Likhith
1BY20CS096
6th B CSE

CGV - Assignment

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

⇒

MC → Construct world coordinate Scene using Modeling 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-ordinate → DC

The mapping of a two-dimensional, world co-ordinate scene descviption 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 scene has been constructed we could set un a separate 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 process independent of the requirements of any output device, graphics systems. construct object descriptions to normalized cordinates 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:

Open GL 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:-

It define a two-dimensional clipping window, we can use the OpenGL utility function:

gluOrtho2D(Xwmin, Xwmax, Ywmin, Ywmax);

OpenGL viewPort function:-

glViewPort(Xvmin, Xvmin, Vpwidth, VpHeight);

Create a GLUT Display window:-

glutInit(&argc, argv);

We have three functions in GLUT for definination 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:-

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

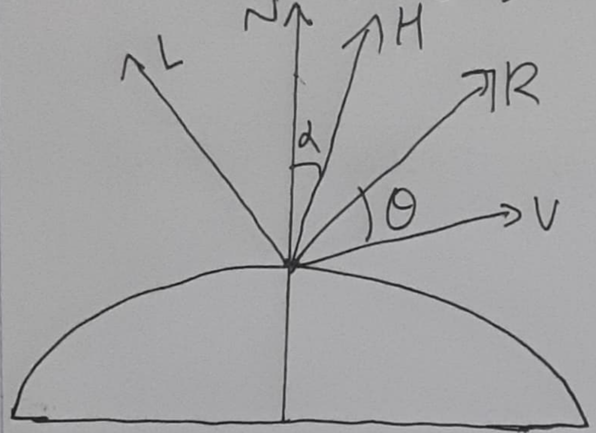
glutDestroyWindow(windowID);

glutSetWindow(window_ID);

glutPositionWindow(xNew TopLeft, yNew Top Left);

2. Build Phong Lighting Model with equations

⇒ Phong reflection is an empirical model of local ilumination. It describes the way a surface reflects light as a combination of the defuse reflection of rough surfaces with the specular reflection of sting surfaces. It is based on Phong's informal observation that shiny surfaces have small intense specular highlights while dulll surfaces have large highlights that fall off more gradually.



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

I_1, specular $= W(\theta) I_c \cos^{n_s} \phi$

$0 \leq W(\theta) \leq 1$ is called specular reflection coefficient.

If light derives l and viewing direction V are on the seence side of the normal N, as if L is the surface, specular effects doesn't exist.

For most opeque materials specular-reflection co-efficient is nearly constant k_s.

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

$$I = I_a K_a + I_d K_d (N \cdot L) + I_s K_s (R \cdot V)^S$$

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

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

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 and 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 3x3 matrix of the illumination model of the ith light source is

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

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

⇒ Translation:

$\cancel{P'_x = P_x + t_x}$ $P' = P + T$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Rotation:

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

$$P'_y = P_x \sin\theta + P_y \cos\theta$$

in matrix form

$$P' = R \cdot P$$

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

Scaling:

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

$$P'_y = S_y \cdot P_y$$

in matrix form:

$$P' = S \cdot P$$

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

$$P' = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

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

Scaling $p' = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

$\Rightarrow$ Translation $p' = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$

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 $p' = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

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 $h = 1$

$(x, y, 1)$

Homogeneous co-ordinates representation for translation, scaling and rotation are as follows:

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

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

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

4 Outline the difference b/w 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 lower than random scan.
• It is costlier than random scan	• cost is lesser.
• ~~Raster 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 scence.

5 Demonstrate OpenGL functions for displaying window management using GLUT

→ glutInit(&argc, argv)

It is used to intialize GLUT library.

• glutInitWindowPosition(xtopleft, ytopleft);

Position of display window on screen.

glutInitWindowSize(width, height);

Size of window.

gluwidth is width of display.

gluheight is height of display

gluCreateWindow("string");

It is used to create disply window with nare glutDisplayfunc();

glutDisplayfunc();

It sets the display for current window.

glutInitDisplayNode();

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 calling on.

glcallface(mode)

It specifies what to call

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 counterclockwise direction.

GL-CCW is default.

