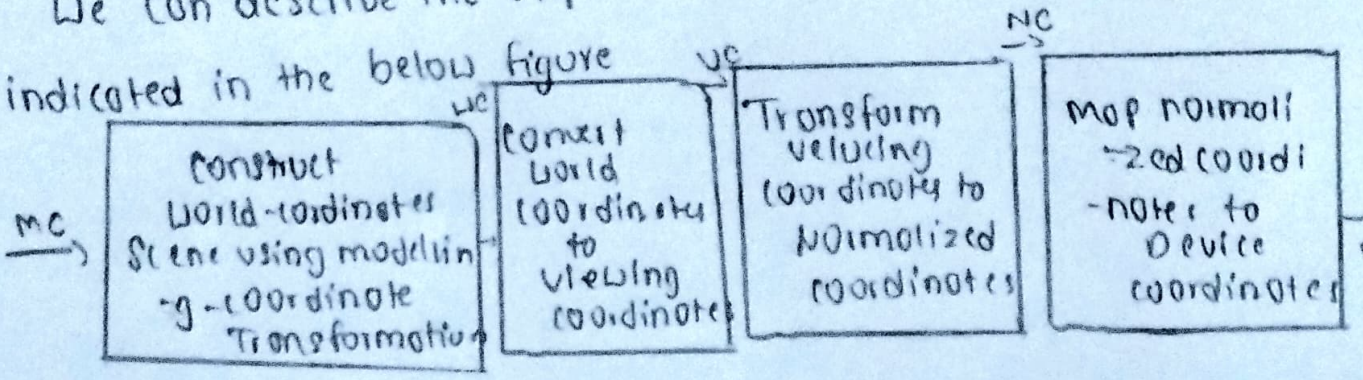
CG Assignment

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

The mapping of two-dimensional, World-cordinates scene description to device coordinates is called a two-dimensional viewing transformation

This transformation is simply refered to as the window-to-viewport transformation or windowing transformation

We can describe the steps for two-dimension viewing as indicated in the below figure



Once a world-coordinate scene has been constructed, we could set up a seporate two-dimensional, viewing coordinate reference for specifying the clipping window

To make the viewing process independent of requirement of any output devices, graphics convert object description to normalized coordinates and apply clipping routines

Systems use normalized coordinates in range from 0 to 1, and others use a normalized coordinates in range from -1 to 1

At the final step of viewing transformation, the contents of the viewport are transfered to positions within the display windows

Clipping is usually performed in normalized coordinates

2D Viewing Functions

We must set the parameters for the clipping window as part of the projection transformation

```
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 function

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

We need to initialize GLUT with the following function:

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

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

```
glutInitWindowPosition(xTopleft, yTopleft);
glutInitWindowSize(dwWidth, dwHeight);
glutCreateWindow("Title of Display window");
```

Various display-window parameters are selected with the GLUT functions

```
glutInitDisplayMode(mode);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glClearColor(red, green, blue, alpha);
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 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 content $I = I_a \times k_a$

Where I_a:- ambient light Intensity (color)

k_a:- ambient reflection coefficient (0 to 1)

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

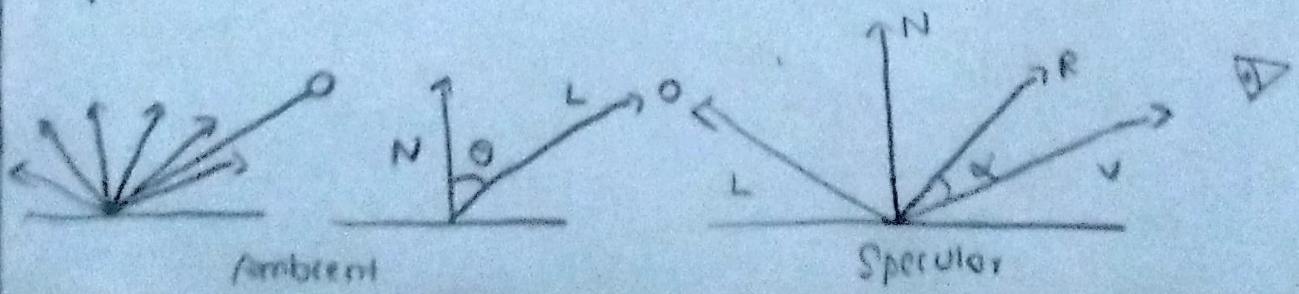
The brightness at each point is proportional to $\cos(\theta)$

