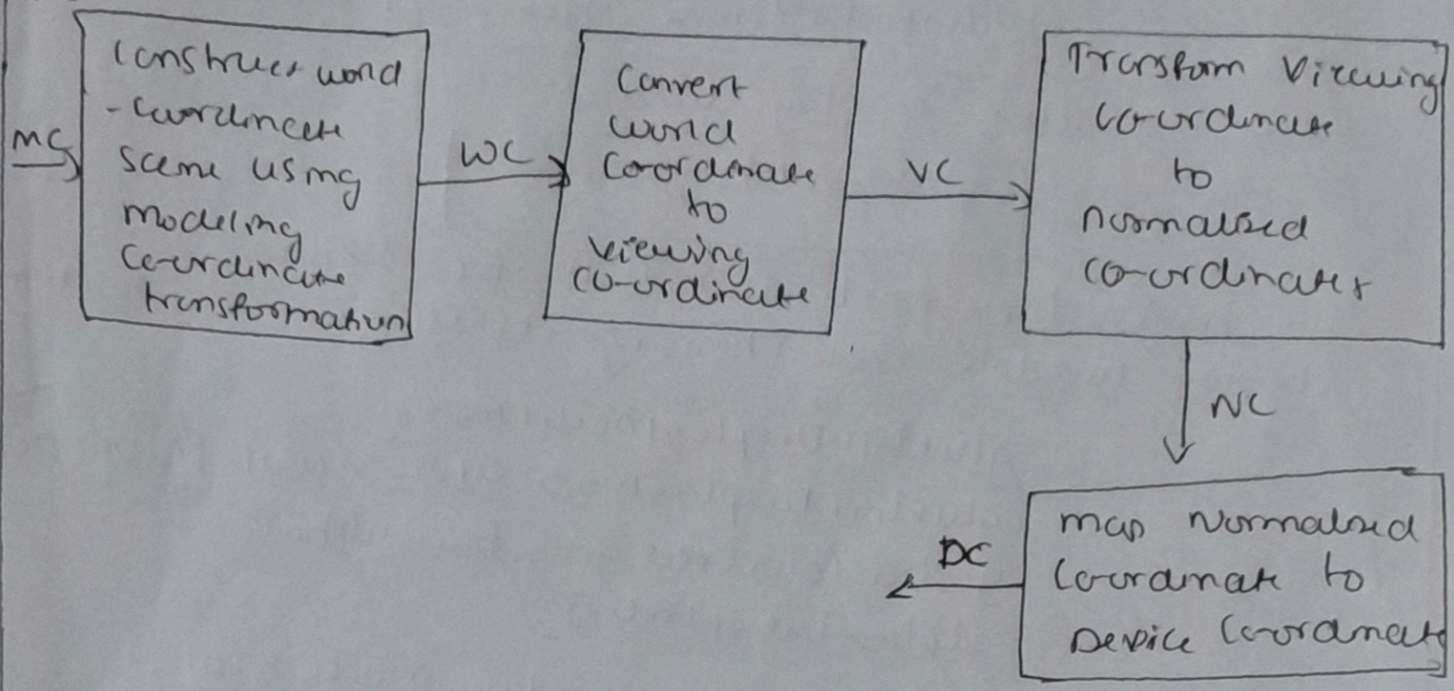
CGV Assignment

Name: Jyoti Rajo Kamati
USN: 1BY20CS076 B/VI

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 to 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 coordinates.

glMatrixMode (GL_PROJECTION);

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

→ GLU Clipping window functions:

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 definition a display window and choosing its dimension and position

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

→ Setting the GLUT Display-window mode & color:

Various display window parameters are selected with GLUT function

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

→ GLUT Display-window identifier:

```
windowID = glutCreateWindow("A display window");
```

→ Current GLUT Display Window:

```
glutSetWindow(windowID);
```

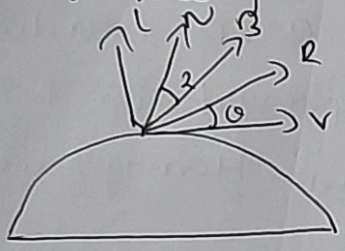
2: Build phong lighting model with equation

Ans:

Phong reflection is an empirical model of local illumination. It describes the way a surface reflects light as a combination of the diffuse of shiny surface. It is based on phong's informal observation that shiny surfaces have small intense specular highlights. while dull surfaces have large highlights while dull surfaces have large highlights, while dull that fall off more gradually.

