07/09/2023

NISAR AHMED P
(1BY21CS412)

CG Assignment

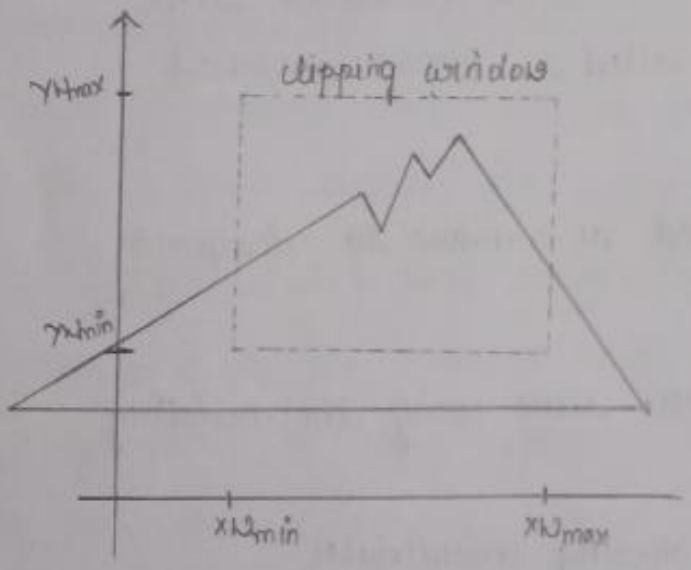
① Build a 2D viewing transformation pipeline and also explain OpenGL 2D viewing functions.

⇒ (i) A section of a two-dimensional scene that is selected for display is called clipping-window.

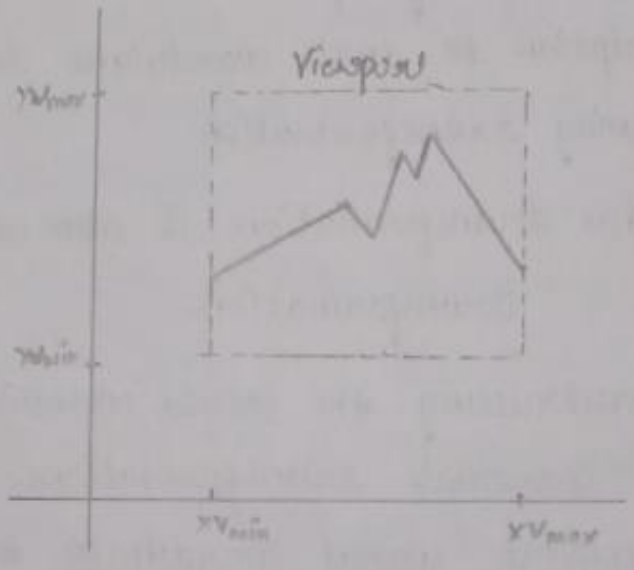
(ii) clipping window is also alluded to as world window (OR) viewing window.

(iii) Graphics packages allow us to control the placement within the display window using another window called as viewport

(iv) Objects that are present in the clipping window are mapped to the viewport.



