CG Assignment

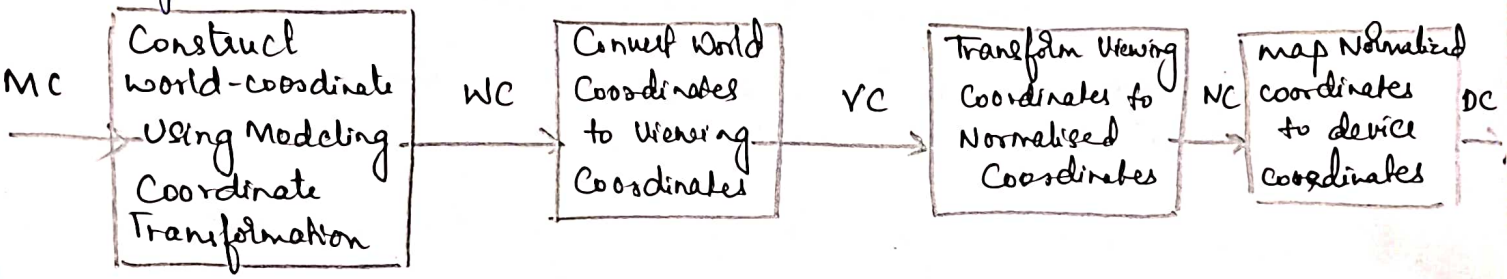
Malavika. S. Patil
1BY20CS102
6th B.

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

The mapping of two dimensional, world co-ordinate 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-dimensional viewing as indicated in the figure given below-



Once 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 coordinates output device, graphics systems convert object descriptions to normalised 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 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 and 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 2-dimensional clipping window, we can use the GLU function

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

We specify the viewport parameters with the openGL function

```
glViewport(xvmin, xvmax, vpwidth, vpheight);
```

We need to initialize the GLUT with following function

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

We have three functions in GLUT for defining a display window and choosing its dimensions and positions:

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 light 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 constant

$$I = I_a \times K_a.$$

where I_a: ambient light intensity (color)

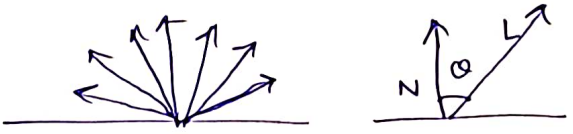
K_a: ambient reflection coefficient (0~1)

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

The brightness at each point is proportional to $\cos(\theta)$.

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

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



where I_p = intensity of 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 point light source

K_s = specular reflection coefficient (0~1)

n = shininess

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

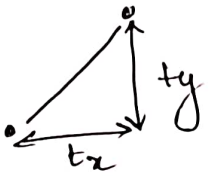
3) Apply homogenous coordinates for translation, rotation and scaling via matrix representation.

Translation moves all points in an object along the same straight line paths to new positions.

We can write the components:

$P_x' = P_x + t$

$P_y' = P_y + t_y$



In matrix form:

$P' = P + T$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix}$$

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

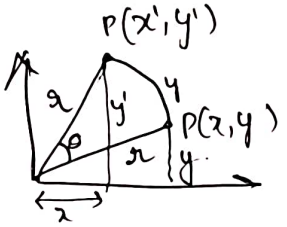
Review Trigonometry

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

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

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

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



We can write components

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

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

In matrix form,

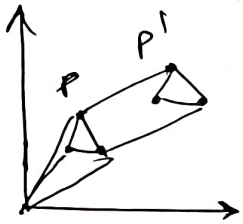
$P' = R.P$ where $R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$

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

Components are:

$P_x' = S_x.P_x$ & $P'y = S_y.P_y$

In matrix $P' = S.P$ 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 $P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

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

Combining above equations, we can say that

$P' = M_1 * P + M_2$

Co-ordinate (x, y) with homogenous coordinate (x_h, y_h) where $x = x_h/h$, $y = y_h/h$ $(h*x_h, h*y_h, h)$ set $h = 1$ $(x, y, 1)$

Homogenous coordinate representation

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

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

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

4) Outline differences between raster and random scan displays.

Raster-Scan Display.

→ electron beam is swept across the screen one row at a time from top to bottom.

→ As it moves across each row, 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 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 the aspect ratio, which is defined as number of pixel columns divided by number of scan lines that can be displayed by the system.

Random scan display

When operated as a random-scan display unit, a CRT has the electron beam directed only to those points of the screen where a picture is to be displayed.

Pictures are generated as line drawings, with the electron beam tracing out the component lines one after the other.

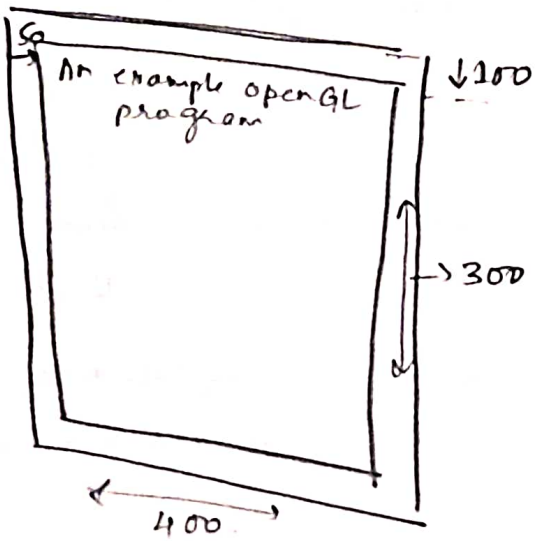
For this reason, random-scan monitors are also called vector displays.