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 effect does not exist.

for most opaque materials specular reflection co-efficient is nearly constant is



$$I_{specular} = \begin{cases} I_s k_s [(V \cdot R)^{n_s}] & V \cdot R > 0 \\ & N \cdot L > 0 \\ 0 & \text{otherwise} \end{cases}$$

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

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

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

If the light source and viewer are sufficiently far from the object α is constant

3 Apply homogeneous co-ordinates for translation, rotation and scaling in a matrix representation

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

$$P' = M_1 * P + M_2$$

P' & P represents column vectors

Matrix $M_1 \rightarrow 2 \times 2$ array containing multiplicative factor

$M_2 \rightarrow$ 2 element column matrix containing translation term $\begin{bmatrix} t_x \\ t_y \end{bmatrix}$

For translation M_1 is identity matrix $P' = P + T$ where $T = M_2$ For rotation and scaling M_2 is contain translation terms associated with pivot point or scaling

HOMOGENOUS CO-ORDINATES: A standard technique to expand the matrix representation for a 2D (x,y) position to a 3-element representation for a 2-D coordinates (x_h, y_h, h) → called Homogenous coordinates.

In homogeneous parameter the non-zero value (x, y) is converted into new co-ordinate values as (x_h, y_h, h) $x = \frac{x_h}{h}$, $y = \frac{y_h}{h}$ $x_h = x.h$ $y_h = y.h$

→ 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 $P' = T(t_x, t_y) \cdot P$ → 3×3 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

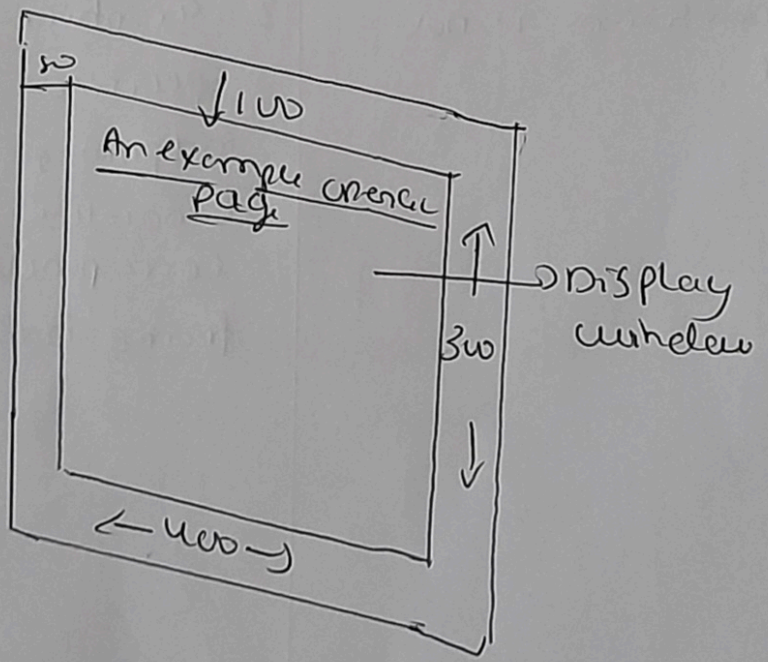
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} bx & 0 & 0 \\ 0 & by & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow P' = S(bx, by) \cdot P$$

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

Ans	Random scan display	Raster scan display
1.	In vector scan display the beam is moved between the end points of the graphics primitives	1. In raster scan display the beam is moved all over the screen one scanline at a time, from top to bottom and then back to top
2.	2. Vector display flickers when the number of primitives in the buffer become too large	In raster display, the refresh process is independent of the complexity of the image.
3.	Scan conversion is not required	3. Graphics primitives are specified in terms of their endpoints and must be scan converted into their corresponding pixel in the frame buffers

4.	Scan conversion hardware is not required	because each primitive must be scan-converted in real time dynamics it for more computational and required separate scan conversion hard-ware
5	vector display derive a continuous and smooth lines	Raster display can display mathematically smooth liner polygons and boundaries of curved primitives only by approximating them with pixels in the raster grid
6	cost it more	cost it Low
7.	vector display only draw lines and characters	Raster display has ability to display area filled with solid colors 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 last one in our program. It displays 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 top edge of the screen.

