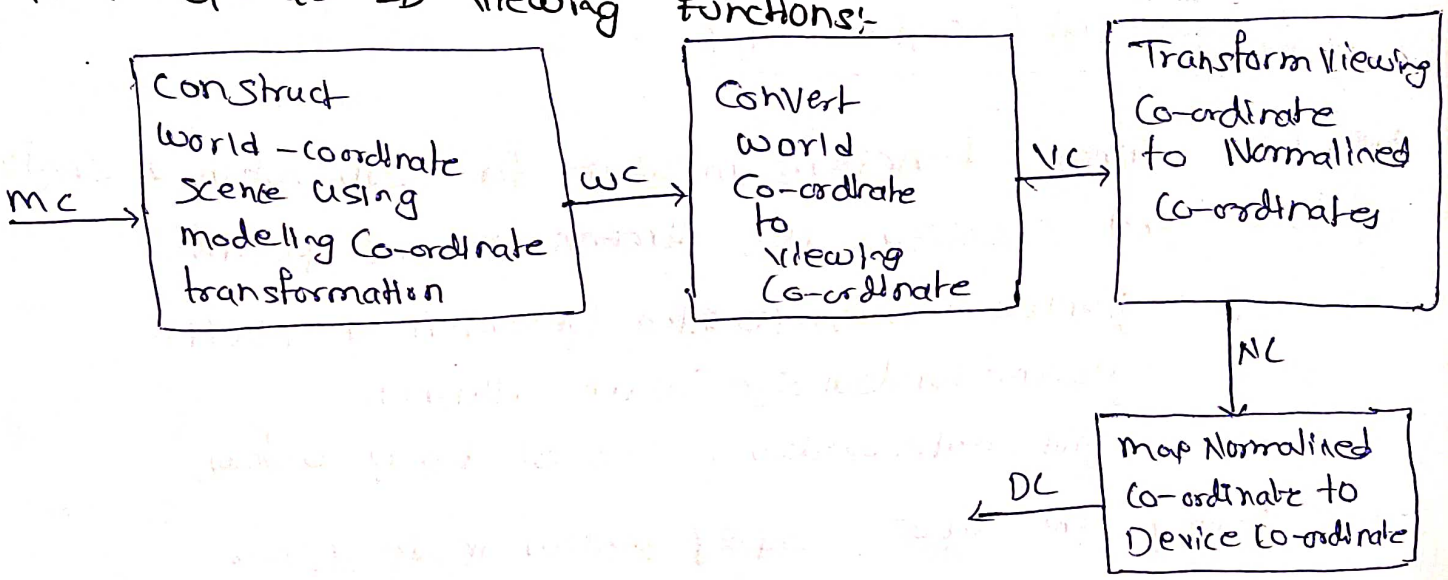
L. Manoj Kumar
1BY21CS410

CG Assignment

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 the opengl 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 transformation from world co-ordinates to screen co-ordinates

```
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.

```
glu ortho2D (xwmin, xwmax, ywmin, xwmax);
```

OpenGL Viewport function

```
glViewport (xwmin, ywmin, 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 (width, dHeight);
glutCreateWindow ("Title of display window");
```

→ Setting the GLUT Display window mode & color

Various display window parameters are selected with the GLUT functions:

```
glutInitDisplayMode (mode);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glClearColor (red, green, blue, alpha);
glClearIndex (index);
```

→ GLUT Display window identifier

```
windowID = glutCreateWindow ("A display window");
```

→ Current GLUT Display window:

```
glutSetWindow (windowID);
```

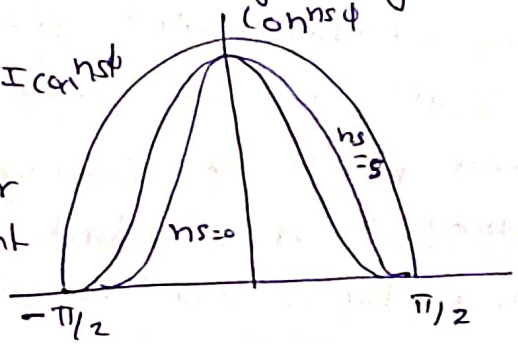
2. Build phong lighting model with equations

→ 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, while dull surfaces that fall off more gradually

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

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

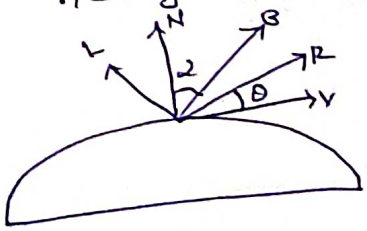
is called specular reflection co-efficient



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 opeque materials specular - reflection co-efficient is nearly constant K_s



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

The normal N way 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

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

→ The three basic 2-D transformations are translation, Rotation. and scaling

$$p' = M_1 + p + m_2$$

p' x p represents column vectors

Matrix m_1 → 2x2 array containing multiplicative factors

m_2 → elements column matrix containing translation term $[x_2 y_2]$

For translation m_1 is identity matrix $p' = p + T$ where $T = m_2$

For rotation and scaling m_2 is contains translational terms associated with pivot point or scaling.

HOMOGENOUS CO-ORDINATES : A standard technique to expand the matrix representation for a 2D co-ordinate (x,y) position to a 3-element representation for a 2-D co-ordinate

(x_h, y_h, h) → called Homogenous co-ordinate

h → homogenous parameter h (non-zero value)

(i.e) (x,y) is converted into new co-ordinate values as (x_h, y_h, h) $x = \frac{x_h}{h}$, $y = \frac{y_h}{h}$ $x_h = x \cdot h$, $y_h = y \cdot h$

Translatation

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

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

3x3 translation matrix

Rotation

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

Scaling Matrix

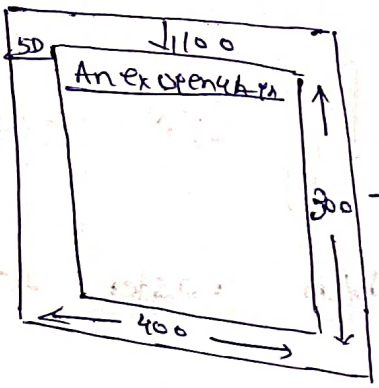
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow P' = S(S_x \cdot S_y) \cdot P$$

4. Outline the difference b/w raster scan displays and random scan displays

→

Random scan display	Roster scan Display
* In vector scan display the beam is moved between the end points of the graphics	* In roster scan display the beam is moved all once the screen one scanline at a time from top bottom and then break to top
* Vector display flickers when the numbers of primitive in the buffer becomes too large	* In raster display the refresh process is independent of the complexity of the image
* Scan conversion is not required	* Graphics primitive are specified in terms of their endpoints and must be scan converted into their corresponding pixel in the frame buffers
* Scan conversion hardware is not required.	* Because each primitive must be scan converted real time dynamics is for more computation and required separate scan conversion hardware
* Vector display derive a continuous and smooth line	* Cost is low
* cost is more	* Roster display has ability to display areas filled with solid colours or patterns
* vector display only draws lines and characters.	
* Vector display flickers when the numbers of primitives in the buffer becomes too large.	
* Scan conversion is not required	

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, this is accomplished with the function
→ glutCreateWindow ("An example opengl program");
where the sigle argument for this function can be any character string.

* The following function call the line segment description to the display window
→ glutWindowFunc (lineSegment);

* glutMainLoop ();
this function must be the last one in our program. It displays the initial graphics and puts the progoam 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 cover of the display window should be placed 50 pixel to the right of the left edge of the screen and 100 pixels down from the top edge of the screen.

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

* glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
the command specifies that a single refresh 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 visibility detection Functions?

→ 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 in glCullFace function has the value GL_BACK.
* The culling routine is turned off with glDisable (GL_CULL_FACE)

b) openGL Depth Buffer Functions

To use the openGL depth-buffer visibility-detection Function, 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.

glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

