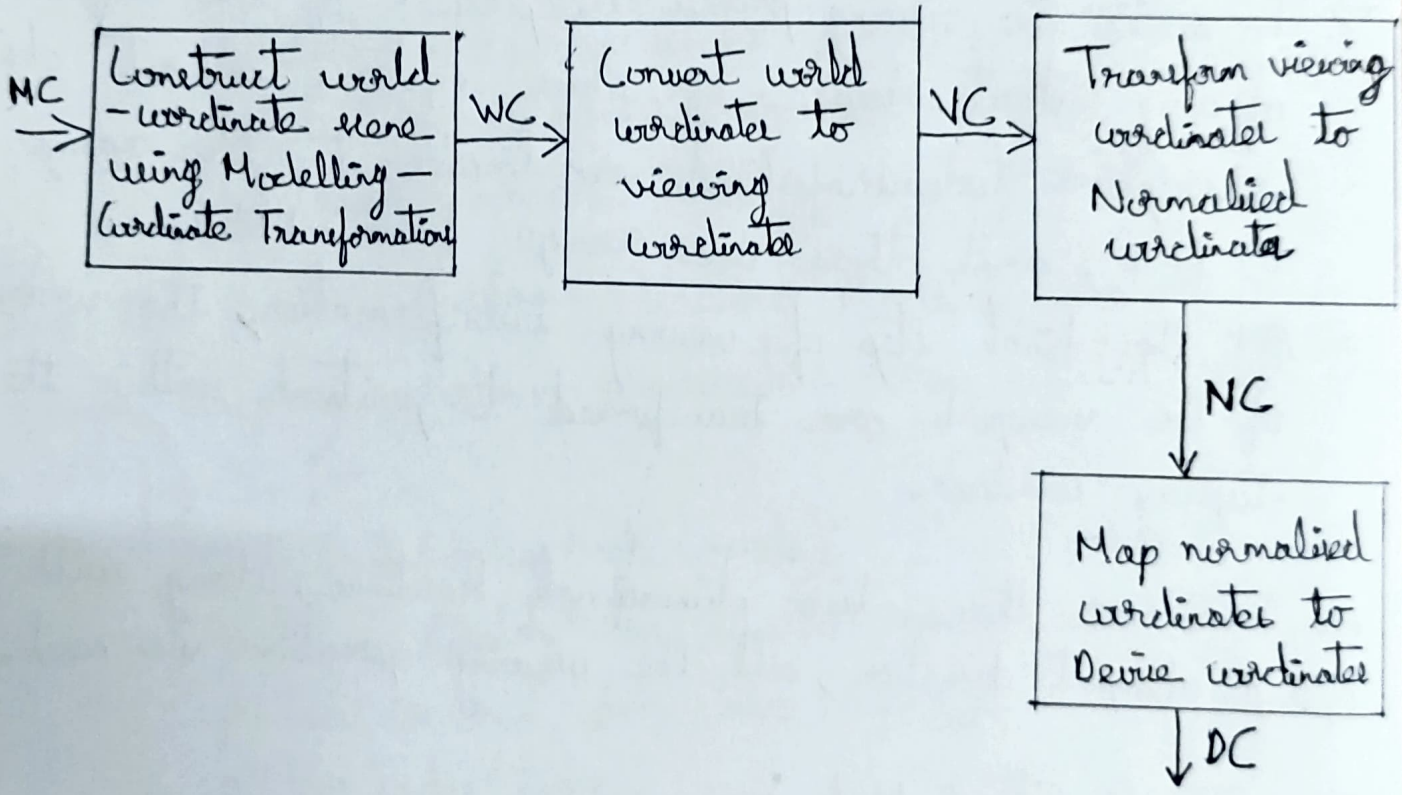
Name :- Jayanth R
USN :- 1BY20CS073
Sem&Sec :- VI 'B'
Dept :- CSE
Course :- Computer Graphics

# Assignment

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

Sol :-




→ A section of a two-dimensional scene that is selected for display is called a clipping window because all parts of the scene outside the selected section are "clipped" off.

→ The mapping of a two-dimensional, world co-ordinate scene description to device co-ordinates is called a two-dimensional viewing transformation.

→ Sometimes this transformation is simply referred to as the window to viewport transformation or window transformation.

→ Once the world-coordinate scene has been constructed we would set up a separate 2D viewing coordinate reference frame for specifying the clipping window. Viewing coordinates for 2D applications are the same as world coordinates.

→ To make the viewing process independent of the requirements of any output device, graphics systems convert object descriptions to normalised coordinates in the range from 0 to 1, and others use range from -1 to 1.