World coordinates



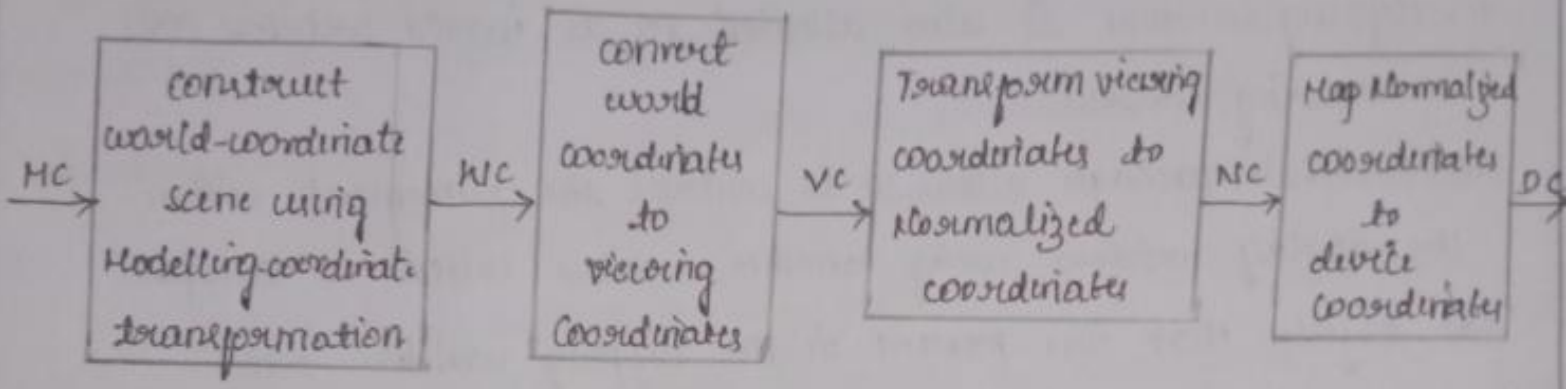
Viewport coordinates

(v) The viewport indicates where the selected object is to be viewed on the output device

(vi) By changing the position of the viewpoint we can view the objects at different positions on the display area of an output device.

(vii) We can use multiple viewports to display different sections at different screen positions.

(viii) Usually clipping window and viewports are rectangles in standard position where the rectangle edges are parallel to the coordinate axis



The mapping of a two-dimensional, world-coordinate scene description to device coordinates is called a two-dimensional viewing transformation.

This transformation is also called as window to viewport transformation.

1. Constructing the world coordinates scene using the modeling coordinate transformations.
2. Convert world coordinates to viewing coordinates
3. Transforming the viewing coordinates to normalized coordinates
4. Mapping the normalized coordinates to device coordinates.

OpenGL 2D viewing functions :-

(i) glViewport(x, y, width, height) :- This function sets the viewport, which defines the output window's dimensions and positions on the screen. It specifies the lower-left coordinates (x, y) and the width and height of the viewport in pixels.

(ii) glMatrixMode(GL_PROJECTION) :- This function sets the matrix mode to the projection matrix. It prepares openGL for defining (or) manipulating the projection transformation.

(iii) glLoadIdentity() :- This function loads the identity matrix into the current matrix stack, resetting any previous transformation.

(iv) glOrtho(left, right, bottom, top, near, far) :- This function sets up an orthographic projection matrix. It defines a rectangular viewing volume and maps it to the canonical view volume, where objects outside this range are clipped.

(v) glPushMatrix() and glPopMatrix :- These functions respectively push and pop the current matrix onto and from the stack. They are useful when applying transformations to different parts of the scene hierarchy, such as objects within objects (or) nested transformation

(vi) glTranslate(), glRotate(), glScale() :- These functions allow for translation, rotation and scaling of the current matrix. They help position and orient the camera (or) objects in the scene

② Build phong lighting model with Equations.