* glutInitWindowSize(400, 300);
  the glutInitWindowSize function is used to set the initial pixel width and height of 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 want to use the color mode which uses red, green and blue (RGB) components to select color values

6 Explain OpenGL visibility Detection functions

a) OpenGL Polygon-Culling Functions

Back-face removal is accomplished with the function glEnable(GL_CULL_FACE);
glCullFace(mode);

* where parameter mode is assigned the value GL_BACK, GL_FRONT, GL_FRONT_AND_BACK
* By default, parameter mode in glCullFace function has the value GL_BACK
* The culling routine is turned off with glDisable(GL_CULL_FACE).

b) OpenGL Depth-Buffer Functions:

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

glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

→ Depth buffer value can be initialized with

glClear(GL_DEPTH_BUFFER_BIT)

By default it is set to 0

→ These routines are activated with the following function

glEnable(GL_DEPTH_TEST);

And we deactivate these depth-buffer routines with glDisable(GL_DEPTH_TEST);

→ We can also apply depth buffer testing using some other initial value for the maximum depth

glClearDepth(maxDepth);

c) OpenGL wire-frame surface visibility methods

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

glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

But this displays both visible and hidden edges

d) OpenGL-depth-cueing function

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

glEnable(GL_FOG);

glFogi(GL_FOG_MODE, GL_LINEAR);

→ This applies the linear depth function to object color using dmin = 0.0 and dmax = 1.0 we can see 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) $z_{prp} = y_{prp} = 0$

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

we get (1) when the projection reference point or limited to positions along the z view axis

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

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

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

we get (2) when the projection reference point is fixed as co-ordinate origin

3) $z_{vp}=0$

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

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

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

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

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

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

we get (4) when the uv plane as the view plane & the projection reference point on the z-D view axis

8) Explain Bezier curve equation along with its properties

Ans

- Developed by French Engineer Pierre Bezier for use in design of Renault automobile bodies
- Bezier have a number of properties that make them highly useful for curve and surface design they are also easy to implement
- Bezier curve section can be fitted to any number of control points