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

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

I_p intensity, k_d diffuse reflection coefficient

Specular component:- smoth surfaces $I = I_p \times k_s \times \cos^n \alpha$

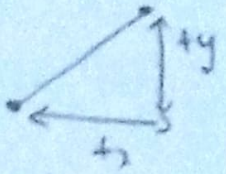


3 Apply homogeneous coordinates for translation, rotation and scaling via matrix representation

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

We can write the components

$P_x' = P_x + t_x \quad P_y' = P_y + t_y$



in matrix form

$P' = P + T$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \rightarrow (1)$$

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

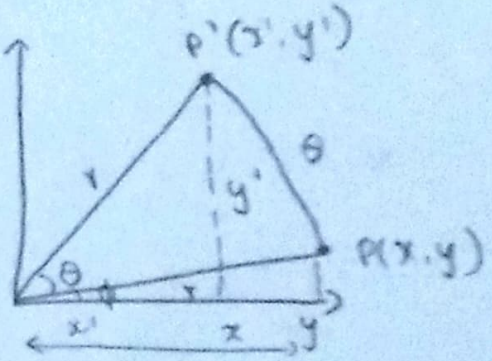
Review Trigonometry

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

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

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

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



We can write components

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

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

matrix $P' = R.P$ where $R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \rightarrow (2)$

Scaling → It changes the size of an object and involves two scale factor s, s_x and s_y for x and y coordinates respectively components are $P_x' = S_x.P_x$ and $P_y' = S_y.P_x$

$P' = S.P$ where $S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \rightarrow (3)$

Combining all above equations we can write

$P' = M_1.P + M_2$

Homogeneous coordinates representation

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

6 Outline the differences between raster scan displays and 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 pattern of illuminated spots

This scanning process is called refreshing. Each complete scanning of a screen is normally called 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 store intensity values for all the screen points. Each screen point is called pixel

Property of raster scan is Aspect ratio, which defined by number of pixels columns divided by number of scan lines that can be displayed by system

Random Scan displays

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

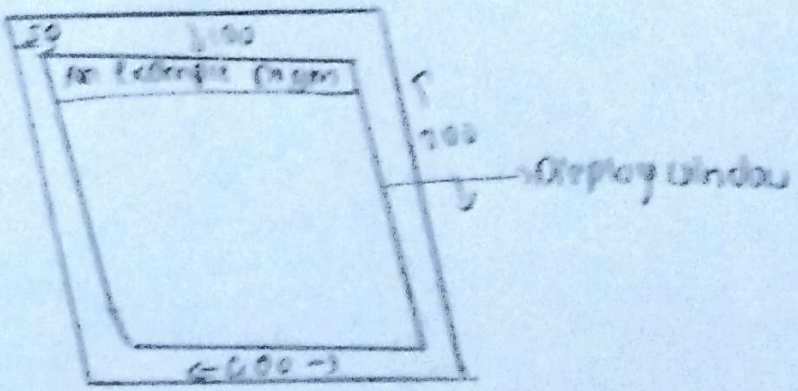
Pictures are generated as line drawings, with the electron beam tracing out the component lines one after the other. For this reason, random-scan monitors are also referred to as vector displays.

The component lines of pictures can be drawn and refreshed by a random-scan system in any specific order.

A 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 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 title bar, this is accomplished with the diagram function

→ glutCreateWindow("An example GL program");

Where the single argument for this function can give 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 program it displays the initial graphics and puts the program into an infinite loop that checks for input from devices such as mouse keyboard

glutInitWindowPosition(50,100);

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

glutInitWindowSize(400,300);

the glutWindowSize function is used to set the initial pixel width and height of display window

6. Explain OpenGL visibility detection functions.

a) OpenGL Polygon - Culling functions.