→ Depth buffer values can initialized with
glClear (GL_DEPTH_BUFFER_BIT)

* By default it is set to 1.0

* These routines are activated with the following functions:

glEnable (GL_DEPTH_TEST);

And we deactivate these depth-buffer routines with gl

glDisable (GL_DEPTHTEST);

* we can also apply depth-buffer testing using some other initial value for the maximum depth

glClearDepth (maxDepth);

* It can be set to any value b/w 0 and 1

* As an option we can adjust normalization values with

glDepthRange (nearNormDepth, farNormDepth);

* we specify a test condition for the depth buffer routines using the following functions

glDEPTHFunc (testCondition);

* we can set the status of the depth buffer so that it is in a read only state or in a read write state gldepth Mask (writeStatus);

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 generated

glPolygonModel (GL_FRONT_AND_BACK_GL_LINE)

But this displays both visible and hidden edges

7 write the special cases that we discussed with respect to perspective projection transformation co-ordinates

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

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

Special cases:-

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

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

we get (1) when the projection reference point is limited to positions along the z view axis.

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

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

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

we get (2) when the projection reference point is fixed at co-ordinate origin

3) $z_{vp} = 0$

$x_p = x\left(\frac{z_{prp}}{z_{prp}}\right) - x_{prp}\left(\frac{z}{z_{prp} - z}\right) \rightarrow (3)a$

$y_p = y\left(\frac{z_{prp}}{z_{prp}}\right) - y_{prp}\left(\frac{z}{z_{prp} - z}\right) \rightarrow (3)b$

we get 3a + 3b if the view plane is the uv plane & there are no restrictions on the placement of the projection reference point.

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

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

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

We get (14) with the uv plane as the view plane + the projection reference point on the zview, axis.

8. Explain Bezier Curve equation along with its properties

→ * Developed by French engineer Pierre Bezier for use in design of Renault automobile bodies

* Bezier 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 an approximate Bezier polynomial function between P_0 and P_n

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

$BEZ_{k,n}(u) = C(n,k) u^k (1-u)^{n-k}$ is the Bernstein 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 $P(0) = P_0$

$P(1) = P_n$

* Curve lies within the convex null of the 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, 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 $(1, 1, 1)$.

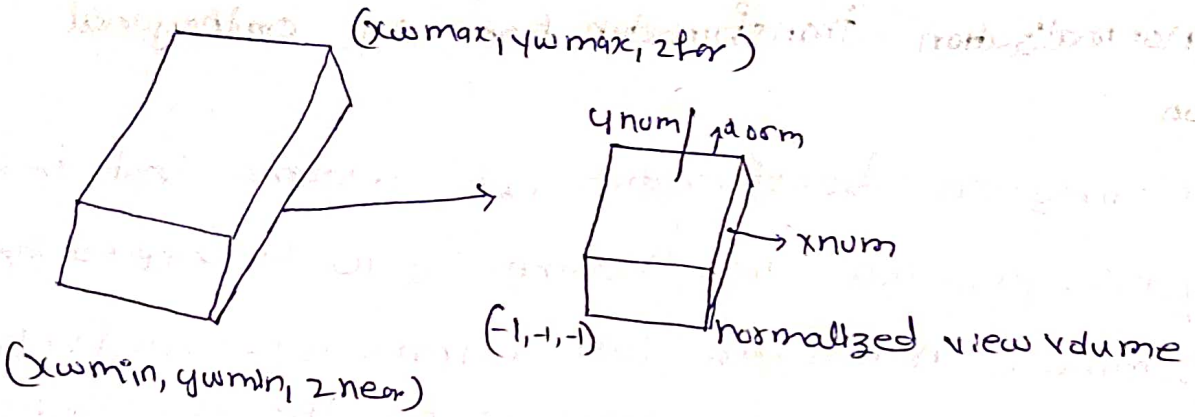
+ transforming the rectangular-parallelpiped view plane to a normalized cube is similar to the method for converting the clipping window into the normalized symmetric square

marthe, 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 the right by the composite viewing transformation R.T to produce the complete transformation from world co-ordinate to normalized ortho-gonal-projection co-ordinate

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 boundary

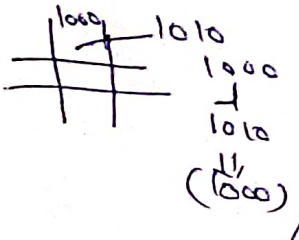
1001	1000	1010
0001	0000 Clipping window	0010
0101	0100	0110

* Once we have established region codes for all line endpoints we can quickly determine which line are completly within clipwindow & 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

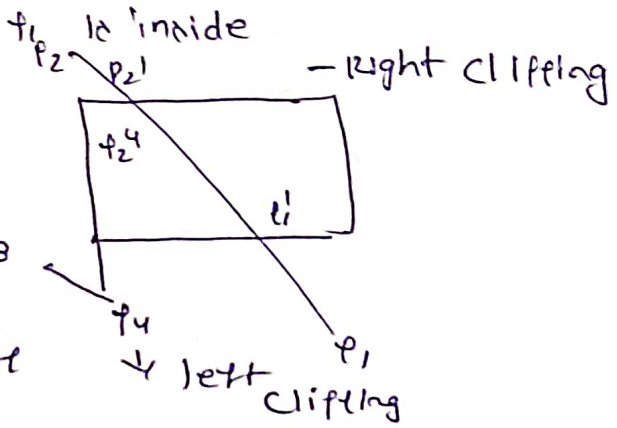
(0000) || (0000)
⇓
(0000)

* when AND operation between 2 end points region codes for a line is true, the line is completly outside the clipping window



* lines that cannot be identified as being completely inside (or) completly outside a clipping window by the region codes tests are next checked for intersection with window border line.

* The region codes 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 are find the remainder of the line is below the clipping window & can be eliminated. To determine a bundary intersection for a line equation the y-co-ordinate 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 co-ordinate is $x = x_0 + \left(\frac{y - y_0}{m}\right)$