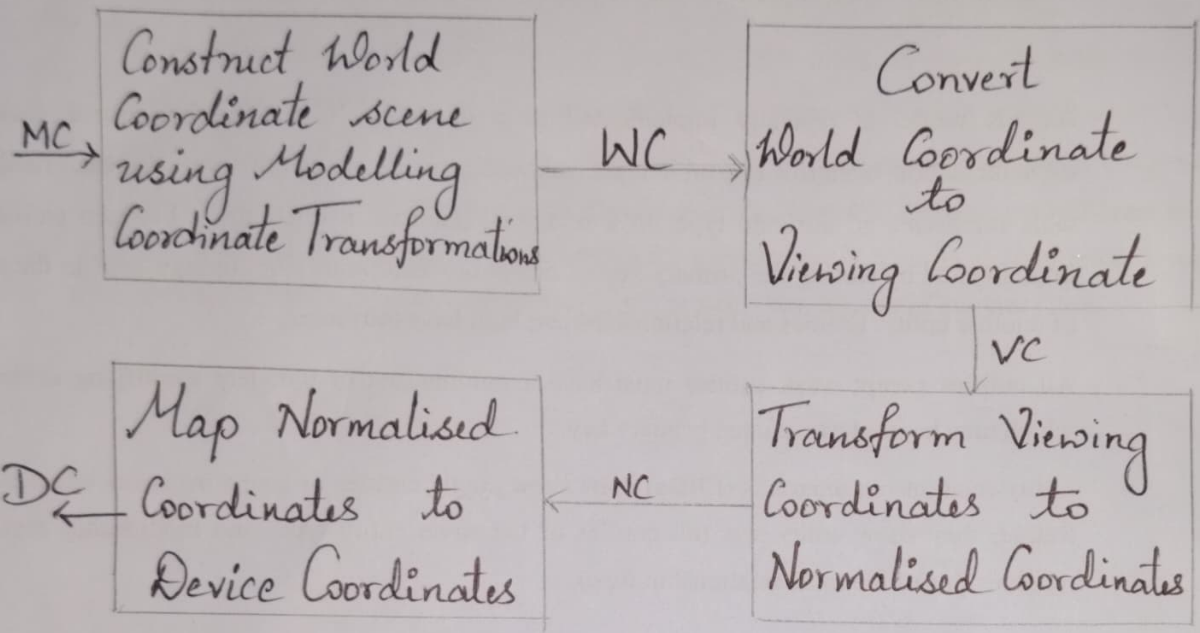
KRUTHIKA. C. K
1BY20CS090

CG Assignment

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



A section of a 2D scene that is selected for display is called a clipping window because all parts of the scene outside the selected section are "clipped off".

The mapping of a 2D world coordinate (WC) description to device coordinates (DC) is called a 2D viewing transformation. Sometimes this transformation is simply referred to as the window to viewport transformation or window transformation.

Once the world coordinate scene has been constructed we could set up a separate 2D viewing coordinate reference frame for specifying the clipping window. Depending upon the graphics library, the viewport is defined in normalised coordinates or screen coordinates.
At the final step of viewing transformation, the contents of the viewport are transferred to portions within the display window.

The OpenGL 2D viewing functions are.

OpenGL Projection Mode.
Before we select a clipping window and a viewport to 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 the identity matrix

GLU Clipping Window function:

glViewport(xvmin, yvmin, vpwidth, vpheight);

Create a GLUT display window:

glutInit(&argc, argv);

We have three functions in GLUT for definition of a display window and choosing its dimension & position

```
glutInitWindowPosition(xTopLeft, yTopLeft);
glutInitWindowSize (dwidth, dheight);
glutCreateWindow("Title of the 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).
```

GLUT Display-Window Identifiers:

