Name: RAMCHANDRA PAUDEL
USN: 1BY20CS145
Subject: CGV, 18CS62
Sec: B
Sem: VI

CG Assignment

1. Build a 2D viewing transformation pipeline 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;

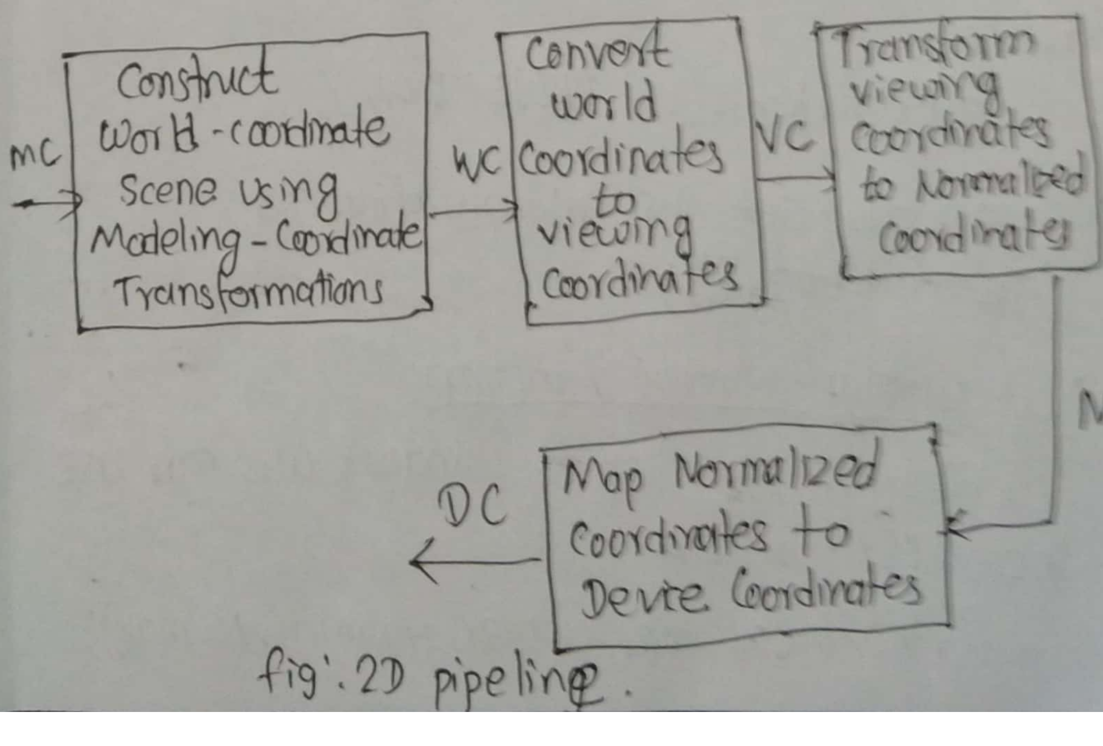


fig: 2D pipeline.

- once a world coordinate scene has been constructed, we could set up a seperate 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 normalized coordinates and apply the clipping routines.
- Systems used normalized coordinates in the range from 0 to 1, and others use a normalized 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 initialization 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) <u>OpenGL viewport Function</u>

we specify the viewport parameters with the OpenGL function;

glViewport(xvmin, yvmin, vpWidth, vpHeight);

glGetIntegerv(GL_VIEWPORT, vpArray);

Others some useful 2D viewing functions are;

⇒ • glutInit (&argc, argv);
- glutInitWindowPosition(xTopLeft, yTopLeft);
- glutInitWindowSize (dwWidth, dwHeight);
- glutCreateWindow ("Title of Display Window");
- glutInitDisplayMode (mode);
- glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
- glClearColor (red, green, blue, alppha);
- glClearIndex (index);
- glutDestroyWindow (windowID);
- glutSetWindow (windowId).
- glutFullScreen ();
- glutDisplayFunc (pictureDescrip);
- glutPostRedisplay( );
- glutMainLoop( );

Q. 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.

$\boxed{I = I_a * k_a}$ where,

I_a = ambient light intensity (color)

k_a = ambient reflection const (0~1)

(ii) Diffuse Component

It describes the diffuse reflection of rough surfaces.

$\boxed{I = I_p * K_d * \cos\theta}$

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.