Create depth buffer by setting GLUT-DEPTH flag in glutInitDisplayMode() or the appropriate flag in the PIXEL FORMAT DESC.

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 $z' = z_{vp}$

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

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

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

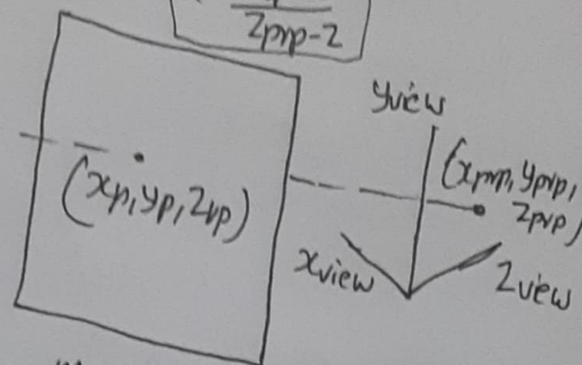
$\boxed{u = \frac{z_{vp} - z}{z_{prp} - z}}$

Substitute this u into x' & y' equations

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

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

• 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



1) if $u=0$, $x'=x$, $y'=y$, $z'=z$

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

. If projection reference point is on zview, means $x_{prp} = y_{prp} = 0$

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

. Sometimes the projection reference point is fixed at the co-ordinate origin and

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

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

. If the view plane is the uv plane and there are no restrictions on the placement of the projection reference point, then we have $z_{vp} = 0$:

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

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

. with the uv plane as the view plane and the projection reference point on the zview axis, the perspective equations are.

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

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

8 Explain Bezier curve equation along with its properties (4)

⇒ 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 approximating 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 polynomials

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

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

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

Eqn $p(u)$ represents a set of three parametric equations for the individual curve co-ordinates:

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

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

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

In 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.

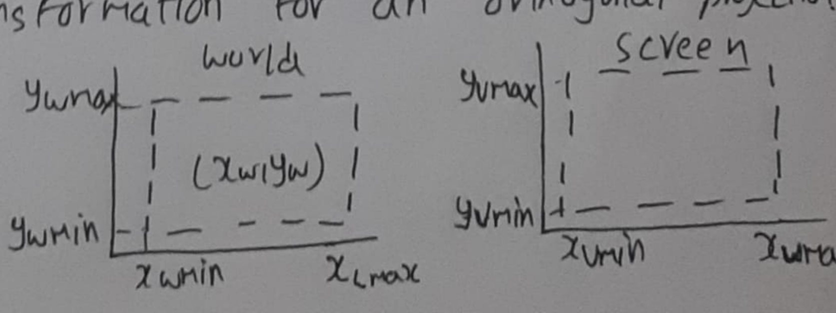
- Recursive calculations can be used to obtain successive binomial-co-efficient values as

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

9 Explain normalization transformation for an orthogonal projection

⇒ Relative position is same.

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



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

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

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

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

$$x_v = x_w s_x + t_x$$

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

similarly

$$y_v = y_w s_y + t_y$$

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

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

For normalized co-ordinated, let us substitute.

-1 for x_{vmin} & y_{vmin}

1 for x_{vmax} & y_{vmax}

For 2D:

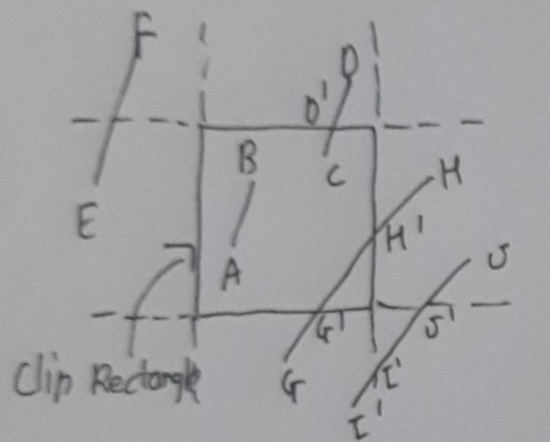
$$M_{window,normsquare} = \begin{bmatrix} \frac{2}{x_{wmax} - x_{wmin}} & 0 & -\frac{x_{wmax} + x_{wmin}}{x_{wmax} - x_{wmin}} \\ 0 & \frac{2}{y_{wmax} - y_{wmin}} & -\frac{y_{wmax} + y_{wmin}}{y_{wmax} - y_{wmin}} \\ 0 & 0 & 1 \end{bmatrix}$$

similarly, for 3D, You'll get this.

