Name: Manojk.A
USN: 1BY20CS106
CSE 6th B sec

CGV ASSIGNMENT

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

MC → [Construct world coordinate scene using modelling co-ordinate transformation.] → WC → [Construct world coordinate to viewing co-ordinate] → VC → [Transform viewing co-ordinates to normalized co-ordinates] → NC → [Map Normalized Co-ordinate to Device Co-ordinates] → DC

- The mapping of a two-dimensional world co-ordinate scence description to device co-ordinate is called two-dimensional viewing transformation.
Sometimes this transformation is simply refered to as the window to viewport transformation or window transformation.
Once the world-coordinate scence has been constructed we could set up an sperate 2D viewing co-ordinates reference frame for specifying the clipping window. Viewing co-ordinates for 2D applications are the same as world co-ordinates. To make the viewing propecess independent of the requirements of any output device, graphics systems. Construct object descriptions to normalized co-ordinates in the range from 0 to 1, and others use range from -1 to 1. Depending upon the graphics library in use, the viewport is defined either in normalized co-ordinates or in screen-co-ordinates after the normalization process.

- 2D Viewing Functions:-

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 to screen.

glMatrixMode(GL-PROJECTION):-

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

GLU clipping-window function:-

If define a two-dimensional 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.

glutInitWindowPosition (xTopleft, yTopleft);

glutInitWindowSize (dwidth, dwheight);

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);

Select Display-Window identifier:-

WindowID = glutCreateWindow ("A display window");

glutDestroyWindow (WindowID);

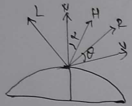
glutSetWindow (Window-ID);

glutPositionWindow (xNewTopleft, yNewTopleft);

2) Build Phong Lighting model with equations:

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



Phong Model sets the intensity of specular reflection to $\cos^n\phi$

I_1, specular $= w(\theta) I_l \cos^n \phi$ $\quad (0 \le w(\theta) \le 1)$ is called specular co-efficient.

If light derives and viewing direction V are on the scene side of normal N, as if L is surface, specular effect does not exist.

For most opaque material specular-reflection co-efficient is nearly constant k_s.

$$I_{light} = I_{ambient} + I_{diffuse} + I_{specular}$$

$$I = I_a k_a + I_d k_d (N.L) + I_s k_s (R.V)^s$$

Here I_a is a combination of red, green, & blue component of

$$I_a = (I_{ra}, I_{g}, I_{rn}).$$

similarly I_d is a combination of red, green, and blue component of diffuse intensity. $I_d = (I_{da}, I_{dg}, I_{db})$

and I_s is a combination of red, green, & blue component of specular reflection intensity.

$$I_s = (I_{sa}, I_{sg}, I_{sb})$$

These can be represented in a matrix form as

$$I = \begin{vmatrix} I_{ar} & I_{ag} & I_{ab} \\ I_{dr} & I_{dg} & I_{db} \\ I_{sr} & I_{sg} & I_{sb} \end{vmatrix}$$

- The 3×3 matrix of illumination model of the i^{th} light source is

$$L_i = \begin{vmatrix} L_{iar} & L_{idg} & L_{iab} \\ L_{idr} & L_{idg} & L_{idb} \\ L_{isr} & L_{isg} & L_{isb} \end{vmatrix}$$

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

→ Translation:-

$$P' = P + T$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix}$$

Rotation:-

$$P'_x = P_x \cos\theta - P_y \sin\theta$$

$$P'_y = P_x \sin\theta - P_y \cos\theta.$$

In matrix form

$$P' = RP$$

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Scaling:

$$P'_x = S_x \cdot P_x$$

$$P'_y = S_y P_y.$$

In matrix form:

$$P' = S \cdot P$$

$$S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

$$P' = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix}$$

Rotation $P' = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

scaling $P' = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

Translation $P' = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix}$

Rotation $P' = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Scaling $P' = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

using homogeneous co-ordinates, the transformations (x_h, y_h, h)
where $x = x_h/h$ $y = y_h/h$
$(h*x, h*y, h)$ set $h=1$
$\Rightarrow (x, y, 1)$

Homogeneous coordinates representation for translation scaling and rotation are as follows:-

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

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

4) Outline the difference between Raster scan display and Random Scan display.

