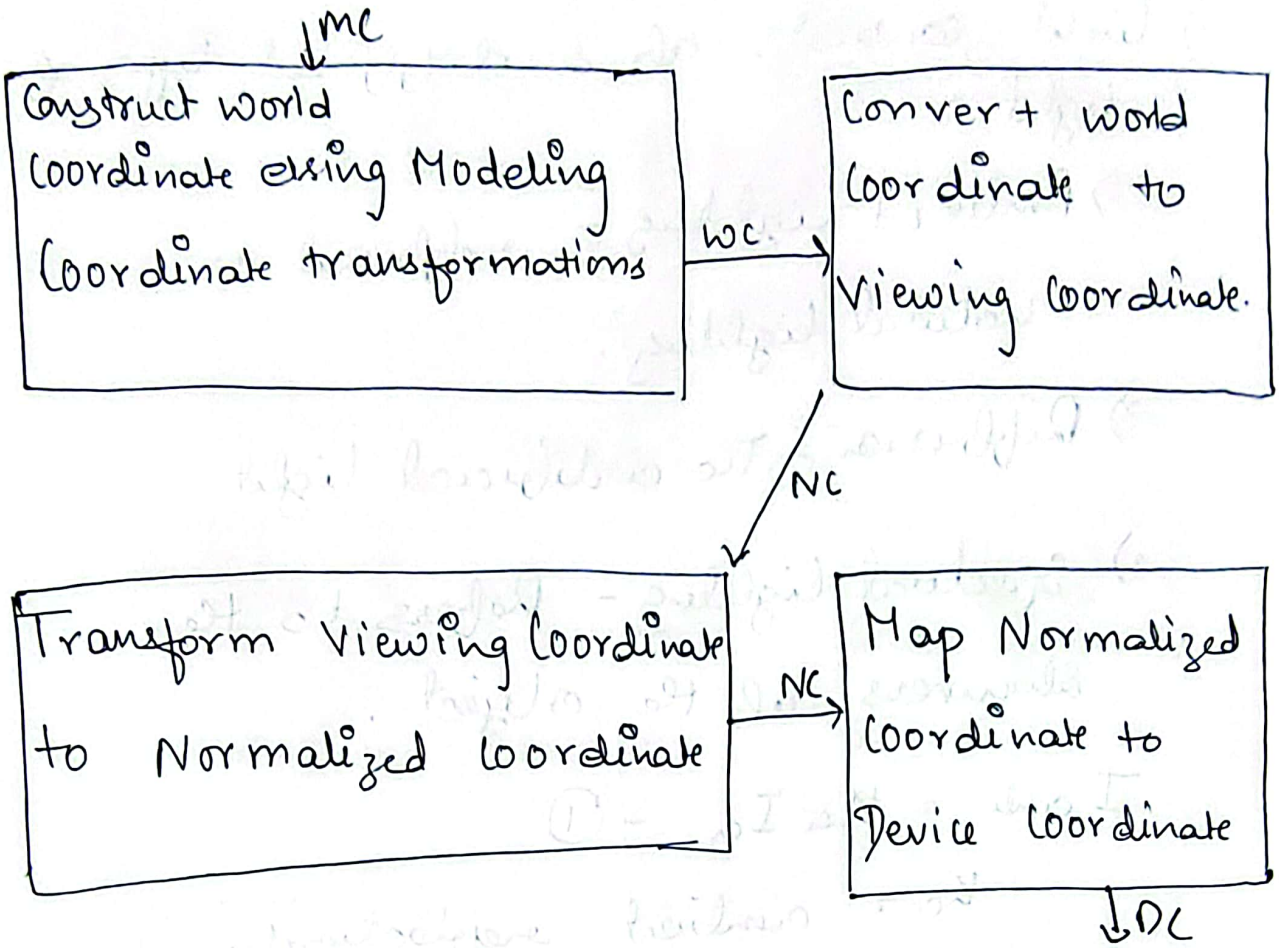
2D Pipeline

Prakyush Raj Shukla
1BY20CS139



- we could set up a separate 2d, viewing coordinate reference frame for specifying clipping window.
- Systems use normalized coordinates in the range from 0 to 1, others used a normalized range from -1 to 1.
- Clipping is usually performed in the normalized coordinates.

② Build a Phong Lighting Model with Eqns

1) Light consists of 3 different types of light :

   1) Ambient lighting - referred as natural lighting.

   2) Diffusion - The artificial light.

   3) Spectral lighting - Refers to the shininess of the object.