Back-face removal is accomplished with the 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 GL_BACK. The calling Routine is turned off with glDisable(GL_CULL_FACE);

b) OpenGL Depth-Buffer-functions

To use openGL depth buffer visibility-detection function we first need to modify the GL Utility Toolkit (GLUT) initialization function for 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 using glClear(GL_DEPTH_BUFFER_BIT). These routines are activated with functions:

glEnable(GL_DEPTH_TEST);

c) Display wire frame Surface visibility method

A wire frame displays 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 cuing functions:

We can vary the brightness of an object as function of its distance from viewing position with glFog():

glFogi(GL_FOG_MODE, GL_LINEAR);

Objects which are closer dmin=0.0, dmax=1.0

glFogf(GL_FOG_START, minDepth);

glFogf(GL_FOG_END, maxDepth);

Write Special cases that we discussed with respect to perspective projection transformation

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

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

Special cases

1. $x_{prp} = y_{prp} = 0$

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

we get (1) when the projection reference point is limited to positions along the view axis

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

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

We get (2) when the projection reference point is fixed to coordinate origin

3. $z_{vp} = 0$

$$x_p = x\left(\frac{z_{prp}}{z_{prp}-z}\right) - x_{prp}\left(\frac{z}{z_{prp}-z}\right), \quad y_p = y\left(\frac{z_{prp}}{z_{prp}-z}\right) - y_{prp}\left(\frac{z}{z_{prp}-z}\right) \rightarrow (3)$$

We get (3) if view plane is the plane and there are no restrictions on placement of projection reference point

4. $x_{prp} = y_{prp} = z_{vp} = 0 \qquad x_p = x\left[\frac{z_{prp}}{z_{prp}-z}\right] \qquad y_p = y\left[\frac{z_{prp}}{z_{prp}-z}\right] \rightarrow (4)$

We get (4) with the uv plane as view plane and the projection reference point on 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 a number of properties that make them highly useful for curve and surface design, these 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 position

P_u = the position vector which describes the path of approxi-mate Bezier polynomial function between P_0 and P_n

Bezier polynomial function between P_0 and P_n

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

$BEZ_{k,n}(u) = C(n,k)\, u^k (1-u)^{n-k}$ is 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 no of defining points
* Curve generally follows 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 control points

9 Explain normalization transformation for orthogonal projection

The normalization transformation we assume that Orthogonal projection view volume is to be merged into Symmetric normalization cube within a left handed reference frame.

Also Z-coordinates 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 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 normalized cube is similar to method for converting the clipping window into normalized symmetric square

The normalization transformation for orthogonal view volume is

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

10. Explain Cohen-Sutherland line clipping algorithm

Every line end point in a picture is assigned a four digit binary value, called region code and each bit position is used to indicate whether the point is inside or outside of the clipping window and which are clearly outside

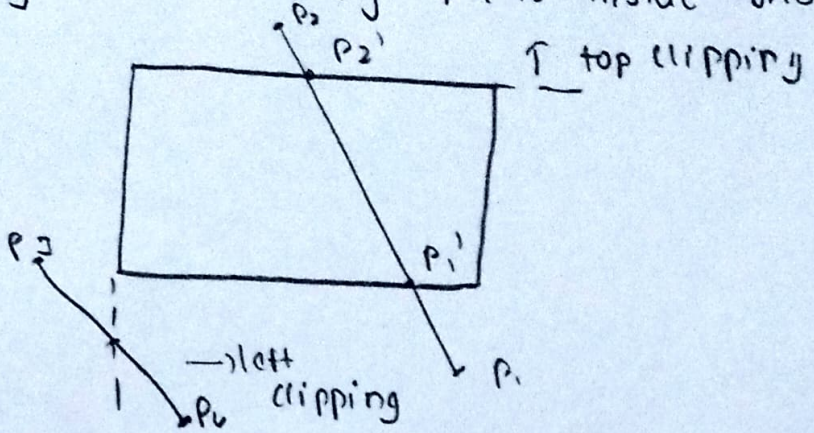
When OR operation between 2 endpoints region codes for a line segment is false (0000), the line is inside the clipping window.

When AND operation between 2 end points regions codes for a line is true the line is completly inside clipping window

1001	1000	1010
0001	0000	0010
0101	0100	0110

lines that cannot be indentified as being completly inside (or) completly outside clipping window by region codes texts 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'' and 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 therefo -re the intersection is P_3 and P_3 to P_3' is clipped off.

By checking the region code of P_3' and P_4 we find remainder of line is below 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 $x = x_{wmin}$ or x_{wmax}

$$\text{slope} = \frac{y_{end} - y_0}{x_{end} - x_0}$$

∴ For intersection with horizontal border then x cordinates

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