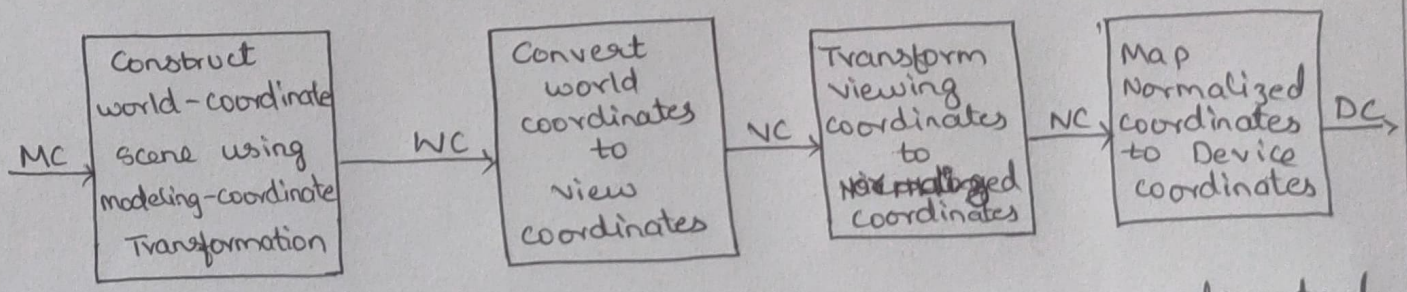
NAME: LAVANYA.N USN: 1BY20CS094	COMPUTER GRAPHICS AND VISUALIZATION ASSIGNMENT	DATE: 07-07-23 SECTION: B

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

Ans. * 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.

Steps for two-dimensional 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 system convert objects descriptions to normalized coordinates and apply the clipping routines.

* Systems use normalized coordinates in the range from 0 to 1 and others users 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.
* Clipping is usually performed in normalized coordinates
* This allows us to reduce computations by first concatenating the various transformation matrices.

The 2D Viewing functions

We must set the parameter 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 function.

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

* we specify the viewport parameters with the openGL funtn

```
glViewport(xvmin, xmin, vpWidth, vpHeight);
```

* we need to initialize GLUT with the following function:

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

* We have three function 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

1. Ambient 2. Diffuse 3. Specular

1. 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 constant

$$I = Ia \times ka$$

where Ia: ambient light intensity (color)
ka: ambient reflection coefficient (0~1)

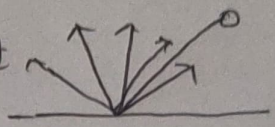
2. Diffuse reflection:

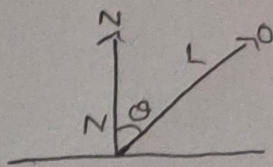
This incident light on the surface is scattered with equal intensity in all directions, independent of the viewing position. Such surface is called ideal diffuse reflection.

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 to 1)

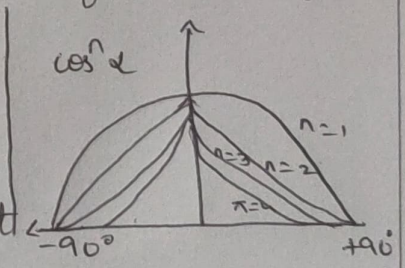
This equation can also be written as

$$I_{diff} = I_p \times k_d \times N.L$$

3. Specular component :

The component describes the specular reflection of smooth surfaces

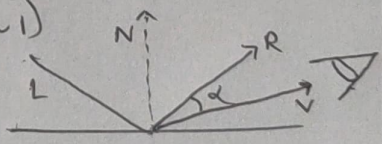
$$I = I_p \times k_s \times \cos^n\alpha$$



where I_p = intensity of the point light source

k_s = specular reflection coeffient (0 to 1)

n = shininess



This equation can also be written as

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

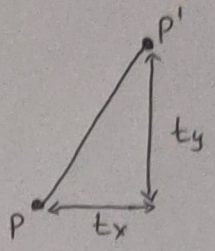
3. Apply homogeneous coordinates for translation, rotation and scaling via matrix representation.

