Geometric Transformations in 3-D space.

2D rotation – rotations about axes that were perpendicular to xy plane.

3D rotation – select any spatial orientation for rotation axis.

3D Translation → how much the object is to be moved in each of the three co-ordinate direction.

3D Scaling → scale an object by choosing a scaling factor for each of three cartesian coordinates.

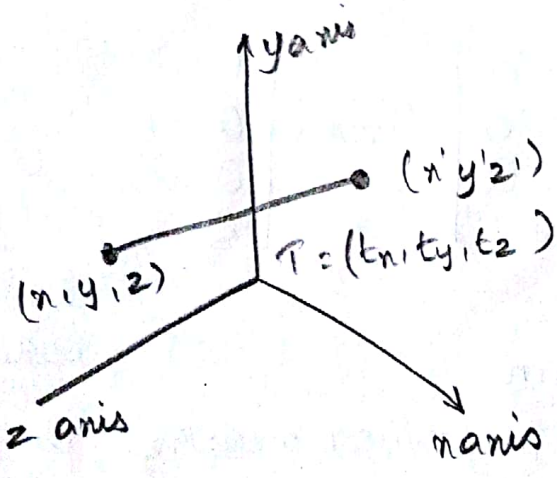
Three Dimensional Translation.

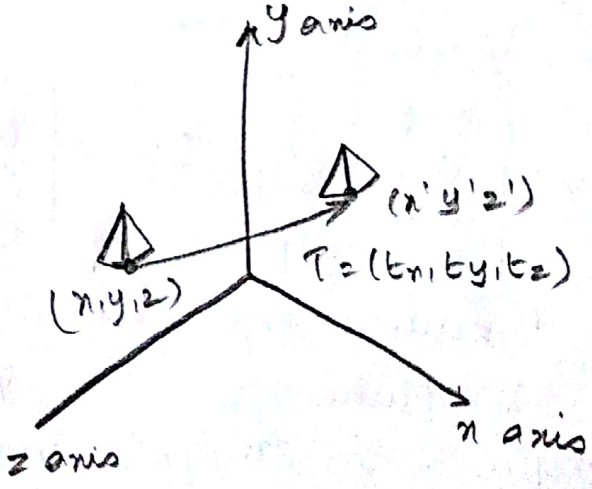
A position $P = (x, y, z)$ in 3D space is translated to a location $P' = (x', y', z')$ by adding translation distances t_x, t_y & t_z.

$$x' = x + t_x \quad y' = y + t_y \quad z' = z + t_z$$

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

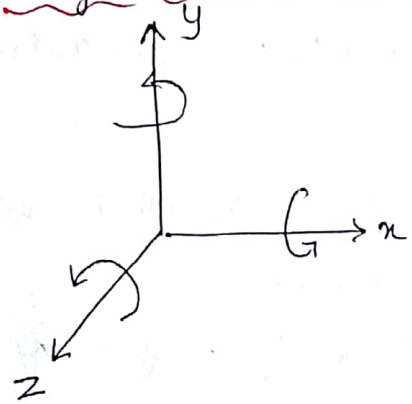
$$P' = T \cdot P$$



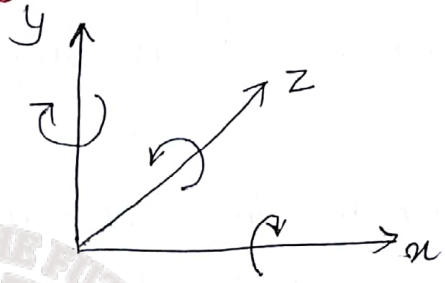


3D → depth of the object that we are viewing.
↳ projection geometry (object taken away becomes small object bringing closer becomes zoomed to viewing eye)

Right handed Space



Left handed Space



Translation matrix in 3D

$$A = \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ g & i & j & r \\ l & m & n & s \end{bmatrix} = \begin{bmatrix} T & K \\ V & 0 \end{bmatrix}$$

$$T = \begin{bmatrix} a & b & c \\ d & e & f \\ g & i & j \end{bmatrix}$$ produce linear transformation: Scaling, shearing, reflection & rotation

$K = [p\ q\ r]^T$, produces translation

$V = [l\ m\ n]^T$, yields perspective transformation

s – responsible for uniform scaling in 3D

3D Reflection

$$T_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_{yz} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_{zx} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Produces reflection about :

XY plane YZ plane ZX plane

A reflection in 3D space can be performed relative to a selected reflection axis or w.r.t reflection plane

Rotation Matrices along an axis.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

x - axis.

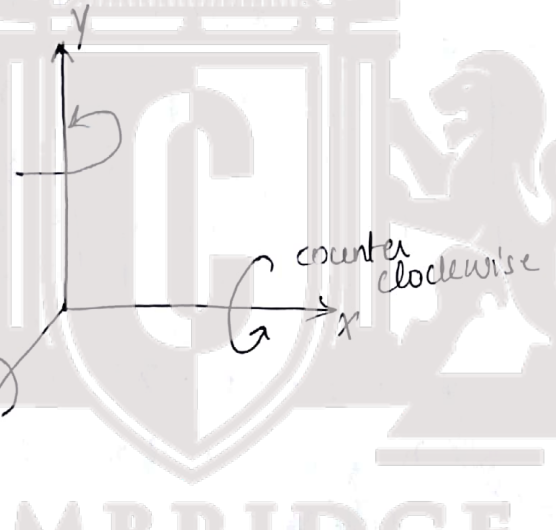
$$\begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Y-axis.

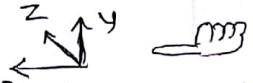
$$\begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Z-axis

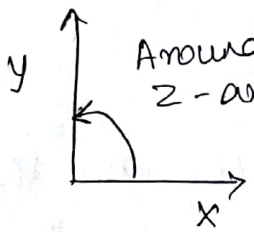
Why is the sign reversed in one case.



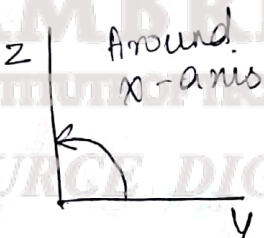
+ve axis



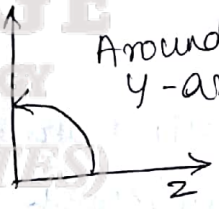
Thumb pointing to arrow from origin; direction is known from the direction of rotatn



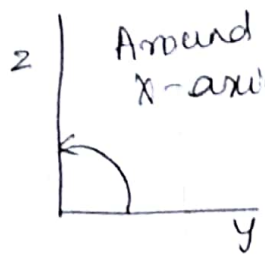
Anti clockwise



Anticlockwise



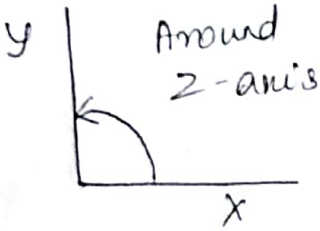
Anti clockwise



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} \leftarrow x \\ \leftarrow y \\ \leftarrow z \end{matrix}$$

from Y to Z

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

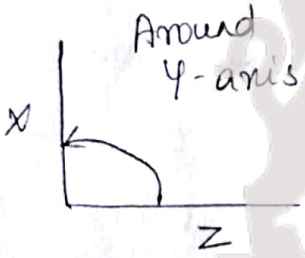


$$\begin{bmatrix} \cos r & -\sin r & 0 & 0 \\ \sin r & \cos r & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} \leftarrow x \\ \leftarrow y \\ \leftarrow z \end{matrix}$$

from X to Y

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

if -ve sign is here, the rotation would be from X to Z



$$\begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} \leftarrow x \\ \leftarrow y \\ \leftarrow z \end{matrix}$$

from Z to X

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

generalising α, β, γ with θ.

General 3D Rotation

An object is to be rotated about the axis that is ||el to one of co-ordinate axis, sequence is

1) Translate the object so that the rotation axis co-incides with the ||el co-ordinate axis
2) Perform the specified rotation about that axis.
3) Translate the object so that the rotation axis is moved back to its original position.

$$P' = T^{-1} \cdot R_x(\theta) \cdot T \cdot P$$

$$R(\theta) = T^{-1} \cdot R_x(\theta) \cdot T$$

General 3D Rotations

Rotation about an Arbitrary Axis in space

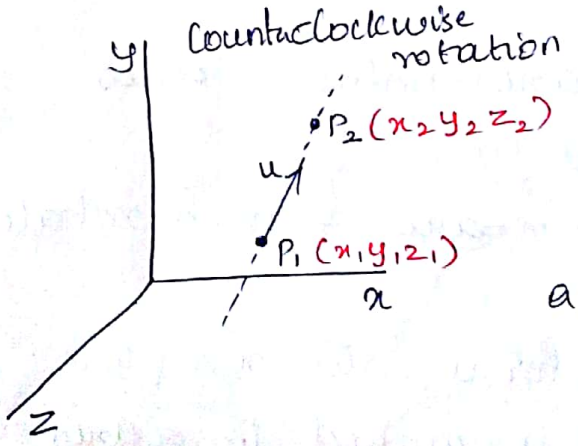
Assume we want to perform a rotation by θ degrees, about an axis in space passing through the point $(x_0\ y_0\ z_0)$ with direction cosines (c_x, c_y, c_z)

1. First translate the object so that the rotation axis coincides with the parallel co-ordinate axis.

*.

1. Translate the object so that the rotation axis passes through the co-ordinate origin.
2. Rotate the object so that the axis of rotation co-incides with one of the co-ordinate axis.
3. Perform the specified rotation about the selected co-ordinate axis.
4. Apply inverse rotations to bring the rotation axis back to its original position/orientation.
5. Apply the inverse translation to bring the rotation axis back to its original spatial position.

* transform the rotation axis onto any one of the three co-ordinate axis. The 'z' axis is often convienient choice



Direction of rotation is counter clockwise from P_2 to P_1

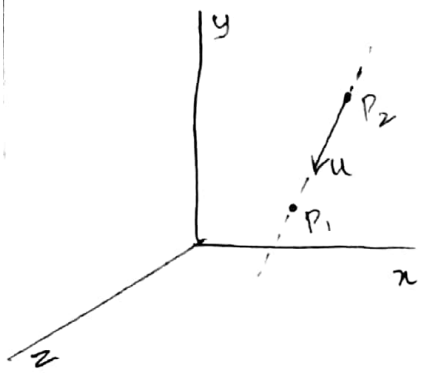
The components of rotation axis vector are computed as

$$V = P_2 - P_1$$
$$= (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

Unit rotation axis vector u is

$$u = \frac{V}{|V|} = (a, b, c)$$

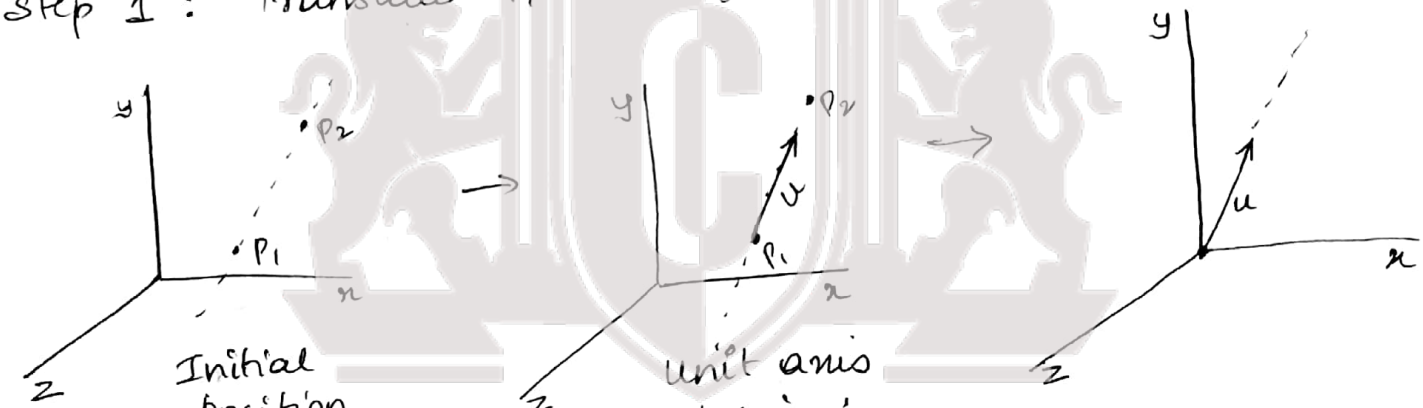
$$a = \frac{x_2 - x_1}{|V|}, \quad b = \frac{y_2 - y_1}{|V|}, \quad c = \frac{z_2 - z_1}{|V|}$$



clockwise rotation

Vector u points in the direction from P_2 to P_1

Step 1 : Translate P_1 to origin



in counter clockwise

$$T = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

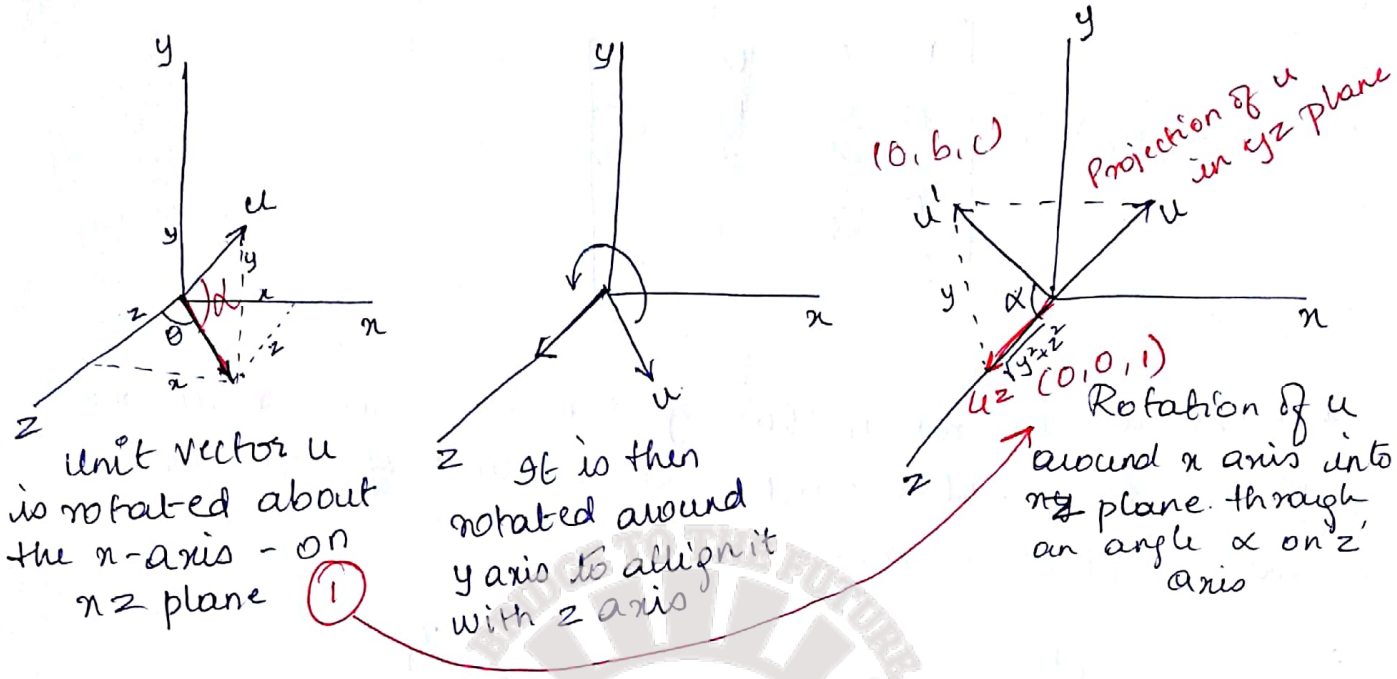
Step 2 : Place the rotation axis onto z axis

This can be accomplished by

1) first rotate about the x-axis & then rotate about the y axis.

The x-axis rotation gets vector u into xz plane.

y-axis rotation swings u around to z axis



* Vector dot product → to determine cosine term.
* Vector cross product → to calculate sine term
* Rotation angle α is the angle between the projection of u in yz plane & the positive z axis.
* projection of u in yz plane as the vector $u' = (0, b, c)$

i.e $\cos\alpha = \dfrac{u' \cdot u_z}{|u'||u_z|} = \dfrac{c}{d}$ $\left\{\dfrac{\text{adjacent}}{\text{hypo}}\right\}$

where $d = \sqrt{b^2 + c^2}$ magnitude of u'

* determine the $\sin\alpha$ — cross product of u' & u_z

$$u' \times u_z = u_x |u'||u_z| \cdot \sin\alpha$$

$$u' \times u_z = u_x \cdot b \quad \text{substitute}$$

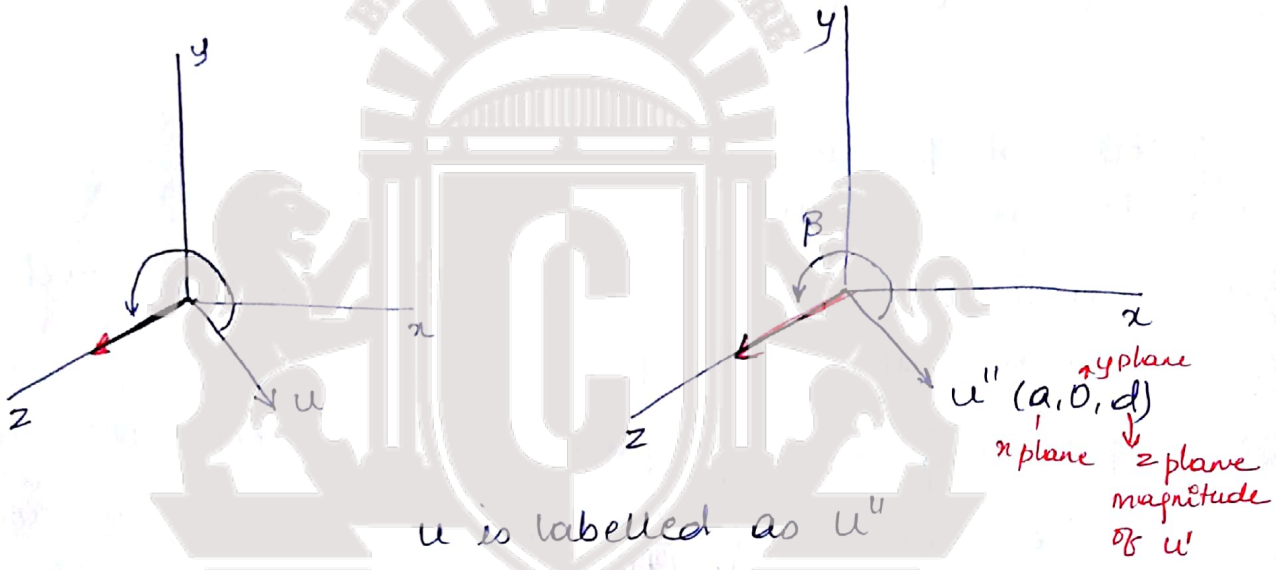
$$u_x \cdot b = u_x \cdot d \cdot \sin\alpha$$

$$b = d \cdot \sin\alpha$$

$$\sin\alpha = \frac{b}{d}$$

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & -\frac{b}{d} & 0 \\ 0 & \frac{b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

* Next is to rotate/swing the unit vector in xz plane ~~to~~ counter clockwise around y axis onto the positive z axis.



rotation angle β since $|u_z| = |u''| = 1$

$$\cos\beta = \frac{u'' \cdot u_z}{|u''| \cdot |u_z|} = d$$

determining sinβ

$$u'' \times u_z = u_y |u''| |u_z| \sin\beta$$

$$u'' \times u_z = u_y \cdot (-a) \quad \text{Subst}$$

hence. $u_y \cdot (-a) \cdot u_y \cdot 1 \cdot \sin\beta$

$$\sin\beta = -a.$$

$$R_y(\beta) = \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 3: Rotate the object with angle θ around the 'z' axis

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

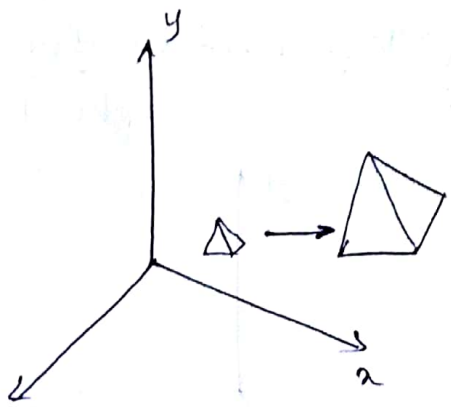
Step 4: Rotate the Axis to its original Orientation.

Step 5: Apply inverse transform to translate the rotation axis to its original position

$$\therefore R(\theta) = T^{-1} R_x^{-1}(\alpha) R_y^{-1}(\beta) . R_z(\theta) . R_y(\beta) . R_x(\alpha) . T$$

Three Dimensional Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Doubling the size of the object with transformation also moves the object farther from the origin.

$P' = S \cdot P$ $(S_x S_y S_z)$ → scaling parameters

$x' = S_x \cdot x \quad y' = S_y \cdot y \quad z' = z \cdot S_z$

* parameter value greater than 1 moves a point farther from the origin
* parameter value less than 1 moves a point closer to the origin.
* If scaling parameter are all not equal, relative dimensions of a transformed object are changed.

Scaling transformation w.r.t fixed point (x_f, y_f, z_f).

Sequence is

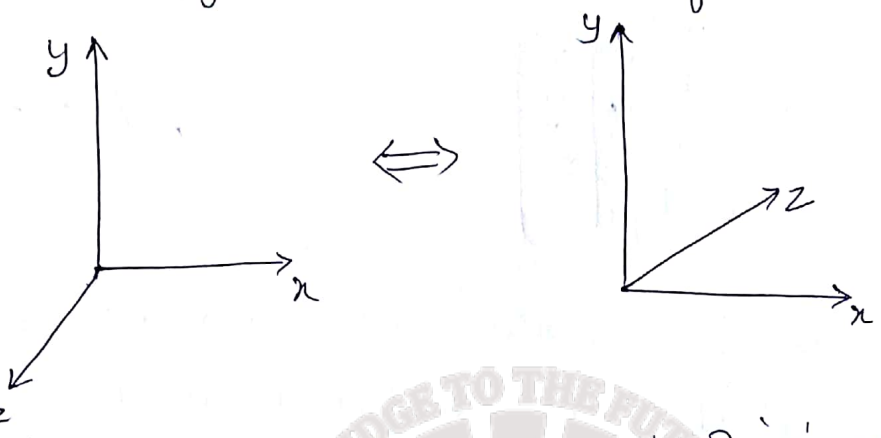
1) Translate the fixed point to origin.
2) Apply the scaling transformation relative to the co-ordinate origin.
3) Translate the fixed point back to its original position

$$T(x_f, y_f, z_f) \cdot S(S_x, S_y, S_z) \cdot T(-x_f, -y_f, -z_f)$$

$$= \begin{bmatrix} S_x & 0 & 0 & (1-S_x)x_f \\ 0 & S_y & 0 & (1-S_y)y_f \\ 0 & 0 & S_z & (1-S_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D reflection Continued.

Right handed → left handed



This transformation changes the sign of 'z' co-ordinates, leaving the values of x & y co-ordinates unchanged.

3D Sheans

- used to modify object shapes.
- applied in 3D viewing transformation for perspective projection.
- sheans relative to z-axis

$$M_{z\,shear} = \begin{bmatrix} 1 & 0 & Sh_{zx} & -Sh_{zx} \cdot z_{ref} \\ 0 & 1 & Sh_{zy} & -Sh_{zy} \cdot z_{ref} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Affine Transformations

A co-ordinate transformation of the form

$$x' = a_{xx}x + a_{xy}y + a_{xz}z + b_x$$
$$y' = a_{yx}x + a_{yy}y + a_{yz}z + b_y$$
$$z' = a_{zx}x + a_{zy}y + a_{zz}z + b_z$$

is called an affine transformation. Each of the transformed coordinates x', y' and z' is a linear function of the original co-ordinates x, y & z.

parameters a_{ij} & b_k are constants determined by the transformation type.

General properties of affine transformations

- parallel lines are transformed into parallel lines.
- finite points map to finite points.

Ex of Affine transformations : Translation, rotation, scaling, reflection & shear.

Basic OpenGL geometric Transformations

- A 4x4 translation matrix is constructed with the following routine

```
glTranslate* (tx, ty, tz);
```

Ex : glTranslatef (25.0, -10.0, 0.0);

translate 25 units in x direction and -10 units in y-direction.

- A 4x4 rotation matrix is generated with

```
glRotate* (theta, vx, vy, vz);
```

vector (vx, vy, vz) defines the orientation for a

rotation axis that passes through the co-ordinate origin.

glRotatef(90.0, 0.0, 0.0, 1.0);

sets up the matrix for a 90° rotation about the z-axis.

- A 4×4 scaling matrix.

glScale* (sx, sy, sz)

This function will also generate reflections when -ve values are assigned to scaling parameters.

glScalef(2.0, -3.0, 1.0);

- It produces a matrix that scales by a factor of 2 in x direction, scales by a factor of 3 in y-direction & reflects with respect to x axis.

** Quaternion Methods for 3-D Rotations

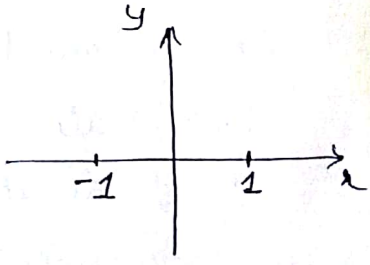
Quaternions are extensions of 2D complex numbers, more often important in animations that requires complicated motion sequences & motion interpolations between two given positions of an object.

Quaternions are ordered pair, consisting of a scalar part & vector part. $q = (s, v)$.

Basics of complex numbers : $a = x + iy$

$x \rightarrow$ real part, y = imaginary part,

$i = \sqrt{-1}$



Imaginary unit

A pure imaginary number with $y = 1$ is called imaginary unit & is denoted as $i = (0, 1)$

$i^2 = (0,1)(0,1)$

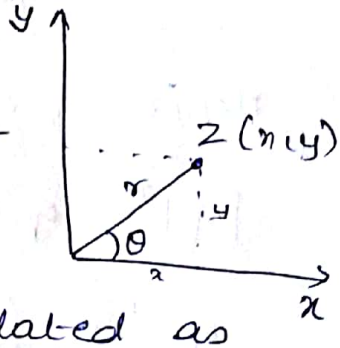
product of two complex number

$(x_1,y_1)(x_2,y_2) = (x_1x_2 - y_1y_2,\ x_1y_2 - x_2y_1)$

$i^2 = (0,1)(0,1) = (-1,0)$

Therefore i^2 is a real number -1, Hence $i = \sqrt{-1}$

Polar-Cordinate representation of complex number.



$x = r\cos\theta$, $y = r\sin\theta$

$z = r(\cos\theta + i\sin\theta)$ where $r = \sqrt{x^2+y^2}$

i.e $x^2+y^2 = r^2\cos^2\theta + r^2\sin^2\theta$

$\boxed{x^2+y^2 = r^2}(\cos^2\theta + \sin^2\theta)$

Hence $r = \sqrt{x^2+y^2}$ & θ can be calculated as

$\frac{y}{x} = \tan\theta \Rightarrow \theta = \tan^{-1}(y/x)$

$z = r(\cos\theta + i\sin\theta)$ can be written as $z = re^{i\theta}$

where $e^{i\theta} = \cos\theta + i\sin\theta$

Hence complex numbers ~~can~~ are extended to higher dimensions using Quaternions.

$q = s + ia + jb + kc$ $\quad v = ia + jb + kc$

a, b, c - co-efficients in the imaginary part.

s - real part called scalar part.

$i^2 = j^2 = k^2 = -1$, $ij = -ji = k$

$jk = -kj = i$

$ki = ik = j$

$q_1 + q_2 = (s_1+s_2) + i(a_1+a_2) + j(b_1+b_2) + k(c_1+c_2)$

Hence for Quaternion of form $q = (s, v)$

$q_1 + q_2 = (s_1+s_2,\ v_1+v_2)$

$q_1 \cdot q_2 = (s_1s_2 - v_1 \cdot v_2,\ s_1v_2 + s_2v_1 + v_1 \times v_2)$

$(s_1,v_1)\cdot(s_2,v_2)$

Ref $q = s + ai + bj + ck$.

$q = (s, v)$

$|q|^2 = q \cdot q = (s, ai + bj + ck) \cdot (s, ai + bj + ck)$

i.e $q \cdot q = (ai + bj + ck)(ai + bj + ck)$

Here $i.j = 0$, $j.k = 0$, $i.k = 0$, $i.i = -1$, $j.j = -1$,

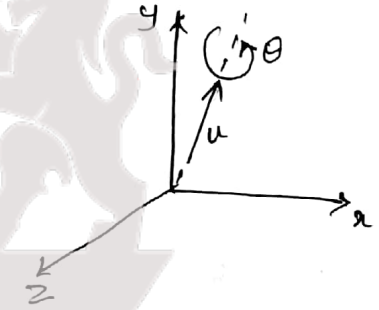
$k.k = -1$, $i.j.k = -1$

Hence $q \cdot q = a^2 + b^2 + c^2$.

$\therefore |q|^2 = q \cdot q = s^2 + v \cdot v$

$q^{-1} = \frac{1}{|q|^2}(s, -v)$

so that $q \cdot q^{-1} = q^{-1} \cdot q = (1, 0)$



$q = (s, v)$

where $s = \cos\frac{\theta}{2}$, $v = u \sin\frac{\theta}{2}$

any point position P that is rotated by this quaternion can be represented in quaternion notation as

$P = (0, p)$

'p' is vector part, $p = (x, y, z)$. The rotation of point is then carried out with quaternion operation

$P' = q P q^{-1}$ 'q' is a quaternion operation, get it to origin rotate & replace it where $q^{-1} = (s, -v)$

This transformation produces a new quaternion

$P' = (0, p')$

where $p' = s^2 p + v(p.v) + 2s(v \times p) + v \times (v \times p)$

Hence the overall composite rotation matrix.

$R_x^{-1}(\alpha) \cdot R_y^{-1}(\beta) \cdot R_z(\theta) \cdot R_y(\beta) \cdot R_x(\alpha)$ in a 3 by 3 form as

$$M_R(\theta) = \begin{bmatrix} 1-2b^2-2c^2 & 2ab-2sc & 2ac+2sb \\ 2ab+2sc & 1-2a^2-2c^2 & 2bc-2sa \\ 2ac-2sb & 2bc+2sa & 1-2a^2-2b^2 \end{bmatrix}$$

Use trignometrical identities to simplify the terms.

$$\cos^2\frac{\theta}{2} - \sin^2\frac{\theta}{2} = 1 - 2\sin^2\frac{\theta}{2} = \cos\theta.$$

&

$$2\cos\frac{\theta}{2}\sin\frac{\theta}{2} = \sin\theta.$$

Using this rewrite the matrix.

$\therefore M_R(\theta) =$

$$\begin{bmatrix} u_x^2(1-\cos\theta)+\cos\theta & u_x u_y(1-\cos\theta)-u_z\sin\theta & u_x u_z(1-\cos\theta)+u_y\sin\theta \\ u_y u_x(1-\cos\theta)+u_z\sin\theta & u_y^2(1-\cos\theta)+\cos\theta & u_y u_z(1-\cos\theta)-u_x\sin\theta \\ u_z u_x(1-\cos\theta)-u_y\sin\theta & u_z u_y(1-\cos\theta)+u_x\sin\theta & u_z^2(1-\cos\theta)+\cos\theta \end{bmatrix}$$

A complete quaternion rotation expression.

$$R(\theta) = T^{-1} \cdot M_R \cdot T.$$

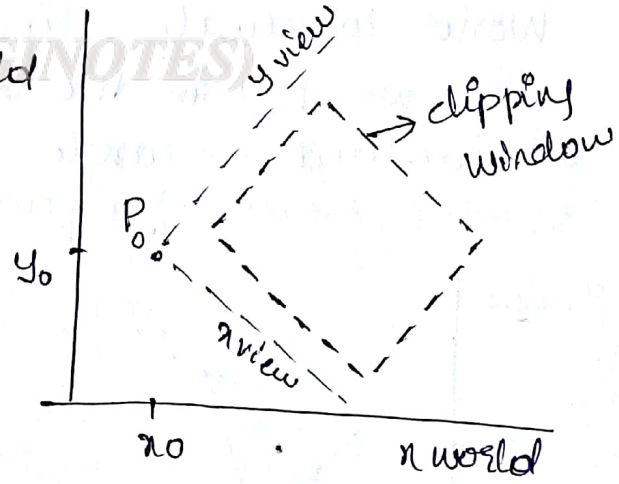
The clipping window

- clipping window can be designed with any shape, size & orientation as we choose.
- clipping with star patterns, an ellipse, or a figure with spline boundary requires more processing than clipping against a rectangle.
- It is hard to determine where the objects intersects a circle than to find out where it intersects a straight line.
- More often graphics packages commonly allow only rectangular clipping windows alligned with x & y axes.
- Rectangular clipping windows in standard position are easily defined by giving the co-ordinates of two opposite corners of each rectangle.

Viewing-coordinate clipping window.

- 2D viewing transformation - is to set up a viewing coordinate system within the world-coordinate frame.
- It provides (viewing frame) provides a reference for specifying a rectangular clipping window with any selected orientation and position.

- We choose the origin for a 2D viewing co-ordinate frame at some world position $P_0 = (x_0, y_0)$ & establish the orientation using vector V that defines the y_{view} direction.



Vector V is called 2D view up vector.

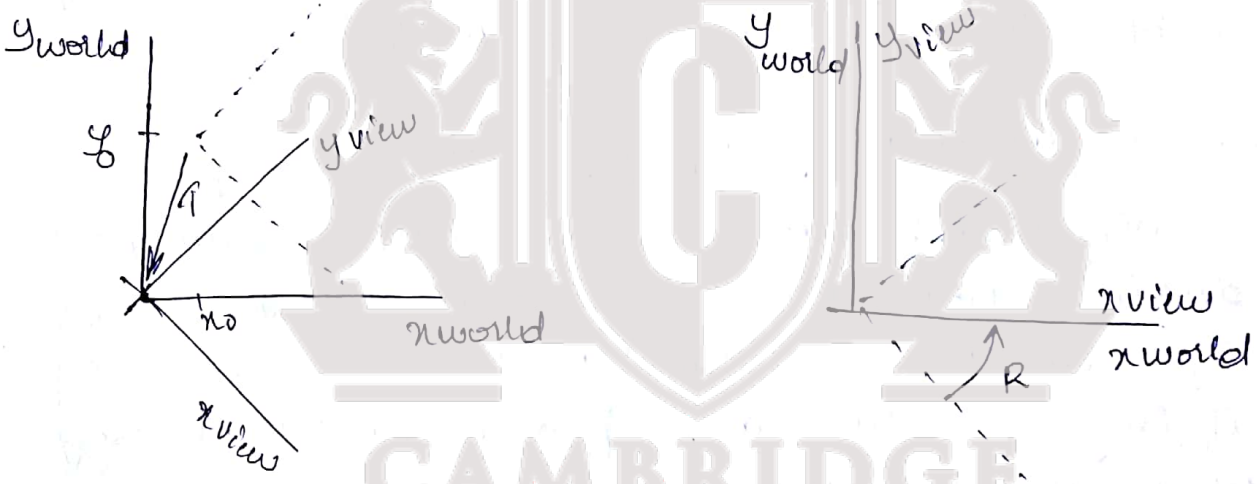
Super Impose viewing frame on world frame

1. Translate viewing origin to the world origin.
2. Rotate the viewing system to allign it with the world frame.

Given the orientation vector V, we calculate the components of unit vectors $V = (V_x, V_y)$ $u = (u_x, u_y)$ for the yview & xview axes respectively.

