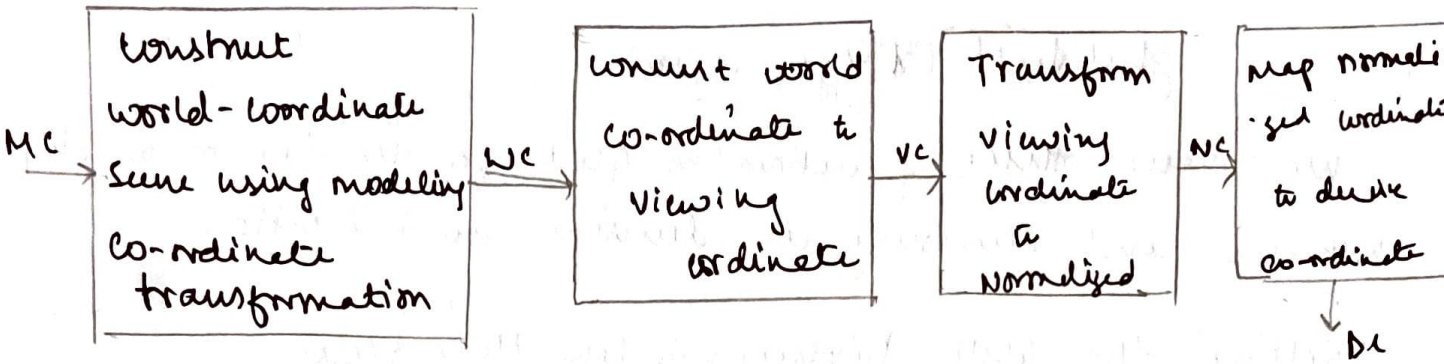
CG - Assignment

1BY20CS075
Jhanavi-H

1)



2D Viewing functions:

We can use these two dimensional routines, along with the opengl viewport fn, 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 2-D clipping window, we can use the opengl utility function

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

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.

→ Setting the GLUT Display-window Mode & Color

Various display window parameters are selected with the GLUT function:

glutInitDisplay(Mode);
glClearIndex(index);
glClearColor(red, green, blue);

→ glut Display window identifier:

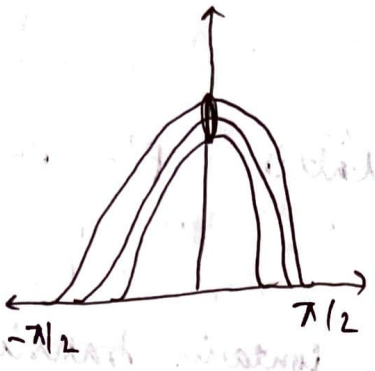
window ID: glutCreateWindow("A display window");

→ Current glut Display window:

glutSetWindow(window ID);

2) 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 intensity.

While dull surfaces have large highlights that falls off more gradually.



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

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

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 material specular reflection co-efficient k_s is nearly constant. k_s

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

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

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

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

3) The three basic 2D transformations are translation and Rotation and Scaling

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

P & P' represent column vectors.

Matrix M_1 → 2×2 array containing multiplicative factors

M_2 → 2 element column matrix containing translation term $\begin{bmatrix} t_x \\ t_y \end{bmatrix}$

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

For rotation and scaling, M_2 contain translational terms associated with pivot point or scaling.

→ Translation :

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

→ Rotation :

