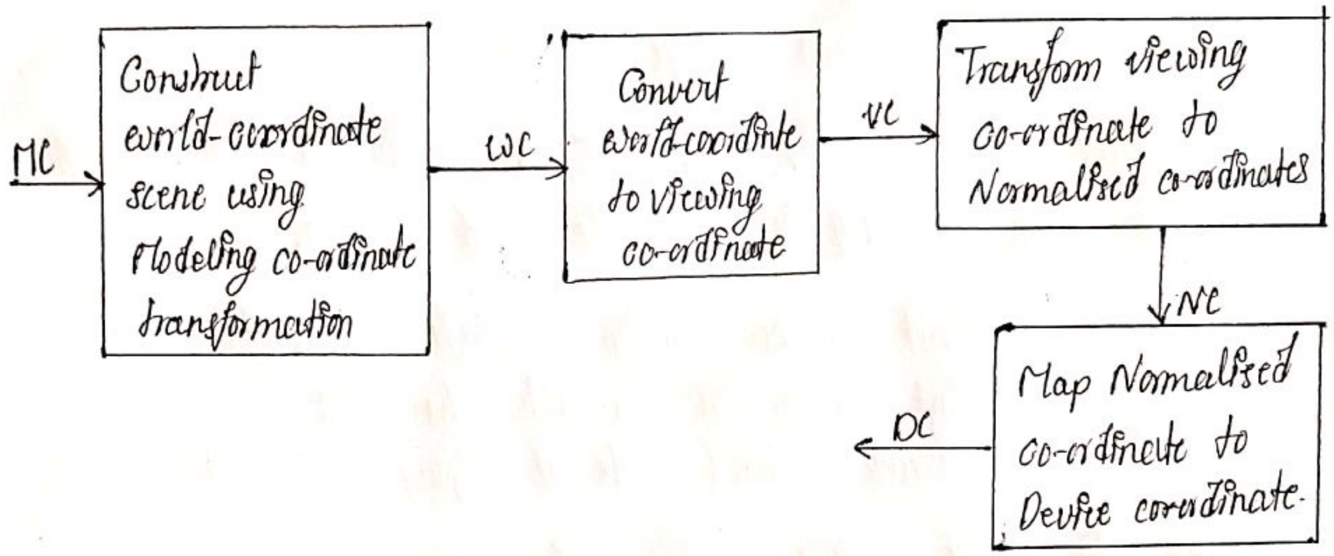
CG Assignment

Name : M. Saye Siddhartha Reddy
USN : 1BY20CS103

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 co-ordinates to screen co-ordinates.

$\boxed{\texttt{glMatrixMode(GL\_PROJECTION);}}$

This designates the Projection Matrix as the current matrix, which is originally set to identity matrix.

→ GLU clipping-window Function :-

To define a 2-D clipping window, we can use the openGL utility function

$\boxed{\text{gluOrtho2D(xwmin, xwmax, ywmin, ywmax);}}$

openGL viewport Function:

$\boxed{\text{glViewPort(xvmin, yvmin, vpwidth, vpHeight);}}$

Create a Glut Display window :-

$\boxed{\text{glutInit(\&argc, argv);}}$

we have three functions in GLUT for defination of a display window and choosing its dimension and position.

```
glutInit windowPosition (xTopleft , yTopleft);
glutInit WindowSize (dewidth, dheight);
glutCreateWindow ("Title of display window");
```

→ Setting the GLUT Display-window Mode & color :-

various display window Parameters are selected with the GLUT function:

