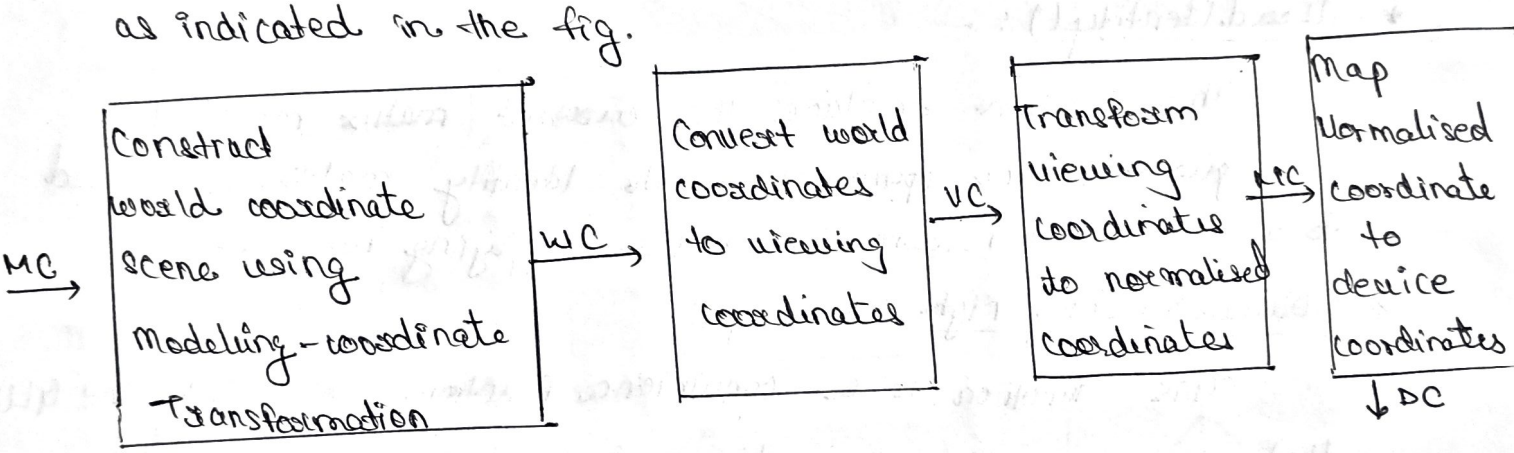
NIKITA BIRADAR
1BY20CS124

CG ASSIGNMENT

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 the 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 description 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 transformed to positions within the display window.

2* Clipping is usually performed in normalised coordinates.

* This allows us to reduce computations by first concatenating the various transformation matrices.

2D viewing functions:

OpenGL provides several functions to set up and control the 2D viewing, functions transformation pipeline.

* glMatrix mode (mode):

This function sets the current matrix mode for subsequent matrix operations. In 2D viewing, you'll typically use the 'GL-MODEL VIEW' and 'GL-PROJECTION' matrix modes

* glLoadIdentity():

This function replaces the current matrix mode for subsequent matrix operations with Identity matrix. It is used to reset the transformation before applying new ones.

* gluOrtho2D (left, right, button, top):

This function is a convinience function provided by the GLU that sets up a 2D orthographic projection matrix similar to 'glOrtho()'.

* We need to initialize GLUT with the following function:

glutInit (&argc, argv);

Other functions in GLUT for defining a display window

1. glutInitWindowPosition (xtopleft, ytopleft);
2. glutInitWindowSize (dwWidth, dwHeight);
3. glutCreateWindow ("Title of Display window");

* Some others are;

1. glutInitDisplayMode (mode)
2. glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
3. glClearColor (red, green, blue, alpha);
4. glClearIndex (index);

k_s = specular reflection coefficient (0~1)

n = shininess

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

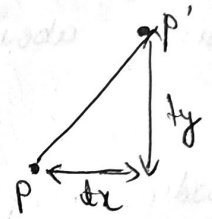
3. Apply homogenous coordinates for translation, rotation and scaling via matrix representation.