Translation

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

We can write the components

$$P'x = Px + t_x$$
$$P'y = Py + ty$$



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

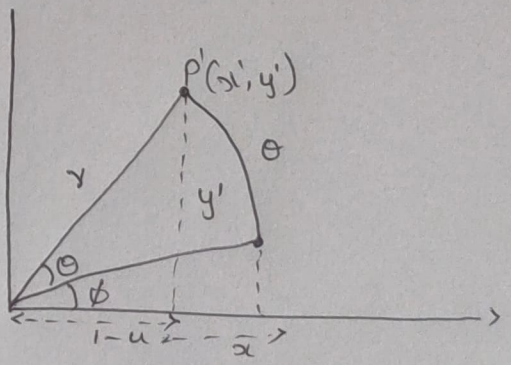
Review trignometry

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

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

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

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



We can write the components

$$P'x = Px\cos\theta - Py\sin\theta$$
$$P'y = Px\sin\theta + Py\cos\theta$$

In matrix form

$$P' = R.P$$

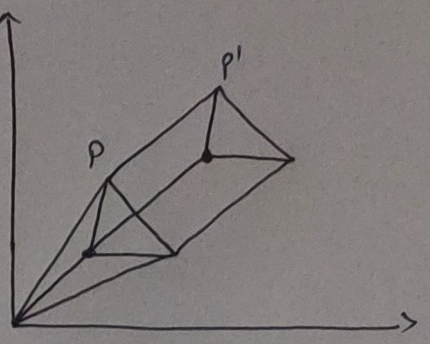
where $$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Scaling

Scaling changes the size of an object and involves two scale factor s, Sx & Sy for the x & y coordinates respectively

$P'_x = S_x \cdot P_x$ &

$P'_y = S_y \cdot P_y$



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

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

$$P' = M_1 P + M_2$$

coordinates (x, y) with homogeneous coordinate (x_h, y_h, h) where $x = x_h/h$, $y = y_h/h$ $(h * x_h, h * y_h, h)$ set $h = 1$ (x_1, y_1)

Homogeneous co-ordinates representation:

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

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

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

4. Outline the difference between raster scan displays & 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 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 the fram 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.
* The property of a roaster scan is aspect ratio, which defined as number of pixel columns divided by number of scan lines that can be displayed by the system

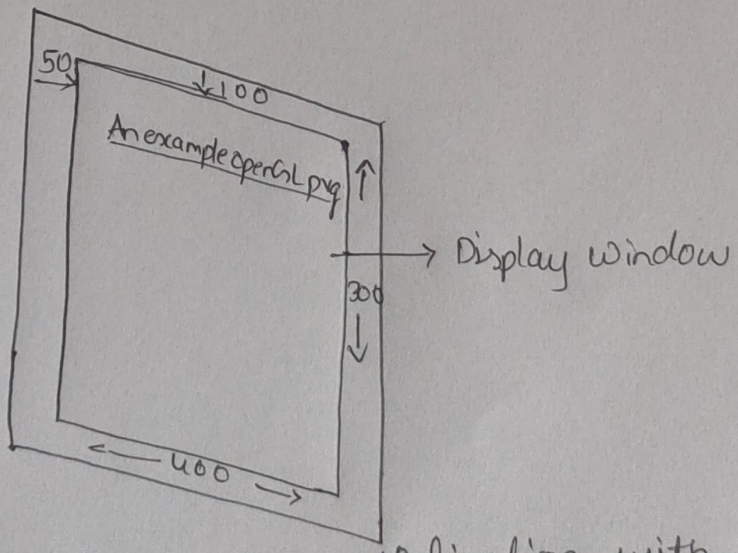
Random Scan displays

* When operated as a random-scan-display unit, a CRT has the electron beam directed only to those points of the screen colour 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 displays 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
* As 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 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 down from the top edge of the screen

* glutInitWindowSize(400,300);
the glutInitWindowSize function is used to set the initial pixel width & height of the display window

6. Explain OpenGL visibility Detection functions.

1] OpenGL Polygon-Culling functions
Back face removal is accomplished with functions.
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 function has the value GL_BACK.

2] 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 refresh buffer.

```
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
```

Depth buffer values can be initialized with glClear (GL_DEPTH_BUFFER_BIT).