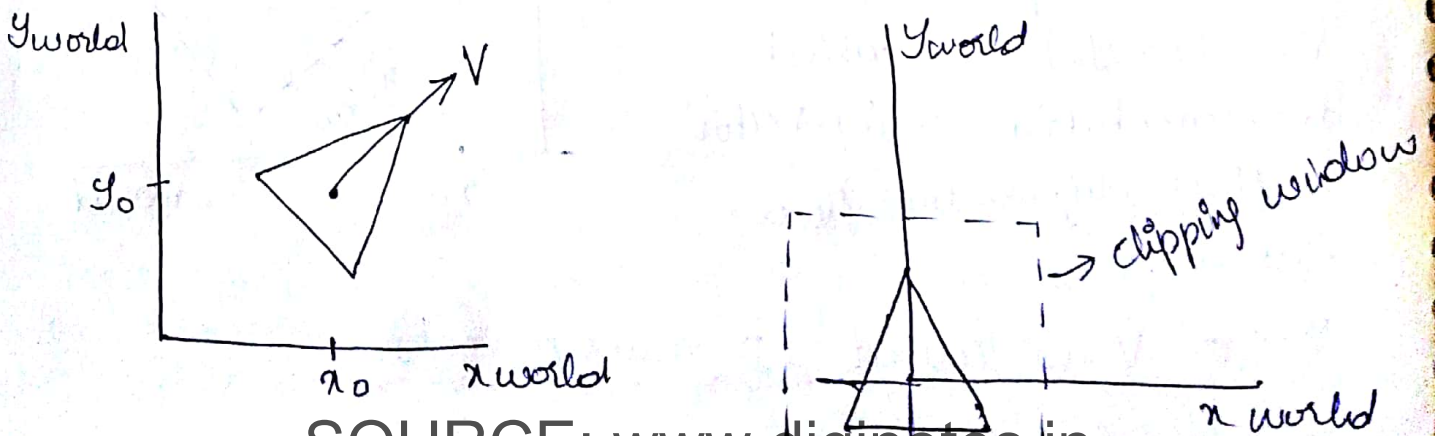
Objects positions in world coordinates are then converted to viewing coordinates with composite 2D transformation matrix.

$$M_{wc,vc} = R \cdot T$$



World coordinate clipping window

We perform the same above steps by considering a standard rectangle but without considering a viewing frame of reference



Normalization and Viewport transformation.

Viewport co-ordinates are often given in range from 0 to 1., hence viewport is positioned with a unit square. After clipping, the unit vector containing viewport is mapped to o/p display device.

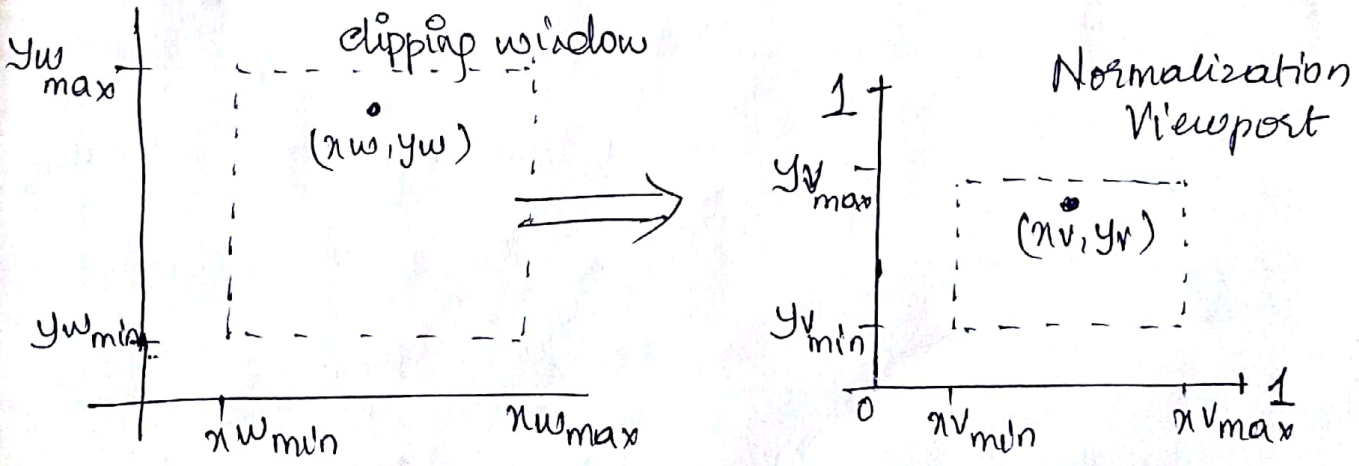
Mapping the clipping Window into a Normalized Viewport

* First a viewport is defined with normalized coordinates values between 0 and 1.
* same relative placement of a point is maintained in the viewport as it had in clipping window i.e the relative position will not change.
* i.e based on the position we need to find relative viewport coordinates based on window coordinates.

Position (x_w, y_w) in the clipping window is mapped into position (x_v, y_v) in associated viewport.

$$\frac{x_v - x_{v_{min}}}{x_{v_{max}} - x_{v_{min}}} = \frac{x_w - x_{w_{min}}}{x_{w_{max}} - x_{w_{min}}}$$

$$\frac{y_v - y_{v_{min}}}{y_{v_{max}} - y_{v_{min}}} = \frac{y_w - y_{w_{min}}}{y_{w_{max}} - y_{w_{min}}}$$



Scaling parameters S_x & S_y are

$$S_x = \frac{x_{Vmax} - x_{Vmin}}{x_{wmax} - x_{wmin}} \quad \& \quad S_y = \frac{y_{Vmax} - y_{Vmin}}{y_{wmax} - y_{wmin}}$$

{ how many times the viewport is greater than window}

As we have all values other than x_v & y_v, which is the coordinate in viewport, hence x_v.

$$x_v - x_{Vmin} = (x_{Vmax} - x_{Vmin}) \frac{(x_w - x_{wmin})}{(x_{wmax} - x_{wmin})}$$

$$= (x_w - x_{wmin}) \frac{(x_{Vmax} - x_{Vmin})}{(x_{wmax} - x_{wmin})}$$

Substitute S_x.

$$x_v - x_{Vmin} = (x_w - x_{wmin}) S_x$$

$$\boxed{x_v = x_{Vmin} + (x_w - x_{wmin}) S_x}$$

Similarly

$$\boxed{y_v = y_{Vmin} + (y_w - y_{wmin}) S_y}$$

$$x_v = x_{Vmin} + S_x \cdot x_w - S_x \cdot x_{wmin}$$

$$= S_x \cdot x_w - S_x \cdot x_{wmin} + x_{Vmin}$$

Substitute S_x

$$= S_x \cdot x_w - \frac{x_{Vmax} x_{wmin} + x_{Vmin} x_{wmin}}{x_{wmax} - x_{wmin}} + x_{Vmin}$$

multiply & take common

$$= S_x \cdot x_w - \frac{x_{Vmax} x_{wmin} + x_{Vmin} x_{wmin} + x_{Vmin} x_{wmax} - x_{Vmin} x_{wmin}}{x_{wmax} - x_{wmin}}$$

$$= S_x \cdot x_w + \frac{x_{wmax} x_{Vmin} - x_{Vmax} x_{wmin}}{x_{wmax} - x_{wmin}}$$

$$x_v = s_x x_w + t_x$$

where $t_x = \dfrac{x_{wmax} x_{vmin} - x_{vmax} x_{wmin}}{x_{wmax} - x_{wmin}}$

Also $y_v = y_{vmin} + (y_w - y_{wmin}) s_y$.