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

→ GLUT Display-window identifier :-

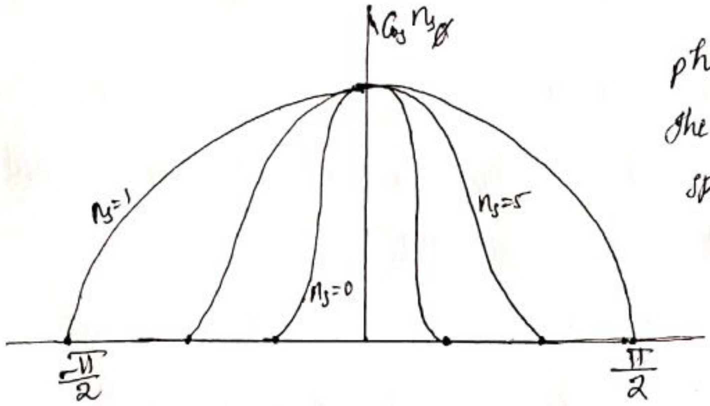
```
windowID = glutCreateWindow ("A display window");
```

→ Current Glut Display window :-

```
glutSetWindow (window.ID);
```

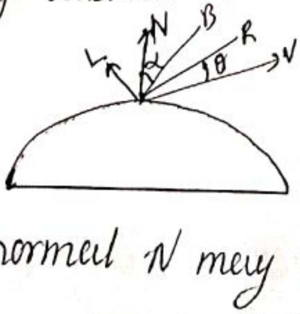
② Build phong lighting model with equations

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 shiny surface. It is based on phong's informal observation that shiny surface have small intense specular highlights, while dull surfaces have large highlights that fall off more gradually.



phong model sets the intensity of specular reflection at $\cos^{n_s}\phi$

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 do not exist. For most opaque materials specular-reflection co-efficient is nearly constant ks



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

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

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

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

CS CamScanner

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

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

Ans: The three basic 2D-transformations are translation, rotation & scaling

$$\boxed{P' = M_1 * P + M_2}$$ P' & P represents column vectors

Matrix $M_1 \rightarrow$ 2x2 array containing multiplicative factors

$M_2 \rightarrow$ elements column matrix containing translation term $\begin{bmatrix} x_t \\ y_t \end{bmatrix}$

For translation, M_1 is identity matrix $P' = P+T$ where $T = M_2$

For rotation and scalling, M_2 contains translational terms associated with pivot pointer scaling.

<u>HOMOGENOUS CO-ORDINATES</u> :- A standard techniques to expand the matrix representation for a 2D co-ordinate (x, y) position to a 3-element representation $(x_h, y_h, h) \rightarrow$ called Homogeneous co-ordinates.

$h \rightarrow$ homogeneous parameter h (non-zero value)

(i.e) (x, y) is converted into new co-ordinate values as (x_h, y_h, h)

$$x = \frac{x_h}{h}, \quad y = \frac{y_h}{h} \qquad x_h = x.h \qquad y_h = y.h$$

→ <u>Translation</u> :

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

This translation operation can be written as $P' = \underset{\substack{\downarrow \\ 3\times 3 \text{ translation matrix.}}}{T(t_x, t_y)} \cdot P$

→ <u>Rotation</u> :

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

→ Scaling Matrix :-

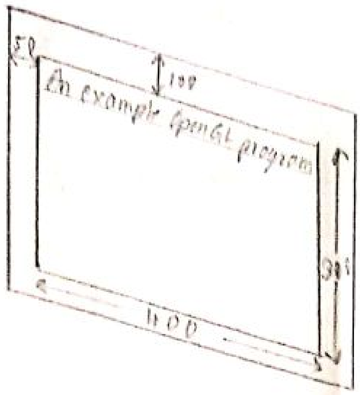
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow P' = S(S_x, S_y) \cdot P$$

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

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 once the screen one scanline at a time, from top bottom and then break to top.
2. Vector display flicker when the number of primitives in the buffer becomes too large	2. 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 connected into their corresponding pixel in the frame buffers
4. Scan Conversion hardware is not required	4. Because each primitive must be scan-converted, real-time dynamics is for more computational and required separate scan conversion hardware.
5. Vector display derive a continuous and smooth lines	5. Raster display can display mathematically smooth lines, polygons, and boundries of curved primitives only by approximating them with pixel on the rastergrid
6. Vector display only draws lines and characters.	6. Raster display has ability to display area filled with solid colours on patterns.

