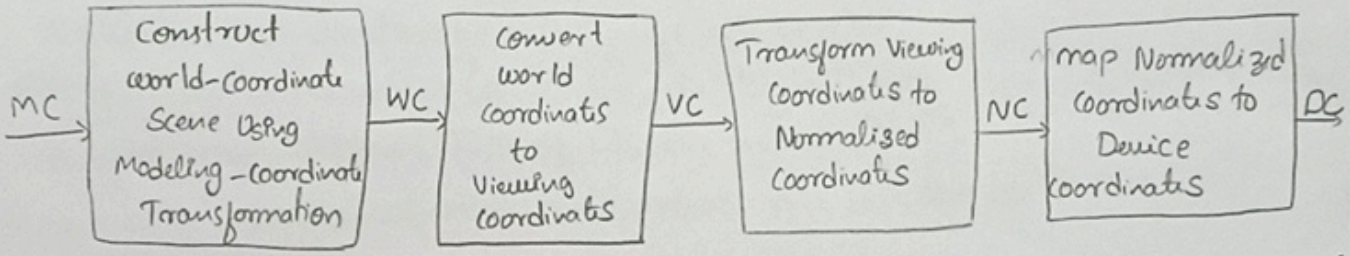
CG Assignment

Name:- N. SANGEETHA
USN:- 1BY20CS120

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

* The mapping of a two-dimensional, world-coordinate scene description to device coordinates is called a two-dimensional viewing transformation.
* This transformation is simply referred to as the window-to-viewport transformation or the windowing transformation.
* we can describe the steps for two-dimension viewing as indicated in fig.



* once a world-coordinate scene has been constructed, we could set up a separate two-dimensional, viewing coordinate reference frame for specifying the clipping window.
* To make the viewing process independent of the requirements of any output device, graphics systems convert object descriptions to normalized coordinates and apply the clipping routines.
* Systems use normalized coordinates in the range from 0 to 1, and others use a normalized range from -1 to 1.
* At the final step of the viewing transformation, the contents of the viewport are transferred to positions within the display window.
* clipping is usually performed in normalized coordinates.
* This allows us to reduce computations by first concatenating the various transformation matrices.

2D viewing functions

we must set the parameters used for the clipping window as part of the projection transformation.

* Function: glMatrixMode(GL_PROJECTION);
we can also set the initialization as
glLoadIdentity();
* To define a two-dimensional clipping window, we can use the GLU fun
gluOrtho2D (xwmin, xwmax, ywmin, ywmax);
* we specify the viewport parameters with the opengl function
glViewport(xvmin, xvmin, vpWidth, vpHeight);
* we need to finitialize GLUT with the following function:
glutInit(&argc, argv);
* we have three functions in GLUT for defining a display window and choosing its dimensions and position:
1. glutInitWindowPosition (xTopLeft, yTopLeft);
2. glutInitWindowSize(dwWidth, dwHeight);
3. glutCreateWindow("Title of Display window");
* Various display-window parameters are selected with the GLUT functions.
1. glutInitDisplayMode(mode);
2. glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
3. glClearColor (red, green, blue, alpha);
4. glClearIndex(index);

2. Build Phong Lighting Model with equations.
* A local illumination model that can be computed rapidly
* There are three components:
-> Ambient
-> Diffuse
-> specular
Ambient lighting -> this produces a uniform ambient lighting that is the same for all objects, and it approximates the global diffuse reflections from the various illuminated surfaces.
The component approximates the indirect lighting by a contant
$I = I_a \times k_a$

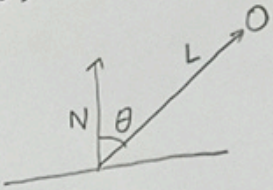
where I_a: ambient light intensity (color)

k_a: ambient reflection coefficient (0~1)

Diffuse reflection → The incident light on the surface is scattered with equal intensity in all directions, independent of the viewing position, Such surface are called ideal diffuse reflection

The brightness at each point is proportional to cos(θ)

The reflected intensity I_{diff} of a point on the surface is



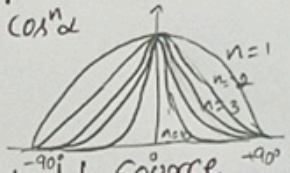
$$I_{diff} = I_p \times k_d \times \cos\theta$$

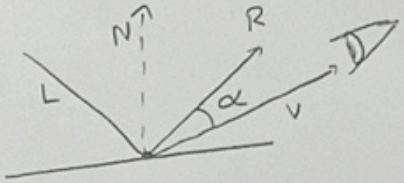
where I_p = intensity of the point light source

k_d = diffuse reflection coefficient (0~1)

This equation can also be written as $I_{diff} = I_p \times k_d \times N \cdot L$

Specular Component → The component describes the specular reflection of smooth surfaces.





$$I = I_p \times K_s \times \cos^n \alpha$$

where I_p = intensity of the point light source

K_s = Specular reflection coefficient (0~1)

n = Shininess

$$I = I_p \times K_s \times (R \cdot V)^n$$

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

Translation → A translation moves all points in an object along the same straight-line path to new positions.