→ At the final step of viewing transformation, the contents of the viewport are transferred to positions within the display-window.

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 viewport in openGL, we need to establish the appropriate mode for constructing the matrix to transform world coordinates to screen coordinates.

glMatrixMode (GL_PROJECTION);

GLU clipping-window Function:-

To define a two-dimensional clipping window, we can use the OpenGL utility function.

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

OpenGL viewport Function:-

```
glViewPort (xvmin, yvmin, vpwidth, vpheight);
```

Creating a GLUT Display Window:-

```
glutInit (&argc, argv);
```

We have three functions in GLUT for defining a display window and choosing its dension and position.

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

Setting the GLUT Display-Window Mode & Color:-

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");
```

Deleting a GLUT Display Window:-

```
glutDestroyWindow (windowID);
```

Current Glut Display Window :-

```
glutSetWindow (windowID);
```

Relocating and Resizing a Glut Display Window :-

```
glutPositionWindow (xNewTopLeft, yNewTopLeft);
glutReshapeWindow (dwNewWidth, dwNewHeight);
glutFullScreen ();
```

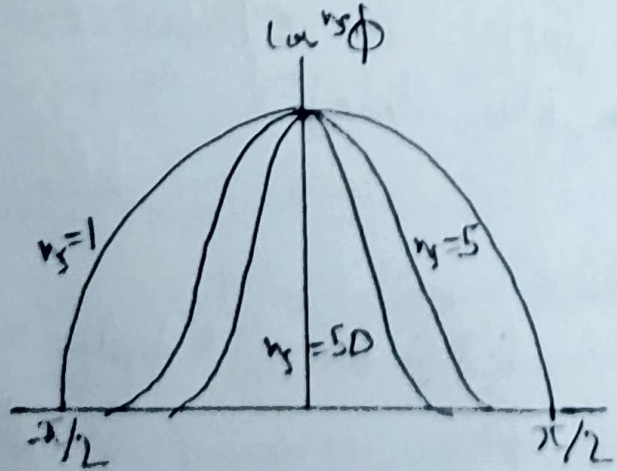
Managing multiple Glut Display Window

