

(2)

to transform from world to screen coordinates.

`glMatrixMode(GL_PROJECTION);`

This designates the Projection matrix as the current matrix which is originally set to identity matrix.

ii) Glu Clipping - window function -

To define a 2D clipping window, we can use the OpenGL utility function:

`gluOrtho2D(Xwmin, Xwmax, Ywmin, Ywmax);
glViewport(Xvmin, Yvmin, Vpwidth, Vpheight);`

Create a glut display function -

`glutInit(&argc, argv);`

We have three functions in glut for definition of a display window and choosing its dimension and position.

`glutInitWindowPosition(Xtopleft, Ytopleft);
glutInitWindowSize(dwidth, dheight);
glutCreateWindow("Title of display window");`

iii) Setting the glut Display Mode of Window and color -

`glutInitDisplayMode(mode);
mode = GLUT_RGB | GLUT_SINGLE | GLUT_DEPTH, ...;
glClearColor(Red, Green, Blue, alpha);
glClearIndex(index);`

iv) glut Display Identifier -

`window ID = glutCreateWindow("display window");`

v) Current GLUT display window:

`glutSetWindow(window ID);`

vi) Other glut Viewing functions (2D)-

`glLoadIdentity();`

`glutDestroyWindow(windowID);`

`glutReshapeWindow(width, height);`

`glutFullScreen();`

`glutPushWindow();`

`glutPopWindow(); , etc.`

Q2 Build Phong Lighting Model with equations.

Ans: Phong reflection is an empirical model of local illumination.

It describes the way a surface reflects light as a combination of the diffuse reflection of rough surfaces with the specular reflection of shiny surfaces. It is based on Phong's informal observation that shiny surfaces have small intense specular highlights, while dull surfaces have small intense specular highlight while dull. Surfaces have large highlights, while dull surfaces have large highlights, while dull surfaces have large highlights that fall off more gradually.

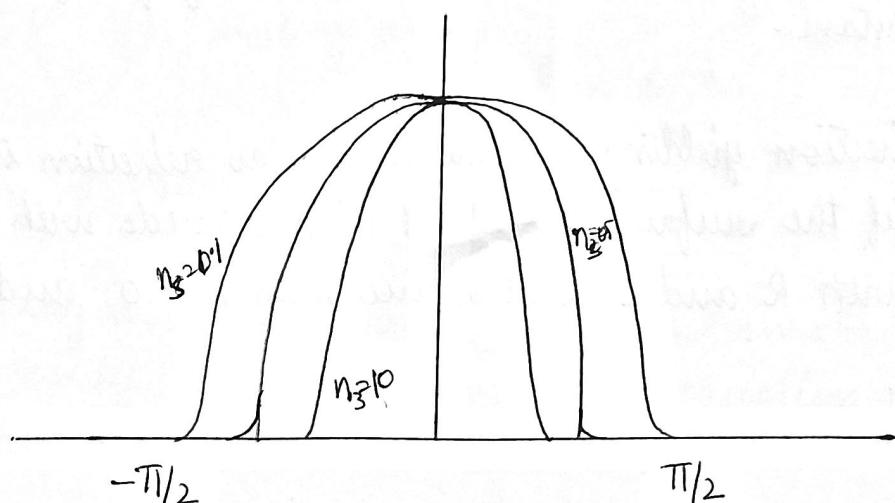


fig effect of the shininess value

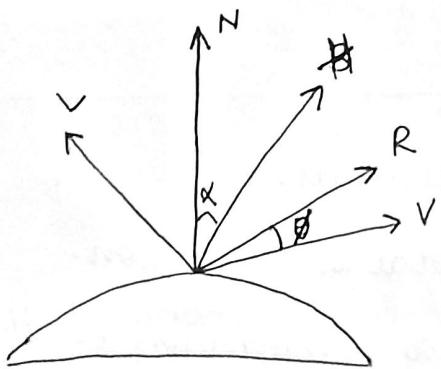
(4)

Phong model sets the intensity of specular reflection to $\cos^{n_s} \phi$.

$$I_{\text{specular}} = W(\theta) I_L \cos^{n_s} \phi, \quad 0 \leq W(\theta) \leq 1 \text{ is specular reflection coefficient.}$$

If light direction L and viewing direction V are on the same side of the normal N , or if L is behind the surface, specular effects do not exist.