Random Scan	Raster Scan
* The resolution of random scan is higher than raster scan	* While the Resolution of Raster scan is better than random scan.
* It is cost is less	* It is costlier than random scan.
* Reflection is easy in comparison of raster scan	* Any reflection is not easy.
* Interviewing is not used	* Interviewing is used.
* It is suitable for applications requiring polygon determings	* It is suitable for creating relative scene.

5) Demonstrate OpenGL functions for displaying window management using GLUT

-> glutInit(&argc, argv)
It is used to initialise GLUT Library.
-> glutInitWindowPosition(xtopleft, ytopleft);
-> position of display window on screen.
-> glutInitWindowSize(width, height)
size of window.
glutwidth is width of display.
glutheight is height of display.
gluCreateWindow("string");
It is used to create display window with name
-> glutDisplayFunc();
It sets the display for current window.
glutInitDisplayMode();
It sets the initial Display Mode;
glutReshapeFunc();
It sets the reshape corelate for current window.
glutsetCursor();
It changes the cursor image of current window.

6) Explain OpenGL visibility detection functions.

⇒ glEnable (GL_CULL_FACE)

It is used for turning culling on.

glCullface (Mode)

It specifies what to cull

Mode = GL_FRONT or GL_BACK

GL_BACK is default

glfrontface(vertexOrder)

It is for order of vertices.

Orientation is changed.

VertexOrder = GL_CW or GL_CCW

GL_CW is for clockwise direction (front)

GL_CCW is for counter clockwise direction)

GL_CCW is default.

Create depth buffer by setting GLUT_DEPTH flag in glutInitDisplayMode() or the appropriate flag in the PIX f.L FORMATDISC

Enable per-pixel depth testing with glEnable(GL_DEPTH_TEST)

Clear depth buffer by setting GL_DEPTH_BUFFER_BIT in glClear()

glDepthfunc(condition);

change the test used.

condition : GL_LESS [closer : resizable (default)]

GL_GREATER [Rather : resizable]

7) Write a special cases that we discussed with respect to perspective projection transformation co-ordinates

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

$y' = y - (y - y_{prp})u \quad 0 \leq u \leq 1$

$z' = z - (z - z_{prp})u$

if it is at Origin

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

On the viewplane, $z' = z_{vp}$, solve this for u

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

$z' = z - (z - z_{prp})u$

substitute $z' = z_{vp}$

$z_{vp} + (z - z_{prp})u = z$

$(z - z_{prp})u = z - z_{vp}$

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

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

substitute this u into x' & y' equations

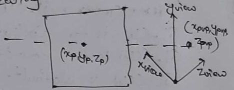
$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) + x_{prp}\left(\frac{z_{vp} - z}{z_{prp} - z}\right)$$

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

∴ final projected point on this plane is x_p, y_p, z_p --- since we are viewing along z axis, z_p can also be written as

i) If $u=0$, $x'=x$, $y'=y$, $z'=z$

2) if $u=1$, $x' = x_{prp}$, $y' = y_{prp}$, $z' = z_{prp}$



1) If projection reference point is on z_{view}, means $x_{prp} = y_{prp} = 0$

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

2) Sometimes the projection reference point is fixed at the coordinate origin and

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

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

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

$z_{vp} = 0$

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

&

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

4) with the uv plane as the view plane and projection reference point on the z view axis, the perspective equations are

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

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

*8) Explain Bezier curve equation along with its properties

we consider the general case of n+1 control points, positions denoted as $p_k = (x_k, y_k, z_k)$ with k varying from 0 to n. These co-ordinate points are blended to produce the following position vector p(u), which describes the path of an approximately Bezier polynomial function b/w p_0 & p_n.

$$p(u) = \sum_{k=0}^{n} p_k BEZ_{k,n}(u) \quad 0 \le u \le 1$$

The Bezier blending functions $BEZ_{k,n}(u)$ are the Bernstein polynomial

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

where parameters C(n,k) are the binomial co-efficients

$$C(n,k) = n! / k!(n-k)!$$

Equ p(u) represents a set of three parametric equations for the individual curve co-ordinates.