Component lines of a picture can be drawn and represented by random scan system in any specified order.

A pen plotter operates similarly and is example of 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 glutInit(&argc, argv) statement. Next, we can state that the display window is to be created on the screen with a given caption for title bar. this is accomplished with the functions

→ glutCreateWindow ("An Example OpenGL program");

where the single argument for the function can be any character string

The following function called the line segment description to the display window

→ glutDisplayFunc (lineSegment);

GlutMainLoop(): this function must be the last one in the program. It displays the initial graphics and puts the program into an infinite loop that checks for input from devices such as a mouse or keyboard.

glutInitWindowPosition(50, 100);

The following statement specifies upper left corner of the display window should be placed 50 pixel to righ of left edge of the window screen and 100 pixel down from top edge of screen.

glutInitWindowSize (400, 300);

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

glEnable (GL_CULL_FACE);

glCullFace (mode);

where parameter mode is assigned the value GL_BACK, GL_FRONT, INT_FRONT_AND_BACK.

By default, the parameter has value GL_BACK.

It is turned off with glDisable (GL_CULL_EYE).

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 depth buffer, as well as for 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 following functions:

glEnable(GL_DEPTH_TEST);

c) OpenGL wireframe surface visibility method:

A wire-frame displays the standard graphics object can be obtained in OpenGL by requesting that only its edges are to begenerated.

glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)

d) OpenGL-Depth-Cuing function

Varies 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 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

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

7) Write the special cases discussed w.r.t perspective projection transformation.

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

Special cases

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

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

we get y_p when projection reference point is limited to positions along the z_{view} axis.

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

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

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

we get x_p, y_p when projection reference point is fixed at coordinate origin

$$z_{vp} = 0$$

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

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

we get x_p and y_p if the view plane is the uv plane and there are no restrictions on the placement of the project reference point.

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

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

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

we get the above equation with uv plane as the view plane and the projection references point on the z_{view} axis.

8) Explain Bezier curve equation with properties.

Developed by french engineer pierre Bezier for use in design of Renault automobile bodies.

Beziers have a number of properties that make them highly useful for curve and surface design. They are 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.

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

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

where $C(n,k) = \dfrac{n_k}{k_k (n-k)_k}$.

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

curve lies within the first convex hull of the control points.

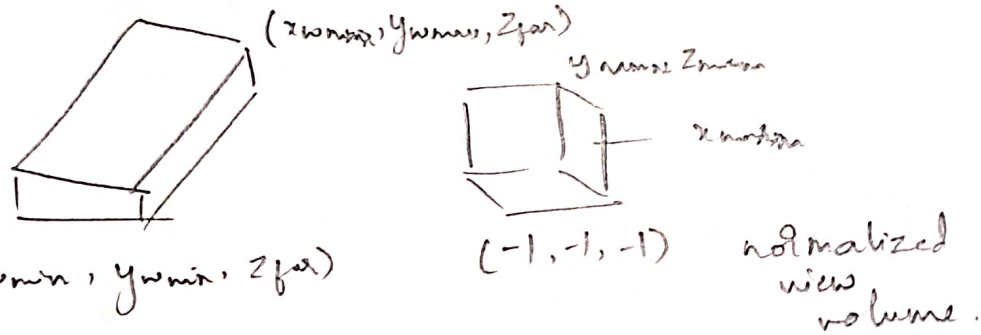
9) Explain normalisation transformation for an orthogonal projection

The normalisation transformation we assume that the orthogonal projection view volume is to be mapped into the symmetric normalization cube with a left handed reference frame. Also, z-coordinate positions for the near and far planes 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 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

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

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

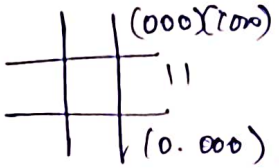


10) Explain Cohen-Sutherland line clipping algorithm.
Every line and point in a picture is assigned a four digit binary value called a region code and each bit position is used to indicate whether point is inside or outside of one of the clipping window boundaries.

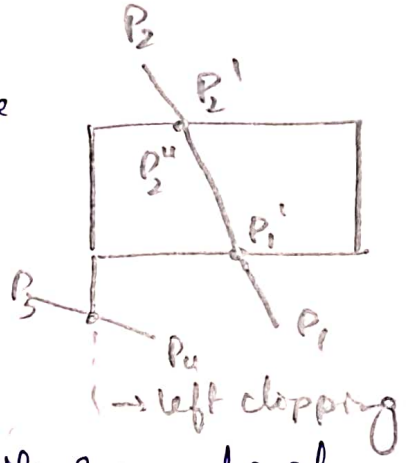
1001	1000	1010
0001	0000	0010
0101	0100	0110

Once we have established region codes for all line and points. we can quickly determine which line are completely within clip windows and which are clearly outside.
When the OR operation between 2 end points region codes for a line segment is false (0000), the line is inside the clipping window, when AND

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 code says P_1 is inside and P_2 is outside The intersection to be P_2'' & P_1 to P_2'' is clipped off for line P_3 to P_4 we find that point P_3 is outside the left boundary and 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' and P_4, we find the remainder of the line is below the clipping window and can be eliminated. To determine a boundary intersection point with vertical clipping border lines can be obtained by

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

where x is either $x_{w\,min}$ or $x_{w\,max}$ and slope is

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

For intersection with horizontal border then x coordinate is

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