Translation → A translation moves all the 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 the 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 a object along a circular path in the plane centered at the pivot point.

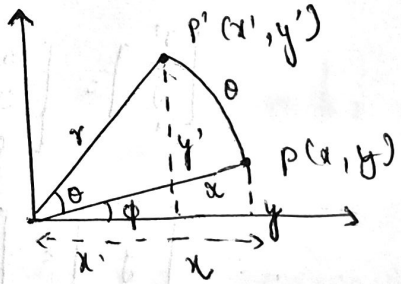
Trigonometry

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

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

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

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



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

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

Scaling → Scaling changes the size of an object and involves two scale factors i.e. s_x and s_y for the x & y coordinates respectively.

Components are

$p'x = s_x p_x$ and $p'y = s_y \cdot p_y$

in matrix $p' = sp$ where $s = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$

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

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.

$$p' = M_1' * p + M_2$$

coordinate (x, y) with homogeneous coordinate (x_h, y_h, h) where

$x = x_h / h \quad y = y_h / h$: $h * x_h, h * y_h, h)$

Homogeneous coordinates set $h = 1$

representation. $(x, y, 1)$

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

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \rightarrow \text{scaling.}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \rightarrow \text{Rotation}$$

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

Raster scan displays

* The electron beam is swept across the beam intensity is turned on and off to create a pattern of illuminated spots
* As it is swept across one row at a time from top to bottom
* This scanning process is called refreshing. Each complete scanning of a screen is normally called a frame.
* The refreshing called the frame rate, ~~called~~ is normally 60 to 80

picture definition is stored in a memory area called the frame buffer.

* This frame buffer stores the intesity values for all the screen points. Each screen point is called as pixel
* Property of raster scan is aspect ratio, which defined a number of pixel columns divided by no. 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 distributed only to those parts of the screen where a picture is to be displayed.
* Pictures are generated as line drawing 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 devices
* 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.

Step1: Initialization of GLUT:-

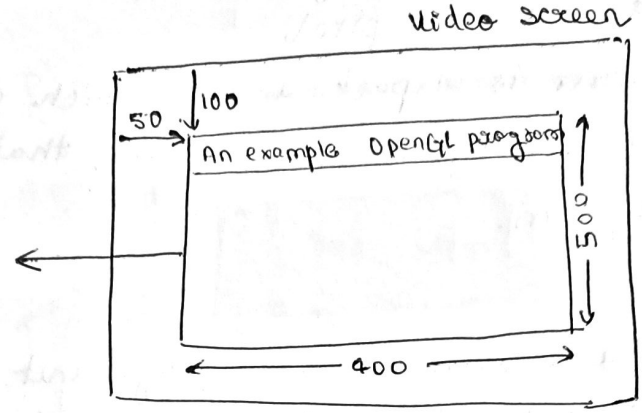
* We are using the OpenGL utility Toolkit
* We perform the GLUT initialization with the statement

```
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");

Here, the single argument for this function can be any character string that we want to use for the display window title.



* The following function call passes 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 graphic and puts the program into an infinite loop that checks for input from devices or mouse or keyboard.
* 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 screen and 100 pixels down from the top edge of screen.
* glutInitWindowSize (400, 300);
  The glutInitWindowSize function is used to set the initial pixel width and height of display window.
* glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB):
  The following command specifies that a single refresh buffer is to be used for window and color mode like red, green and blue component to select color values

6. Explain the OpenGL visibility detection functions.

OpenGL polygon-culling functions

OpenGL wire frame surface visibility method.

- A wire-frame display 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);
glfog:(GL_fOG_mode, GL_LINEAR)
```

→ This applies the linear depth function to object colours using dmin = 0.0 and 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. $z_{prp} = y_{prp} = 0$

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

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

Back-face removal is accomplished

```
glEnable (GL_CULL_FACE);
glCullFace (mode);
```

* where parameter mode is assigned the value GL_BACK, GL_FRONT, GL_FRONT_AND_BACK.
* By default, parameter mode in glCullFace functions 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 function:

```
glEnable (GL_DEPTH_TEST);
```

And we deactivate these depth buffer routines with

```
glDisable (GL_DEPTH_TEST);
```

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

```
glDepthMask (writeStatus);
```

$z_{vp} = 0$

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

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

We get 3a & 3b if the view plane is the uv plane & there are no restriction 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 reference point on the z view 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 as 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_u = the position vector which describes the path of an approximate Bezier polynomial function between p_0 and p_n

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

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

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 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 $(1,1,1)$.

