goes from white at the top to black at the bottom, with other neutral colors in between. The angle from the axis depicts the hue, the distance from the axis depicts saturation, and the distance along the axis depicts value.



Q.6 (b) Find a normalization transformation from the window whose lower left corner is at (1,1) and upper right corner is at (3,5) onto the viewport with lower left corner at (0,0) and upper right corner at (½, ½).

Ans. $wx_{min} = 1$, $wy_{min} = 1$

$wx_{max} = 3$, $wy_{max} = 5$

$vx_{min} = 0$, $vy_{min} = 0$

$vx_{max} = ½$, $vy_{max} = ½$

$$Sx = vx_{max} - vx_{min} / wx_{max} - wx_{min}$$

$$= ½ - 0 / 3 - 1$$

$$= ¼$$

$$Sx = vy_{max} - vy_{min} / wy_{max} - wy_{min}$$

$$= ½ - 0 / 5 - 1$$

$$= 1/8$$

$$N = \begin{pmatrix} 1 & 0 & vxmin \\ 0 & 1 & vymin \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -wxmin \\ 0 & 1 & -wymin \\ 0 & 0 & 1 \end{pmatrix}$$

$$N = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1/4 & 0 & 0 \\ 0 & 1/8 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} ¼ & 0 & -¼ \\ 0 & ½ & -1/8 \\ 0 & 0 & 1 \end{pmatrix}$$

Q.7 Write short notes on:

a. Diffuse Illumination

Diffuse reflection is the reflection of light from a surface such that an incident ray is reflected at many angles rather than at just one angle as in the case of specular reflection. An illuminated *ideal* diffuse reflecting surface will have equal luminance from all directions which lie in the half-space adjacent to the surface. A surface built from a non-absorbing powder such as plaster, or from fibers such as paper, or from a polycrystalline material such as white marble, reflects light diffusely with great efficiency. Many common materials exhibit a mixture of specular and diffuse reflection.

The visibility of objects, excluding light-emitting ones, is primarily caused by diffuse reflection of light: it is diffusely-scattered light that forms the image of the object in the observer's eye.

Diffuse reflection from solids is generally not due to surface roughness. A flat surface is indeed required to give specular reflection, but it does not prevent diffuse reflection. A piece of highly polished white marble remains white; no amount of polishing will turn it into a mirror. Polishing produces some specular reflection, but the remaining light continues to be diffusely reflected.

b. Computer animation

Animation means giving life to any object in computer graphics. It has the power of injecting energy and emotions into the most seemingly inanimate objects. Computer-assisted animation and computer-generated animation are two categories of computer animation. It can be presented via film or video.

The basic idea behind animation is to play back the recorded images at the rates fast enough to fool the human eye into interpreting them as continuous motion. Animation can make a series of dead images come alive. Animation can be used in many areas like entertainment, computer aided-design, scientific visualization, training, education, e-commerce, and computer art.

Computer animation is essentially a digital successor to the stop motion techniques used in traditional animation with 3D models and frame-by-frame animation of 2D illustrations. Computer-generated animations are more controllable than other more physically based processes, constructing miniatures for effects shots or hiring extras for crowd scenes, and because it allows the creation of images that would not be feasible using any other technology. It can also allow a single graphic artist to produce such content without the use of actors, expensive set pieces, or props. To create the illusion of movement, an image is displayed on the computer monitor and repeatedly replaced by a new image that is similar to it, but advanced slightly in time (usually at a rate of 24 or 30 frames/second). This technique is identical to how the illusion of movement is achieved with television and motion pictures.

c. Inverse transformation

If a general transformation (which may be composite) is given by the 3 × 3 matrix M, then the inverse transformation, which maps the new image back to the original, untransformed one is given by M^{-1} . The matrix inverse is defined so that $MM^{-1} = I$ where I is the 3 × 3 identity matrix

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Inverses only exist for one to one transformations, for many operations they are not defined

The translation matrix has inverse:

$$T^{-1} = \begin{pmatrix} 1 & 0 & -tx \\ 0 & 1 & -ty \\ & & \end{pmatrix}$$

0 0 1

The scaling matrix has inverse:

$$S^{-1} = \begin{pmatrix} 1/Sx & 0 & 0 \\ 0 & 1/Sy & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

For a composite transformation matrix C = AB the inverse is less obvious.

We know $(AB)^{-1} = B^{-1}A^{-1}$ from linear algebra thus: $C^{-1} = B^{-1}A^{-1}$

The inverse transformations are applied in the opposite order.

d. Frame buffer

A frame buffer (frame buffer, or sometimes framestore) is a portion of RAM containing a bitmap that is used to refresh a video display from a memory buffer containing a complete frame of data.

The information in the buffer typically consists of color values for every pixel to be shown on the display. Color values are commonly stored in 1-bit binary (monochrome), 4-bit palettized, 8-bit palettized, 16-bit high color and 24-bit true color formats. An additional alpha channel is sometimes used to retain information about pixel transparency. The total amount of memory required for the frame buffer depends on the resolution of the output signal, and on the color depth and palette size.

Frame buffers used in personal and home computing often had sets of defined "modes" under which the frame buffer could operate. These modes would automatically reconfigure the hardware to output different resolutions, color depths, memory layouts and refresh rate timings.

Frame buffers have traditionally supported a wide variety of color modes. Due to the expense of memory, most early frame buffers used 1-bit (2-color), 2-bit (4-color), 4-bit (16-color) or 8-bit (256-color) color depths. The problem with such small color depths is that a full range of colors cannot be produced. The solution to this problem was to add a lookup table to the frame buffers. Each "color" stored in frame buffer memory would act as a color index; this scheme was sometimes called "indexed color".

e. DVST

Two electron guns are used in a Direct View Storage Tube. A primary electron gun is used where electrons are shot for image pattern generation. When high speed electrons hit the storage grid it displaces the electrons creating a positive charge. The purpose of storage grid is to store

image info in the form of charge distribution. The displaced electrons are attracted towards the collector. A flood gun is used for picture display. Now, the continuous flowing slow speed electrons from flood electron gun are attracted to the positively charged regions of the storage grid. They penetrate the storage grid and hit the phosphor coating in CRT generating the output. Here, the collector is used to control the flow of flood electrons.