$\boxed{I = I_p * k_s * \cos^n \alpha}$, n = shininess

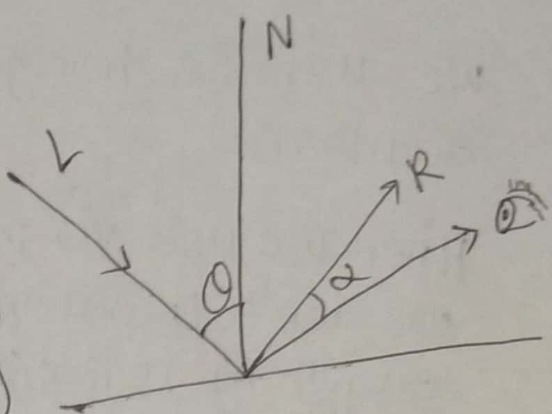
where, I_p = Intensity of the point light source

k_s = specular reflection coeff. (0-1)

Similarly, from the figure,

$\cos\theta = N \cdot L$

$\cos\alpha = RV$



So,

$I = Ip * Kd * N \cdot L$ (Diffusion)

$I = Ip * Ks * (RV)^n$ (Speculor)

Therefore, phong model eq^n is,

I = Ambient + Diffusion + Speculor

$\Rightarrow I = Iaka + Ipkd \cos\theta + Ipks \cos^n\alpha$

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

$$\boxed{I = Iaka + IpK_dNL + IpK_s(RV)^n}$$

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

---

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

<u>Ans:</u> A 2/3D point p is represented in homogeneous coordinates by a 4-dim. vect:

$$p = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{or, } p = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

We use the homogenous coordinate to eliminate additions.

When we use a point Homogenous Coordinate, we don't lose anything, it is easier to compose & everything is matrix multiplication.

(i) Translation;

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

For 3D;
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

(ii) Rotation:-

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

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

(iii) Scaling:-

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

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

The 3D scalling can done as below in homogeneous coord inate;

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

#

---

10)

4. Outline the differences between raster scan displays and random scan displays.

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 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 modifications.	In it, it is difficult to do any modifications
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 functions.	When it comes to image rendering, we prefer pixels here. 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.	It is easy to fill the solid pattern
8.	Ex: Pen Plotter	Ex: TV sets
9.	Designed for line drawing & can't display realistic shaded scenes	Designed for realistic display of scenes contain shadow & color patterns.

5. Demonstrate OpenGL functions for displaying window management using GLUT.

Ans: Since we are using the OpenGL utility toolkit, we need to initialize GLUT. ~~the~~ 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 justy the display window with these functions;

- glutInitWindowPosition(xTopLeft, yTopLeft);
- glutInitWindowSize(~~dw~~Width, dwHeight);
- glutInitDisplayMode(GLUT_SINGLE/DOUBLE|GLUT_RGB);

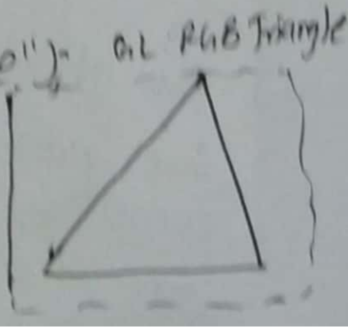
→ But the display window is not yet on the screen we need ~~one~~ more ~~and~~ GLUT function to complete the window-processing operations. After execution of the following statement, all display window that we have created, including their graphic content, are now activated.

```
glutMainLoop();
```

Ex:-

```
#include<GL/glut.h>

void display(){
    glClearColor(0,0,0,1);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    glColor3f(1,0,0);
    glVertex2f(-0.8,-0.8);
    glColor3f(0,1,0);
    glVertex2f(0.8,-0.8);
    glColor3f(0,0,1);
    glVertex2f(0,0.9);
    glEnd();
    glutSwapBuffers();
}
int main(int argc,char**argv){
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("GL RGB Triangle");
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```



6. Explain the OpenGL visibility Detection functions.

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

(i) OpenGL polygon culling functions

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

```
• glCullFace(mode);
```

Parameter mode is;

```
• glEnable(GL_CULL_FACE);   // active culling
• glDisable(GL_CULL_FACE);  // deactive culling
```

ii) OpenGL Depth-Buffer Functions:

```
• glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
```

- This initialization function will request for depth buffer and refresh buffer.

```
• glClear(GL_DEPTH_BUFFER_BIT);
• glEnable(GL_DEPTH_TEST);   } for enable & disable
• glDisable(GL_DEPTH_TEST);  } the depth buffer
```

```
• glDepthRange(nearNormDepth, farNormDepth);
```

(iii) OpenGL wireframe surface visibility Function:

This function is used for wire frame display of the light, But display both visible and hidden edges;

```
• glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
```

iv) OpenGL Depth Cueing Functions:

$\boxed{\text{glFogi(GL\_FOG\_MODE, GL\_LINEOF);}}$

- This function is used to vary the brightness of an object.
- It 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);
```

$\boxed{\text{glEnable(GL\_FOG)}}$ → Active fog function.