These routines are actuated with following function

```
glEnable(GL_DEPTH_TEST);
```

3] OpenGL wireframe 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);
```

4] OpenGL Curing 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 & dmax with following

```
glFog(GL_FOG_START, MinDepth);
glFog(GL_FOG_END, MaxDepth);
```

7. Write the special cases that we discussed with respect to perspective 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. $(x_{prp}, y_{prp}, z_{prp}), (0,0,0)$

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

$$y_p = y\left(z_{vp}/z\right) \longrightarrow (2)$$

we get (2) when the projection reference point is fixed at coordinate 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 and 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

* Deployed 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_u = the position vector which describes the path of an approximate Bezier polynomial function blw P_0 & P_n

$$P[U] = \sum_{k=0}^{n} P_k BEZ_{k,n}(U) \quad 0 \le U < 1$$

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

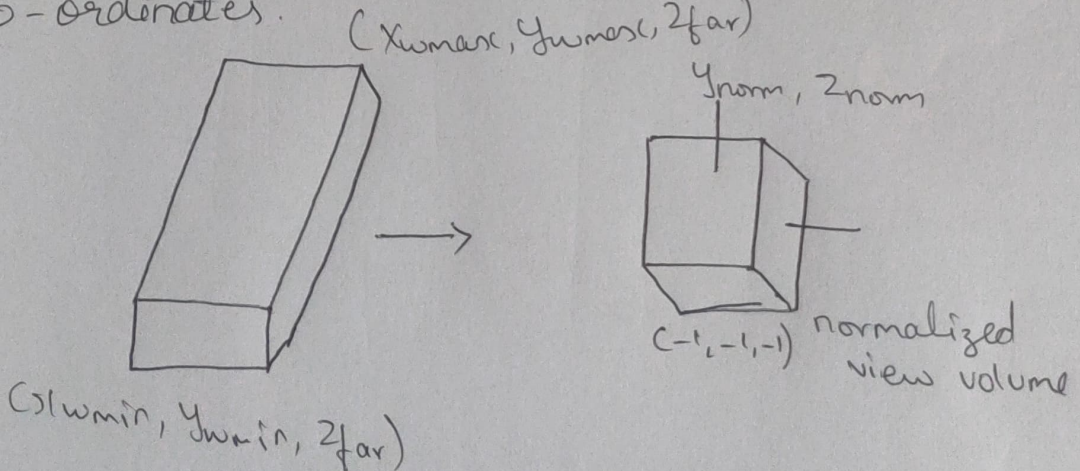
The normalization transformation we assume that the orthogonal projection view volume is to be mapped into the symmetric normalization cube within a left handed reference frame. Also, Z-coordin-ate positions of the near and far planes are denoted as Znear and Zfar.

Respectively this position $(x_{max}, y_{max}, z_{far})$ is mapped to (1, 1, 1) cube position (-1, -1, -1) & position $(x_{max}, y_{max}, z_{near})$ is mapped to (-1,-1,-1)

Transforming the rectangular parallel piped view plane 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

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

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



10. Explain Cohen-Sutherland Line Clippin Algorithm

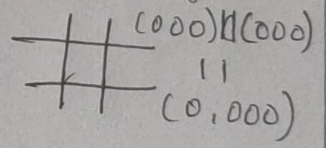
1001	1000	1010
0001	0000	0010
0101	0100	0110

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

Once we have established region codes for all line end points. we can quickly determine which line are completely within clip window R which are clearly outside

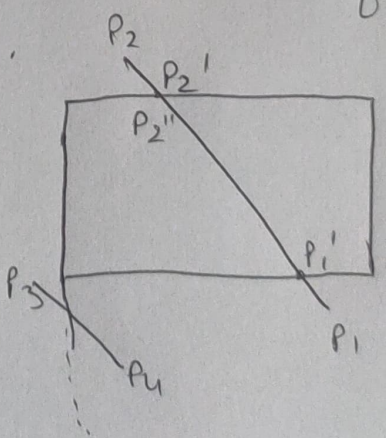
When the OR operation between 2 endpoints region codes for a line segment is false (0000), the line is inside the clipping window.

When And operation between 2 end points region code 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 code tests are next checked for intersection with border lines.

The region codes says P_1 is inside & P_2 is outside



The intersection to be P_2'' & P_1 to P_2'' is clipped off four 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 reminder of the line is below the clipping window and 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} & slope is

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

∴ For intersection with horizontal border then x coordinates

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