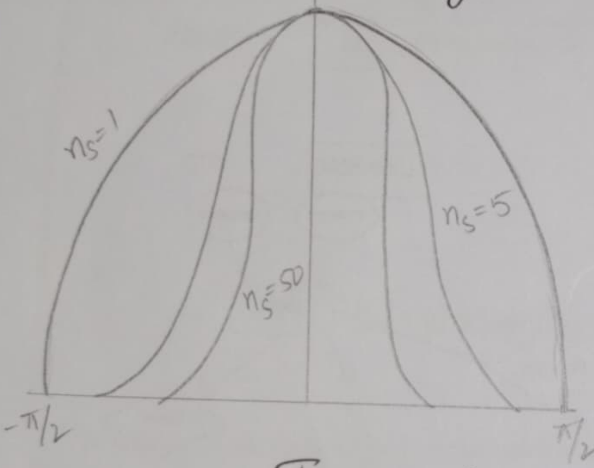
```
Window ID = glutCreateWindow("A display window")
```

Managing Multiple GLUT Display Window:

```
glutIconifyWindow ("New Window Name")
glutSetWindowTitle ("New Window Name")
```

2) Build Phong lighting Model with equations

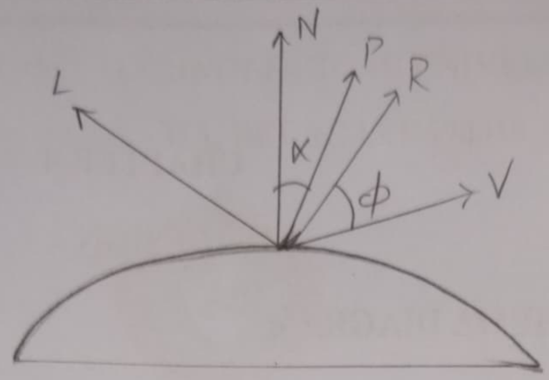
Phong reflection is an empirical model of local illumination. It describes the way a surface reflects light as a combination of the diffuse reflection of rough surfaces with the specular reflection of shiny surface. It is based on Phong's informal observation that shiny surface have small intense specular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



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

$$I_{l,\,specular} = W(\theta)\, I_l \cos^{n_s}\phi$$

$0 \le W(\theta) \le 1$ is called specular reflection coefficient.
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 coefficient is nearly constant K_s.



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

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

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

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

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

H is the direction yielding maximum specular reflection in the viewing direction V, if the surface Normal N would coincide with H.

If V is coplanar with R and L (and hence with N too) $\alpha = \phi/2$.

3) Apply homogenous coordinates for translation, rotation and scaling via matrix representation.

A standard technique for accomplishing 2D or 3D Transformation is to expand each two-dimensional coordinate-position representation (x, y) to a three-dimensional or three-element representation (x_h, y_h, h) called homogenous coordinates.

where,

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

Translation:

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

$$P' = T(t_x, t_y) \cdot P$$

Scaling.

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

$$P' = S(S_x, S_y) \cdot P$$

Rotation:

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

$$P' = R(\theta) \cdot P$$

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

Random Scan	Raster Scan.
→ It has high resolution	→ It's resolution is low
→ It is more expensive	→ It is less expensive
→ Any modification if needed is easy	→ Modification is tough.
→ Solid pattern is tough to fill	→ Solid pattern is easy to fill
→ Refresh rate depends on resolution	→ Refresh rate does not depend on the picture.
→ Beam Penetration technology comes under it	→ Shadow mark technology comes under this
→ It is restricted to line drawing applications	→ It is suitable for realistic display.

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 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");

where the single argument for this function can be any character string that we want to use for the display-window title.

The following function call passes the line-segment description to the display window

glutDisplayFunc(linesegment);

glutMainLoop():

This function must be the last one in the 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 pixels to the right of the left edge & 100 pixels from the top edge of the screen.