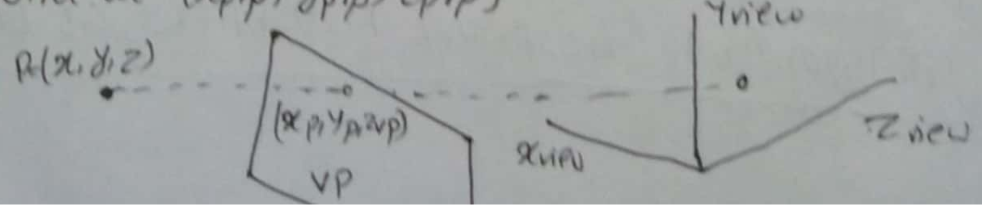
Here, By using above functions, we can detect the visibility of the objects.

---

7). Write the special cases that we discussed with respect to Prespective 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 prespective-projection line in parametric form as,

$$x' = x - (x - x_{prp})u$$
$$y' = y - (y - y_{prp})u \qquad 0 \le u \le 1$$
$$z' = z - (z - z_{prp})u$$

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 equations for x' & y', we obtain the general prespective transformation equations,

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

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

⇒ Prespective projection Equations: Special cases:

Case I:

The projection reference point could be limited to position along the z_{view} axis, then

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

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

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

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

<u>Case 3</u>: 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, $$x_p = x\left(\frac{z_{prp}}{z_{prp}-z}\right) - x_{prp}\left(\frac{z}{z_{prp}-z}\right)$$

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

<u>Case 4</u>: With the uv plane as the view plane and ~~the~~ the projection reference point on the z_{view} axis, the prespective equations are;

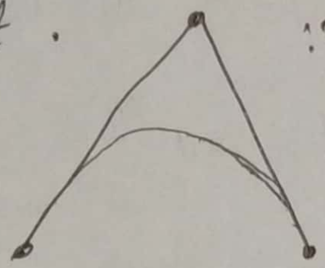
$$x_{prp} = y_{prp} = z_{vp} = 0.$$

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

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

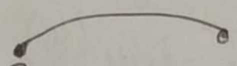
8. Explain Bézier curve equation along with its properties.

<u>Ans</u>: Bezier curve is drawn by considering control points. Approximate target is drawn by using control point.



:3-control points of bezier curve

Simplest Form of Bezier curve is connection two endpoints.



→ Bezier curve eq^n is also known as Bezier spline curve. It is developed by french engineer Pierre Bézier.

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

→ Bezier curve equation;

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

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

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

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

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

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

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

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

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

⇒ **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.

$n \rightarrow$ control points (4)

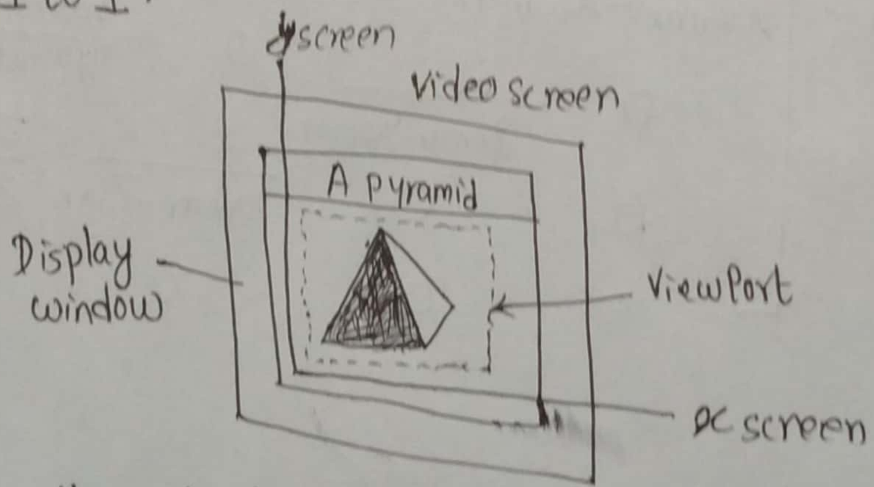
$n-1 \rightarrow$ Degree of polynomial equation (3).

9. Explain normalization transformation for an orthogonal projection.

Ans: Once we have established the limits for the view volume, coordinate descriptions inside this rectangular parallelpiped 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.