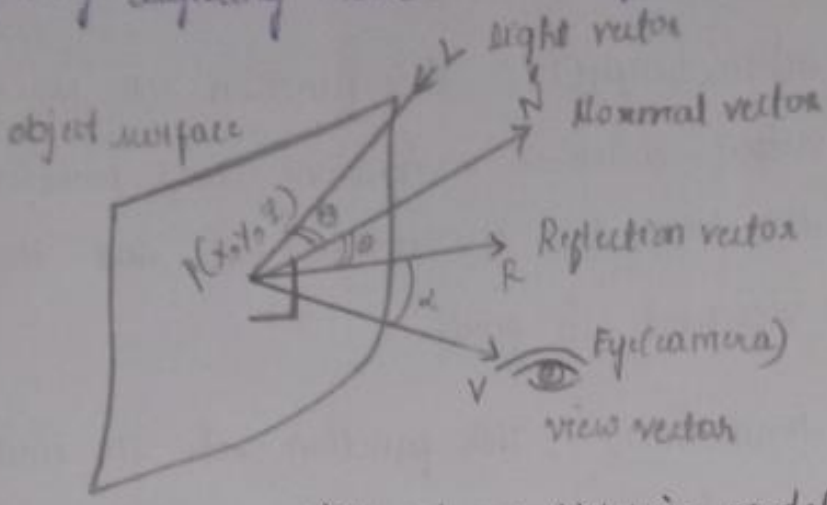


fig:- phong lighting model

The local illumination model (or) the phong lighting model gives the color of a point on the object surface. It is a combination of ambient, diffuse and speculas component of the light.

The illumination equation of the phong lighting model is given as:

$$I_{light} = I_{ambient} + I_{diffuse} + I_{speculas}$$

$$I = I_a K_a + I_d K_d (N.L) + I_s K_s (R.V)^s$$

Here I_a is a combination of red, green and blue component of ambient intensity written as

$$I_a = (I_{ra}, I_{rg}, I_{rb})$$

similarly I_d is a combination of red, green, and blue component of diffuse intensity written as

$$I_d = (I_{da}, I_{dg}, I_{db})$$

and I_s is a combination of red, green and blue component of specular reflection intensity written as

$$I_s = (I_{sr}, I_{sg}, I_{sb})$$

These can be represented in a matrix form as

$$I = \begin{bmatrix} I_{ar} & I_{ag} & I_{ab} \\ I_{dr} & I_{dg} & I_{db} \\ I_{sr} & I_{sg} & I_{sb} \end{bmatrix}$$

The 3×3 matrix of the illumination model for the i^{th} light source is written as

$$L_i = \begin{bmatrix} L_{iar} & L_{iag} & L_{iab} \\ L_{idr} & L_{idg} & L_{idb} \\ L_{isr} & L_{isg} & L_{isb} \end{bmatrix}$$

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

$\Rightarrow$ $P' = P + T$ (Translation)

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

$P' = R \cdot P$ (Rotation)

Rotation $P' = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ ——— ②

$p' = S \cdot P$ (Scaling)

Scaling $P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ ——— ③

Equation ①, ② & ③ could be generalized as in equation ④

$$P' = M_1 \cdot P + M_2 \quad \text{——— ④}$$

Using homogeneous coordinates, the transformations could be combined easily. Here we reformulate equation ④ to eliminate matrix addition.

In homogeneous coordinate system, we combine multiplicative and translational terms by expanding the 2x2 matrix representation to 3x3 matrices. Also expand the matrix representation for coordinate position.

We represent each cartesian coordinates (x, y) with homogeneous coordinate (x_h, y_h, h)

where $x = x_h/h$, $y = y_h/h$

$(h \cdot x, h \cdot y, h)$

let $h = 1$

$(x, y, 1)$

Homogeneous coordinates representation for translation, rotation & scaling are

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

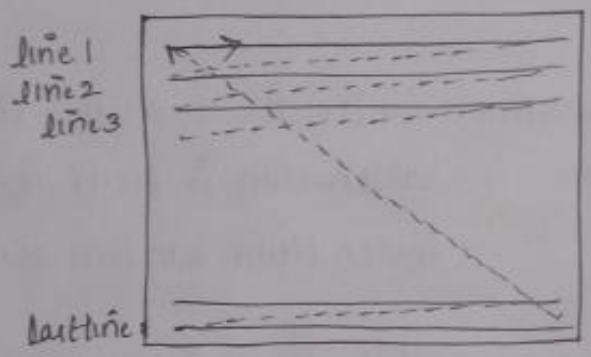
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

(4) outline the differences between raster scan displays and random scan displays.

	Base of difference	Random scan	Raster scan
1.	Resolution	The resolution of random scan is higher than raster scan	while the resolution of raster scan is lesser (or) lower than random scan
2.	cost	it is costlier than raster scan	while the cost of raster scan is lesser than random scan
3.	Modification	In random scan, any alteration is easy in comparison of raster scan	while in raster scan, any alteration is not so easy.
4.	Picture definition	it stores pictures definition as a set of line commands in the refresh buffer	it stores picture definition as a set of intensity values of the pixels in the frame buffer.

8.	Refresh rate	refresh rate depends on the number of lines to be displayed ie 30 to 60 times per second	Refresh rate is 60 to 80 frames per second and is independent of picture complexity.
9.	Solid pattern	In random scan, solid pattern is tough to fill	In raster scan, solid pattern is easy to fill
10.	Example	pen plotter	TV sets



Raster scan

Random scan

⑤ Demonstrate OpenGL functions for displaying window management using GLUT.

⇒ we can consider a simplified example, minimal number of operations for displaying a picture.

Step 1 :- Initialization of GLUT

(i) we are using the OpenGL utility toolkit, our first step is to initialize GLUT.

(ii) This initialization function could also process any command line arguments, but we will not use these parameters for our first example programs

(iii) We perform the GLUT initialization with the statement

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

Step 2: title

(i) We can state that a display window 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");
```

Step 3: Specification of the display window

(i) Then we need to specify what the display window is to contain.

(ii) For this, we create a picture using openGL functions and pass the picture definition to the GLUT routine glutDisplayFunc, which assigns our picture to the display window.

(iii) Example: suppose we have the openGL code for describing a line segment in procedure called lineSegment.

(iv) Then the following function call passes the line-segment description to the display window:

```
glutDisplayFun(lineSegment);
```

Step 4: one more GLUT function

(i) After the execution of the following statement, all display windows that we have created, including their graph content,

are now activated.

```
glutMainLoop();
```

Step5: Additional glut functions.

(i) We use glutInitWindowPosition function to give an initial location for the upper left corner of the display window

(ii) We can also set the number of other options for the display window, such as buffering and a choice of color modes, with the glutInitDisplayMode function.

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

(6) Explain OpenGL visibility detection functions.

→ (1) OpenGL polygon-culling functions :-

(i) 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

→ the culling routine is turned off with.

```
glDisable( GL_CULL_FACE);
```

(2) OpenGL depth-Buffer-functions :-

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

(ii) Depth Buffer values can then be initialized with

```
glClear( GL_DEPTH_BUFFER_BIT);
```

(iv) The OpenGL depth-buffer visibility detection routines are activated with the following function:

```
glEnable(GL_DEPTH_TEST);
```

and we deactivate the depth-buffer routines with

```
glDisable(GL_DEPTH_TEST);
```

(3) openGL depth-cueing functions.

(1) 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 and dmax = 1.0. But we can set different values for dmin and dmax with the following function calls;

