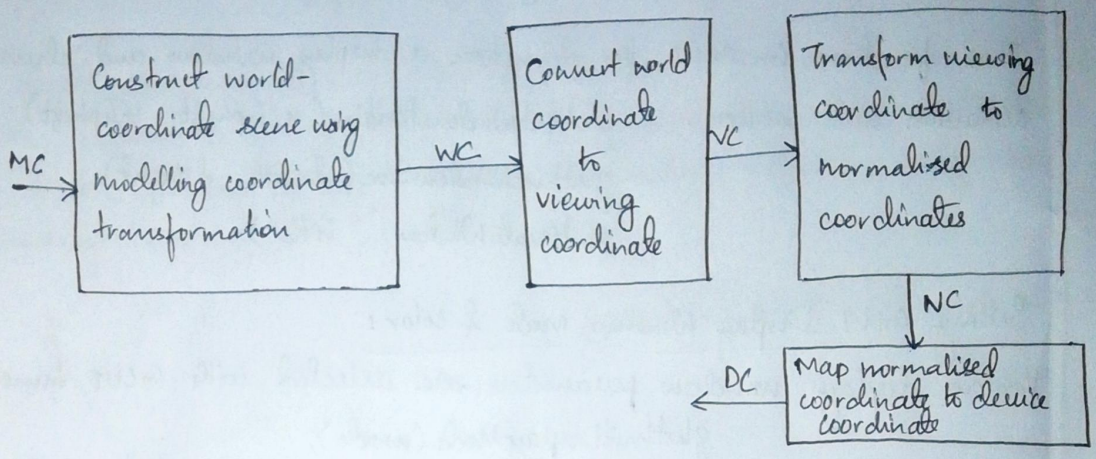
CGV ASSIGNMENT

LAVENIA VIJAY KUMAR
1BY20CS095

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



* 2D Viewing functions:

We can use these two dimensional routines, along with open-gl 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 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 Function:

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

```
gluOrtho2D (xwmin, xwmax, ywmin, ywmax);
```

OpenGL Viewport Function :

glViewPort(xvmin, yvmin, VpWidth, VpHeight);

Create a GLut Display window

glutInit(&argc, argv);

Three functions in GLUT for definition a display window and choosing its dimension and position.

glutInitWindowPosition(xTopleft, yTopleft)
glutInitWindowSize(dWidth, dHeight)
glutCreateWindow("Title");

Setting 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)

2) Build Phong lighting Model with equations

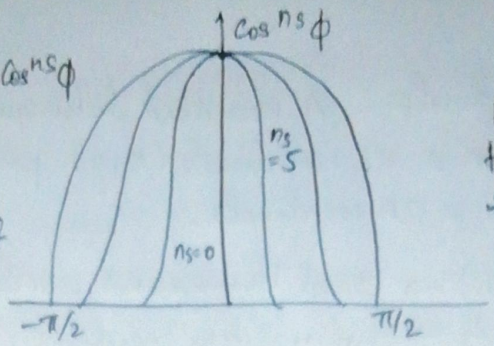
Phong reflection is an empirical model of local illumination. It describes the way a surface reflects light as a combination of the diffuse reflection of rough surfaces with the specular reflection of shiny surface.

It is based on Phong's informal observation that shiny surfaces have small intense specular highlights while dull surfaces have large highlights that fall more gradually

I_l, specular $= w(\theta) I_l \cos^{ns}\phi$

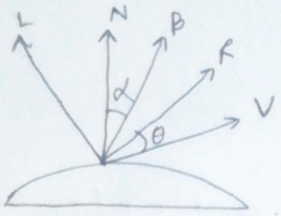
$0 \le w(\theta) \le 1$

is called specular reflection coefficient



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

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 coefficient is nearly constant K_s



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

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

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

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

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

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

The three basic 2D transformations are translation, rotation and scaling

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

P' & P represents column vectors

Homogenous Coordinates : A standard technique to expand the matrix representation for a 2D coordinate (x, y) position to a 3-element representation for a 2D coordinates.