$I_{amb} = K_a I_a$ - ①

K_a : ambient reflectivity
I_a : intensity of ambient light

Similarly,

$I_{diff} = K_d I_p \cos(\theta)$ - ②
$= K_d I_p N.L$

$I_{spec} = K_s I_L \cos^n \phi$

∴ The Phong Model is the equation of all Combined

Total Intensity → $I = K_a I_a + K_d . I_p \cos\theta + K_s I_L \cos^n \phi$

3) Apply homogeneous coordinates for translation, rotation & scaling with matrix representation

→ The Matrix Representations of Translation Rotation & Scaling are :

$P' = P + T$

↓

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

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

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

Generic Eqn → $S.P$

$P' = M_1 * P + M_2$

But $x = \frac{x_h}{h}$, $y = \frac{y_h}{h}$ $\quad h = 1$

∴ consider $(h*x, h*y, h)$

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

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

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

④ Difference b/w Raster Scan & Random Scan

Raster Scan	Random Scan
1) Produces jagged lines that are plotted as a discrete point sets	1) Random System produces smooth line drawings
→ less expensive	→ More Expensive
→ Modification difficult	→ Modification easy
→ Resolution is low	→ Resolution is high
→ Solid pattern is easy to fill	→ Solid pattern is difficult to fill

⑤ Demonstrate OpenGL functions for Window management using GLUT.

→ glutCreateWindow - used to create a new window

→ glutCreateSubWindow - used to create another window within same window

- glut SetWindow - used to set an particular id for the window
- glut GetWindow - used to get the Window ID.
- glut DestroyWindow - to delete the window that was created.
- glut PostRedisplay - To display the window again again, continuously until forcibly closed.

glut ReShapeWindow - Used for transformation of world Coordinates to view Coordinates & displaying it.

glut FullScreen - To represent Window in Full Screen Mode.

glut PopWindow / glutPushWindow - works just like a matrix on window.

glut HideWindow - To hide the windows from being displayed on Screen.

glut DisplayFunc - To display

- glut MainLoop

Init()

5) OpenGL visibility Detection Functions

a) OpenGL polygon culling Functions:
Remove backface, frontface of an object

```
glCullFace(mode);
glEnable(GL_CULL_FACE);
glDisable(GL_CULL_FACE);
```

b) Depth-Buffer Functions

```
glutInitDisplayMode(GLUT_SINGLE |
GLUT_RGB | GLUT_DEPTH)
glClear(GL_DEPTH_BUFFER_BIT)
```

this works as initialization for each of the depth buffer & refresh buffer.

```
glDepthRange(nearNormDepth, farNormDep
glClear(GL_DEPTH_BUFFER_BIT)
glClearDepth(MaxDepth)
glEnable(GL_DEPTH_TEST)
glDisable(GL_DEPTH_TEST)
```

c) OpenGL wireframe surface visibility m

```
glPolygonMode(GL_FRONT_AND_BACK,
GL_LINE), - visible & hidden edges
```

display

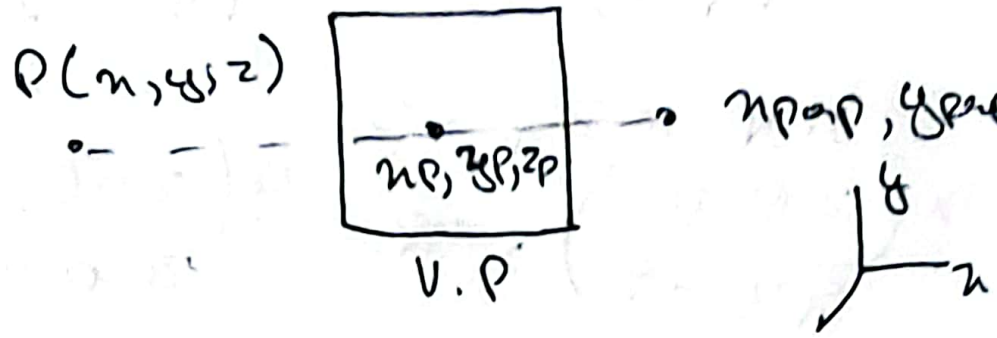
(k) OpenGL Depth Using fucs

glFogi (GL_FOG_MODE GL_LINEAR)

glEnable (GL_FOG)