```
glFogf(GL_FOG_START, minDepth);
glFogf(GL_FOG_END, maxDepth);
```

(7) write the special cases that we discussed with respect to perspective projection.

→ case1: To simplify the perspective calculations, the projection reference point could be limited to position along zview axis, then

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

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

case 2:

sometimes the projection reference point is fixed at the coordinate origin, and

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

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

case 3: If the view plane is the uv plane and there are no restrictions on the placement of the projection reference point, then we have $z_{vp} = 0$

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

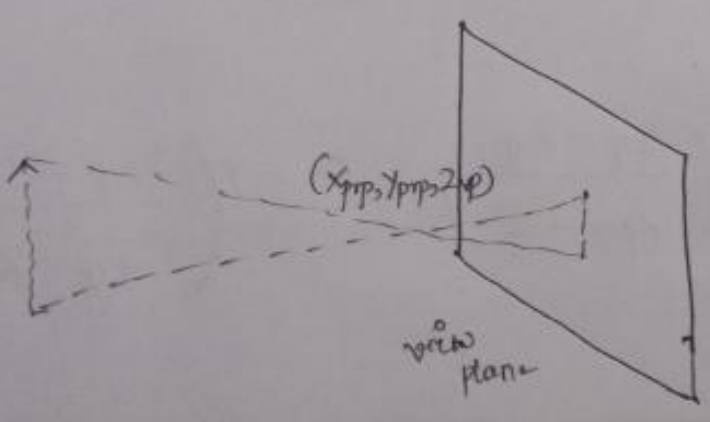
$$y_p = y\left(\frac{z_{prp}}{z_{prp} - z}\right) - y_{prp}\left(\frac{z}{z_{prp} - z}\right)$$

case 4: with the uv plane as the view plane and the projection reference point on the zview axis, the perspective projection is,

$$x_{prp} = y_{prp} = z_p = 0:$$

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

if the projection reference point is the view plane and the same objects are inverted on the view plane



⑧ Explain Bezier curve equation along with Example.

⇒ Bezier curve equations:

(i) we first consider the general case of n+1 control-point positions denoted as $P_k = (x_k, y_k, z_k)$, with k varying from 0 to n