For most opaque materials specular-reflection coefficient is nearly constant k_s .



$$I_{\text{specular}} = \begin{cases} k_s I_L (V \cdot R)^{n_s}, & V \cdot R > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$R = (2N \cdot L) N - L.$$

The normal N may vary at each point to avoid N computation. Angle ϕ is replaced with an angle α defined by a halfway vector H between L and V .

$$\text{Coefficient} \Rightarrow H = \frac{L + V}{|L + V|}$$

If the light source and viewer are relatively far from the object, α is constant.

H is the direction yielding maximum specular reflection in the viewing direction V if the surface normal N coincides with H . If V is coplanar with R and L and hence with N too and thus

$$\alpha = \frac{\phi}{2}.$$

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

Ans :- The three basic 2-D transformations are translation, rotation and scaling

$$P' = M_1 \cdot P + M_2$$

P' & P represents column vectors.

Matrix $M_1 \rightarrow 2 \times 2$ array containing multiplicative factors
 $M_2 \rightarrow 2$ elements column matrix containing translation term $\begin{bmatrix} x_t \\ y_t \end{bmatrix}$.

For translation, M_1 is identity matrix $P' + T$ where $T = M_2$.
 For rotation and scaling, M_2 contains translational terms associated with pivot point or scaling.

Homogenous coordinates

$$P' = P + T \quad \text{for translation where } T = \begin{bmatrix} tx \\ ty \end{bmatrix}$$

$$P' = R \cdot P \quad \text{for rotation where } R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$P' = S \cdot P \quad \text{for scaling where } S = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix}$$

$$\text{In general, } P' = M_1 \cdot P + M_2$$

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

In homogenous coordinate system, we combine multiplicative and translational terms by expanding the 2×2 matrix representation.

(6)

to 3×3 matrix representation. Also expand the matrix representations for coordination position.

We represent each Cartesian co-ordinate (x, y) with homogenous co-ordinate (x_h, y_h, h) where $x = x_h/h$, $y = y_h/h$

$$(h^*x, h^*y, h)$$

Set $h = 1$ then $(x, y, 1)$

Homogenous coordinates representation for translation, scaling and rotation are as follows:

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

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \rightarrow \text{Scaling}$$

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

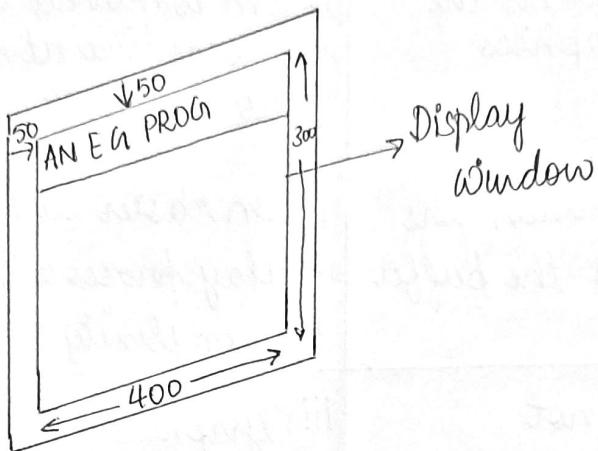
(7)

Q4 Outline the differences between Raster Scan displays and Random Scan displays.

RANDOM SCAN DISPLAY	RASTER SCAN DISPLAY
i) In random scan display, the beam is moved between the end points of the graphics primitives.	i) In raster scan display, the beam is moved all over the screen one scanline at a time from top bottom and then break to loop.
ii) Vector display flickers when the number of primitives in the buffer becomes too large.	ii) In raster display the refresh display process is independent of the complexity of the image.
iii) Scan conversion is not required.	iii) Graphics primitives are specified in terms of their endpoints and must be converted into their corresponding pixel in the frame buffers.
iv) Scan conversion hardware is not required.	iv) Because each primitive must be scan converted, real time dynamics is for more computational and required separate scan conversion hardware.
v) Vector display derives a continuous and smooth lines.	v) Raster display can display mathematically smooth lines, polygons and boundaries of curved primitives only by approximating them with pixels on the raster-grid.
vi) Cost is more.	vi) Cost is low.
vii) Vector display only draws lines and characters.	vii) Raster display has ability to display areas filled with solid colors or patterns.

