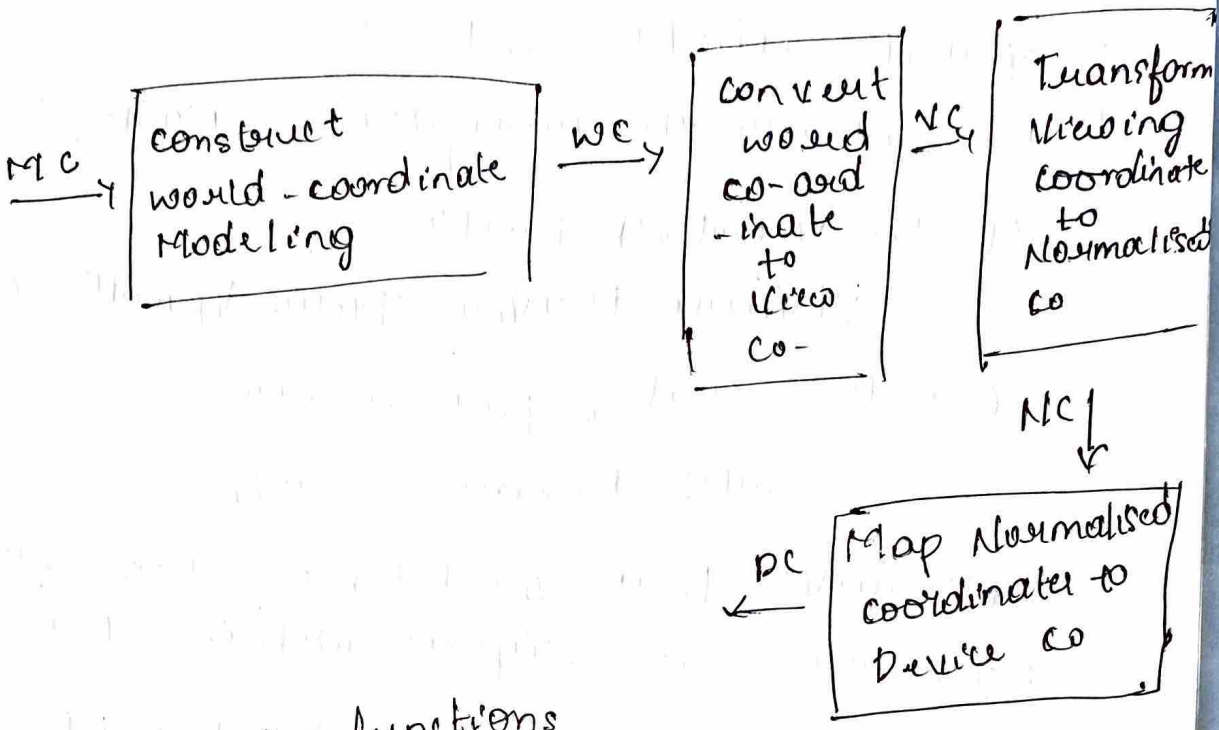
Web 2.0 → Web 3.0

2) ~~Non-F~~ NFT? diff b/w ~~to~~ Fungible & non fungible (Honelia)

CG Assignment

Nitheesh M S
1BY20CS128
6th 'B' CSE

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 funct,

OpenGL Projection Mode

Before we select a clipping window and a viewport in OpenGL, we need to establish appropriate mode for constructing the matrix to transform from world coo. to screen coordinates.

glMatrixMode(GL_PROJECTION)

→ OpenGL Wire-Frame Surface Visibility Method

A wire-frame display of a standard graphics object can be obtained in OpenGL.

diff b/w ~~to~~ Functional & nonfunctional ...

glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)

→ GLU clipping window, we can use the openGL utility function

gluOrtho2D(xmin, xmax, ymin, ymax);

openGL ViewPort Function

glViewPort(xmin, ymin, Vp width, Vp height);

Create a GLut Display window

glutInit(&argc, argv);

we have three functions in GLUT for definination a display within the window

→ Setting the GLUT Display-Window Mode & color:

Various display window Parameters are selected with the GLUT function.

glutInitDisplayMode(mode);
glClearColor(red, green, blue, alpha);
glClear(index);

→ GLUT Display-window identifier

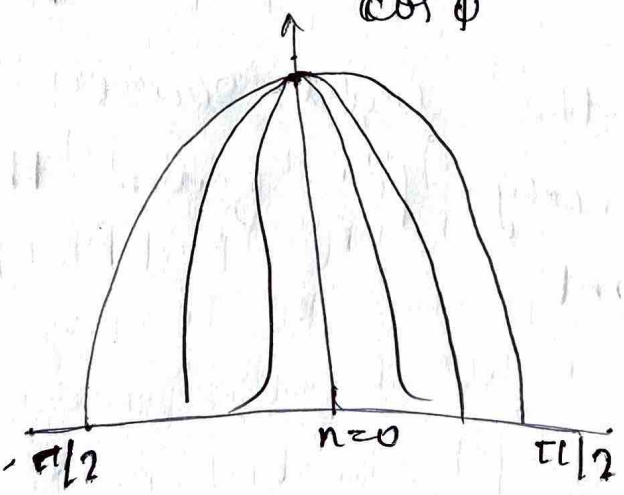
windowID = glutCreateWindow(");

→ Current GLUT Display window

glutSetWindow(windowID);

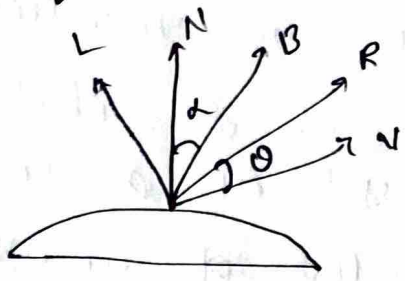
2). Build phong lighting Model with equations?

