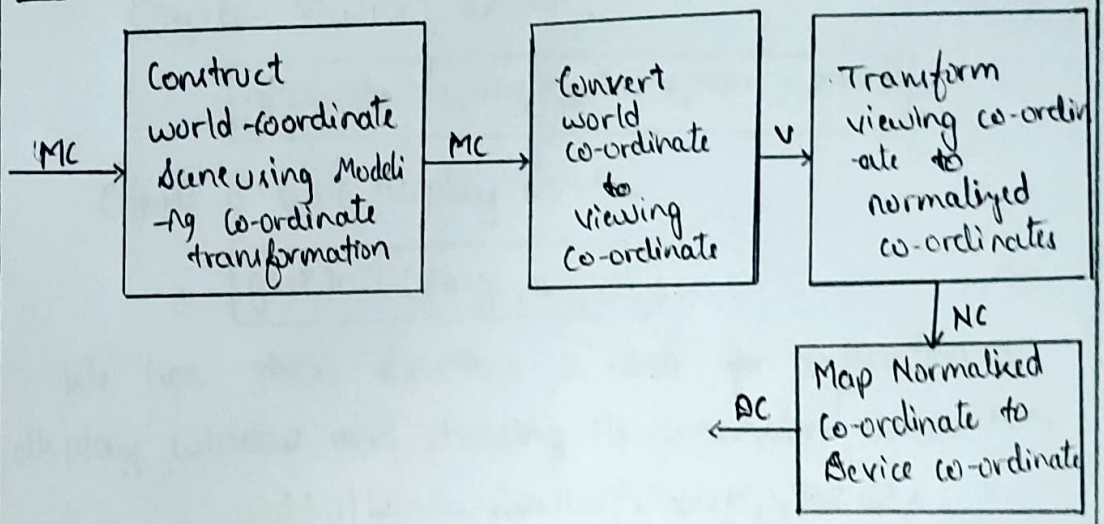
CG Assignment

Name: Kulashekar Alwar. N
USN : 1BY21CS409

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

Ans



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 transform from world co-ordina -tes 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 2D clipping window, we can use the openGL utility function.

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

OpenGL ViewPort function;

```
glViewPort(xvmin, xwmin, yvmin, yvmax);
```

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.

```
glutInitWindowPosition(xTopLeft, yTopLeft);
glutInitWindowPosition(dwidth, dHeight);
glutCreateWindow("Title of display window");
```

-> Setting the GLUT Display-window mode & Colour;

Various display window parameters are selected with the GLUT function.

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

-> GLUT display-window identifier:

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

-> Current GLUT display window:

```
glutSetWindow(window.ID);
```

---

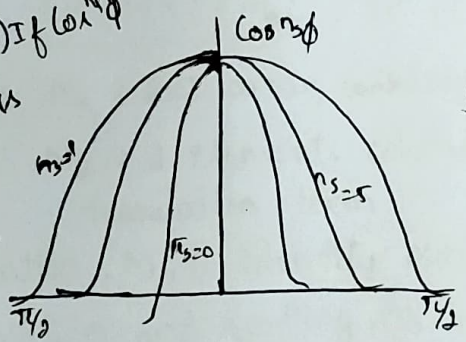
2) Build Phong Lighting model with equations?

Ans

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 unformal observation that shiny surfaces have small intense specular highlights, while dull surfaces have large highlights, while that fall off more gradually.

If specular = $w(\theta) I \cos^{n_s}\phi$

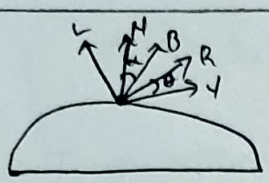
$0 \le w(\theta) \le 1$ is called Specular reflection co-efficient.



Phong model sets the intensity of specular reflection to constant.

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



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

The Normal N may vary at each process point. To avoid N computation angle ϕ is replaced by 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.

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

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

$P' \times P$ represents column vectors

Matrix $M_1 \rightarrow 2 \times 2$ array containing multiplicative factors.

$M_2 \rightarrow 2$ Elements column matrix containing translation term

For translation, M_1 is identity Matrix $P' = P + T$ where $T = M_2$. For rotation and scalling, M_2 is contains translational terms associated with pivot point or scaling.

Homogeneous 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 2D co-ordinates. $(x_h, y_h, h) \rightarrow$ called Homogeneous co-ordinates.

$h \rightarrow$ homogeneous parameter h (non-zero value)

Ie (x,y) is converted into new co-ordinates values.

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

-> Translation:

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

This translation operation can be written as

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

↳ 3x3 translation Matrix.

-> ROTATION:

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

-> Scaling Matrix:

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

4.) Outline the difference between raster scan displays and random scan display.

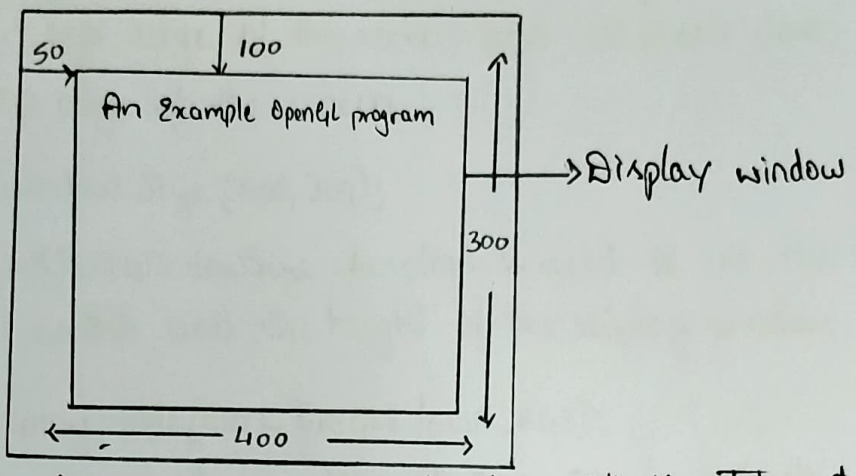
Ans

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 once the screen one scanline at a time from top bottom and then break to top.