⑤ Demonstrate OpenGL functions for displaying window management using GLUT



* We perform the GLUT initialization with the statement

$\boxed{\text{glutInit(\&argc, argv);}}$

* 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 calls the line segment description to the window display window → glutDisplayFunc(line segment);

* 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 60 pixel to the right of the left edge of the screen and 100 pixel down from the top edge of the screen.

* glutInitWindowSize(400, 300);

The 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 colour 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, 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-function :-

To use the OpenGL depth-buffer visibility-detection function, 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 values can be initialized with

glCreate(GL_DEPTH_BUFFER_BIT);

* By default it is set to 1.0

→ These routines are activated with the following function:-

glEnable (GL_DEPTH_TEST);

And we deactivates these depth-buffer routines with

glDisable (GL_DEPTH_TEST);

→ We can also apply dept-buffer testing using some other initial value for the maximumdepth

glClearDepth (maxDepth);

* It can be set to any value b/w 0 and 1

→ As an option, we can adjust normalization values with

glDepthRange (nearNormalDepth, farNormDepth);

→ We specify a test condition for the depth buffer routines using the following functions

glDepthFunc(test Condition)

→ We can stet the status of the depth buffer so that it is in a read-only sate 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 are to be generated.

glPolygonMode (GL_FRONT_AND_BACK, GL_LINE)

But this displays both visible and hidden edges.

d] openGL - DEPTH - cuing 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 colors using $d_{min} = 0.0$ and $d_{max} = 1.0$ we can set different value for dmin and dmax with the following

glFogf (GL_FOG_START, minDepth);
glFogf (GL_FOG_END, maxDepth);

7 Write the special case that we discussed with respect to perspective projection transformation co-ordinates.

Ans

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

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

special case :-

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

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

we get 1 when the projection reference point is limited to positions along the ~~zview~~ zview axis.

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

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

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

we get 2 when the projection reference point is fixed at co-ordinate origin.

3. $z_{vp} = 0$

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

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

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

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

$$y_p = y\left[\frac{z_{prp}}{z_{prp} - z}\right] \quad \text{— (4)}$$

we get (4) with the uv plane as the view plane & the projection reference point on the z 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.

Equation:

$P_k = (x_k, y_k, z_k)$ P_k = General (n+1) control point position

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

$$P(u) = \sum_{K=0}^{n} P_K \, BEZ_{K,n}(u) \qquad 0 \le u < 1$$

$BEZ_{K,n}(u) = C(n,k)\, u^k (1-u)^{n-k}$ is the Bernstein polynomial.

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

properties :-

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

thus $P(0) = P_0$

$P(1) = P_n$

* curve lies within the convex hull of the control points

(9) Explain normalization transformation for an orthogonal projection.

Ans:- The normalization transformation, 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. This position $(x_{min}, y_{min}, z_{near})$ is mapped to the normalized position $(-1,-1,-1)$ and position $(x_{max}, y_{max}, z_{far})$ is mapped to $(1,1,1)$.

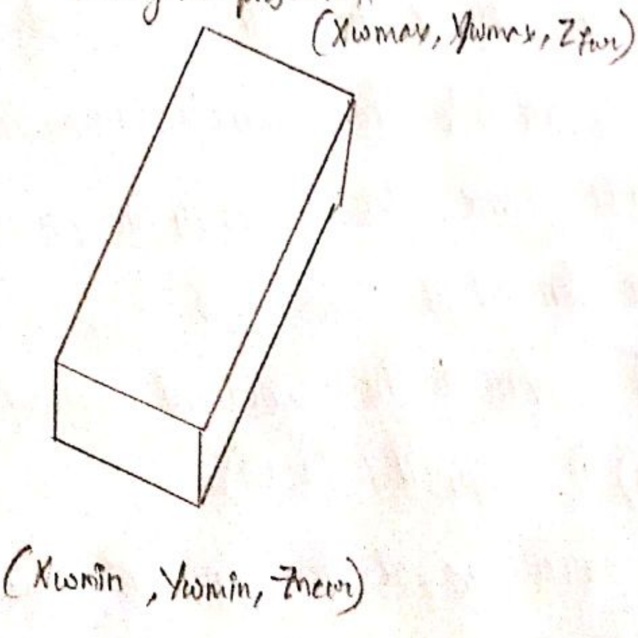
Transforming the rectangular-parallelepiped view ~~plane~~ volume to a normalized cube is similar to the method for converting the clipping window into the normalized symmetric square.

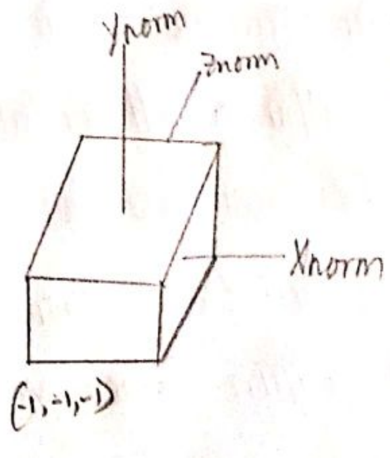
The normalization transformation for the orthogonal view volume is

$$norm = \begin{bmatrix} \frac{2}{Xwmax - Xwmin} & 0 & 0 & -\frac{Xwmax + Xwmin}{Xwmax - Xwmin} \\ 0 & \frac{2}{Ywmax - Ywmin} & 0 & -\frac{Ywmax + Ywmin}{Ywmax - Ywmin} \\ 0 & 0 & \frac{-2}{Znear - Zfar} & \frac{Znear + Zfar}{Znear - Zfar} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The matrix is multiplied on the right by the composite viewing transformation from world co-ordinates R.T to produce the complete transformation from world co-ordinates to normalize orthogonal-projection co-ordinates.

Orthogonal projection





(10) Explain cohen-sutherland line clipping algorithm

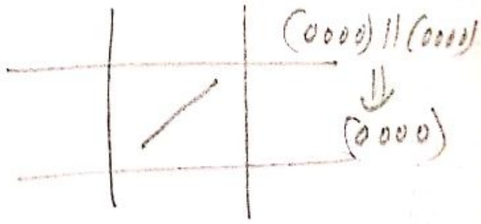
Every line endpoint in a picture is assigned a four digit binary value called a region code and each bit position is used to indicate whether the part is inside or outside of one of the clipping window boundaries

1001	1000	1010
0001	0000 clipping window	0010
0101	0100	0110

Once we have established region code for all the line endpoints. we can quickly determine which line are completly within clipwindow & which are clearly outside.

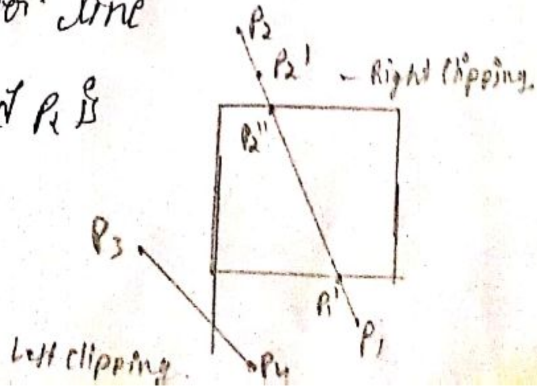
when the OR operation between 2 endpoints region codes for a line segment is false (0000), the line is inside the clipping window.

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



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

The region code says P_1 is inside and P_2 is outside



The intersection to be P_2'' & P_2' to P_2 is clipped off. For line

For line P_3 to P_4 we find that point P_3 is outside the left boundry and P_4 is inside. Therefore, the intersection is P_3 & P_3 to P_3' is clipped off. By checking the region codes of P_3' & P_4, we find the remainder of the line is below the clipping window and can be eliminated. To determine a boundry intersection for a line equation the y. co-ordinate of intersection point with vertical clipping bordel line can be obtained by

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

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 border, the x co-ordinate is

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