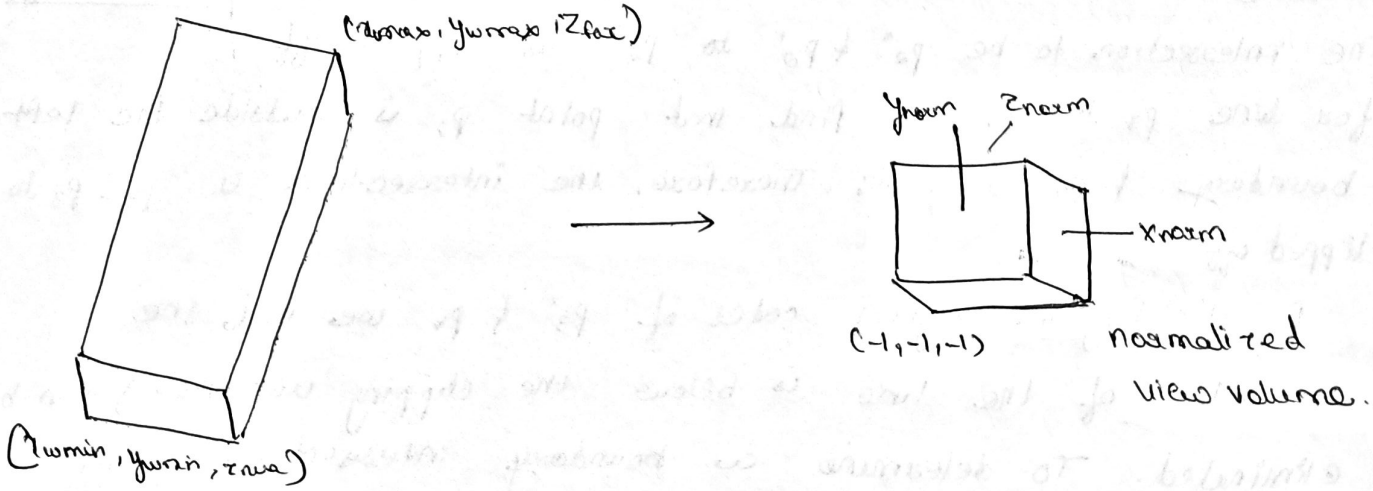
Transforming the rectangular parallelepiped view volume to a normalized cube is similar to the method for converting the clipping window into the normalized symmetric square.

The normalization transformation for the orthogonal view volume is

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

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

Orthogonal-projection



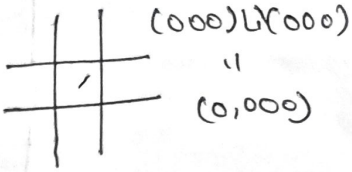
10. Explain cohen-sutherland line clipping algorithm.

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 one of the clipping window boundaries. Once we have

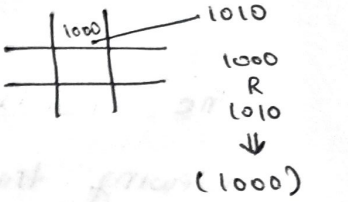
1001	1000	1010
0001	0000 cliping window	0010
0101	0100	0110

established region codes for all line endpoints. We can quickly determine which line are completely within clip window and 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.

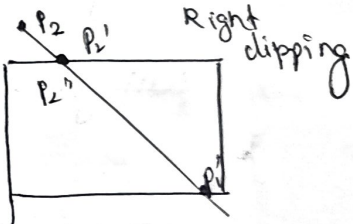


When AND operation between 2 ends points region codes for a line is true, the line is completely outside the clipping windows.



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 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 & 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, the x coordinate is

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