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

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

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

The most cases, a Bezier curve is a polynomial of a degree that is one less than the designated number of control points. Three points generates a parabola four points a cubic curve and so forth.

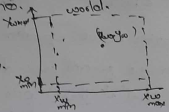
Recursive calculates can be used to obtain successive binomial co-efficient values as.

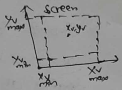
$$C(n,k) = \frac{n-k+1}{k} C(n, k-1) \text{ for } n \ge k.$$

9) Explain normalization transformation for an orthogonal projection

=> Relative position is same.

$$\frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}} = \frac{x_w - x_{wmin}}{x_{wmax} - x_{wmin}}$$





$$x_v - x_{vmin} = (x_{vmax} - x_{vmin})\left(\frac{x_w - x_{wmin}}{x_{wmax} - x_{wmin}}\right)$$

$$x_v = x_w\left(\frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}\right) + \frac{x_{vmin} + x_{wmin}x_{vmin} - x_{wmin}x_{vmax}}{x_{wmax} - x_{wmin}}$$

$$x_v = x_w\left(\frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}\right) + \frac{x_{wmax}x_{vmin} - x_{wmin}x_{vmax}}{x_{wmax} - x_{wmin}}$$

$$x_v = x_w s_x + t_x$$

where $s_x = \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}$ $\quad t_x = \frac{x_{wmax}x_{vmin} - x_{wmin}x_{vmax}}{x_{wmax} - x_{wmin}}$

Similarly for $\quad y_v = y_w s_y + t_y$.

where $s_y = \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}}$ $\quad t_y = \frac{y_{wmax}y_{vmin} - y_{wmin}y_{vmax}}{y_{wmax} - y_{wmin}}$

$$M_{window,normview} = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

For normalized co-ordinates, let us substitute,

−1 for x_{vmin} & y_{vmin}.

1 for x_{vmax} & y_{vmax}.

for 2D:-

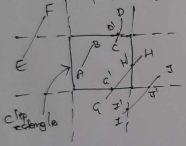
$$M_{window,normsquare} = \begin{bmatrix} \frac{2}{x_{wmax} - x_{wmin}} & 0 & \frac{-x_{wmax} + x_{wmin}}{x_{wmax} + x_{wmin}} \\ 0 & \frac{2}{y_{wmax} - y_{wmin}} & \frac{-y_{wmax} + y_{wmin}}{y_{wmax} + y_{wmin}} \\ 0 & 0 & 1 \end{bmatrix}$$

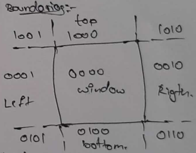
Similarly for 3D, will get this.

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

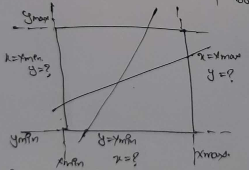
10) Explain Cohen-Sutherland line clipping algorithm.

To clip the pixels outside the window, lets first calculate the intersection point then redraw the line from inner window point to this intersection point.





consider :-

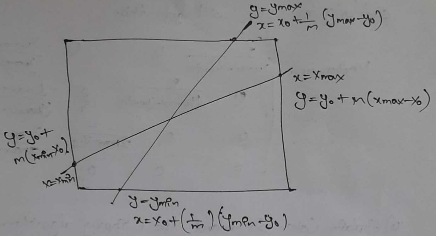


$$m = (y-y_0)/(x-x_0)$$

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

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

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



y = ymax
x = x0 + 1/m (ymax - y0)
x = xmax
y = y0 + m(xmax - x0)
y = y0 + m(xmin - x0)
x = xmin
y = ymin
x = x0 + (1/m)(ymin - y0)