(ii) These coordinates points are blended to produce the following position vector $P(u)$, which describes the path of an approximating Bezier polynomial function between P_0 and P_n:

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

(iii) The Bezier blending function $BEZ_{k,n}(u)$ are the Bernstein polynomials

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

where parameters $C(n,k)$ are the binomial coefficients

$$C(n,k) = \frac{n!}{k!\,(n-k)!}$$

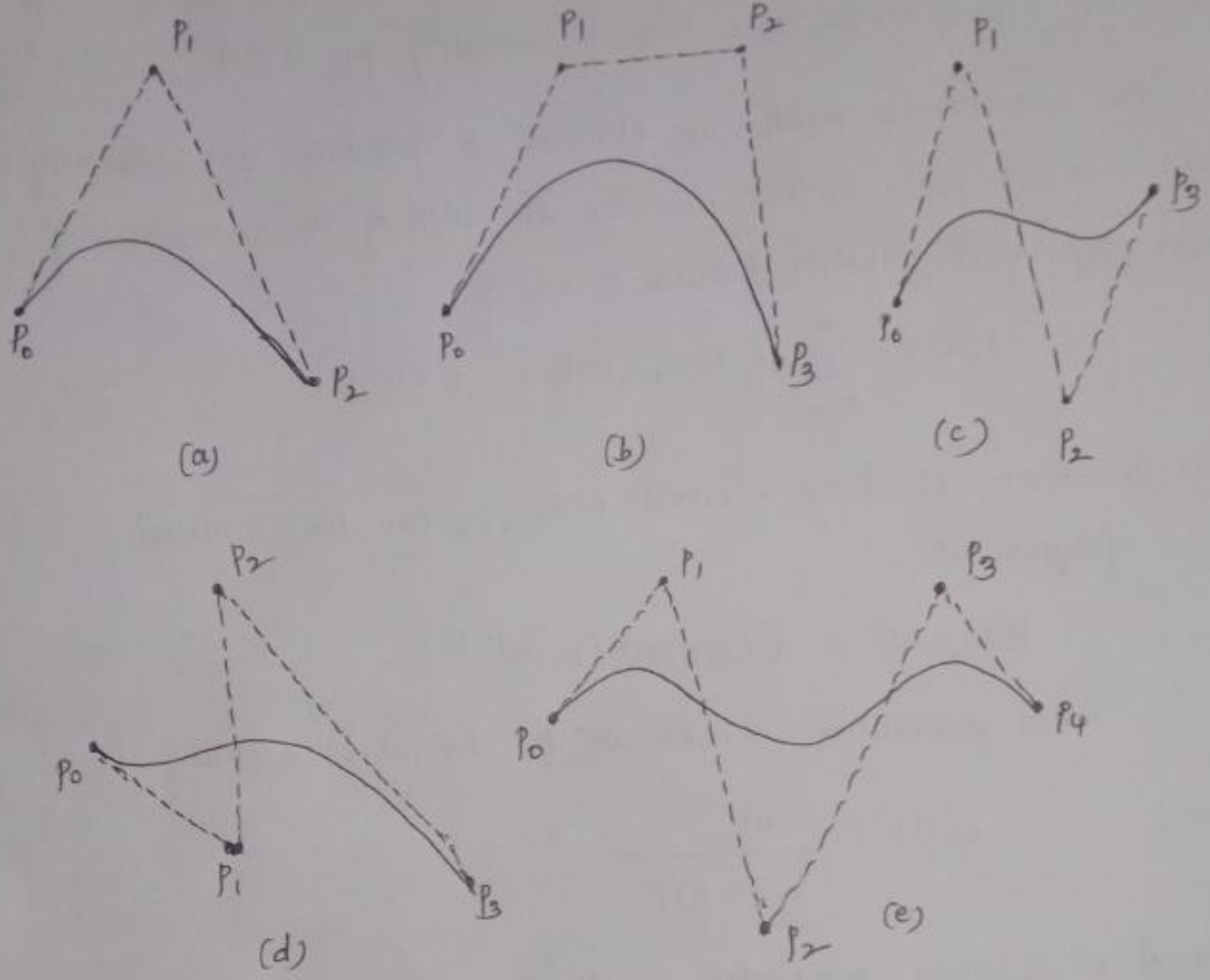
(iv) A set of three parametric equations for the individual curve coordinates can be represented as

$$x(u) = \sum_{k=0}^{n} x_k \, BEZ_{k,n}(u)$$

$$y(u) = \sum_{k=0}^{n} y_k \, BEZ_{k,n}(u)$$

$$z(u) = \sum_{k=0}^{n} z_k \, BEZ_{k,n}(u)$$

(v) Below figures demonstrate the appearence of some Bezier curve for various selections of control points in xy plane ($z=0$)



(vi) Recursive calculations can be used to obtain successive binomial coefficient values as

$$C(n,k) = \frac{n-k+1}{k} C(n,k+1)$$

for $n \geq k$. Also, the Bezier blending functions satisfy the recursive relationship.

$$BEZ_{k,n}(u) = (1-u) BEZ_{k,n+1}(u) + u BEZ_{k+1,n+1}(u), \quad n > k \geq 1$$

with $BEZ_{k,k} = u^k$ and $BEZ_{0,k} = (1-u)^k$

properties of Bezier curve :-

property1 :

A very useful property of Bezier curve is that the curve connects the first and last control points

Thus, a basic characteristics of any Bezier curve is that

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

Values for the parametric first derivatives of a Bezier curve at the endpoints can be calculated from control-point coordinates as