```
glutIconifyWindow ();
glutSetWindowTitle ("New Window Name");
```

---

2. Build Phong Lighting Model with equations.

Sol:- Phong reflection is an empirical model of local illumination. It describes the way of a surface reflects light as a combination of the diffuse reflection of rough surface with the specular reflection of shiny surface. It is based on Phong's informal observation that shiny surfaces will have small intense specular highlights, while dull surfaces have large highlights that fall off more gradually.



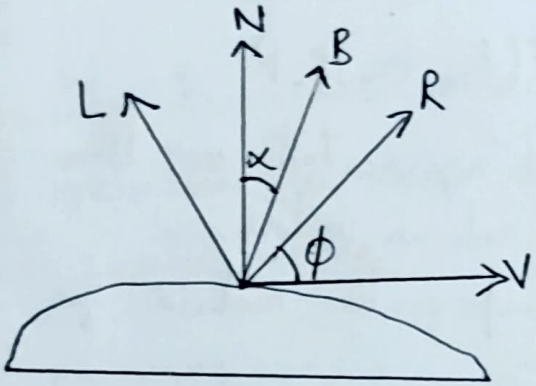

Phong model sets the intensity of specular reflection to $\cos^{n_s}\phi$.

$I_{\lambda, \text{specular}} = W(\theta) I_\ell \cos^n \phi$

$0 \le w(\theta) \le 1$ is called 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 materials, specular-reflection co-efficient is nearly constant $k_s$




$$I_{\text{specular}} = \begin{cases} k_s I_\ell (V \cdot R)^{n_s}, & V \cdot R > 0 \text{ and } N \cdot L > 0 \\ 0.0 & \text{otherwise} \end{cases}$$

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

The normal N may vary at each point. To curiel N computation angle $\phi$ is replaced by an angle $\alpha$ defined by a halfway vector H between L and V.

Efficient computation: $H = \dfrac{L+V}{|L+V|}$

If the light source and viewer are relatively far from the object, $\alpha$ is constant.

H is the direction yielding maximum specular reflection in viewing direction V if the surface normal N would coincide with H.

If V is coplanar with R and L (and hence with N too) $\alpha = \phi/2$.

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

Sol:- *Translation

→ If two successive translation vectors $(t_{1x}, t_{1y})$ and $(t_{2x}, t_{2y})$ are applied to a two dimensional coordinate position P, the final transformation location P' is calculated as

$$P' = T(t_{2x}, t_{2y}).\{T(t_{1x}, t_{1y}).P\}$$
$$= \{T(t_{2x}, t_{2y}).T(t_{1x}, t_{1y})\}.P$$

where P and P' are represented as three-element, homogeneous-coordinate column vectors.

→ Also, the composite transformation matrix for this sequence of translation is

$$\begin{bmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{1x}+t_{2x} \\ 0 & 1 & t_{1y}+t_{2y} \\ 0 & 0 & 1 \end{bmatrix}$$

$$T(t_{2x}, t_{2y}).T(t_{1x}, t_{1y}) = T(t_{1x}+t_{2x}, t_{1y}+t_{2y})$$

* Rotation

→ Two successive rotations applied to a point P produce the transformed position.

$$P' = R(\theta_2).\{R(\theta_1).P\}$$
$$= \{R(\theta_2).R(\theta_1)\}.P$$

where P and P' are represented as three-element, homogeneous-coordinate column vectors.

→ By multiplying the two rotation matrices, we can verify that two successive rotations are additive:

$$R(\theta_2) \cdot R(\theta_1) = R(\theta_1 + \theta_2)$$

→ Final rotated coordinates of a point can be calculated with composite rotation matrix as,

$$P' = R(\theta_1 + \theta_2) \cdot P$$

## *Scaling

→ Concatenating transformation matrices for two successive scaling operations in two dimension produces the following composite scaling matrix.

$$\begin{bmatrix} S_{2x} & 0 & 0 \\ 0 & S_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} S_{1x} & 0 & 0 \\ 0 & S_{1y} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} S_{1x} \cdot S_{2x} & 0 & 0 \\ 0 & S_{1y} \cdot S_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S(S_{2x}, S_{2y}) \cdot S(S_{1x}, S_{1y}) = S(S_{1x} \cdot S_{2x}, S_{1y} \cdot S_{2y})$$

---

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

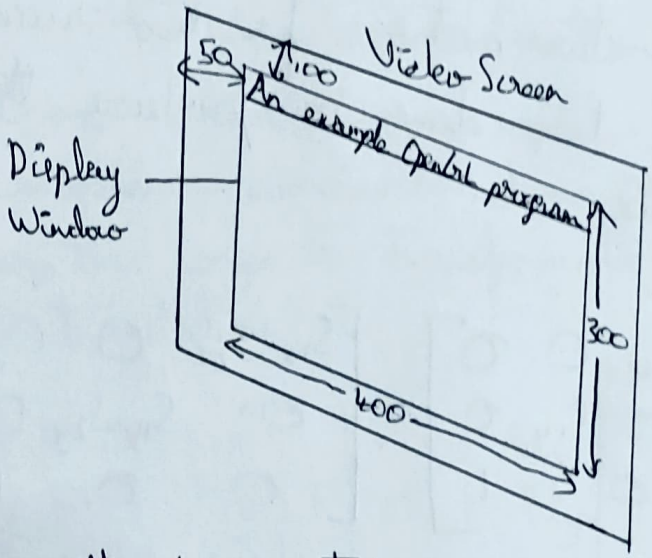
Sol:-

| Random Scan Display | Raster Scan Display |
|---|---|
| In random scan display, the beam is moved between the end point of the graphics primitives. | In raster scan display, the beam is moved all over the screen one scan line at a time |
| Vector display flickers when the number of primitives in the buffer becomes too large. | In raster scan display, the refresh process is independent of the complexity of the image. |
| Scan conversion is not required. | Graphics primitives are all specified in terms of their endpoints. |

| | |
|---|---|
| Scan conversion hardware is not required | Each primitives must be scan converted. |
| Vector display draws a continuous and smooth line | Raster display can display mathematically smooth line, polygon and boundaries. |
| Cost is more | Cost is low |

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

Sol:-




→ 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.
glutCreateWindow("An example OpenGL program");

→ The following function call passes the line-segment description to the display window.
glutDisplayFunc(display);

→ 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 the upper-left corner of the display window should be placed 50 pixels to the right of left edge of the screen and 100 pixels down from the top edge of the screen.

→ glutInitWindowSize(400,300);
Used to set the initial pixel width and height of the display window.

→ glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
