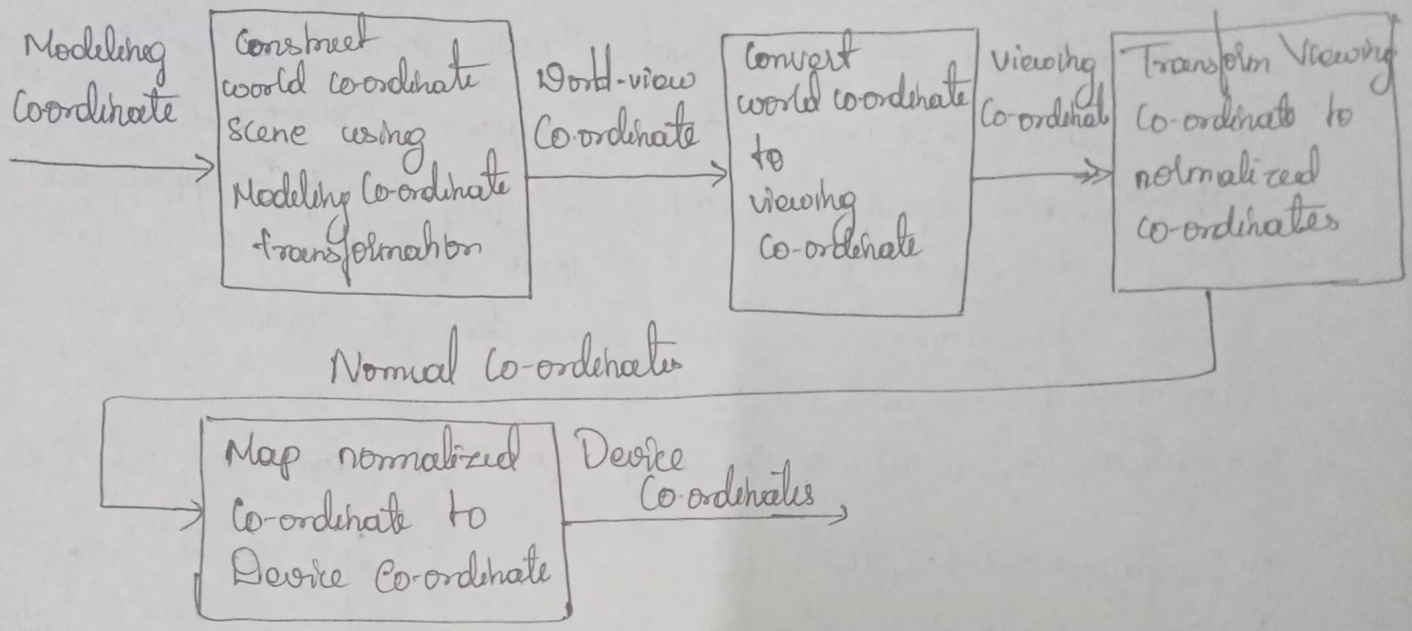
CGV Assignment

Name: Punith Kumar.B
USN: 1BY20CS140

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

ans



2D-Viewing functions

We can use different two dimensional routines along with the openGL viewing functions for all the viewing operations we need.

OPENGL Projection Mode:

Before we select a clipping window and a viewport in openGL, we need to establish the appropriate mode for constructing the matrix to transform from world co-ordinates to screen or viewing Co-ordinates

```
glMatrixMode(GL_PROJECTION);
```

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

3) glLoadIdentity():

This function ensures that each time we enter the projection mode, the matrix will be reset to identity matrix so that the new viewing parameters are not combined with the previous ones.

4) GLU clipping-Window Function:

To define a two-dimensional clipping window, we use

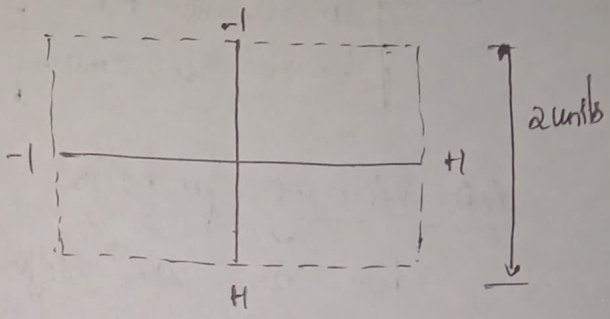
gluOrtho2D(xwmin, xwmax, ywmin, ywmax);

This function specifies an orthogonal projection for mapping the scene to the screen the orthogonal projection has no effect on our two-dimensional scene other than to convert object positions to normalized coordinates.