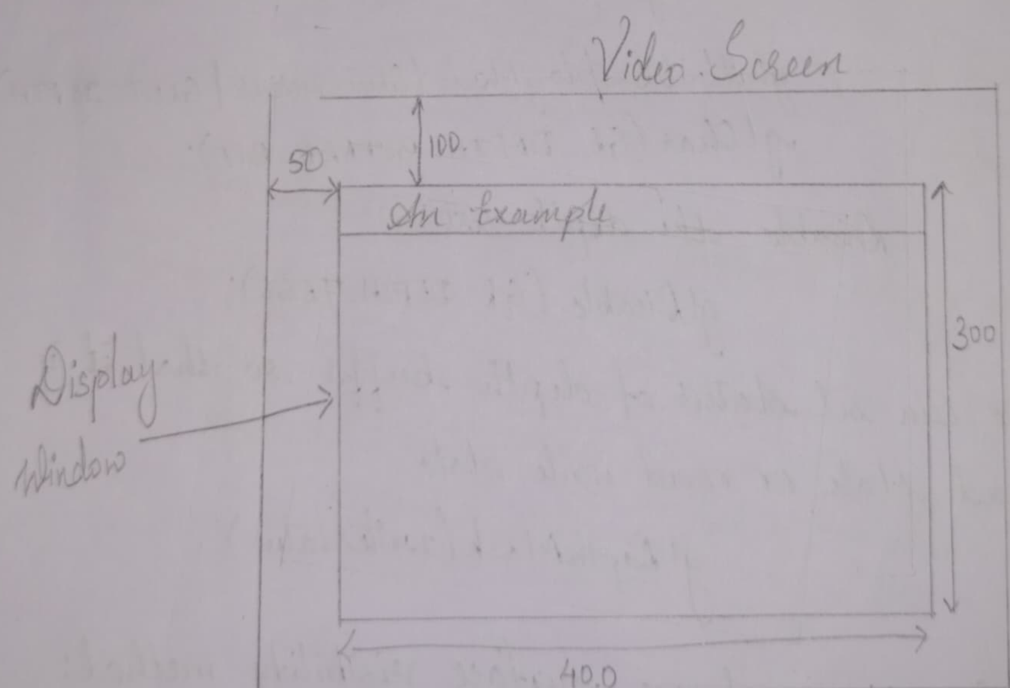
glutInitWindowSize(400,300)

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

glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);

This command specifies that a single 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



6> Explain OpenGL Visibility Detection functions

a. OpenGL polygon culling functions:

Back face removal with functions

```
glEnable (GL_CULL_FACE);
glCullFace (Mode);
```

Mode can be GL_BACK, GL_FRONT, GL_FRONT_AND_BACK.

Disable with

```
glDisable (GL_CULLFACE);
```

b) OpenGL Depth Buffer functions:

To use OpenGL depth buffer visibility detection function, we need to modify GLUT initialization function.

```
glutInitDisplayMode (GLUT_SINGLE| GLUT_DEPTH);
glClear (GL_DEPTH_BUFFER_BIT);
```

Disable the depth_test:

```
glDisable (GL_DEPTH_TEST);
```

we can set status of depth buffer so that it is only in read state or read write state.

```
glDepthMask(writestatus)
```

c) OpenGL wireframe surface visibility method:

A wireframe display can be obtained in OpenGL by requesting that only its edges are generated.

```
glPolygonModel (GL_FRONT_AND_BACK, GL_LINE);
```

d). OpenGL. Depth. CURING function:

It is used to vary the brightness of an object as a function of its distance from viewing position with

glEnable(GL_FOG);
glfogi(GL_FOG_MODE, GL_LINEA);

It applies to depth f^n $d_{min} = 0.0$ and $d_{max} = 1.0$ and set different values for d_{min} and d_{max}

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

7) Write special cases that we discussed with respect to perspective projection transformation coordinates.

$$x_p = x\left[\frac{Z_{prp} - Z_{vp}}{Z_{prp} - Z}\right] + x_{prp}\left[\frac{Z_{vp} - Z}{Z_{prp} - Z}\right]$$

$$y_p = y\left[\frac{Z_{prp} - Z_{vp}}{Z_{prp} - Z}\right] + y_{prp}\left[\frac{Z_{vp} - Z}{Z_{prp} - Z}\right]$$

Cases:

1) Projection reference point is limited along z-view axis

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

$$x_p = x\left[\frac{Z_{prp} - Z_{vp}}{Z_{prp} - Z}\right] \qquad y_p = y\left[\frac{Z_{prp} - Z_{vp}}{Z_{prp} - Z}\right]$$

2. When projection reference point is at coordinate origin

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

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

3. If view plane is uv plane and no restriction on placement of projection reference point.

$z_{vp} = 0$.

