Name: Pratham Kalwani

USN: 1BY20CS138

Subject: CAV, 18CS62

Sec: B

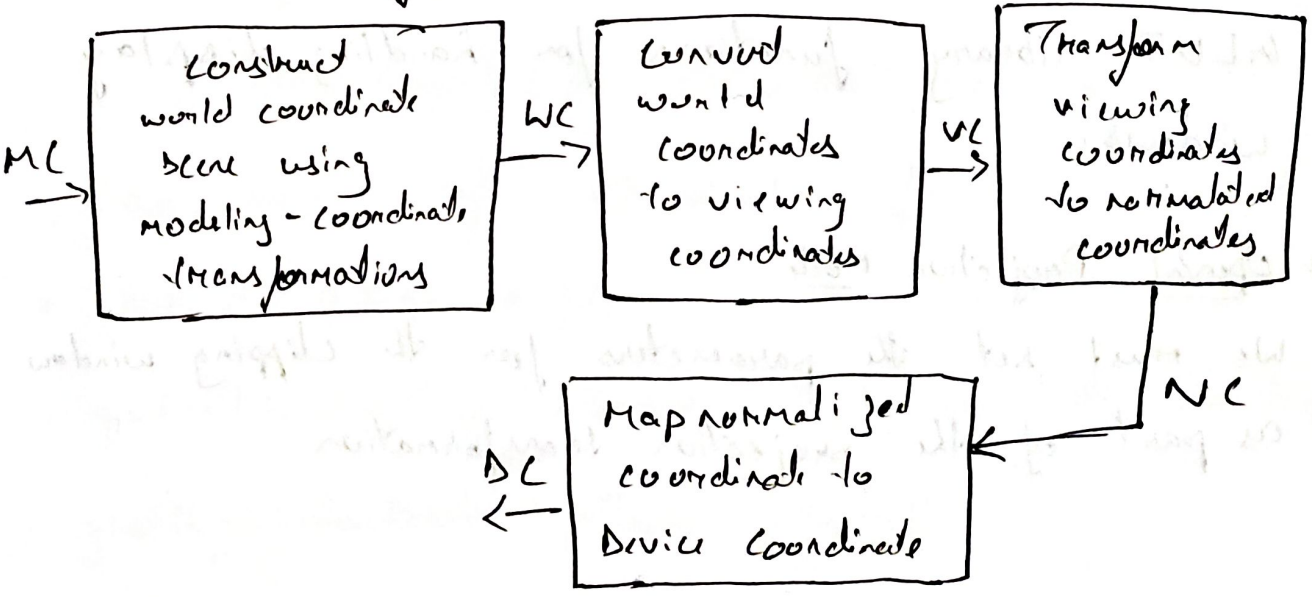
Sem: VI

Q1 Build a 2D viewing transformation pipline and also explain OpenGL 2D viewing functions

Ans Viewing pipeline means the mapping of a two-dimensional world-coordinate scene description to device coordinates is called a two dimensional viewing transformation.

This transformation is simply referred to as the window to viewport transformation or the windowing transformation.

We can describe the steps for 2D viewing pipeline as indicated in figure:



2D Pipelining

→ Once a world coordinate scene has been constructed, we could set up a separate two-dimensional viewing coordinate reference frame for specifying the clipping window.

→ To make the viewing process independent of the requirements of any output device, graphics system convert object descriptions to normalised coordinates and apply the clipping routines.

→ System used normalized coordinates in the range from 0 to 1, and others use a normalised range from -1 to 1.

OpenGL 2D viewing functions:

The GLU Library provides a function for specifying a two-dimensional clipping window and we have GLUT library functions for handling display windows.

(i) OpenGL Projection Mode

We must set the parameters for the clipping window as part of the projection transformation

Function:

glMatrixMode(GL_PROJECTION);

We can also set the initialisation as

glLoadIdentity();

ii) GLU clipping-window Function

To define 2D clipping window, we can use the GLU function:

gluOrtho2D(xwmin, xwmax, ywmin, ywmax);

The normalized coordinates in the range from -1 to 1 are used in the openGL clipping routines.

iii) openGL viewport Function

We specify the viewport parameters with the openGL function:

glViewport(xvmin, yvmin, vp width, vp height);

glGetIntegerv(GL_VIEWPORT, vpArray);