we can write the components:

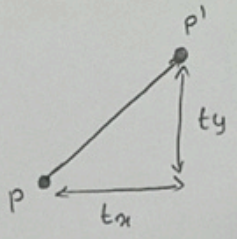
$$P'_x = P_x + t_x$$

$$P'_y = P_y + t_y$$

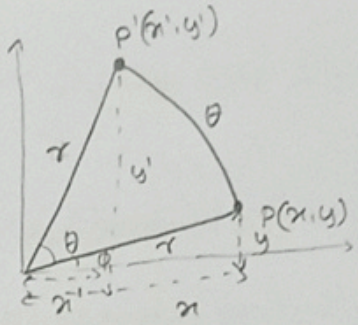
In matrix form:

$$P' = P + T$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$



Rotation → A Rotation repositions all points in an object along a circular path in the plane centered at the pivot point



Review Trigonometry

$\Rightarrow \cos\phi = x/r$, $\sin\phi = y/r$

$x = r.\cos\phi$, $y = r.\sin\phi$

$x' = x.\cos\theta - y.\sin\theta$

$y' = x\sin\theta - y\cos\theta$

we can write the components:

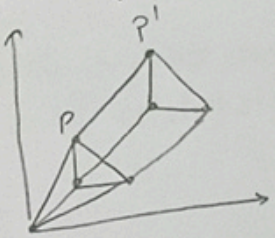
$P'_x = P_x\cos\theta - P_y\sin\theta$

$P'_y = P_x\sin\theta + P_y\cos\theta$

in matrix form

$P' = R \cdot P$ where $R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$

Scaling → Scaling changes the size of an object and involves two scale factor s, S_x & S_y for the x- & y- coordinates respectively.



components are

$P'_x = S_x \cdot P_x$ & $P'_y = S_y \cdot P_y$

in matrix $P' = S \cdot P$ where $S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$

we can write the equations as

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

Rotation $P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

scalling $P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

combining above equations, we can say that

$P' = M_1 * P + M_2$

Co-ordinate (x,y) with homogenous co-ordinate (x_h, y_h, h) where $x = x_h/h$, $y = y_h/h$

Homogenous co-ordinates representation $(h*x_h, h*y_h, h)$

set $h = 1$

$(x, y, 1)$

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

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

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

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

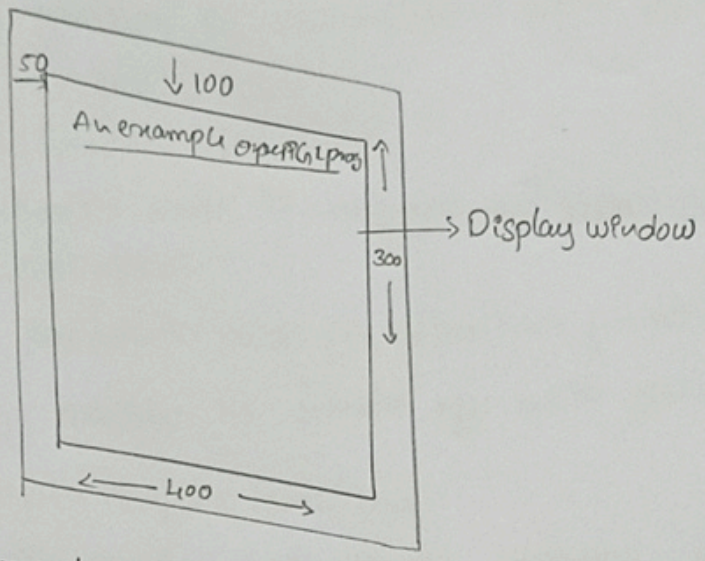
Raster-Scan Displays

→ The electron beam is swept across the screen one row at a time from top to bottom.
→ As it moves across each raw, the beam intensity is turned on and off to create a pattern of illuminated spots.
→ This scanning process is called refreshing. Each complete scanning of a screen is normally called a frame.
→ The refreshing rate, called the frame rate, is normally 60 to 80 frames per second, or described as 60Hz to 80Hz.
→ Picture definition is stored in a memory area called the frame buffer.
→ This frame buffers stores the intensity values for all the screen points. Each screen point is called a pixel
→ property of raster scan is Aspect ratio, which defined as number of pixel columns devided by number of scan lines that can be displayed by the system.

Random-Scan Displays

→ when operated as a random-scan Display unit, a CRT has the electron beam directed only to those points of the screen where a pictures is to be displayed.
→ Pictures are generated as line drawings, with the electron beam tracing out the component lines one after the other.
→ for this reason, random-scan monitors are also referred to as vector displays
→ The component lines of a pictures can be drawn and refreshed by a random-scan system in any specified order.
→ A pen plotter operates in a similar way and it is an example of a random-scan, hard-copy device.
→ Refresh rate on a random-scan system depends on the number of lines to be displayed on that system.

