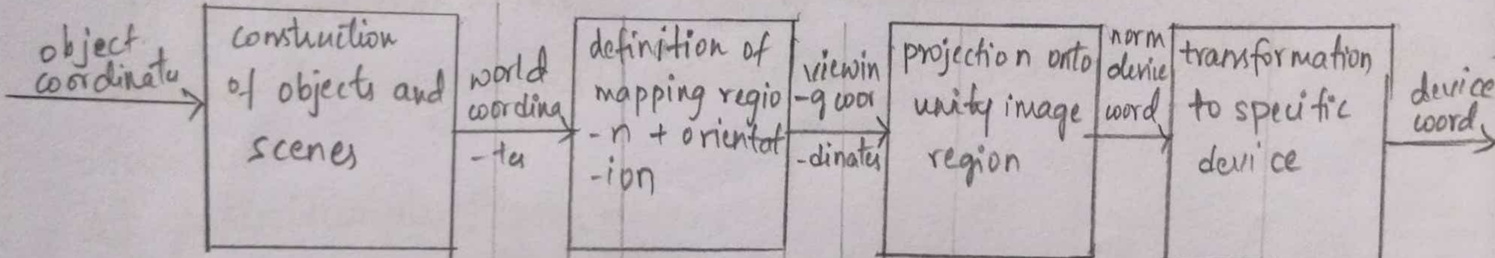
Computer Graphics Assignment

Kavya.S
1BY20CS082

1] Build a 2D Viewing transformation pipeline and also explain the Opengl 2D Viewing functions.

* The mapping of a two-dimensional, world-coordinate scene descrip-tion to device co-ordinates 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 a two-dimensional viewing as indicated in figure.



* Model transformation :- this transformation is applied to individual objects or models within the scene to position, scale, and orient them in a virtual 2D space. It involves applying translation, rotation, and scaling operation
* Viewport Transformation:- The viewport Transformation maps the 2D Viewing space onto the screen and (or) output devices. It defines the final region of the screen where the rendered scene will be displayed. This transformation involves scaling and translating the 2D coordinates of the scene to match the dimensions and position of the viewport on the screen.
* After these transformations, the resulting transformed 2D coordinates are rasterized and mapped to specific pixels on the screen. Various techniques like scanline rendering or ray tracing are then used to determine the color and intensity values for each pixel, taking into account factors such as lighting, shading and textures.

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 two-dimensional clipping window, we can use the GLU function, gluOrtho(xwmin, xwmax, ywmin, ywmax);

* we specify the viewport parameters with the openGL function.

  glViewport(xwmin, xwmax, vpwidth, vpHeight);

* we have three functions in GLUT for defining a display window and choosing it's dimensions and position:

  1] glutInitWindowPosition(xTopLeft, yTopLeft);

  2] glutInitWindowSize(dwWidth, dwHeight);

  3] glutInitCreateWindow("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.

→ It consists of three components:

→ Ambient

→ diffuse

→ specular.

Ambient Lighting → 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

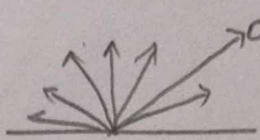
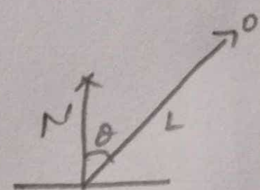
$$I = I_a \times K_a$$

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 in all directions independent of the viewing position, 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.

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

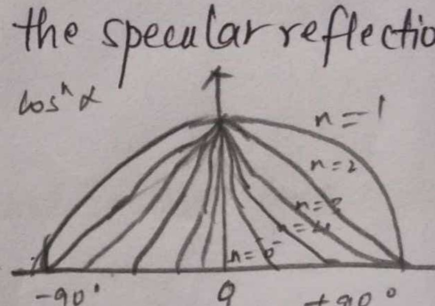



where, I_p = intensity of the point light source
K_d = diffuse reflection coefficient $(0 \sim 1)$.

This equation can also be written as $I_{diff} = I_p \times K_d \times N \cdot L$.

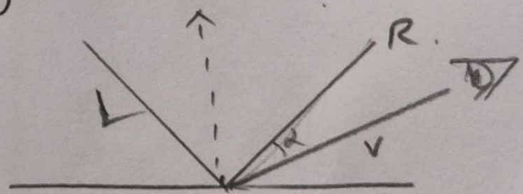
Specular component:- The component describes the specular reflection of smooth surfaces

$$I = I_p \times K_s \times \cos^n \alpha.$$



where, I_p = intensity of the point light source
K_s = specular reflection co-effient $(0 \sim 1)$
n = shininess

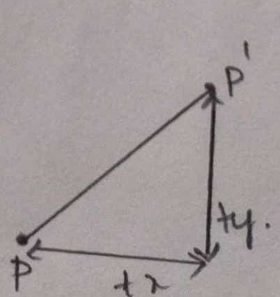
$$I = I_p \times K_s \times (R \cdot V)^n$$