Other some useful 2D viewing function are

- glutInit(&argc, argv);
- glutInitWindowSize(xTopleft, yTopleft, dw width, dw height);
- glutInitWindowPosition(xTopleft, yTopleft);

- glut CreateWindow ("Title of Display Window");
- glut DisplayMode (mode);
- glut InitDisplayMode (GLUT_SINGLE | GLUT_RGB);
- glClearColor (red, green, blue, alpha);
- glClearIndex (index);
- glutDestroyWindow (windowID);
- glutSetWindow (windowID);
- glutFullScreen();
- glutDisplayFunc (pictureDescrip);
- glutPostRedisplay();
- glutMainLoop();

Q2 Build phong lighting model with equations.

Ans A phong lighting model is a local model that can be computed rapidly. It's an empirical model for calculating the specular reflection range, developed by phong.

Angle ϕ can be assigned values in the range 0° to 90°, so that $\cos\phi$ varies from 0 to 1.0.

It has 3 components:

(i) Ambient component

- The component approximates the indirect lighting by a constant

$I = I_a * K_a$ where,

I_a = ambient light intensity (color)

K_a = ambient reflection const (0 to 1)

(ii) Diffuse Component

It describes the diffuse reflection of rough surfaces

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

where,

I_p = intensity of point light src

K_d = diffuse reflection coefficients (0-1)

$\cos\theta$ = lambertian cosine law

(iii) Specular Component

It describes the specular reflection of smooth (shiny) surfaces

$I = I_p * K_s * \cos^{\wedge}\alpha$ α = shininess

where,

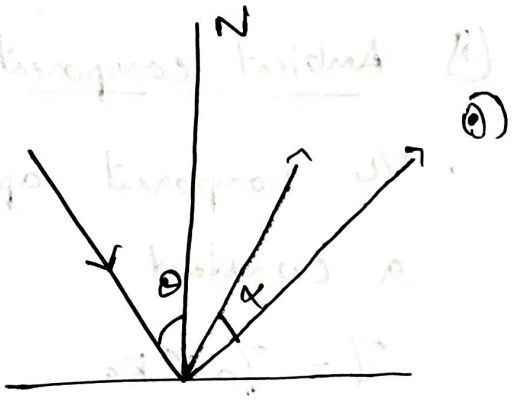
I_p = Intensity of point light source

K_s = specular reflection coeff (0-1)

Similarly from the figure

$\cos\theta = N \cdot L$

$\cos\alpha = RV$



So,

$I = I_p * K_d * N \cdot L$ (diffusion)

$I = I_p * K_s * (RV)^{\wedge}$ (specular)

Therefore, phong model eq^ is

$I = I_p * K_d * NL$ (diffusion)

$I = I_p * K_s * (RV)^{\wedge}$ (specular)

$I =$ Ambient + Diffusion + specular

$I = I_a K_a + I_p K_d \cos\theta + I_p K_s \cos^{\wedge}\alpha$

It also can be written in terms of Normal, reflection & view vector as,

$$\boxed{I = I_a K_a + I_p K_d NL + I_p K_s (RV)^{n}}$$

This gives the perfect phong model for a object under the illumination.

3) Apply homogenous coordinates for translation, rotation and scaling via matrix representation.

Ans A 2/3D point P is represented in homogenous coordinates by a L_1-dim.

Used: $P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$ or $P_1 = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

we use the homogenous coordinate to eliminate additions

when we use a point homogenous coordinate, we don't lose anything, it is easier to compute & everything is matrix multiplication.

(i) Translation

for 2D; $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} +x \\ +y \end{bmatrix} = P' \begin{bmatrix} x' \\ y' \end{bmatrix}$

for 3D:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

ii) Rotation :

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \rightarrow \text{Clockwise}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$ ⇒ anticlockwise

(iii) Scaling :-

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

In all transformation x, y, are the current points of x', y' are the resulting points after the transaction, rotation and scaling.

The 3D scaling can done as below in homogenous coordinate;

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Q4 Outline the differences between the raster scan display and random scan display.