q) Explain normalization transformation for an orthogonal projection

Ans

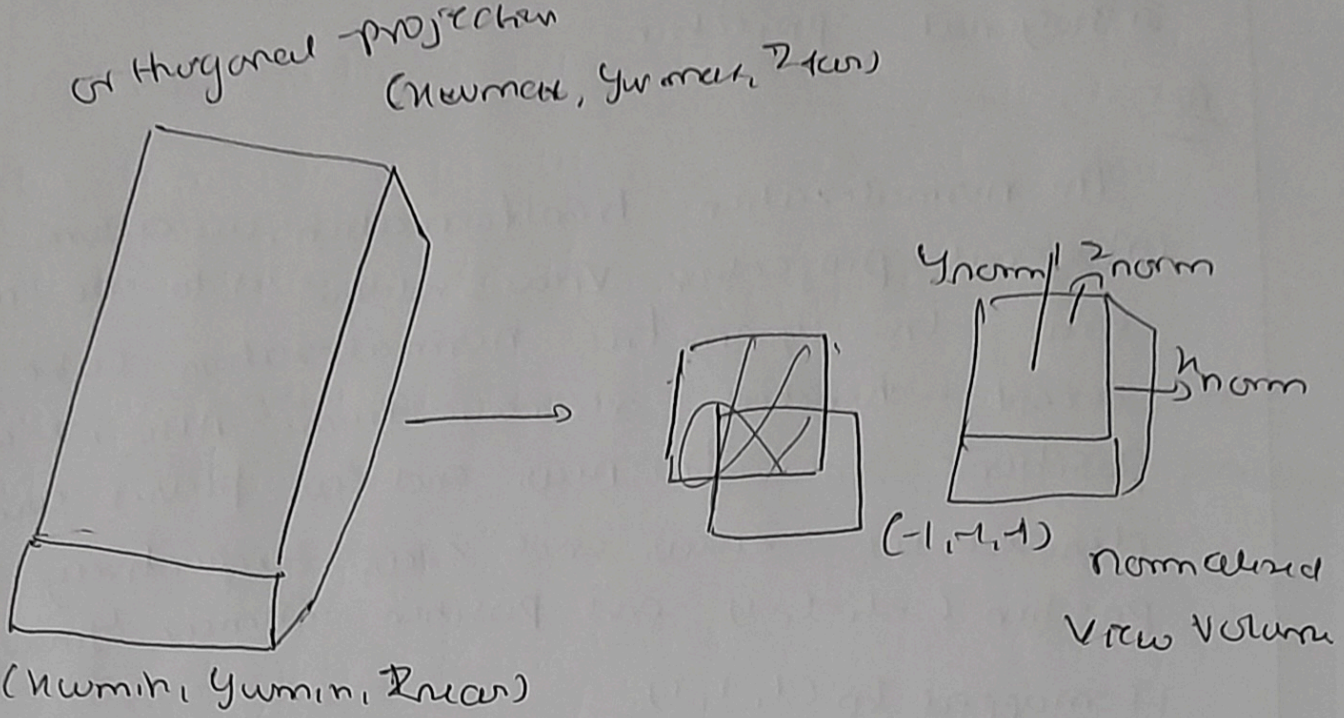
The normalization transformation is used to transform the orthogonal projection view volume into a symmetric normalization cube within a left-handed reference frame. Also, z-coordinate positions for the near and far planes are denoted as $-z_{near}$ and z_{far} respectively. This is position $(-1, -1, -1)$ and position $(x_{max}, y_{max}, z_{max})$ 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

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

The matrix is multiplied on the right by the composite viewing transformation R-T to produce the composite transformation from world coordinates to normalized orthogonal-projection coordinates



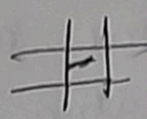
10 Explain Cohen-Sutherland line clipping algorithm

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

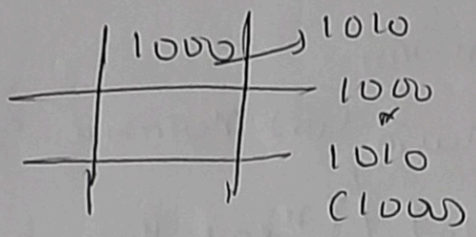
1001	1000	1010
0001	0000 clipping window	0010
0101	0100	0110

Once we have established region codes for all line endpoints we can quickly determine which lines are completely within clip window & which are clearly outside

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

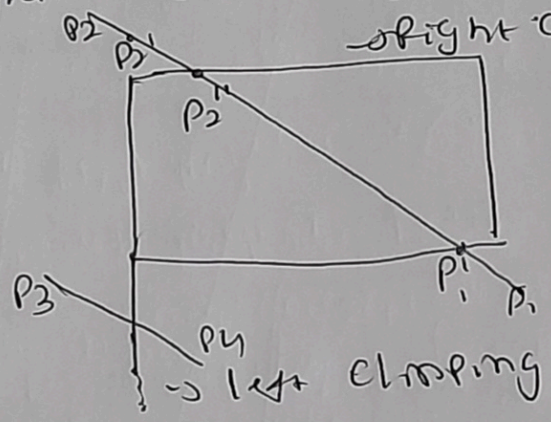


when AND operation between 2 endpoints region codes for a line is true, the line is completely outside the clipping window



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

The region codes says P_2 is inside and P_2 is outside



The intersection to be P_2'' & P_2' to P_2'' is clipped off for line P_3 to P_4 we find that point P_3 is outside the left boundary & P_4 is inside Therefore the intersection is P_3 & P_3 to P_3' clipped off

P_2

By checking the region codes of P_3 & P_4 we find the remainder of the line is below the clipping window & can be eliminated. To determine a boundary intersection for a line equation the y-coordinate of intersection point with vertical clipping border line can be obtained by

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

where x is either x_{win} (or) x_{wmax} and slope is

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

∴ for intersection with horizontal border, the x coordinate is

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