Q5 Demonstrate OpenGL functions for displaying window management using GLUT?

Ans: GLUT supports two types of windows: top level windows and subwindows. Both types support OpenGL rendering and GLUT callbacks. There is a single identifier space for both types of windows.



- We perform the GLUT initialization with the statement
 $\underline{\text{glutInit}}(\&\text{argc}, \&\text{argv});$
- 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
 $\underline{\text{glutCreateWindow}}("An EG Prog");$
 where single argument for this function can be any character string.
- The following functions call the line segment description to the display window,
 $\underline{\text{glutDisplayFunc}}(\text{display});$
 ↳ line segment.
- The function $\underline{\text{glutMainLoop}}();$ must be the last one 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.
- The function $\underline{\text{glutInitWindowPosition}}(50, 100);$ specifies that the

Q5 Upper left corner of the display window should be placed 50 pixels to the right of the left edge of the screen and 100 pixels down from the top edge of the screen.

- The function `glutInitWindowSize(400, 300);` is used to set the initial pixel width and height of the display window.
- The function `glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);` specifies that a single refresh buffer to be used for the display window and that we convert to use the color mode which uses red, green and blue (RGB) components to select values of color.

Q6 Explain OpenGL Visibility Detection Functions?

Sol:- a) OpenGL Polygon-Culling Functions-

Back face removal is accomplished with the functions
 `glEnable(GL_CULL_FACE);`
 `glCullFace(mode);`

where mode is assigned the values `GL_BLACK`, `GL_FRONT`,
`GL_FRONT_AND_BACK`

By default, parameter mode in `glCullFace()` has 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 refresh buffer.

`glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);`

Depth buffer value can be initialized with

`glClear(GL_DEPTH_BUFFER_BIT);`

By default glClear is set to 10.

These routines are activated by following

glEnable(GL_DEPTH_TEST);

and can be deactivated using

glDisable(GL_DEPTH_TEST);

We can also apply depth buffer testing using some other initial values for maximum depth

glClearDepth(maxDepth);

- It can be set to any value between 0 and 1.

As an option, we can normalize values with

glDepthRange(near NormDepth, far NormDepth);

We specify a test condition for the depth buffer routines using

glDepthFunc(test condition);

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

glDepthMask(write status);

c) OpenGL Wireframe Surface Visibility Methods -

A wireframe 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);

But this displays both visible and hidden edges.

d) OpenGL Depth Culling function -

We can vary the bright of an object as a function of its distance from the viewing position with

glEnable(GL_FOG);

glFogf(GL_FOG_MODE, GL_LINEAR);

This applies the linear depth function to object colors using
 $d_{\min} = 0.0$ and $d_{\max} = 1.0$.

(11)

`glfogf(GL_FOG - START, minDepth);
 glfogf(GL_FOG - END, maxDepth);`

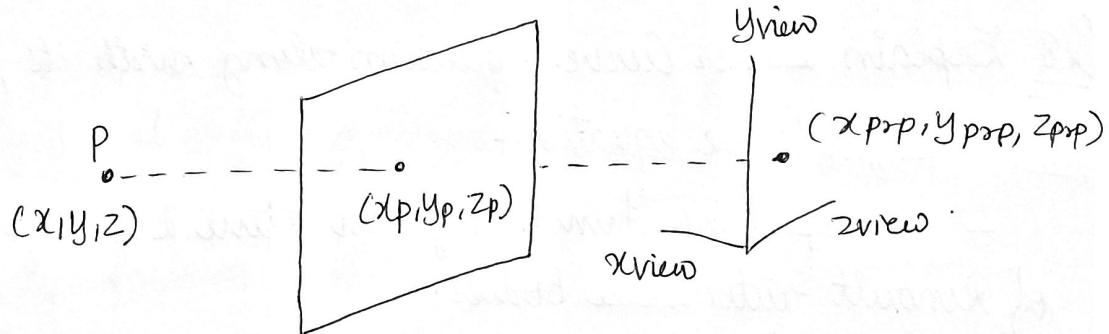
Q7. Write the special cases discussed with respect to Perspective projection transformation coordinates.