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

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

4. If uv plane is on projection reference point on Z-view axis

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

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

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

8) Explain Bezier Curve equation along with its properties

* Developed by French Engineer, Pierre Bezier for use in design of Renault automobile bodies.

* It has number of properties that make them highly useful for curve and surface design

* It can be filled to any number of control points.

Equations.

$P_k = (x_k, y_k, z_k)$ P_k = general control point positions

P_u = position vector that describes the path

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

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

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

Properties:

* Basic functions are real

* Degree of polynomial is one less than number of control point

* Curve generally follows shape of defining polygon.

* It connects first & last control points:

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

* Curve lies within convex null of control points.

9) Explain normalization transformation for an orthogonal projection.

We assume that orthogonal projection view volume 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 $(1, 1, 1)$

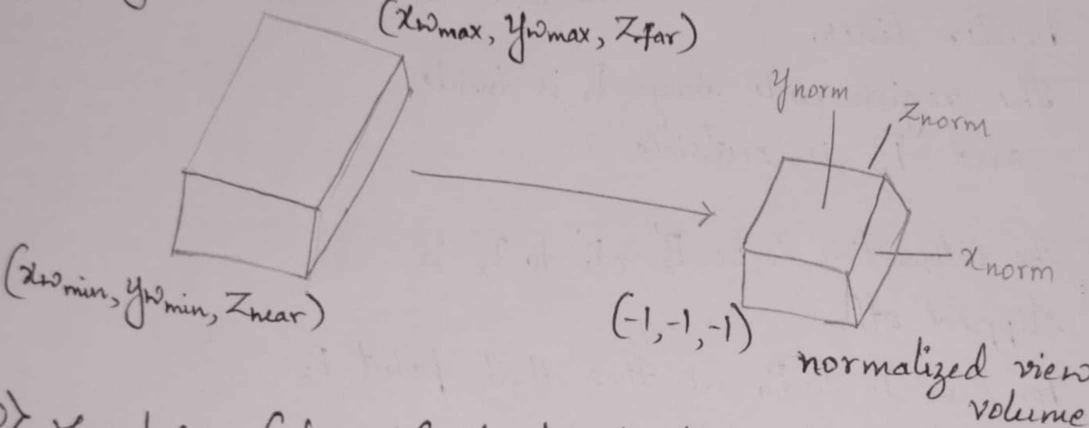
Transforming the rectangular-parallelopiped view volume to a normalized cube is similar to the method for converting the clipping window into the normalized symmetric square.

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

The normalization transformation for the orthogonal view volume is:

$$M_{ortho,norm} = \begin{bmatrix} \frac{2}{xw_{max}-xw_{min}} & 0 & 0 & -\frac{xw_{max}+xw_{min}}{xw_{max}-xw_{min}} \\ 0 & \frac{2}{yw_{max}-yw_{min}} & 0 & -\frac{yw_{max}+yw_{min}}{yw_{max}-yw_{min}} \\ 0 & 0 & \frac{-2}{z_{near}-z_{far}} & \frac{z_{near}+z_{far}}{z_{near}-z_{far}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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 point is inside or outside of one of the clipping window boundaries

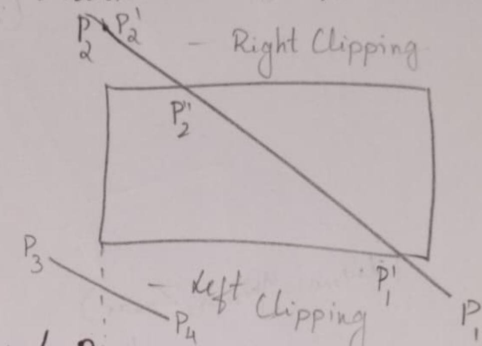
1001	1000	1010
0001	Clipping Window 0000	0010
0101	0100	0110

Once we have established region codes for all line endpoints, we can quickly determine which line are completely within Clipping window & 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.

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



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 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' 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 & can be eliminated. To determine a boundary intersection for a line equation the y-coordinate of the intersection point with vertical clipping border 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 coordinate is

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