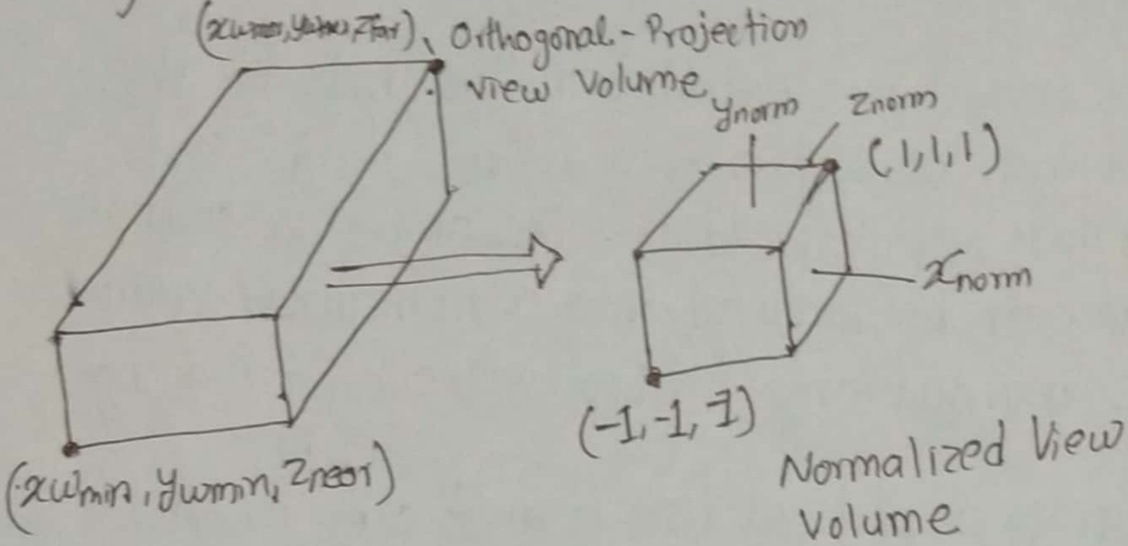
Another normalization-transformation approach is to use a symmetric cube, with coordinates in the range from -1 to 1.



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 znear and zfar, respectively.

Figure below illustrates this normalization transformation.



The normalization transformation for the orthogonal view volume is;

$$M_{ortho,norm} = \begin{bmatrix} \frac{2}{xw_{max} - xw_{min}} & 0 & 0 & -\frac{xw_{max} + xw_{min}}{xw_{max} - xw_{min}} \\ 0 & \frac{2}{yw_{max} - yw_{min}} & 0 & -\frac{yw_{max} + yw_{min}}{yw_{max} - yw_{min}} \\ 0 & 0 & \frac{2}{z_{near} - z_{far}} & \frac{z_{near} + z_{far}}{z_{near} - z_{far}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

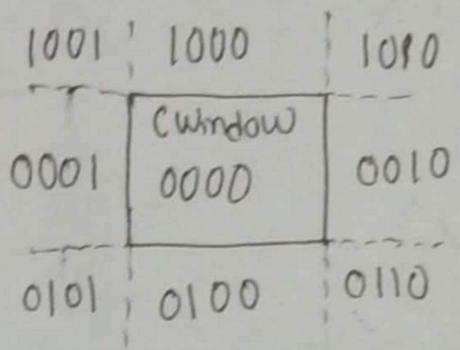
#.

10. Explain Cohen-Sutherland line Clipping Algorithm.

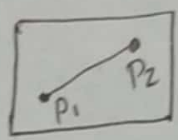
Ans: Cohen sutherland algorithm works on Region Code.

Region code is 4 Bit code (ABRL ; above, Below, Right, left)

(TBRL; ~~above~~, Top, Bottom, Right, Left)



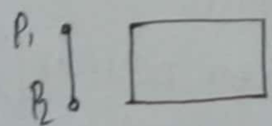
Case1: If both endpoint Region code is zero, completely inside & visible.



$P_1 = 0000$ (zero)
$P_2 = 0000$ (zero)

AND 0000 (zero)

Case2: If both end point Region code is nonzero, apply the logical AND and Result is nonzero. completely outside & invisible.

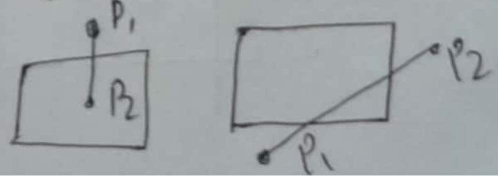


$P_1 = 0001$
$P_2 = 0001$

AND 0001

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

Find the Intersection Point.



Finding Intersection points:

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

$$\therefore \boxed{y = y_1 + m(x - x_1)}$$

here, $x = xw_{min}$ (left border) & $x = xw_{max}$ (right border).

2) Find x value of horizontal line.

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

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

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

$$\boxed{\therefore x = x_1 + \left(\frac{y - y_1}{m}\right)}$$

here, $y = y_{max}$ (upper boundry)

$y = y_{min}$ (lower boundry)

ex: