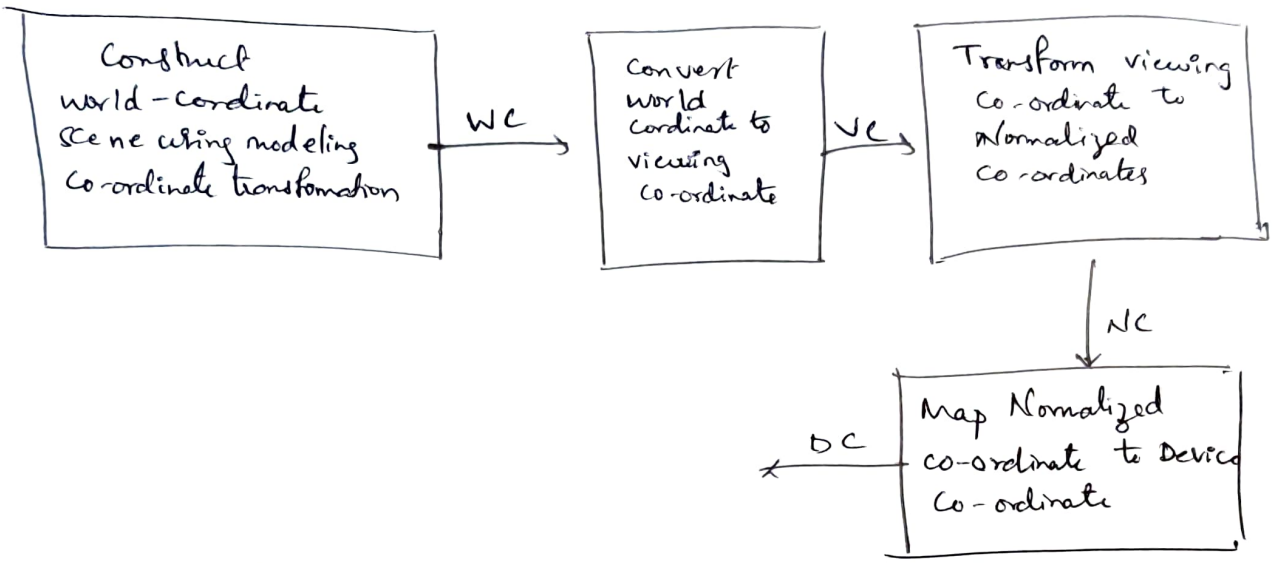
CGV Assignment

Name: K.V. Diwakar Reddy
USN: 1BY20CS079

① Build a 2D viewing transformation pipeline and also explain OpenGL 2D viewing functions.

Ans →



2D-viewing functions :-

We can use these two dimensional routines, along with the OpenGL viewport function, all the viewing operations we need

OpenGL projection Mode :-

Before we select a clipping window and a viewport in openGL, we need to establish the appropriate mode for constructing the matrix to transform from world coordinates to screen co-ordinates

```
glMatrixMode (GL_PROJECTION);
```

This designates the projection matrix as the current matrix, which is orginally set to identity matrix.

→ GLU Clipping-window Function :-

To define a 2-D Clipping window, we can use the OpenGL utility function.

```
gluOrtho2D ( xwmin , xwmax , ywmin , ywmax );
```

OpenGL Viewport Function

```
glViewport (xvmin , yvmin , Vpwidth , Vpheight );
```

Create a GLUT Display window

```
glutInit ( &argc, argv );
```

we have three functions in GLUT for defination a display window and Choosing its dimention and position

```
glutInitWindowPosition ( xTopLEFT , yTopLEFT );
glutInitWindowSize ( dwidth , dheight );
glutCreateWindow ("Title of display window");
```

→ Setting the GLUT Display window Mode Color :-

Various display window parameters are selected with the GLUT function.

```
glutInitWindow
glutInitDisplayMode (mode);
glutInitDisplayMode ( GLUT_SINGLE | GLUT_RGB );
glClearColor ( red, blue, green, alpha );
glClearIndex ( index );
```

→ Glut display window Identifier :

Window ID = glutCreateWindow("A display window");

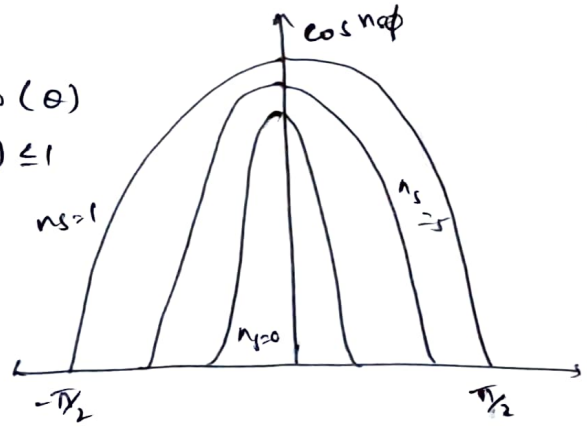
→ Current GLUT Display window :

glutSetWindow (window ID);

② Build Phong Lighting Model with equation?

Ans → Phong reflection is an empirical model of local illumination It describes the way a surface reflects light as a combination of the diffuse reflection of rough surfaces with the specular reflection of shinny surfaces have large highlights, while dull surfaces have large highlights that fall off more gradually

If, Specularity = w(θ)
If $\cos^{n_s}\phi$ $0 \le w(\theta) \le 1$
is called
Specular reflection
Co-efficient



Phong model sets the intensity of specular reflection to $\cos^{n_s}\phi$

If light direction L and viewing direction V are on the same side of the normal N, or if L is behind the surface, specular effects effec do not exists

for most opeque material Specular-reflection Co-efficient is nearly Const k_s

$$I_{Specular} = \begin{cases} k_s I_e (V \cdot R)^{n_s}, & V \cdot R > 0 \text{ and } N \cdot L > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$R = (2N \cdot L)N - L$$

The normal N may vary at each point to avoid as computation angle ϕ is replaced by an angle α defined by a halfway vector H between L and V

$$\text{Efficient} \Rightarrow H = \frac{L+V}{|L+V|}$$

If the light source and viewer are relatively far from the object, α is constant.

3) Apply homogeneous co-ordinates for translation, rotation and scaling via matrix representation.

Ans → The three basic 2-D transformations are translation, rotation and scaling

$$\boxed{P' = M_1 * P + M_2}$$

$P' \times P$ represents column vectors

Matrix $M_1 \rightarrow$ 2 x 2 Array containing multiplicative factors
$M_2 \rightarrow$ 2 elements column matrix containing translation term $\begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$ translation, M1 is identity matrix $P' = P + T$ where $T = M_2$ rotation and scaling, M_2 contains translational terms associated with pivot point or scaling

HOMOGENOUS CO-ORDINATES :- A standard technique to expand the matrix representation for a 2D co-ordinate (x,y) position to a 3 element representation for a 2D coordinate (x_h, y_h, h)
→ called Homogeneous Coordinates
h is homogeneous parameter h (non-zero value)
i.e (x,y) is converted into new coordinate values

as (x_h, y_h, h) $x = \frac{x_h}{h}$, $\frac{y_h}{h}$ $x_h = x \cdot h$ $y_h = y \cdot h$

Translation :- $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

This translation operation can be written as $P' = T(t_x, t_y) \cdot P$

3 x 3 translation matrix

Rotation :-

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow P' = R(\theta) \cdot P$$

Scaling Matrix :-

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow P' = S(S_x, S_y) \cdot P$$

(4) Outline the difference between raster scan displays and Random scan displays.

Ans →

Random Scan display	Raster Scan display
* In vector scan display the beam is moved between the end points of the graphics primitive.	* In raster scan display, the beam is moved all over the screen over scanline at a time, from top bottom and then back to top.

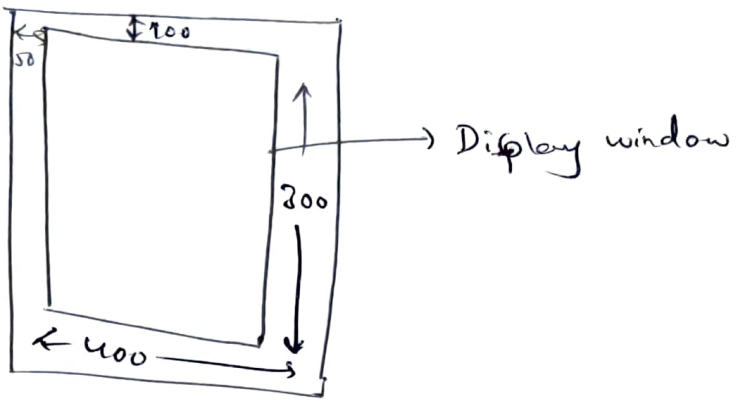
Random Scan display

* Vector display flickers when the number of primitives in the buffer becomes too large
* Scan conversion is not required
* Scan conversion hardware is not required
* vector display draws a continious and smooth lines
* Cost is more
* vector display only draw lines and characters

Raster Scan display

* In raster display, the refresh process is independent of the complexity of the image.
* because each primitive must be scan-converted, real-time dynamics is for more computational and required seperate scan conversion hardware
* Graphics primitives are specified in terms of their end points and must be scan converted into their corresponding pixel in the frame buffer
* Raster display can display mathematically smooth lines, polygons and boundaries of curved primitives only by approximating them with pixels on the raster grid
* Cost is low
* Raster display has ability to display areas filled with solid colours or patterns

5) Demonstrate OpenGL functions 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. this is accomplished with the function.

→ glutCreateWindow ("An Example OpenGL Program");

where the single argument for this function can be any character string.

* The following function call the line segment description to the display window.

→ glutDisplayFunc (lineSegment);

* glutMainLoop();

This Function must be the last one in our program. It display the initial graphics and puts the program

into an infinite loop that checks for input from devices such as mouse or keyboard.

* glutInitWindowPosition (50,100);

The following statement specifies that the upper-left corner of the display window should be placed 50 pixel to the right of the left edge of the screen and 100 pixels down from the top edge of the screen.

* glutInitWindowSize (400,300);

The fol glutInitWindowSize function is used to set the initial pixel width and height of the display window.

* glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

The command specifies that a single refresh buffer is to be used for the display window and that we convert to use the color mode which uses red, green, and blue (RGB) components to select color values.

6. Explain OpenGL Visibility Detection Functions?

Ans a] OpenGL polygon- Culling Functions

Back-face removal is accomplished with the functions

glEnable (GL_CULL_FACE);
glCullFace (mode);

* where parameter mode is assigned the value GL_BLACK, GL_FRONT_AND_BACK

* By default, parameter mode in glCullFace function has the value GL_BLACK
* The Culling routine is turned off with glDisable (GL_CULL_FACE);

b] OpenGL Depth-Buffer Functions :-

To use the OpenGL depth-buffer visibility detection function, we first need to modify the GLutility Toolkit. (GLUT) initialization function for the display mode to include a request for the depth, as well as for the refresh buffers

glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

→ Depth Buffer values can be initialized with

glClear (GL_DEPTH_BUFFER_BIT);

* By default it is set to 1.0.

→ These routines are activated with the following functions.

glEnable (GL_DEPTH_TEST);

And we deactivate these depth-buffer routines with

glDisable (GL_DEPTH_TEST);

→ we can also apply depth-buffer so that it is read only state or in a read-write state.

glDepthMask (writeStatus);

c] OpenGL Wire-Frame Surface Visibility Methods

→ A wire-frame displays of a standard graphics object can be obtained in OpenGL by requesting that only its edges one to be generated.

glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

But this displays both visible and hidden edges

d] OpenGL_DEPTH_Curing Function

→ we can vary the brightness of an object as a function of its display distance from the viewing position with

```
glEnable(GL_FOG);
glFogi(GL_FOG_Mode, GL_LINEAR);
```

→ This applies the linear depth function to object colors using dmin = 0.0 and dmax = 1.0 we can set different values for dmin and dmax with the following.

```
glfogf(GL_FOG_START, minDepth);
glfogf(GL_FOG_END, maxDepth);
```

7) write the special cases that we discussed with respect to perspective projection transformation Co-ordinates

Ans →

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

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

Special Cases:

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

$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right), \quad y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) \quad -(1)$$

we get (1) when the projection reference point is limited to positions along the z_{view} axis

2) $(x_{prp}, y_{prp}, z_{prp}) = (0, 0, 0)$

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

$$y_p = y\left(\frac{z_{vp}}{z}\right) \quad -(2)$$

3) $z_{vp} = 0$

$$x_p = x\left(\frac{z_{prp}}{z_{prp} - z}\right) - x_{prp}\left(\frac{z}{z_{prp} - z}\right) \quad -(3a)$$

$$y_p = y\left(\frac{z_{prp}}{z_{prp} - z}\right) - y_{prp}\left(\frac{z}{z_{prp} - z}\right) \quad -(3b)$$

we get 3a and 3b if the view plane is the uv plane and there are no restrictions on the placement of the projection reference point.

4) $x_{prp} = y_{prp} = z_{vp} = 0$

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

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

(8) Explain Bezier Curve Equation along with its properties

Ans → * Developed by French Engineer Pierre Bezier for use in design of Renault automobile.

* Beziers have a number of properties that make them highly useful for curve and surface design. They are also easy to implement

Equation :-

$P_k = (x_k, y_k, z_k)$ P_k = General (n+1) Control-point positions

P_u = the position vector which describes the path of an approximate Bezier polynomial function between P_0 and P_n

$$P(u) = \sum_{k=0}^{n} P_k BEZ_{k,n}(u) \quad 0 < u < 1$$

$BEZ_{k,n}(u) = C(n,k) u^k (1-u)^{n-k}$ is the Bernstein Polynomial

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

Properties :-

* Basic functions are real.
* Degree of polynomial defining the curve is one less than number of defining points.
* Curve generally follows the shape of defining polygon
* Curve connects the first and last control points

Thus $P(0) = P_0$, $P(1) = P_n$

9) Explain normalization transformation for an orthogonal projection

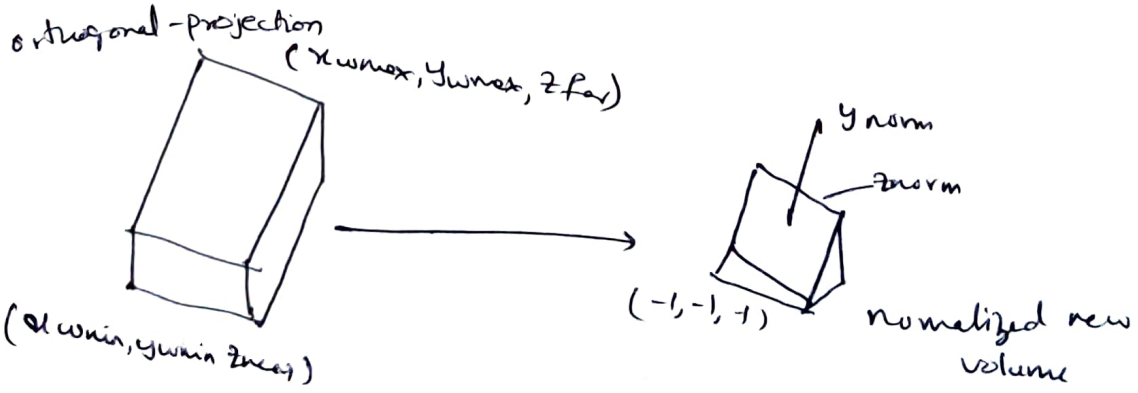
Ans → The Normalization transformation, we assume that orthogonal Projection view volume is to be mapped into symmetric normalization cube with in a left handed reference frame

$(x_{max}, y_{max}, z_{far})$ is mapped to $(1, 1, 1)$

Transforming the rectangular - parallelepiped view volume to a normalized cube is similar to the method for Converting the Clipping window into the normalized Symmetric Square

The normalization transformation for the orthogonal view volume is

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



10) Explain Cohen-Sutherland line clipping Algorithm?

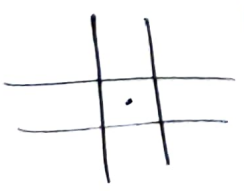
Ans → Every line endpoint in a picture is assigned a four digit binary ~~number~~ value called a region code and each bit position is used to indicate whether a point is inside or outside of one of the clipping window boundaries

1001	1000	1010
0001	0000	0010
0101	0100	0110

Once we have established region codes for all the line exponents. we can quickly determine which line are completely within clipwindow and which are clearly outside

when the OR operation between 2 endpoints region codes for a line segment is false (0000), the line is inside the clipping window

When AND operation between 2 endpoints region codes for a line is true, the line is completely outside the clipping window

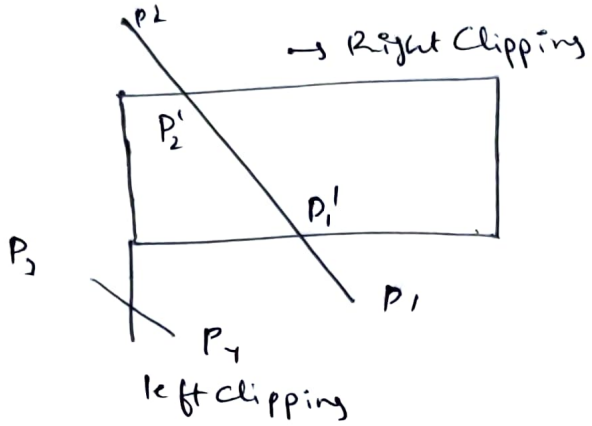


(0000) || (0000)
⇓
(0000)

1010
1000 & 1010 ⇒ (1000)

lines that cannot be identified as being completely inside (or) completely outside a clipping window by region codes test are next checked for intersection

The region codes says P_1 is inside and P_2 is outside



The intersection to be P_2' and P_2' to P_3' is clipped off

For line P_3 to P_4 we ~~define~~ find that point P_3 is outside the left boundary and P_4 is inside. Therefore the intersection is P_3 & P_3' is clipped off

$$y = y_0 + m(x - x_0)$$

where x is either x_{wmin} (or) x_{wmax} and Slope is

$$m = (y_{end} - y_0) / (x_{end} - x_0)$$

for intersection with horizontal borders, the x co-ordinate is

$$x = x_0 + \left(\frac{y - y_0}{m}\right)$$