(x_h, y_h, h) → called Homogenous coordinate

h → homogenous coordinates

(x, y) is converted into new coordinate values

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

Translation :

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

This translation operation can be written as $P' = T(t_x, t_y) P$

Rotation :

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

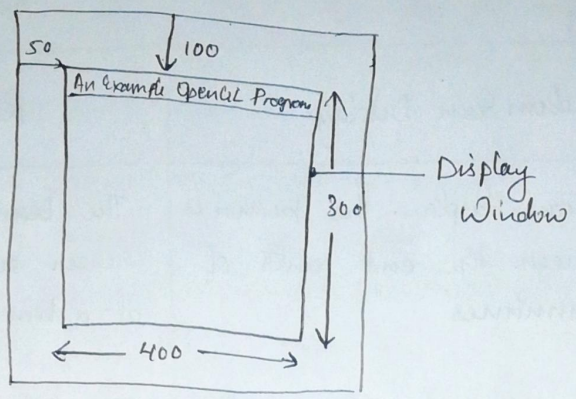
Scaling Matrix :

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

4) Difference between raster scan and random scan display

Random Scan Display	Raster Scan Display
* In vector scan display the beam is moved between the end points of graphics primitives	The beam is moved all once the screen once. the screen one scanline at a time, from top bottom then back to top
* Vector display flickers when the numbers of primitives in the buffer becomes too large	. The refresh process is independent of complexity of image
* Scan conversion is not required	Graphics primitives are specified in terms of their endpoints and must be scan converted into their corresponding pixel in the frame buffers
* Scan conversion is not required	Real time dynamics is far more computational and require separate scan conversion hardware
* Vector display derives a continuous and smooth lines	Can display mathematically smooth lines polygons and boundaries of curved primitives only by approximating them with pixels on raster grid.
* Cost is more	Cost is low
* Vector display only draws lines and characters	Has ability to display areas filled with solid color 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.
  glutCreateWindow("An Example OpenGL program");

* The following function call the line-segment description to the display function
  glutDisplayFunc(lineSegment)

* glutMainLoop()
  This function must be the last one in our program. It displays the initial graphics and puts the program into an infinite loop that checks the input from devices such as mouse & keyboard.

* glutInitWindowPosition(80, 100)
  Specifies that the upper-left corner of the display window should be placed 80 pixel to the right of the left edge of the screen and 100 pixels down from top edge of screen

* glutInitWindowSize(400, 300)
  Function is used to set the initial pixel width and height of display window

* glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
  Command specifies that single refresh buffer is to be used for display window and connect to use color mode which uses red, green, blue components to select values

6) Explain OpenGL Visibility Detection Functions

a) OpenGL Polygon - Culling Functions

Back face removal is accomplished with functions

```
glEnable(GL_CULL_FACE)
glCullFace(mode);
```

where mode is assigned the value GL_BACK, GL_FRONT, GL_FRONT_AND_BACK

By default, parameter mode in glCullFace function has value

Culling routine is turned off with glDisable(GL_CULL_FACE)

b) OpenGL Depth-Buffer Functions

We first need to modify the GL utility Toolkit (GLUT) initialization function for display mode to include a request for depth buffer, as well as for the refresh buffer.

```
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH)
```

Depth buffer values can be initialized with

```
glClear(GL_DEPTH_BUFFER_BIT)
```

These routines are activated with following functions

```
glEnable(GL_DEPTH_TEST)
```

We deactivate these depth buffer routines with glDisable(GL_DEPTH_TEST)

We can also apply depth-buffer testing using some other initial value

glClearDepth(maxDepth) can be set b/w 0 & 1

We can adjust normalization values with:

```
glDepthRange(nearNormDepth, farNormDepth);
```

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

```
glDepthFunc(test condn)
```

We can set status of buffer using glDepthMask(write status)

so is in read-only state or read-write state.

c) OpenGL wire-frame surface Visibility Methods

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)

This displays both visible and hidden flags.

d) OpenGL - DEPTH - Culling Function

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

glEnable (GL_FOG)

glFog (GL_FOG_MODE, GL_LINEAR)

Applies linear depth function to object colors using dmin = 0.0 and dmax = 1.0 and we can set different values

glFogf (GL_FOG_START, minDepth)

glFogf (GL_FOG_END, maxDepth)

7) Write the special cases that we discussed with respect to Perspective projection transformation coordinates

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

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

Special cases:

i) $z_{prp} = y_{prp} = 0$

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

Projection reference point is limited to positions along z_{view} axis

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

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

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

Projection reference point is fixed at coordinate 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 projection reference point.