Ans:-

We know that,

$$x_p = x \left[\frac{z_{\text{prp}} - z_{\text{vp}}}{z_{\text{prp}} - z} \right] + x_{\text{prp}} \left[\frac{z_{\text{vp}} - z}{z_{\text{prp}} - z} \right]$$

$$y_p = y \left[\frac{z_{\text{prp}} - z_{\text{vp}}}{z_{\text{prp}} - z} \right] + y_{\text{prp}} \left[\frac{z_{\text{vp}} - z}{z_{\text{prp}} - z} \right]$$



Special Cases with respect to Perspective projection transformation

① If projection reference point is on Z_{view} $\Rightarrow x_{\text{prp}} = y_{\text{prp}} = 0$, then

$$x_p = x \left[\frac{z_{\text{prp}} - z_{\text{vp}}}{z_{\text{prp}} - z} \right]$$

$$y_p = y \left[\frac{z_{\text{prp}} - z_{\text{vp}}}{z_{\text{prp}} - z} \right]$$

② Sometimes the projection reference point is fixed at coordinate at origin and $x_{\text{prp}} = z_{\text{prp}} = y_{\text{prp}} = 0$. Then

$$x_p = x \left[\frac{z_{\text{vp}}}{z} \right], \quad y_p = y \left[\frac{z_{\text{vp}}}{z} \right]$$

(12)

③ If the view plane is uv plane and there are no restrictions on placement of projection reference point then we have

$$z_{vp} = 0 :$$

$$x_p = x \left(\frac{z_{pp}}{z_{pp}-z} \right) + (-y_{pp}) \left(\frac{z}{z_{pp}-z} \right)$$

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

④ With the uv plane as the view plane and the projection plane reference point on Z_{view} axis, the perspective equations are

$$x_{pp} = y_{pp} = z_{pp} = 0 :$$

$$x_p = x \left(\frac{z_{pp}}{z_{pp}-z} \right),$$

$$y_p = y \left(\frac{z_{pp}}{z_{pp}-z} \right).$$

Q8 Explain Bezier Curve equation along with its properties.

Ans :- Bezier Curve equation-

- Developed by French engineer Pierre Bezier for use in design of Renault automobile bodies.

- They have lots of properties that make them highly useful for curve and surface design. They are easy to implement.

- Bezier curve section can be fitted to any number of control points.

Equation:

$$P(u) = \sum_{k=0}^n p_k BEZ_{k,n}(u)$$

where n is no of control pts - 1.

k is $n+1$ (no of control points) varying from 0 to n

p_k is position vector coordinate (x_k, y_k, z_k) .

$BEZ_{k,n}(u)$ is Bernstein Polynomial and is defined as

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

and $0 \leq u \leq 1$.

eqn $P(u)$ can be represented as set of three parametric equation for the individual curve -

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

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

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

Properties-

- ① Basic functions are real.
- ② Degree of polynomial defining curve is one less than number of defining points.
- ③ Curve generally follows shape of defining polygon.
- ④ Curve will pass through ^{end} control point but not all control point.
- ⑤ Polygon equation depends on number of control points.
- ⑥ Curves lie within convex hull of the control points.

OQ Explain normalization transformation of Orthogonal Projection.

Ans:- Normalization transformation of Orthogonal projection can be defined as mapping coordinates into a normalized view volume with coordinates in range -1 to 1.

In this, 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 by z_{near} and z_{far} respectively.

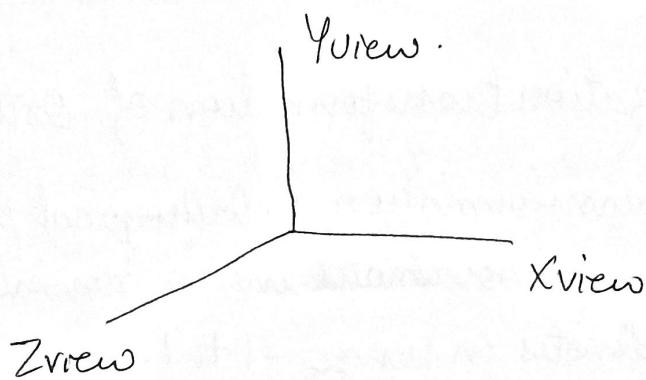
This position $(x_{\text{min}}, y_{\text{min}}, z_{\text{near}})$ is mapped to be the normalized position $(-1, -1, 1)$ and position $(x_{\text{max}}, y_{\text{max}}, z_{\text{far}})$ to $(1, 1, 1)$.