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 single argument for this function can be any character string
* The following function call the line-segment description to the display window
  → 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 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 pixel down from the top edge of the screen
* glutInitWindowSize(400,300);
  the glutInitWindowSize function is used to set the initial pixel width and height of the display window

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_BLACK, 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 be initialized with glClear(GL_DEPTH_BUFFER_BIT)
These routines are activated with the following functions:

glEnable(GL_DEPTH_TEST);

c] openGL wire-Frame Surface visibility method.

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-Cueing function

we can vary the brightness of an object as a function of its distance from the viewing position with glEnable(GL_FOG);
glFogi(GL_FOG_MODE, GL_LINEAR);

This applies the linear depth function to object colors using dmin=0.0 dmax = 1.0 we can set different values for dmin and dmax with the following

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

7. write the special cases that we discussed with respect to perspective projection transformation.

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

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

Special cases

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

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

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

2. $(x_{prp}, y_{prp}, z_{prp}) \cdot (0,0,0)$

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

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

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

3. $z_{vp} = 0$

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

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

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

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

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

we get (4) with the uv plane as the view plane & the projection references 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 filled to any number of control points.

equation:

$P_k = (x_k, y_k, z_k)$ P_k = General (n+1) control point positions

P_v = the position vector which describes the path of an approximate Bezier polynomial function between P_0 and P_n

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

$BEZ_{k,n}(v) = C(n,k)\, v^k (1-v)^{n-k}$ 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 no. 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 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 znear and zfar.

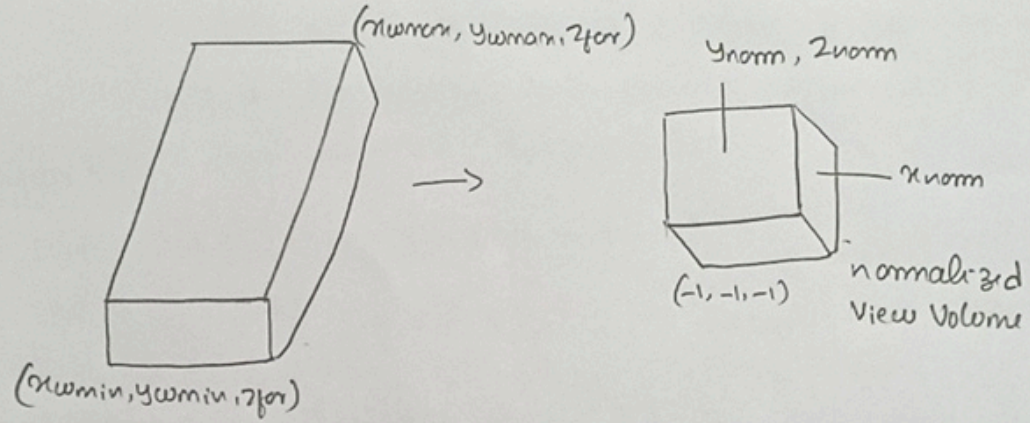
respectively this position $(x_{min}, y_{min}, z_{near})$ is mapped to the normalized position $(-1,-1,-1)$ and position $(x_{max}, y_{max}, z_{far})$ is mapped to $(1,1,1)$

Transforming the rectangular parallel piped view volume to a normalized cube is similar to the method for converting the clipping window into the normalized Symmetric square.

The normalization transformation for the orthogonal view volume is

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

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



10] explain cohen-sutherland line clipping algorithm

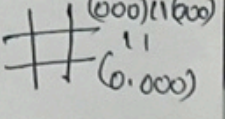
every line end point in a picture is assigned a four digit binary value. called a region code and each bit position is used to indicate whether the point is inside or outside of one of the clipping window boundaries.

1001	1000	1010
0001	0000	0010
0101	0100	0110

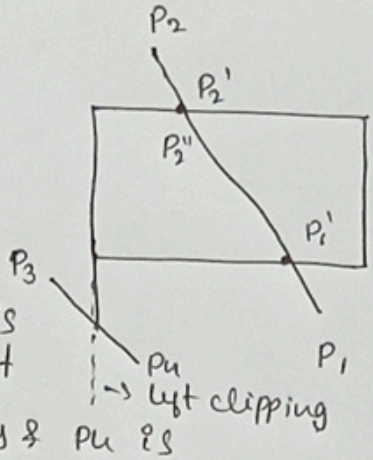
once we have established region codes for all line endpoints. we can quickly determine which line are completely within clip window & which are clearly outside.

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

when AND operation between 2 end points 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 tests are next checked for intersection with border lines.



The region codes says P1 is inside and P2 is outside

The intersection to be P_2'' & P1 to P_2'' is clipped off for line P3 to P4 we find that Point P3 is outside the left boundary & P4 is inside therfore the intersection is P3 & P3 to P3' is clipped off.

By checking the region codes of P3' & P4 we find the remainder of the line is below the clipping window & can be eliminated. To determine a boundary 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 then x coordinate is

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