→ Normalized coordinates in the range from -1 to 1 are used in OpenGL

eg) gluOrtho2D(-1, +1, -1, +1);

(gluOrtho2D(Left, Right, Bottom, Top);



5) OpenGL Viewport Function:

→ We specify the viewport parameters with the OpenGL function

glViewport(xvmin, yvmin, vpWidth, vpHeight);

where

i) 'xvmin' and 'yvmin' specify the position of the lowerleft corner of the viewport relative to the lowerleft corner of the display window

ii) 'vpWidth' and 'vpheight' are pixel width and height of the viewport

→ We can obtain the parameters for the currently active viewport using the query function,

glGetIntegerv(GL_VIEWPORT, vpArray);

where,

vpArray[0] = xv_{min} vpArray[2] = vpWidth

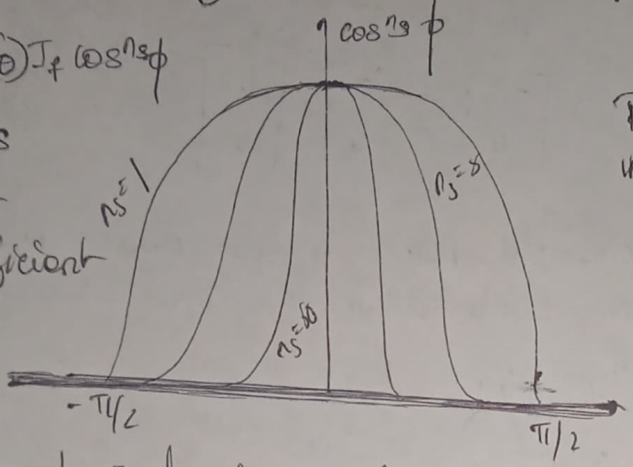
vpArray[1] = yv_{min} vpArray[3] = vpHeight

2. Build Phong Lighting Modul 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 surface. It is based on Phong's unformal observation that shiny surfaces have small intense specular highlights, while dull surfaces have large highlights, that fall off more gradually

If specular: $w(\theta) I_f \cos^{n_s}\phi$

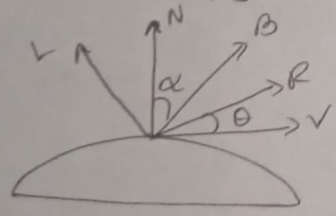
$0 \le w(\theta) \le 1$ is called specular reflection co-efficient



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

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 co-efficient is nearly constant ks



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

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

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

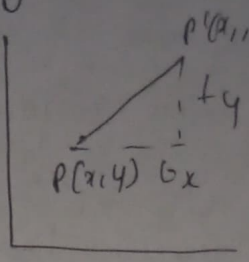
$$\text{Efficient} \Rightarrow \quad H = \frac{L+V}{|L+V|}$$

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 would coincide with H. if V is coplanar with R and L. $\alpha = \phi/2$

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

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



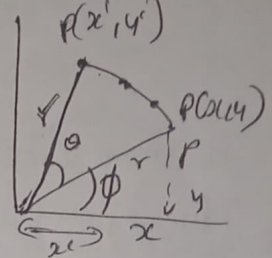
we can write the components :

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

in matrix form, $P' = P + T$

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

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



$\cos\phi = \frac{x}{r}$, $\sin\phi = y/r$

$x = r\cos\phi$ $\quad y = r\sin\phi$

$\Rightarrow \cos(\phi+\theta) = x'/r$.