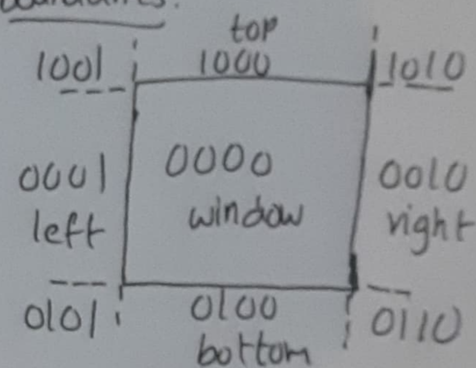
$$M_{orthonorm} = \begin{bmatrix} \frac{2}{x_{wmax}-x_{wmin}} & 0 & 0 & -\frac{x_{wmax}+x_{wmin}}{x_{wmax}-x_{wmin}} \\ 0 & \frac{2}{y_{wmax}-y_{wmin}} & 0 & -\frac{y_{wmax}+y_{wmin}}{y_{wmax}-y_{wmin}} \\ 0 & 0 & \frac{-2}{z_{near}-z_{far}} & \frac{z_{near}+z_{far}}{z_{near}-z_{far}} \\ 0 & 0 & 0 &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

Explain Cohen-Sutherland line clipping algorithm.

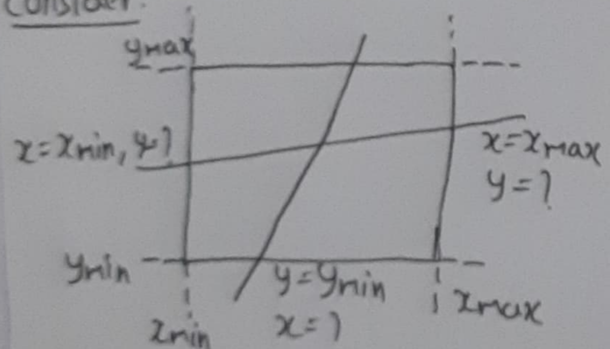
→ To clip the pixels outside the window, lets first calculate the intersection point, then redraw the line from inner window point to this interection point.



Boundaries:



consider:

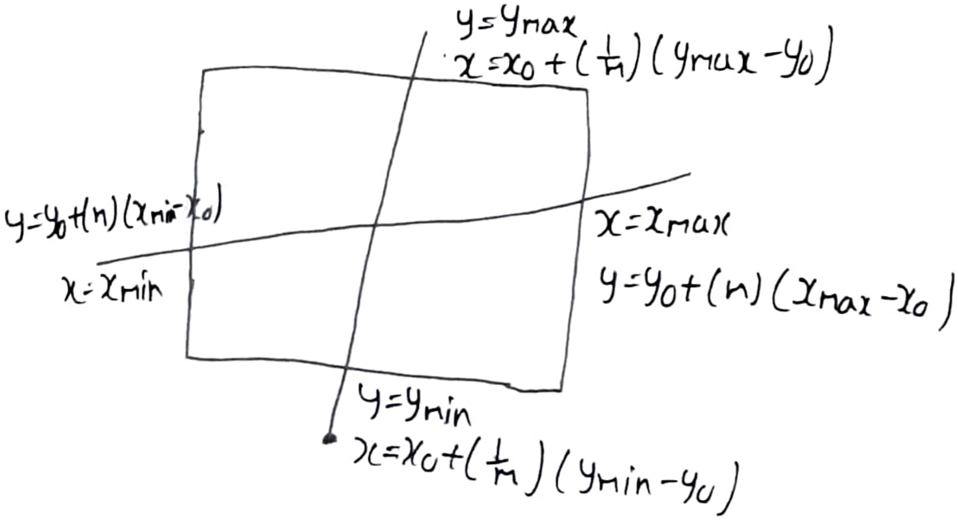


$m = (y-y_0)/(x-x_0)$

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

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

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


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