$$P'(0) = -nP_0 + nP_1$$
$$P'(1) = -nP_{n-1} + nP_n$$

The parametric second derivative of a Bezier curve at the end points are calculated as

$$P''(0) = n(n-1)[(P_2 - P_1) - (P_1 - P_0)]$$
$$P''(1) = n(n-1)[(P_{n-2} - P_{n-1}) - (P_{n-1} - P_n)]$$

property2 :

Another important property of any Bezier curve is that it lies within the convex hull of the control points.

$$\sum_{K=0}^{n} BEZ_{K,n}(u) = 1$$

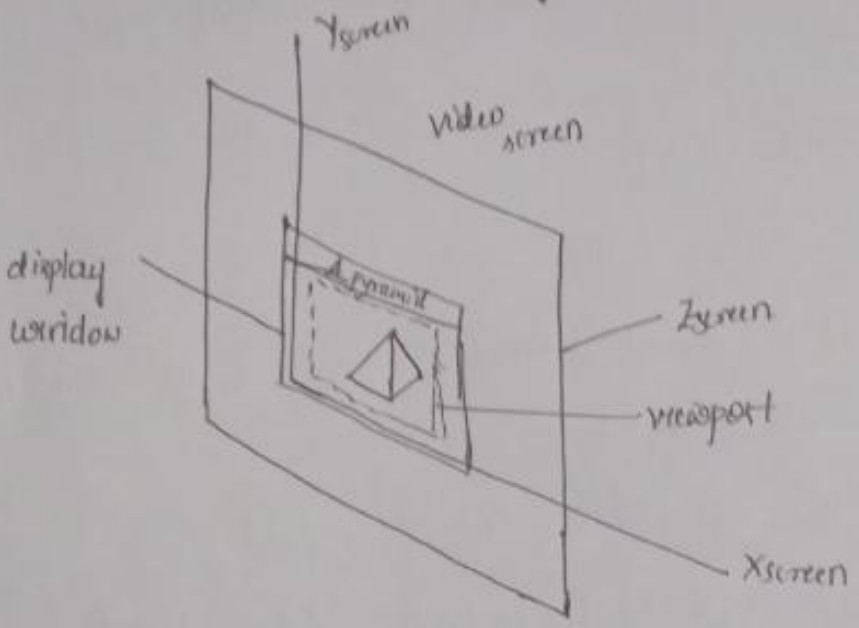
The Bezier blending functions are all positive and their sum is always 1.

① Explain normalization transformation for an orthogonal projection.

→ Once we have established the limits for the view volume, coordinate descriptions inside this rectangular parallelepiped are the projection coordinates, and they can be mapped into a normalized view volume without any further projection processing.

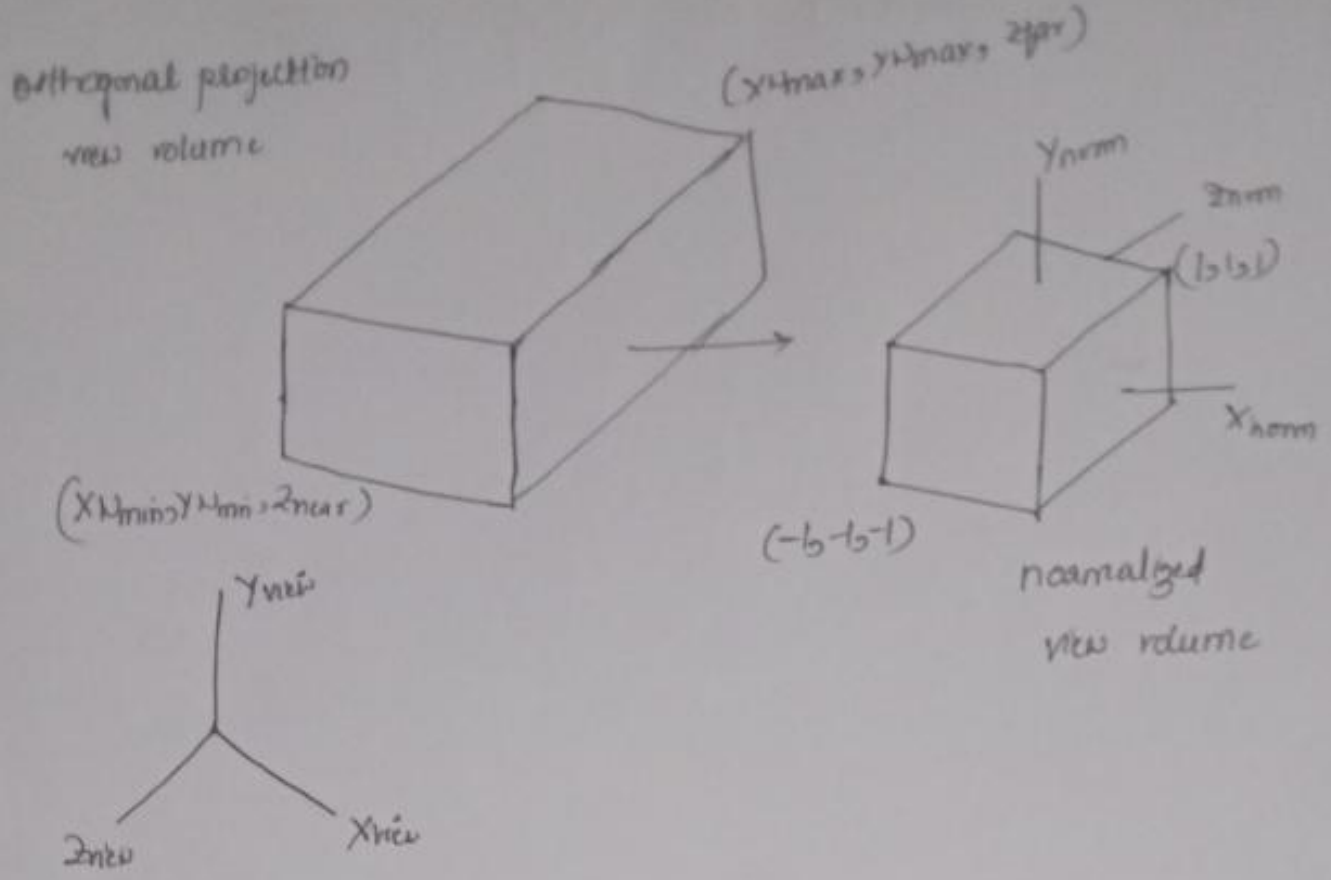
The $x_v y_v z$ coordinates normalized in range from 0 to 1

Another normalization transformation approach is to use a symmetric cube, with coordinates in the range from -1 to 1



we can convert projection coordinates into positions within a left handed normalized coordinate reference frame, and these coordinate positions will then be transferred to lefthanded screen coordinates by the viewport transformations.

Also 2-coordinate positions for the near and far planes are denoted as z_{near} and z_{far}, respectively.



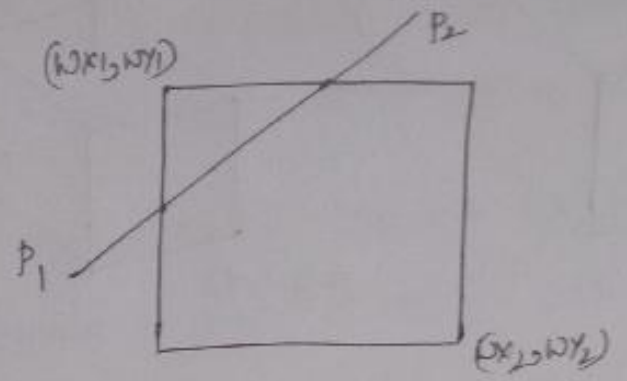
The normalization transformation for the orthogonal view volume is