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

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

→ Scaling Matrix

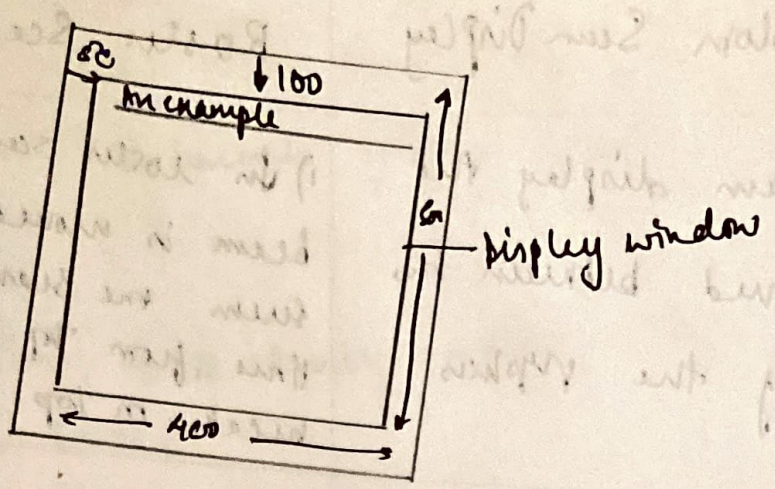
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P' = S(s_x \cdot s_y) P$$

4)

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 over the screen one scan line at a time, from top to bottom and then back to top.
2) Vector display flickers 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.
4) Scan conversion hardware is not required	4) Each primitive must be scan-converted, real-time dynamics
5) Vector display draws a continuous and smooth lines	5) Raster display can display mathematically smooth lines
6) cost is more	6) cost is low
7) Vector display only draws lines and character	7) Raster display has ability to display area filled with solid colors or patterns.

5)



→ We can perform the GLUT initialization with the statement

glutInit(&argc, argv);

→ Next state that a display window is to be created on the screen with a given caption.

→ The following functions call 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.

→ glutInitWindowPosition(50,100);

The following statement specifies that upper left corner of the display window

→ glutInitWindowSize(50,100);

This function is used to set the initial pixel width & height.

6) 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_BACK, GL_FRONT, GL_FRONT_AND_BACK.
- By default, parameter mode is assigned the value ~~GL_BACK;~~ function has the value GL_BACK.

b) OpenGL Depth-Buffer Functions

To use the OpenGL depth-buffer visibility-detection fn 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.

→ Depth buffer values can be initialized with

glClear (GL_DEPTH_BUFFER_BIT)

* By default it is set to 1.0

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)

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);
glFog (GL_FOG_MODE, GL_LINEAR);
```

This applies the linear depth function to object colors using dmin = 0.0 and dmax = 1.0

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

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

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

Special cases:

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

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

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

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

3) $z_{vp} = 0$

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

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

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

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

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

8). Developed by French engineer Pierre Bezier for use in design of Renault auto mobile bodies.

- Bezier have a no. of properties that make them highly useful for curve and surface design.

Equation:

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

P_u = the position vector which describes the path of the an approximate Bezier polynomial fn b/w P_0 & P_n

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

9) The normalization transformation, we assume that the orthogonal projection view volume is to be mapped. z-positions for the near and far planes are denoted as znear and zfar. The position (xmin, ymin, zmin) is mapped to the normalized position (-1, -1, -1) and the position to be (xmax, ymax, zfar) is mapped to (1, 1, 1). The normalization transformation for the orthogonal view volume is

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

10)

1001	1000	1010
0001	0000	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 the clipping window boundaries.

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

The intersection to be

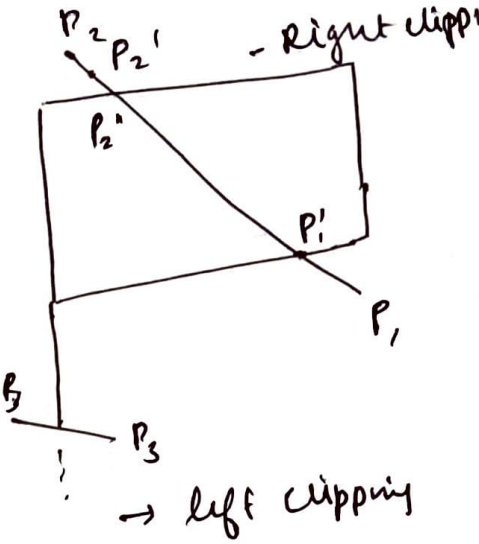
P_2' & P_2' to P_2'' is clipped off

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



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

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

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