$y_v = y_{vmin} + s_y \cdot y_w - s_y \cdot y_{wmin}$.

$= s_y \cdot y_w - \left[\dfrac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}}\right] \cdot y_{wmin} + y_{vmin}$.

$= s_y \cdot y_w - \dfrac{y_{wmin} y_{vmax} + y_{vmin} y_{wmin}}{y_{wmax} - y_{wmin}} + y_{vmin}$.

$= s_y \cdot y_w - \dfrac{y_{wmin} y_{vmax} + y_{vmin} y_{wmin} + y_{vmin} y_{wmax} - y_{vmin} y_{wmin}}{y_{wmax} - y_{wmin}}$

$= s_y y_w + \dfrac{y_{wmax} \cdot y_{vmin} - y_{vmax} y_{wmin}}{y_{wmax} - y_{wmin}}$.

$$y_v = s_y \cdot y_w + t_y$$

where $t_y = \dfrac{y_{wmax} \cdot y_{vmin} - y_{vmax} \cdot y_{wmin}}{y_{wmax} - y_{wmin}}$.

We can get/obtain the transformation from world co-ordinates to viewport co-ordinates with sequence

1) Scale the clipping window to size of viewport using fixed point position of (x_{wmin}, y_{wmin})
2) Translate (x_{wmin}, y_{wmin}) to (x_{vmin}, y_{vmin})

The scaling transformation in ① can be represented

$$\text{as} \quad S = \begin{bmatrix} S_x & 0 & x_{wmin}(1-S_x) \\ 0 & S_y & y_{wmin}(1-S_y) \\ 0 & 0 & 1 \end{bmatrix}$$

The 2D matrix representation for translation of lower left corner of clipping window to lower-left viewport corner is

$$T = \begin{bmatrix} 1 & 0 & x_{vmin} - x_{wmin} \\ 0 & 1 & y_{vmin} - y_{wmin} \\ 0 & 0 & 1 \end{bmatrix}$$

And the composite matrix representation for transformation to normalized viewport is

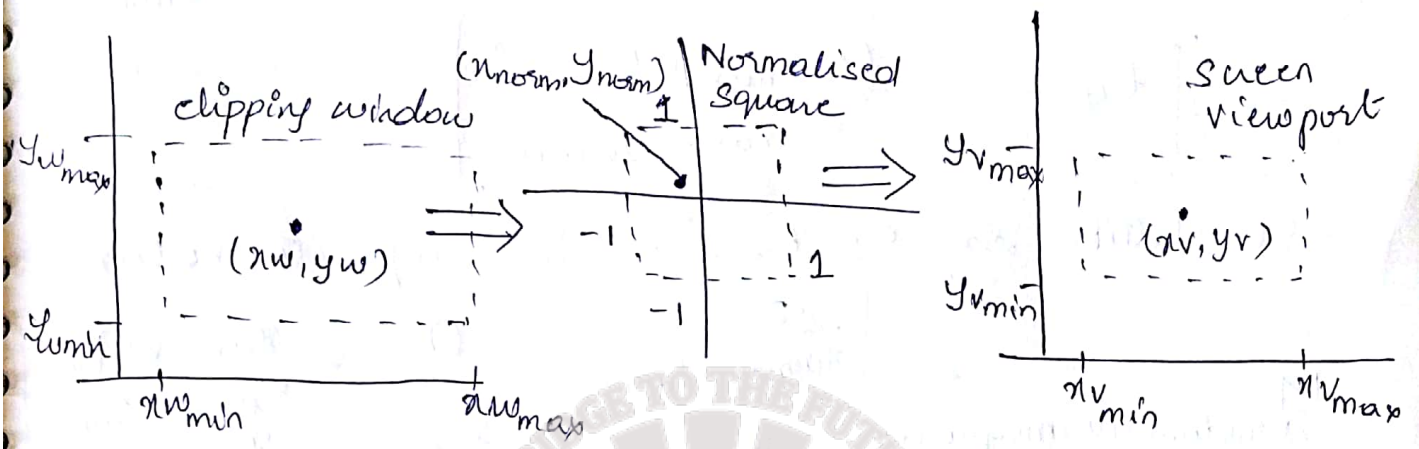
$$M_{window,\ viewp}^{norm} = T \cdot S = \begin{bmatrix} S_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Mapping clipping window into a Normalized Square

Transform the clipping window into a normalised square. → clip in normalized co-ordinates → then transfer the scene description to a viewport specified in screen co-ordinates.

* Normalized co-ordinates are in range from -1 to 1.
* clipping algorithms are standardized such that objects outside the boundaries $x = \pm 1$ & $y = \pm 1$ are detected & removed from scene description.

finally the viewing transformation, has the objects in the viewport positioned within display window.



A point (x_w, y_w) in the clipping window is mapped to normalized coordinate position (x_{norm}, y_{norm}), then to a screen co-ordinate position (x_v, y_v) in a viewport.

Consider the composite matrix.

$$M = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

for s_x & s_y, t_x & t_y, substitute -1 for x_{vmin} and y_{vmin}, $+1$ for x_{vmax} & y_{vmax}

where $$s_x = \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}} = \frac{1-(-1)}{x_{wmax} - x_{wmin}} = \frac{2}{x_{wmax} - x_{wmin}}$$

$$s_y = \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}} = \frac{1-(-1)}{y_{wmax} - y_{wmin}} = \frac{2}{y_{wmax} - y_{wmin}}$$

we also know.

$$t_x = \frac{x_{wmax}\, x_{vmin} - x_{vmax}\, x_{wmin}}{x_{wmax} - x_{wmin}} = \frac{x_{wmax}(-1) - 1 \cdot x_{wmin}}{x_{wmax} - x_{wmin}}$$

$$t_x = -\left(\frac{x_{wmax} + x_{wmin}}{x_{wmax} - x_{wmin}}\right)$$

$$t_y = \frac{y_{wmax}\, y_{vmin} - y_{vmax}\, y_{wmin}}{y_{wmax} - y_{wmin}} = \frac{y_{wmax}(-1) - y_{vmax}(1)}{y_{wmax} - y_{wmin}}$$

$$\boxed{t_y = -\frac{(y_{wmax} + y_{wmin})}{y_{wmax} - y_{wmin}}}$$

Substitute s_x, s_y, t_x & t_y in composite matrix.

$$M_{window,\,normsquare} = \begin{bmatrix} \frac{2}{x_{wmax} - x_{wmin}} & 0 & 0 \\ 0 & \frac{2}{y_{wmax} - y_{wmin}} & 0 \\ -\frac{x_{wmax} + x_{wmin}}{x_{wmax} - x_{wmin}} & -\frac{y_{wmax} + y_{wmin}}{y_{wmax} - y_{wmin}} & 1 \end{bmatrix}$$

After clipping algorithms are applied, the normalized square with edge length equal to 2 is transformed into specified viewport by substituting -1 for x_{wmin} & y_{wmin} and $+1$ for x_{wmax} & y_{wmax}.