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

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

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

we get (4) 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

Developed by French Engineer Pierre Bezier for use in design of Renault
Beziers 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 positions

P_u = the position vector which describes the path of appropriate Bezier polynomial function P_0 and P_n

$$P(u) = \sum_{k=0}^{n} P_k \, BEZ_{k,n}(u) \qquad 0 \leq u \leq 1$$

$BEZ_{k,n}(u) = C(n,k) u^k (1-u)^{n-k}$ → Berstein polynomial

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

Properties:

Basic functions are real

Degree of polynomial defining the curve is one less than number of defining points

Curve generally follows the shape of defining polygon

Curve connects the first and last control points

thus $P[0] = P_0$

$P[1] = P_n$

Curves lies within the convex hull of control points

9) Explain normalization transformation for an Orthogonal projection

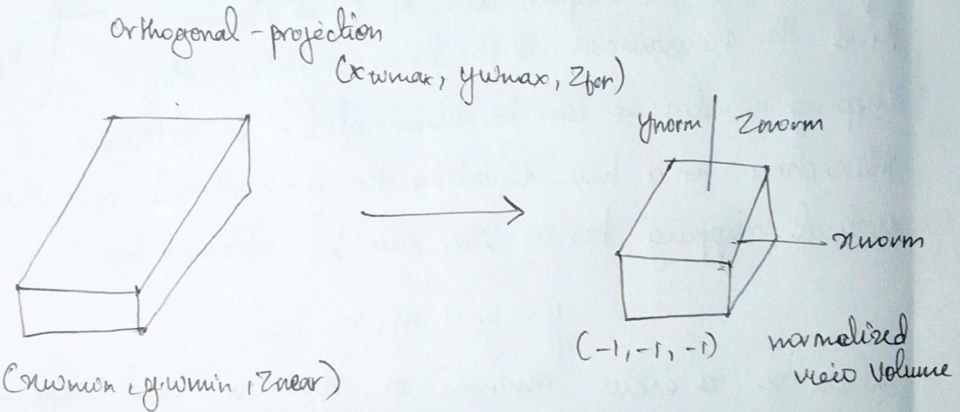
The normalization transformation we assume that the orthogonal projection view volume is to be mapped into the symmetric normalization cube within a left-handed reference frame. Also, 2-coordinate positions for the near and far planes are denoted as z_{near} and z_{far} resp. This position $(x_{min}, y_{min}, z_{near})$ is mapped to be normalized position $(-1, -1, -1)$ and position $(x_{max}, y_{max}, z_{far})$ is mapped to $(1, 1, 1)$

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

The normalization transformation for 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 right by the composite viewing transformation R·T to produce the complete transformation from world coordinates to normalized orthogonal-projection coordinates



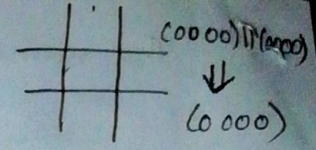
10) Explain Cohen-Sutherland line clipping algorithm

1001	1000	1010
0001	0000 clipping window	0010
0101	0100	0110

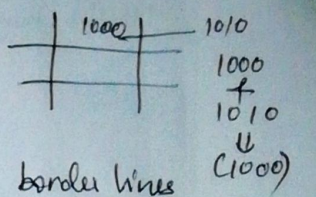
Every line endpoint in a picture is assigned a four digit binary value called a region code and each bit position is used to indicate whether the point is inside or outside of one of clipping window boundries

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

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

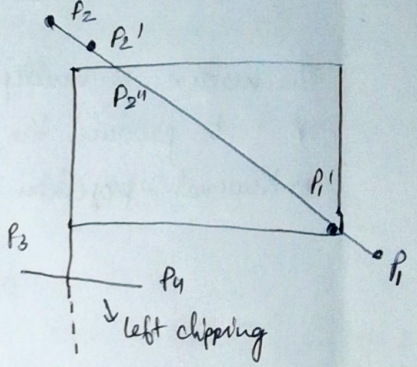


When AND operation b/w 2 end points region codes for a line is true, the line is completely outside clipping window.



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_1 is inside and P_2 is outside The intersection to be P_2' and 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.

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_{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

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