3] Apply Homogeneous co-ordinates 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

$$P'_x = P_x + t_x$$
$$P'_y = P_y + t_y$$



in matrix form:

$$P' = P + T$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

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

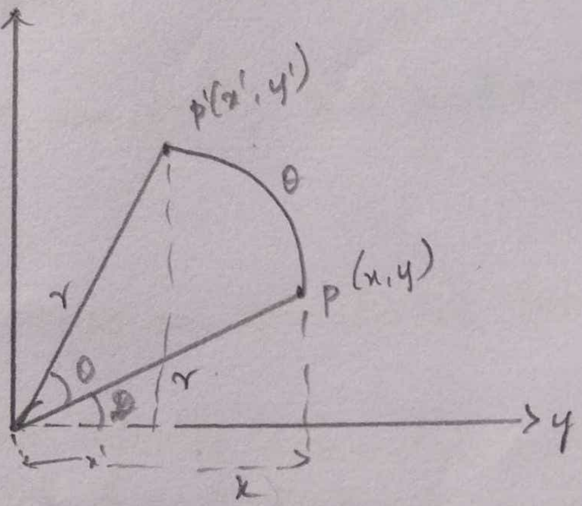
Review Trignometry

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

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

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

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



we can write the Components.

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

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

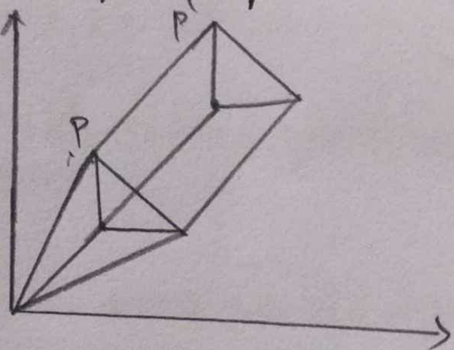
In matrix form

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

Scaling -> scaling changes the size of an object and involves two scale factors, S_x and S_y for the x- and y- coordinates respectively

Components are,

$$P'_x = S_x . P_x \text{ and } P'_y = S_y . P_y$$



in matrix $P' = S.P$, where $S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$

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 homogeneous. co-ordinate (x_n, y_n, h) where x_n

$x = x_n/h$, $y = y_n/h$, where, set $h=1$

Homogeneous co-ordinates 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.

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

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

4] outline the differences between raster scan displays and random scan displays.

Raster scan displays:-

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

→ As it moves across each row, the beam intensity is formed 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 frame rate, is normally 60 to 80 frames per second, or described as 60Hz to 80Hz. picture defination is stored in a memory are 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 roaster scan is Aspect ratio, which defined as number of scan lines that can be displayed by the system.

Random-scan-displays

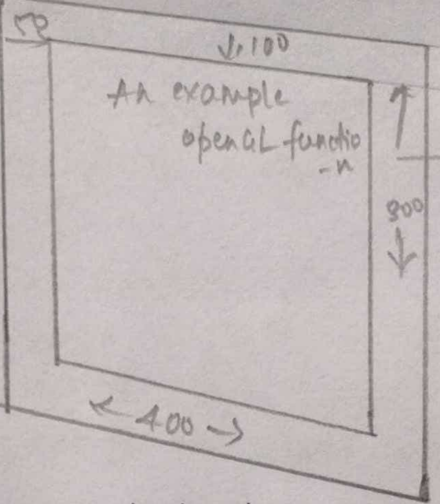
* when operated as a random-scan display unit, a CRT has the electron beam directed only to those pairs of the screen colours a pictures is to be displayed.

* 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 refferred to as vector displays.

* the component lines of a pictures can be drawn and refreshed by a randow scan system in any specified order.

* A pen plotter operates in a similar way and it is an example of a random-scan and hard-copy device

5] Demonstrate the openGL functions for displaying window manage -ment 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 tittle bar. this is accomplis -hed with the function.
* glutCreateWindow("An example"), where the single argument for this function can be any character string.
* The following function call the line-segment description to the display windo -w.
  → glutDisplayFunc(LineSegment);
* glutMainLoop();
  This function must be the last one in out 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 corner of the display window should be place 50 ~~plac~~ pixel to the right of the left edge of the screen and 100 pixel down from the top edge of the screen. the s.
* glutInitWindowSize(400, 300)
  The glutInitWindow size function is used to set the initial pixel width and height of the ~~depository~~ display window.

6] Explain openGL visibility detection function.

a] OpenGL Polygon-Culling Functions.