Ans Difference between Random scan and Raster scan displays:

Random Scan display	Raster Scan display
1) The resolution is higher than raster scan display.	The resolution of raster scan is big lower than random scan display.
2) It is more expensive & less affordable.	It is affordable as compared to random scan display
3) In a random scan display, it is easy to proceed with modification.	In it, it is difficult to do any modification
4) In it, we don't prefer interlacing	In raster scan display, we prefer interlacing.
5) When it comes to image rendering we prefer mathematical function	When it comes to image rendering, we prefer pixels there. It is good for constructing lifelike scenes.
6) Only an area of the screen with a picture is displayed.	The entire screen is scanned & displayed.
7) It is difficult to fill the solid pattern in it.	The entire screen is scanned & displayed
8) Ex: pen plotter	Ex TV sets

Q5 Demonstrate OpenGL functions for displaying window management using GLUT.

Ans Since we are using the OpenGL utility toolkit, we need to initiate GLUT. Steps for displaying window management using GLUT;

→ We perform the GLUT initialization with the statement

- glutInit(&argc, argv);

→ Next, we can state that a display window is to be created on the screen with a given caption for the title bar.

- glutCreateWindow("An example for openGL Program");

→ Then the following function call passes the line segment description to the display window.

- glutDisplayFunc(lineSegment);

→ Before this, we need to justify the display window with these functions

- glutInitWindowPosition(xTopleft, yTopleft);
- glutWindowSize(dwWidth, dwHight);
- glutInitDisplayMode(GLUT_SINGLE | DOUBLE | GLUT_RGB);

→ But the display window is not yet on the screen. We need one more GLUT function to complete the window-processing operations. After execution of the following statement, all display window that we have created, including their graphic content are now activated.

```
glutMainLoop();
```

Ex
```
#include <GL/glut.h>
void display() {
    glClearColor(0.0, 0.0, 0.0, 1);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    glColor3f(1.0, 0.0, 0);
    glVertex2f(-0.8, -0.8);
    glColor3f(0.0, 1.0, 0);
    glVertex2f(0.8, -0.8);
    glColor3f(0.0, 0.0, 1);
    glVertex2f(0.0, 0.9);
    glEnd();
    glutSwapBuffers();
}
```

```
int main (int argc, char * * argv) {
    glut Init (& argc, argv);
    glut Init Display Mode ( GLUT_ SINGLE | GLUT_ RGB);
    glut Init Window Size ( 500, 500);
    glut Init Window position (100, 100);
    glut ( reate Window (" GL RGB Triangle")
    glut Displayfunc (display);
    glut Main Loop ();
    return 0;
}
```

Q6 Explain the OpenGL visibility Detection functions.

Ans The open GL has the different functions for the visibility detection in GLUT library. They are:

(i) Open GL polygon culling function

This function is used to remove Back face, front face or both faces of the object.

- gl Cull Func (mode);

  Parameter mode is

- gl Enable (GL_ CALL_ FACE);
- gl Disable (GL_ CALL_ FACE);

(ii) OpenGL Depth-Buffer Function

- glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

This initialization function will request for depth buffer

- glClear (GL_DEPTH_BUFFER_BIT);
- glEnable (GL_DEPTH_TEST);
- glDisable (GL_DEPTH_TEST);
- glDepthRange (nearNormDepth, farNormDepth);

(iii) OpenGL wireframe surface visibility Function

This function is used for wireframe display of the lighted, but display both visible and hidden edges;

- glPolygonMode (GL_FRONT_AND_BACK, GL_LINE);

(iv) OpenGL Depth Cueing Functions

- glFogi (GL_FOG_MODE, GL_LINEAR);
  - This function is used to vary the brightness of an object.
  - If applied linear depth function to object colors using dmin = 0.0 & dmax = 1.0 by default.
- glFogf (GL_FOG_START, minDepth);
- glFogf (GL_FOG_END, maxDepth);

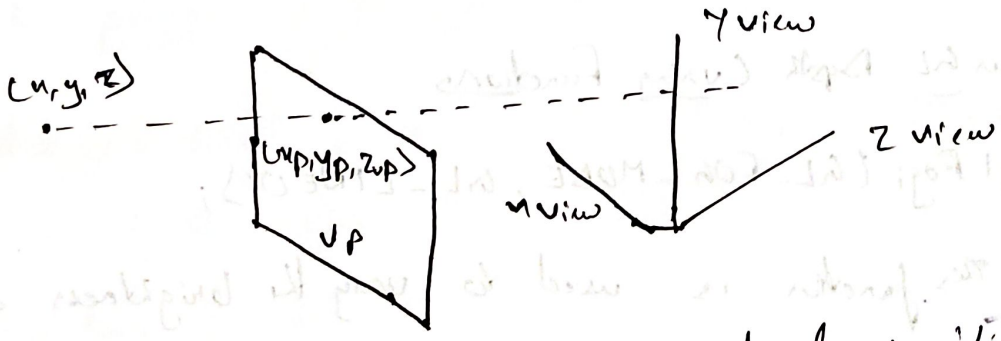
- glEnable (GL_FOG)

Hence by using above function, we can detect the visibility of the object.