To increase or decrease the Brightness

) Write special cases & discussed with Perspective Projections



Consider $x' = x - (x - x_{prp})u$

$y' = y - (y - y_{prp})u$

$z' = z - (z - z_{prp})u$

$u = \frac{z_{vp} - z}{z_{prp} - z}$

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

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

Special Cases - ① $x_{prp} > y_{prp} = 0$

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

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

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

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

③ $z_{vp} = 0$

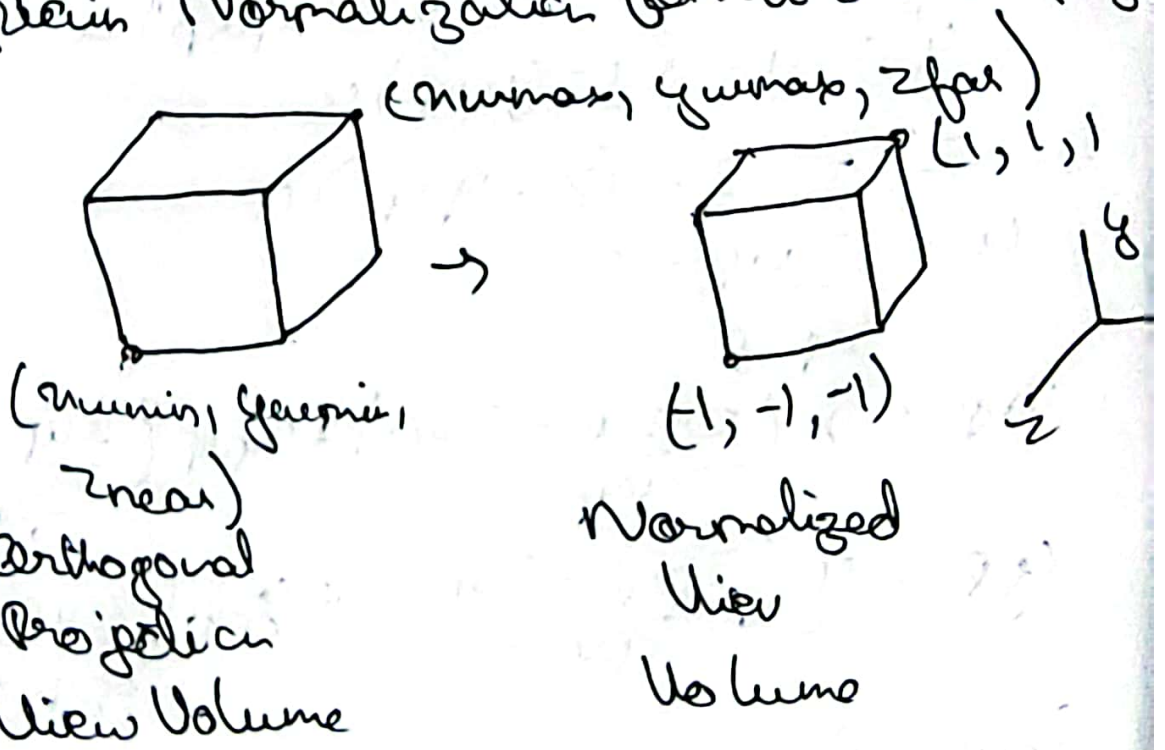
$x_p = x\left(\frac{z_{prp}}{z_{prp}-z}\right) - x_{prp}\left(\frac{z}{z_{prp}-z}\right)$

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

④ $x_{prp} = y_{prp} = z_{vp} = 0$

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

⑧ Explain Normalization for an ortho. proje



We consider a unitcube for this normal
view volume with each x y z coordinates
normalized in the range 0 to 1

And the normalization transformation approach is to use symmetric cube with coordinates

-1 to 1

∴ we get the normalization transformation for the orthogonal view volume

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

) Explain Bezier Curve & its properties with eqns

Bezier curves are parametric curves that are generated with the help of control points .. this widely used in graphics & other related industry.

They are named after the French Engineer Pierre Bezier who discovered it.

Bezier Curves are rep. as :

$$\sum_{k=0}^{n} P_i B_i^n(t)$$

$B_i^n(t)$ represents Bernstein Polynomial

$$B_{i,n}(t) = \binom{n}{i}(1-t)^{n-t}t^i$$

n - polynomial degree

t - variable

i - index

They are be of 2 control points - linear

curve 3 - control points - cubic curve

4 control points - quadratic curves

we used the above mentioned formulas

Bezier curve $= \binom{n}{i} * (1-t)^{n-t} t^i$ ⇒ for curve point

n = control points number - 1

t = 0 - 1 (Range)

(19) Cohen - Sutherland line Clipping Algo

→ Cohen Sutherland algo works on Region Code

→ RegionCode is 4-bit code

(TBRL)