Specifies a single refresh buffer is to be used for the display window and that the color mode which uses red, green and blue (RGB) components to select color values.

---

6. Explain OpenGL Visibility Detection Functions.

Sol:- a) OpenGL Polygon-culling Functions

Back-face removal is accomplished with the functions
glEnable(GL_CULL_FACE);
glCullFace(mode);

→ 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 function visibility detection functions, we first need to modify the GL-Utility Toolkit (GLUT) initialization function for 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);
* By default it is set to 1.0
→ These routines are activated with the following function:

glEnable (GL_DEPTH_TEST);

And we deactivate the depth-buffer routines with glDisable (GL_DEPTH_TEST);

→ 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 btw 0.0 and 1.0

→ As an option, we can adjust normalization values with glDepthRange (nearNormDepth, farNormDepth);

*By default, nearNormDepth = 0.0 and farNormDepth = 1.0

→ We can specify a test condition for the depth buffer routine using the following function.

*Parameter: glDepthFunc (testCondition);

* Parameter that can be used are:
GL_LESS, GL_GREATER, GL_EQUAL, GL_NOTEQUAL, GL_LEQUAL, GL_GEQUAL, GL_NEVER and GL_ALWAYS

*Default parameter is GL_LESS

→ We can set the status of depth buffer so that it is in a read-only state or in a read-write state.

glDepthMask (writeStatus);

---

7. Write the special cases that we discussed with respect to Perspective projection transformation coordinates.

Sol:- i) If projection reference point is on $z_{view}$, means $x_{prp} = y_{prp} = 0$

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

ii) Sometimes the projection reference point is fixed at the coordinate origin, and

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

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

iii) If the view plane is uv plane and there are no restrictions on the placement of the projection reference point, then we have

$z_{vp} = 0$:

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

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

iv) With the uv plane as the view plane and the projection reference point on the z view axis, the perspective equations are

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

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

---

8. Explain Bezier curve equation along with its properties.

Sol:- We first consider the general case of $n+1$ control points denoted as $P_k = (x_k, y_k, z_k)$, with $k$ varying from 0 to $n$. These coordinate points are blended to produce the following position vector $P(u)$, which describes the path of an approximating Bezier polynomial function between $p0$ and $pn$.

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

The Bezier blending functions $BEZ_{k,n}(u)$ are the Bernstein polynomials

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

where parameter $C(n,k)$ are binomial coefficients

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

Eqn P(u) represents a set of three parametric equations for the individual curve coordinates:

$$x(u) = \sum_{K=0}^{n} x_K BEZ_{K,n}(u)$$

$$y(u) = \sum_{K=0}^{n} y_K BEZ_{K,n}(u)$$

$$z(u) = \sum_{K=0}^{n} z_K BEZ_{K,n}(u)$$

In most cases, a Bezier curve is a polynomial of a degree that is one less than the designated number of control points.

---

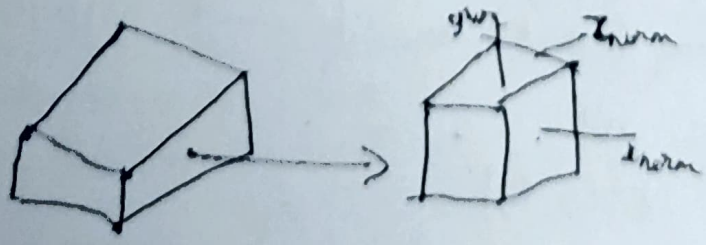
9. Explain normalization transformation for an orthogonal projection.

Sol:- There are 2 approaches to Normalize a 3D object & transformation for an Orthogonal Projection:-

→ Use a unit cube for the normalized view volume, with each of the x, y and z coordinates normalized in the range from 0 to 1.

→ Another normalization transformation approach is to use a symmetric cube, with w-ordinates in the range from −1 to 1.

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



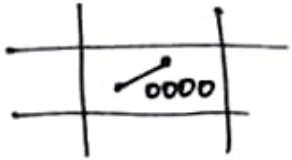
Q. Explain Cohen-Sutherland line clipping algorithm.

Sol:- Every line endpoint in a picture is assigned a four-digit binary value, called a region code.

| | | |
|---|---|---|
| 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 lines are completely contained 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. example:




(0000) || (0000)
↓
(0000) false.

when the AND operation between 2 endpoints region codes for a line is true (not 0000), the line is completely outside the clipping window & can be eliminated. Example:


(1000) & (1010)
↓
1000

By checking region codes of $P_3'$ & $P_4$, we find the remainder of the line is below the clipping window & can be eliminated. To determine a boundary intersection for a line segment, we can use the slope-intercept form of line equation. For a line with endpoint coordinates $(x_0, y_0)$ & $(x_{end}, y_{end})$.

the y co-ordinates of the 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

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