Q7 What the special cases that we discussed with respect to perspective projection transformation coordinates.

Ans The projection line intersects the view plane at the coordinate position (x_p, y_p, z_{vp}) where z_{vp} is some selected position for the view plane on the z view axis.

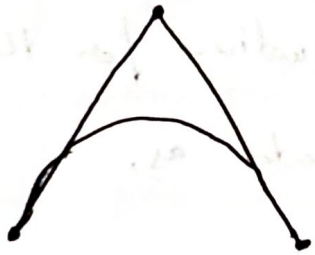
Figure below show the projection path of a spatial position (x, y, z) to a general projection reference point at $(x_{prp}, y_{prp}, z_{prp})$



we can write equations describing coordinate position along this perspective projection line in parametric form as,

Q8 Explain Bezier curve equation along with its properties.

Ans Bezier curve is drawn by considering control points. Approximate target is drawn by using control point.



3 - control points & bezier curve

Simplest form of Bezier curve is connection two endpoints

→ Bezier curve equation is also known as bezier spline curve. It is developed by French engineer Pierre Bezier.

Bezier curve can be fitted to any number of control points, although some graphic package limit to four control points

→ Bezier curve equations

$$Q(u) = \sum_{K=0}^{n} P_K \cdot BEZ_{K,n}(u)$$

where P_K = Position vector coordinate, $K = 0 \ldots n$

$BEZ_{K,n}(u)$ = Bezier Blending function

$$BEZ_{k,n}(u) = C(n,k) \cdot u^k \cdot (1-u)^{n-k}$$

where $C(n,k) = \dfrac{n!}{k!\,(n-k)!}$

→ A set of 3 parametric equations for the individual curve coordinates can be represented as,

$$x(u) = \sum_{k=0}^{n} x_k \, BEZ_{k,n}(u)$$

$$y(u) = \sum_{k=0}^{n} y_k \cdot BEZ_{k,n}(u)$$

$$z(u) = \sum_{k=0}^{n} z_k \cdot BEZ_{k,n}(u)$$

→ Bezier Curve Properties

- Depends on number of control points
- Curve will pass through end points, but not all control point.
- Polynomial equation of curve depends on no. of control points.

n → control points (+1)

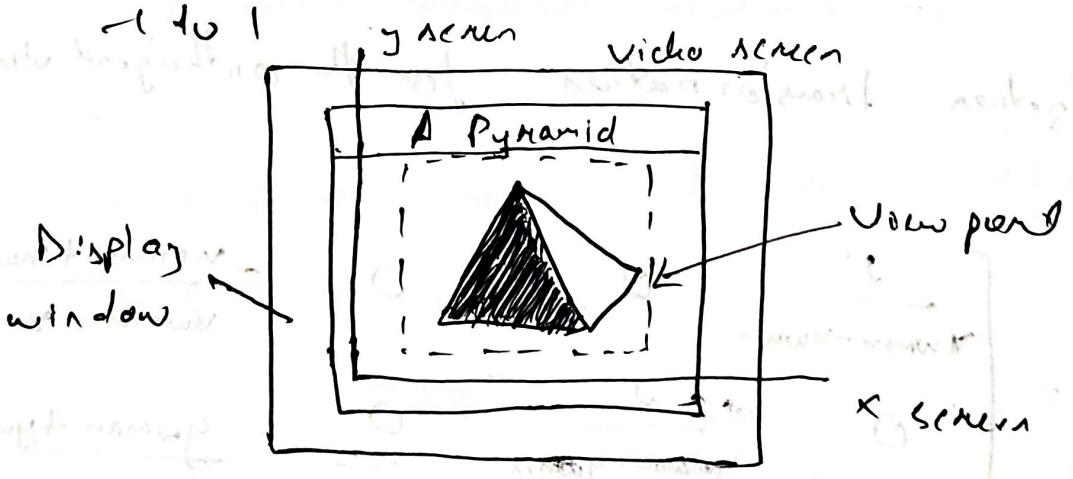
$n-1$ → Degree of polynomial equation (1)

Q9 Explain Normalization transformation for an orthogonal projection

Ans Once we have established the limits for the view volume, coordinate descriptions inside this rectangular parallel piped are the projection coordinates, and they can be mapped into a normalized view volume without any further projection processing.