Back-face removal is accomplished with the function.

glEnable (GL_CULL_FACE);

glCullFace (mode);

* Where parameter mode is assigned the values GL_BLACK, GL_FRONT, GL_FRONT_AND_BACK. By default, parameter mode in glCullface function has the values GL_BACK. The culling routine is formed 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 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 while glclear (GL_DEPTH_BUFFER_BIT)

These modules /routines are attached with the following functions:

glEnable (GL_DEPTH_TEST);

c] openGL wire-Frame Surface visibility method.

A wire-frame displays of a standard graphics object can be obtained in OpenGL by requesting that only it's edges are to be generated.

glPolygonMode (GL_FRONT_AND_BACK, GL_LINE).

d] openGL_DEPTH-Curing function.

we can vary the brightness of an object as a function of it's 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 and dmax with the following

glFogf (GL_FOG_START, minDepth);

glFogf (GL_FOG_END, maxDepth);

7] Write the special areas (or) cases that we discussed with respect to perspective projection transformation.

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

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

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 ~~rexutts~~ 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 at co-ordinate origin.

3] $z_{vp} = 0$

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

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

we get 3a and 3b if the view plane is the uv plane and there are no restrictions on the placement of the projection reference point.

4. $x_{prp} = y_{prp} = z_{vp} = 0$

$$x_p = x\left[\frac{z_{prp}}{z_{prp}-z}\right], \quad y_p = y\left[\frac{z_{prp}}{z_{prp}-z}\right] \rightarrow ④$$

we get ④ with the uv plane as the viewplane and the projection references point onthe zview axis.

8] explain the Bezier curve equation along with it's properties.

* developed by french engineer pierre Bezier for close use in design of Renault autuomobile bodies.
* Bezier have a number of problems/properties that make themhighly 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_v = 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 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 the shape of defining polygon.
* Curve connects the first and last control points thus $P(0) = P_0$, $P(1) = P_n$
* Curve lines lies within the convex null 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 system symmetric normalization cube within a lefthanded reference frame. Also z-coordinate positions for the near and far planes are denoted as z_{near} and z_{far} respectively.

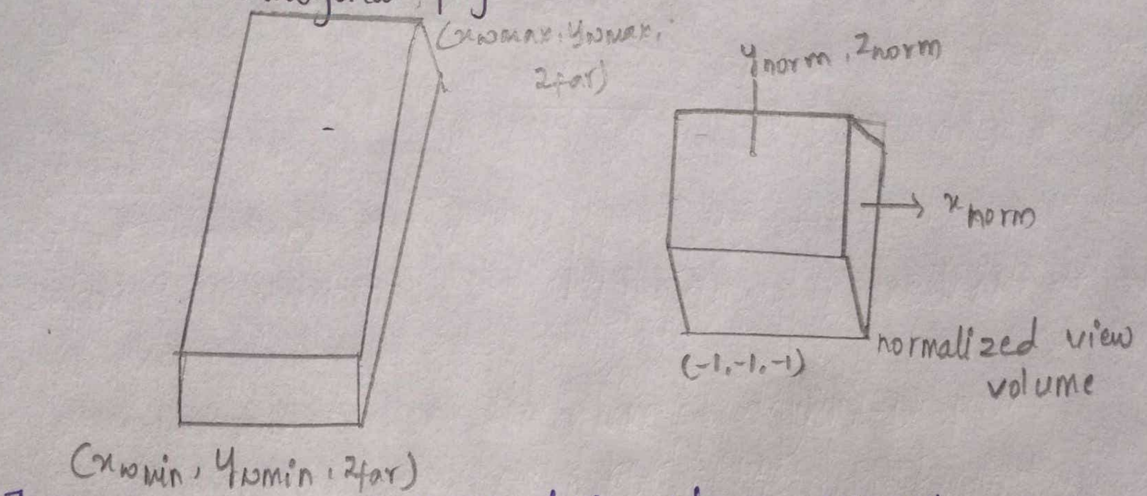
This position $(x_{min}, y_{min}, z_{near})$ is mapped to the normalized position $(-1,-1,-1)$ and position $(x_{max}, y_{max}, z_{far})$ is mapped to $(1,1,1)$.

Transforming the rectangular 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

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

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



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.

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 window. and which are clearly outside.

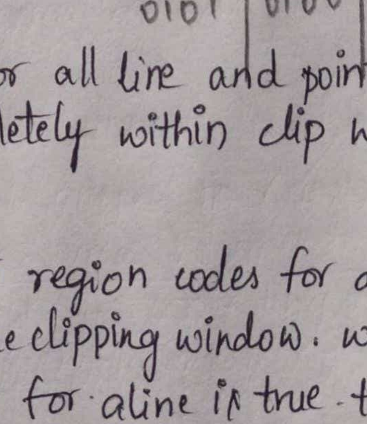
when the OR operation between two endpoints region codes for a line segment is false (0000), the line is inside the clipping window. when AND operation between 2 end points region codes for a line is true. the line is completely outside the clipping window

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