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

(10) Explain Cohen-Sutherland line clipping algorithm

⇒ Algorithm:

1. Read the two end points of the line say $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$



2. Read two corners (left-top and right-bottom) of the window, say (WX1, WY1 and WX2, WY2)

3. Assign the region codes for two endpoints P_1 and P_2 using following steps:

Initialize code with bit 0000

set Bit 1 - if ($x < WX1$)

set Bit 2 - if ($x > WX2$)

set Bit 3 - if ($y < WY2$)

set Bit 4 - if ($y > WY1$)

4. check for visibility of line P_1P_2

a) the region codes for both endpoints P_1 and P_2 are zero then the line is completely visible

b) If region codes for endpoints are not zero and the logical ANDing of them is also non-zero then the line is

completely invisible

(c) If region codes for two endpoints do not satisfy the conditions in (a) & (b) the line is partially visible

5. Determine the intersecting edge of the clipping window by inspecting the region codes for two end points

(a) If region codes for both the end points are non-zero find intersection points P_1' and P_2' with boundary edge for clipping window with respect to P_1 and P_2 respectively

(b) If region code for any one end point is non-zero then find intersection point P_1' (or) P_2' with the boundary edge of clipping window with respect to it

6. Divide the line segments considering intersection points.

7. Reject the line segment if any one end point of it appears outside the clipping window

8. Draw the remaining line segments

9. Stop.