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 combination of the diffuse with specular reflection of shiny surface. It is based on Phong's informal observation that shiny have small intense specular highlights - while dull surfaces have large highlights that fall of more gradually



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



$$I_{specular} = \begin{cases} k_s I_R [(L-R)^n, \\ 0 \end{cases}$$

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

The Normal N may vary at each point to avoid N computation angle ϕ is replaced by an angle ~~or~~ 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 homogenous coordinates for translation, rotation & scaling via matrix representation.

The three basic 2D transformations are translation, rotation & 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 coordinate

(x_h, y_h, h) → called homogenous coordinate

h → homogenous coordinate

(x,y) is converted into new coordinate values

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

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

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

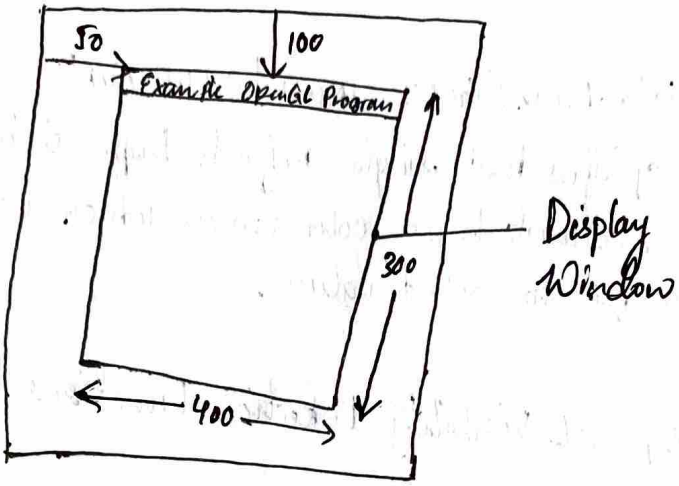
Scaling matrix :

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

Difference between raster scan & 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 over the screen one scan line at a time, from top to bottom then back to top
* Vector display flickers when the number 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 must be scan converted into their corresponding pixel in the frame buffer
* Scan conversion is not required	* Real time dynamics is far more computational & requires separate scan conversion hardware
* Vector display derives a continuous & smooth lines	* Can displays mathematically smooth lines polygons & boundaries of curved primitives only by approximating them with pixels on raster grid
* cost is more	* cost is low
* Vector display only draws lines & characters	* Has ability to display areas filled with solid colors or patterns

5) Demonstrate OpenGL functions for displaying window management using GLUT



- We perform the GLUT initialization with the statement

  glutInit(&argc, argv)

- We can state that a display window is to be created on the screen with a given caption for title bar.

  glutCreateWindow("Example OpenGL program")

- Following function call the line-segment description to the display function

  glutDisplayFunc(lineSegment)

- glutMainLoop()

  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 upper-left corner of display window should be placed 80 pixel to the right of left edge of screen and 100 pixels down from top edge of screen

- glutInitWindowSize (400, 300)
  Function is used to set the initial pixel width/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 ~~for~~ 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 (cmode)
```

mode is assigned the values 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 need to modify the GLUT initialization function for display mode to include a request for depth buffer & refresh buffer

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

Depth buffer values can be initialized with

```
glClear (GL_DEPTH_BUFFER_BIT)
```

Routines are admitted with following functions

```
glEnable (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
glDepthFunc(test condn)

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 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 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.

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

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

Special cases :

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

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

projection reference point is limited to positions along z view axis.

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

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

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

projection reference point is fixed at coordinate origin.

3) $z_{vp} = 0$

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

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

we get 3a & 3b if the view plane is the uv plane & there are no restrictions on the placement of projections 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 Beizer curve equation along with its properties.

→ Developed by French Engineer Piere Bezier for use in design of Renault Bezier have a number of properties that make them highly useful for curve & surface design. They are also easy to implement.

Bezier curve section can be filled to any number of control points.

Equations:

$P_K = (x_K, y_K, z_K)$ P_K = General $(n+1)$ control point positions.

P_q = the position vector which describes the path of appropriate Bezier polynomial func. P_0 & P_n.

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

$$BEZ_{k,n}(u) = C(n,k)\, u^k (1-u)^{n-k} \longrightarrow \text{Bernstein polynomial.}$$

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.

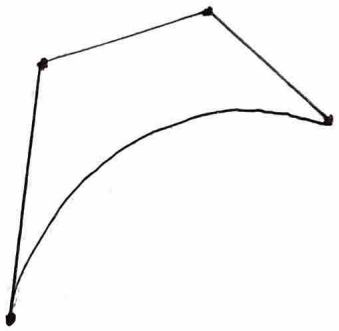
Curve connects the first & last control points thus

$P[0] = P_0$

$P[i] = P_n$

Curves lies within the convex null of control points.

$n=3$



$n=4$

9. Explain normalization transformation for an orthogonal projection

We assume that orthogal projection view volume to be mapped into the symmetric normalization cube within a left-handed reference frame. Also, z-coordinate positions for the near & far places 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 $(1,1,1)$.

The normalization transformation for the orthogonal view volume is:

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

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.

10. Explain Cohen-Sutherland line clipping. Algorithm.

Every line endpoint in a picture is assigned a 4 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.

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, then the line is completely outside

To determine a boundary intersection for a line segment, the y-coordinate of the intersection point can be obtained by.

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

where, x is either x_{0min} or x_{0max} & slope is

$$m = (y_{end} - y_0) / (x_{end} - x_0)$$

for the x-coordinate,

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