Some graphics package use a unit cube for this normalized view volume with each of the x, y, and z coordinates normalized in the range from 0 to 1.

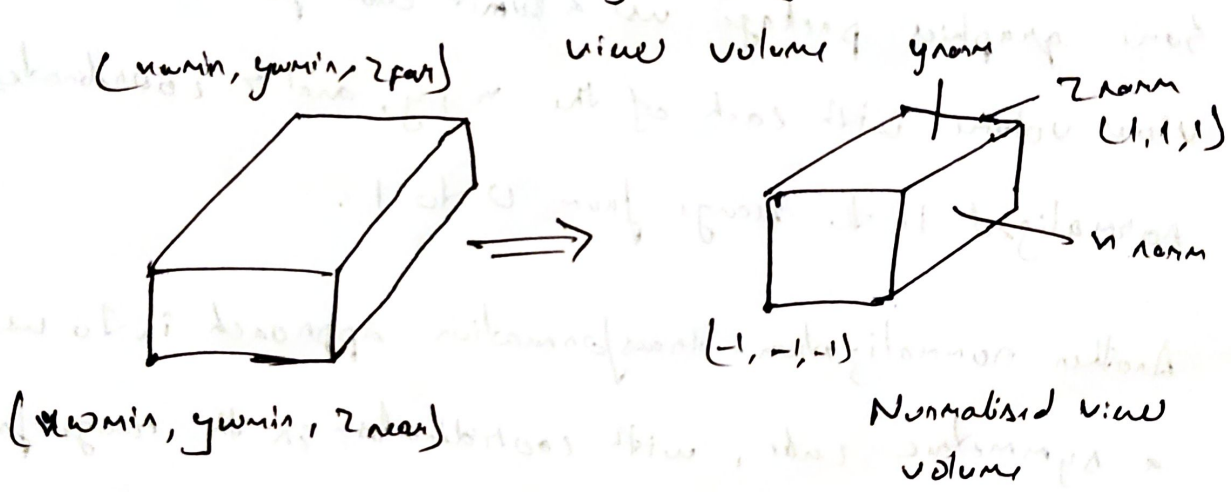
Another normalization-transformation approach is to use a symmetric cube, with coordinates in the range from -1 to 1



To illustrate the normalization, we assume that the orthogonal projection view volume is to be mapped into the symmetric normalization cube within a left-handed reference frame.

Also z-coordinate positions for the near and far planes are denoted as z_{near} and z_{far}, respectively.

Figure below illustrates this normalization transformation



The normalization transformation for the orthogonal view volume is

$$M_{ortho,norm} = \begin{bmatrix} \frac{2}{x_{wmax}-x_{wmin}} & 0 & 0 & -\frac{x_{wmax}+x_{wmin}}{x_{wmax}-x_{wmin}} \\ 0 & \frac{2}{y_{wmax}-y_{wmin}} & 0 & -\frac{y_{wmax}+y_{wmin}}{y_{wmax}-y_{wmin}} \\ 0 & 0 & \frac{2}{z_{wmax}-z_{far}} & \frac{z_{near}+z_{far}}{z_{near}-z_{far}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$x' = x - (x - x_{prp})u$$

$$y' = y - (y - y_{prp})u$$

$$z' = z - (z - z_{prp})u \qquad 0 \leq u \leq 1$$

On the view plane $z' = z_{vp}$, we can write u as,

$$u = \frac{z_{vp} - z}{z_{prp} - z}$$

By substituting this value of u into the equation for x' & y', we obtain the general perspective transformation equation

$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) + x_{prp}\left(\frac{z_{vp} - z}{z_{prp} - z}\right)$$

$$y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) + y_{prp}\left(\frac{z_{vp} - z}{z_{prp} - z}\right)$$

perspective projection equations: Special cases:

Case 1:

The projection reference point could be limited to position along the z view axis, then

$$x_{prp} = y_{prp} = 0$$

$$\therefore x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right), \quad y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right)$$

Case 2: Sometimes the projection reference point is fixed at coordinate origin and $(x_{prp}, y_{prp}, z_{prp}) = (0, 0, 0)$

then

$$x_p = x\left(\frac{z_{vp}}{z}\right) \qquad y_p = y\left(\frac{z_{vp}}{z}\right)$$

Case: 3 If the view plane is the uv plane and there are no restrictions on the placement of the projection reference point, then we have,

$$z_{vp} = 0$$

so $$x_p = x\left(\frac{z_{prp}}{z_{prp} - z}\right) - x_{prp}\left(\frac{z}{z_{prp} - z}\right)$$

$$y_p = y\left(\frac{z_{prp}}{z_{prp} - z}\right) - y_{prp}\left(\frac{z}{z_{prp} - z}\right)$$

Case: 4 With the uv plane as the view plane and projection reference point on the z view axis, the perspective equations are;

$$x_{prp} = y_{prp} = z_{prp} = 0$$

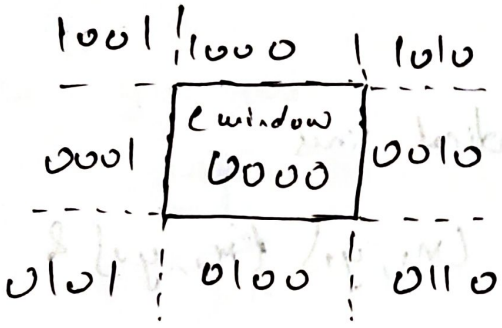
$$x_p = x\left(\frac{z_{prp}}{z_{prp} - z}\right)$$

$$y_p = y\left(\frac{z_{prp}}{z_{prp} - z}\right)$$

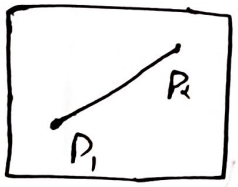
10) Explain Cohen-Sutherland line clipping Algorithm

Ans Cohen Sutherland algorithm works on Region code

Region code is 4 Bit code [ABRL; above, below, Right, Left]
[TBRL; Top, Bottom, Right, Left]



Case 1: If both endpoints region code is zero, completely inside & visible



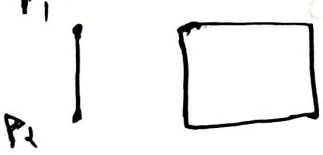
P_1 = 0000 (zero)

P_2 = 0000 (zero)

AND · 0000 (zero)

Case 2: If both end points region code is non zero, apply the logical AND and result is non zero

completely outside & invisible



P_1 = 0001

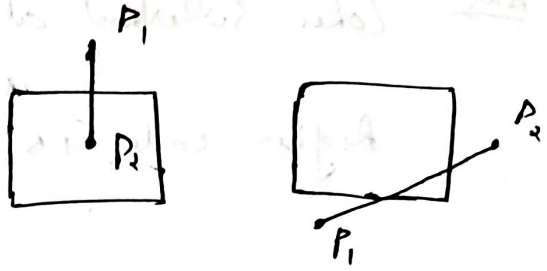
P_2 = 0001

AND 0001

Case 3: a) If one of P_1 & P_2 is zero } logical AND is zero

b) both non-zero

Find the intersection point



Finding Intersection point

1) Find y-value of vertical lines

Consider a line segment (x_1, y_1) (x_2, y_2) &

Intersection point (x, y)

→ Find slope of (x_1, y_1) & (x, y)

$$m = \frac{y - y_1}{x - x_1}$$

then $(y - y_1) = m(x - x_1)$

$$\therefore y = y_1 + m(x + x_1)$$

Here $x = x_{min}$ (left boundary) & $x = x_{max}$ (right boundary)

2) Find x value of horizontal ~~value~~ line

Find slope of (x_1, y_1) & (x, y)

$$m = \frac{y - y_1}{x - x_1}$$

$$m(x - x_1) = y - y_1$$

$$\therefore x = x_1 + \frac{y - y_1}{m}$$

Here $y = y_{max}$

(upper boundary)

$y = y_{min}$

(lower boundary)