Random Scan display	Raster Scan display.
2. Vectors displaying 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 in terms of their endpoints and must be scan connected into their corresponding pixel in the frame buffers.
4. Scan conversion hardware is not required.	4. Because each primitive must be scan-converted real-time dynamics is for more computational and required separate scan conversion hardware.
5. Vector displays derives a continuous and smooth time	5. Raster displays can display mathematically smooth lines, polygons, and boundaries of curved primitives only by approximating them with pixel on the background.
6. Cost is more.	6 Cost is low.
7. Vector display only draws lines and characters	7. Raster display has ability to display areas filled with solid colours or patterns

5.) Demonstrate OpenGL functions for displaying window management using GLUT.

Ans



* 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 pixels down from the top edge of the screen.

* glutInitWindowSize (400, 300);

The glutInitWindowSize function is used to set the initial pixel width and the 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 convert to use the color mode which uses red, green and Blue (RGB) components to select color values.

6.) Explain OpenGL Visibility Detection Functions?

Ans a] OpenGL Polygon - Calling Functions

Backface 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-Buffers 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.

```
glutInitDepth.
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
```

-> Depth buffer values can be initialized with

```
glClear(GL_DEPTH_BUFER_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 glDisable(GL_DEPTH_TEST);

-> we can also apply depth buffers 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, farnomDepth);

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

glDepthRangeFunc(test condition);

-> we can set the states of the depth buffer so that if it

it in a read only state or in a read write state

glDepthMask(write_status);

c] OpenGL wire-frame surface visibility methods.

→ wire-frame displays of a standard graphics object can be obtained in OPENGL by requesting that only i/p edges are to be generated.

glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

But this displays both visible and hidden edges

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 demo 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 co-ordinates.

Ans:

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

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

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

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

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

we get (2) when the projection reference model 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 ~~references~~ 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 z view, axis.

8.) Explain Bezier Curve Equation along with its properties?

Ans → 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 selection can be filled to any number of control points.

Equation:

$P_k = (x_k, y_k, z_k)$ P_k = General ($n+1$) Control points positions

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

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

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

$$P(0) = P_0$$
$$P(1) = P_n$$

* Curves lies within the convex hull of the control points.

---

9) Explain normalization transformation for an Orthogonal projection.

Ans:

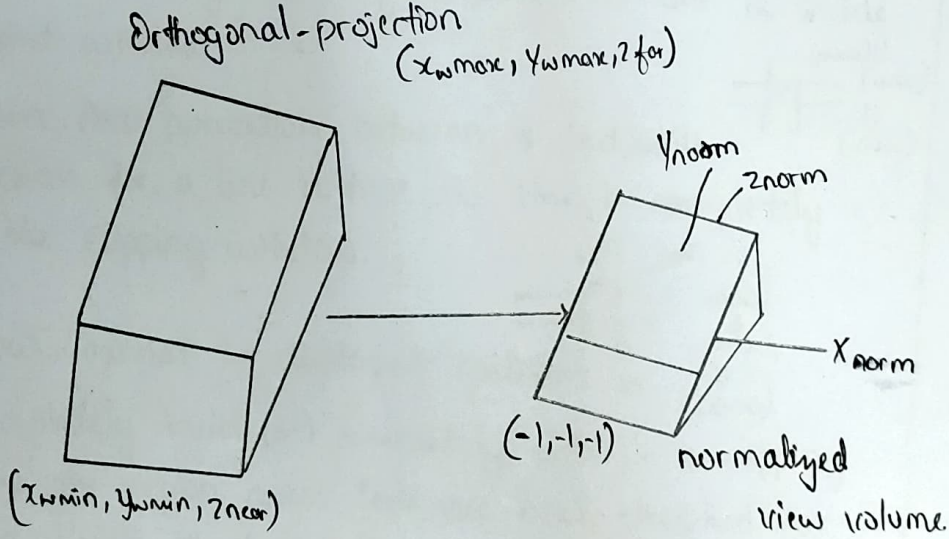
The normalization transformation, we assume that the orthogonal-projection view volume is to be mapped into the symmetric normalization cube within a left-handed reference frame. Also, z-coordinate positions for the near and far planes are denoted as z-near and z-far respectively. this position $(x_{min},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

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

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



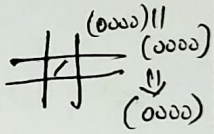
10.) Explain Cohen-Sutherland line clipping algorithm?

Ans! 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 the clipping window boundaries.

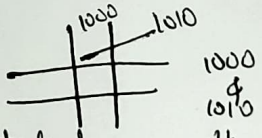
1001	1000	1010
0001	0000 clipping window	0010
0101	0100	0110

Once we have Established region codes for all line endpoints, we can quickly determine which line are 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 clippind window.

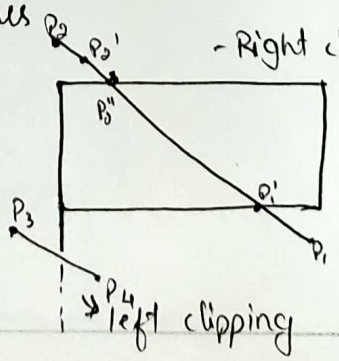


When AND operation between 2 Endpoints regions codes for a line is true, the line is completely outside the clipping window.



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

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. we find the remainder of the line is below the clipping window and can be eliminated. To determine a boundary intersection point for a line equation the y-co-ordinates 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

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