(TBRL) → Top, Bottom, Right, Left

1001	1000	1010
0001	0000	0010
0101	0100	0110

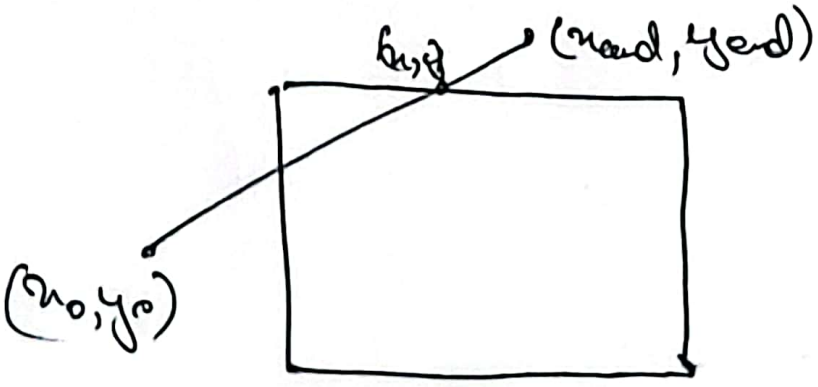
for a line - (x_0, y_0) to (x_{end}, y_{end})

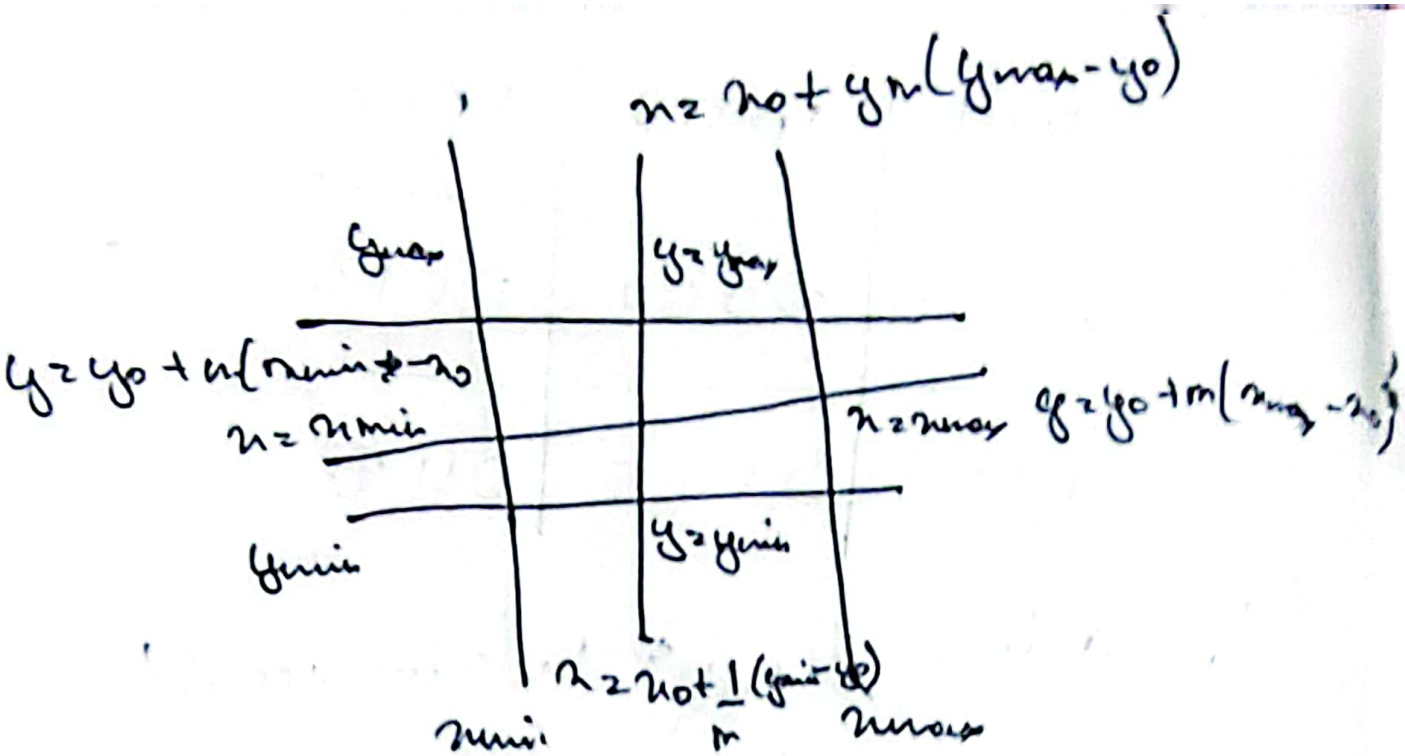
$$m = (y - y_0) / (x - x_0)$$

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

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

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





Thus the above formulas to be applied when a particular line needs to be clipped.