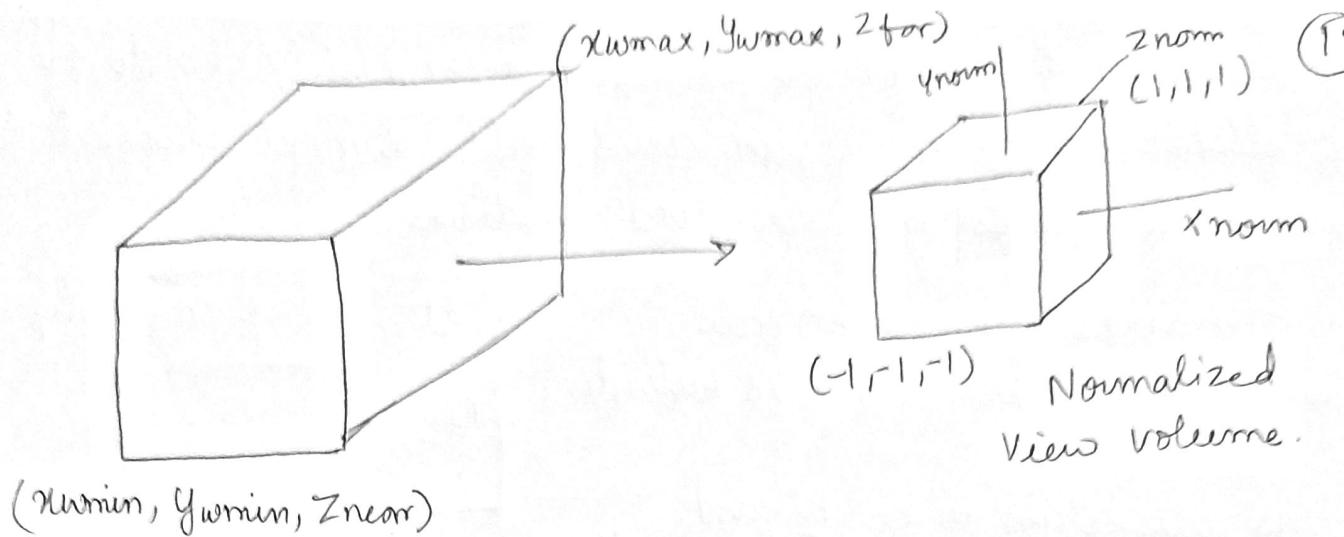
Transforming the rectangular parallelopiped is similar to the method for converting clipping window into the normalized symmetric square.

The normalization transformation for the orthogonal projection can be defined as -

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

The matrix is multiplied on the right by the composite viewing transformation $R T$ to produce the complete transformation from world coordinates to normalized orthogonal projection coordinates.





Q10 Explain Cohen-Sutherland line clipping algorithm.

Ans:- Every line endpoint 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 insider or outside the boundaries of clipping window.

TOP		
1001	1000	1010
0001	0000	0010
Left	Window clipping	Right
0101	0100	0110
		Bottom

Once we have established region codes for all lines endpoints, we can quickly determine which lines are completely within or outside the clipping window.

- * When the OR operation between 2 endpoints region code is false(0000) for a line segment, the line is inside the clipping window.
- * When AND operation between 2 endpoints region code for line segment is true, the line is completely outside the clipping window.

* lines that cannot be identified as completely inside or outside a clipping window by region codes tests are next checked for intersection points with window border lines ⑯

For example - the region code says P_1 is inside and P_2 is outside.

The intersection to be P_2'' and P_1' is clipped off.

For line P_3 to P_4 we define that Left Clipping

P_3 is outside left boundary & P_4 is inside therefore intersection of P_3 & P_4 is clipped off.

* To determine boundary point intersection for line equation y coordinate of intersection point with vertical clipping border line can be obtained as -

$$y = y_0 + m(x - x_0).$$

$x = x_{w\min}$ if left border

$x = x_{w\max}$ if right border

$$\text{and } m = \frac{y - y_0}{x - x_0}.$$

and similarly for x coordinate of intersection point with horizontal clipping border line is -

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

$y = y_{w\min}$ if bottom boundary

$= y_{w\max}$ if upper boundary.