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

Similarly for, $\sin(\phi+\theta) = y'/r$

$\rightarrow y' = r\cdot\sin(\phi+\theta)$

$y' = r\cdot\cos\phi\sin\theta + r\cdot\sin\phi\cos\theta$

$y' = x\cdot\sin\theta + y\cdot\cos\theta$

In matrix form,

$$P' = R \cdot P$$

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

Scaling

Components of scaling is given by,

$$P'_x = S_x \cdot P_x$$
$$P'_y = S_y \cdot P_y$$

Scale matrix as:

$$S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

or in matrix form:

$$P' = S \cdot P$$

Homogenous co-ordinates is given by

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

Rotation $P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

Scaling $P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

which can rewritten as

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

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, $P' = M_1 * P + M_2$

Homogenous co-ordinates can be represented as

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

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

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

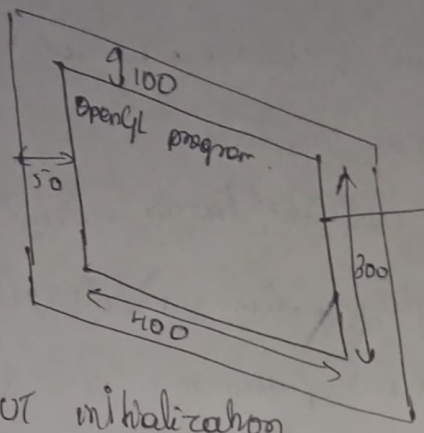
4. Outline the differences between raster scan displays and random scan displays ?

ans

Random Scan Display	Raster Scan Display
In vector scan display the beam is moved between the end points of the graphics primitive	In raster scan display the beam is moved all over the screen one scan line at a time, from top to bottom and then back to top
Vector display flickers when the number of primitives in the buffer becomes too large	In raster display, the refresh process is independent of the complexity of the image
Scan conversion is not required	Graphics primitives are specified in terms of their endpoints and must be scan converted into their corresponding pixel in the frame buffers
Scan conversion hardware is not required	Because each primitive must be scan-converted, real-time dynamics is far more computational and required separate scan conversion hardware
Vector display draws a continuous and smooth lines	Raster display can display mathematically smooth lines, polygons and boundaries of curved primitives only by approximating them with pixels on the raster grid
Cost is more	Cost is low
Vector display only draw lines and characters	Raster display has ability to display areas filled with solid colours or patterns

5. Demonstrate OpenGL functions for displaying window management using GLUT.

Any



-> We perform the GLUT initialization with the statement

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

-> 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");
```

-> The following function call the line segment description to the display window

```
glutDisplayFunc(lineSegment);
```

-> glutMainLoop();

this function 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.

-> glutInitWindowPosition(50,100);

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

-> GLU clipping-window Function:

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

gluOrtho2D (xwmin, xwmax, ywmin, ywmax);

Create a GLUT display window:

glutInit (&argc, argv);

Setting the GLUT display-window mode & color:

Various display window parameters are selected with the GLUT function:

glutInitDisplayMode(mode);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glClearColor (red, green, blue, alpha);
glClearIndex(index);

6.) Explain OpenGL visibility - Detection Functions?

ans a) OpenGL Polygon-culling Functions:

Back face removal is accomplished with the functions

glEnable (GL_CULL_FACE);
glCullFace (mode);

i) where parameter mode is assigned the value GL_BACK, GL_FRONT, GL_FRONT_AND_BACK

ii) By default, parameter mode in glCullFace function has the value GL_BACK

iii) 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 the refresh Buffer

glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

d) OpenGL_DEPTH_Cueing 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 and 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 cases that we discussed with respect to perspective projection transformation coordinates?

ans

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

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

Special Cases:

1.) $z_{prp} = y_{prp} = 0$

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

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

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

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

$$y_p = y\left(\frac{z_{vp}}{z}\right) \quad -(2)$$

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

3) $z_{vp} = 0$

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

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

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

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

$$\left.\begin{aligned} x_p &= x\left(\frac{z_{prp}}{z_{prp}-z}\right) \\ y_p &= y\left(\frac{z_{prp}}{z_{prp}-z}\right) \end{aligned}\right\} (4)$$

we get eq (4) with the uv plane as the view plane & the projection reference point on the z-view axis

8.) Explain Bézier curve equation along with its properties?

ans ⇒ 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 fitted to any number of control points

Equations:

$R_k = (x_k, y_k, z_k)$

R_k = General $(n+1)$ control-point positions

P_k = the position vector which describes the path of an approximate bezier polynomial function between P_0 and P_n

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

$BEZ_{k,n}(u) = C(n,k)\, u^k (1-u)^{n-k}$ is the Bernstein polynomial

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

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?

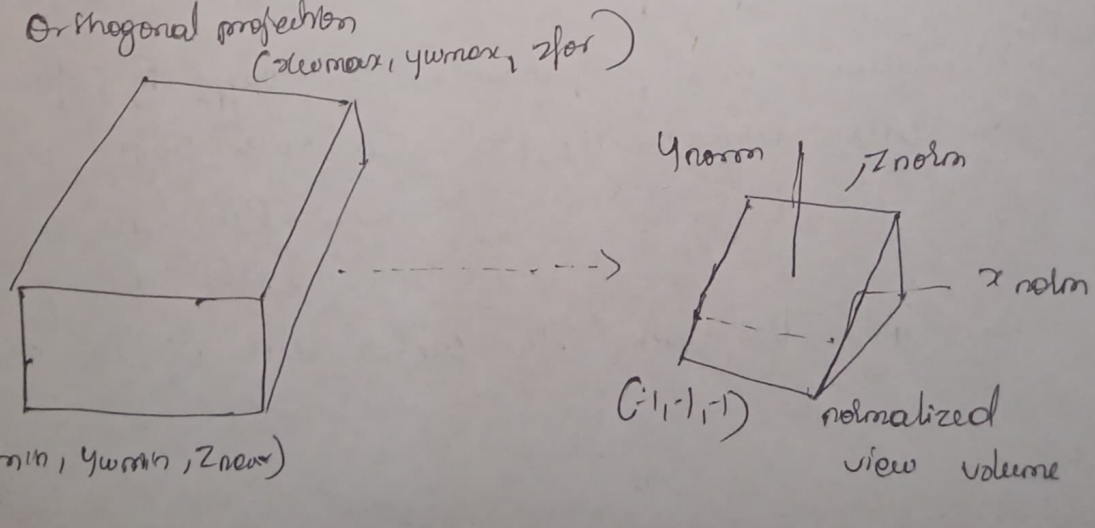
Any

— The normalization transformation we assume that the original 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 as z_{near} and z_{far} respectively. Thus 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 $(1, 1, 1)$.

← The normalized transformation for the Orthogonal view volume is

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

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

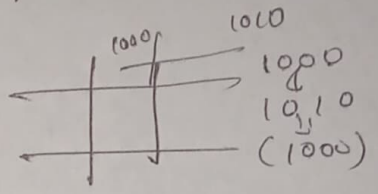


10) Explain Cohen-Sutherland line clipping algorithms?

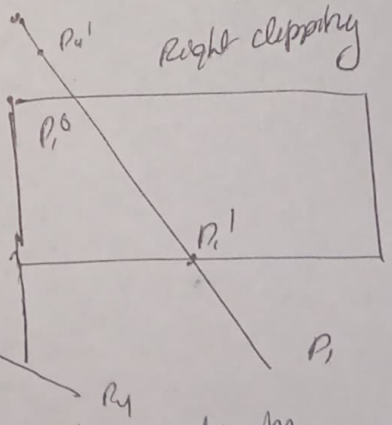
ans Every line endpoint in a picture is assigned a four digit binary value called a region code and each bit position is used to indicate whether the point is inside or outside of one of the clipping window boundaries

Once we have established region codes for all the endpoints we can quickly determine which line are completly within clip window & which are clearly outside

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



Lines that cannot be identified as lying completely inside (or) completely outside a clipping window by the region codes tests are next checked for intersection with window border lines



- The region codes says P_1 is inside and P_2 is outside

- The intersection to the P_2' & P_1' to P_2'' is clipped off for the 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 remainder of the line is below the clipping window & can be eliminated

To determine a boundary intersection for a line equation line y co-ordinate of intersection point with vertical clipping border line can be obtained by

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

where x is either xwmin or xwmax and slope is

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

For intersection with horizontal border, the x coordinate is

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