$$\therefore s_x = \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}} = \frac{x_{vmax} - x_{vmin}}{1 - (-1)} = \frac{x_{vmax} - x_{vmin}}{2}$$

$$s_y = \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}} = \frac{y_{vmax} - y_{vmin}}{1 - (-1)} = \frac{y_{vmax} - y_{vmin}}{2}$$

$$t_x = \frac{x_{wmax}\, x_{vmin} - x_{vmax}\, x_{wmin}}{x_{wmax} - x_{wmin}}$$

$$= \frac{1(x_{vmin}) - x_{vmax}(-1)}{1 - (-1)} = \frac{x_{vmin} + x_{vmax}}{2}$$

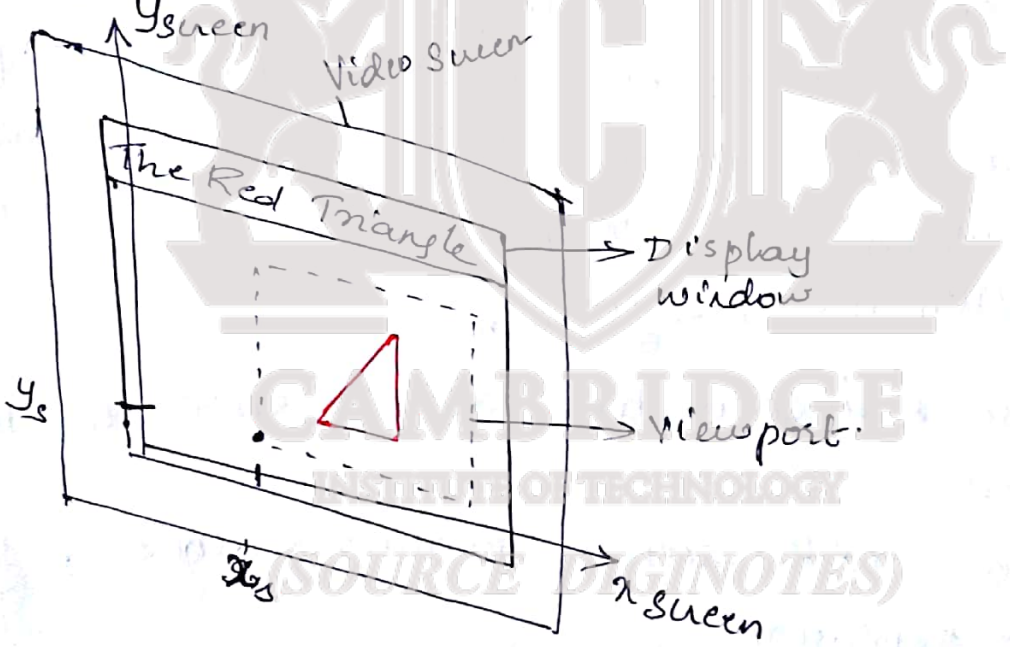
$$t_y = \frac{y_{wmax}\, y_{vmin} - y_{vmax}\, y_{wmin}}{y_{wmax} - y_{wmin}} = \frac{1(y_{vmin}) - y_{vmax}(-1)}{1 - (-1)}$$

$$t_y = \frac{y_{vmin} + y_{vmax}}{2}$$

Substitute s_x, s_y, t_x & t_y in composite matrix

$$M_{\text{normsquare, viewport}} = \begin{bmatrix} \frac{x_{vmax} - x_{vmin}}{2} & 0 & \frac{x_{vmin} + x_{vmax}}{2} \\ 0 & \frac{y_{vmax} - y_{vmin}}{2} & \frac{y_{vmin} + y_{vmax}}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

* The last step in the viewing process is to position the viewport area in the display window
* Also choosing the aspect ratio of the viewport are to be same as the clipping window. if not objects may be stretched or contracted in the x or y directions.



Viewport at co-ordinate position (x_s, y_s) within a display window.

Clipping Algorithms

Any procedure that eliminates those portions of a picture that are either inside or outside of a specified region of space is refered to as clipping Algorithm or simply clipping. [Rectangle shape] - x_{wmin}, x_{wmax} y_{wmin}, y_{wmax}.

Application of clipping is in viewing pipeline - to extract the designated portion of a scene for display.

clipping algorithms are applied in 2D viewing procedures to identify those parts of a picture that are within the clipping window & everything outside it is eliminated.

Two Dimensional Point clipping.

a point $P = (x, y)$ in 2D would be saved for display if the following inequalities are satisfied.

$$x_{wmin} \leq x \leq x_{wmax}$$
$$y_{wmin} \leq y \leq y_{wmax}$$

If these 4 inequalities is not satisfied, the point is clipped

Applications : with scenes involving clouds, sea foam, smoke or explosions - - .

Two Dimensional Line clipping

A line - clipping algorithm processes each line in a scene through a series of tests & intersection calculation to determine whether the entire line or any part is to be saved.

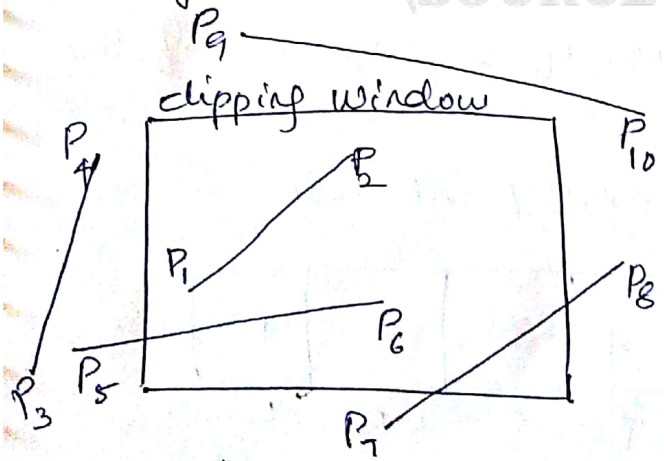
Expensive part of line clipping is in calculating the intersection position of line with the window edges. Hence major goal is to minimize the intersection calculation.

Inorder to do so, perform tests to determine whether a line segment is completely inside or completely outside the clipping window.

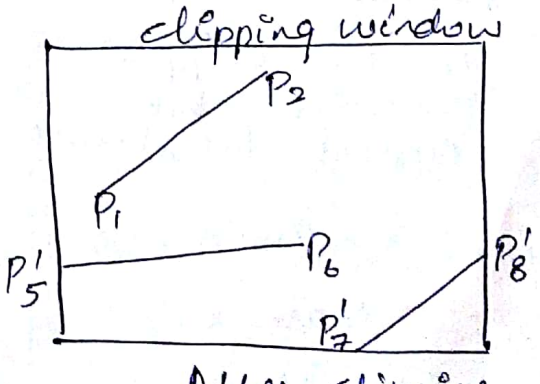
If we are unable to identify a line as complety inside or outside a clipping rectangle, we would then perform intersection calculations to determine whether any part of the line crosses the window Intersection

By applying point clipping tests:

- when both the endpoints of a line segment are inside all 4 clipping boundary. Ex: P_1 to P_2, the line is complety inside the clipping window.
- when both endpoints of a line segment are outside any of the 4 clipping boundaries Ex: P_3 to P_4, that line is complety outside the window & is is eliminated from the scene.
- If both of these tests fail, the line segment intersects at least one clipping boundary & it may or may not cross into the interior of clipping window



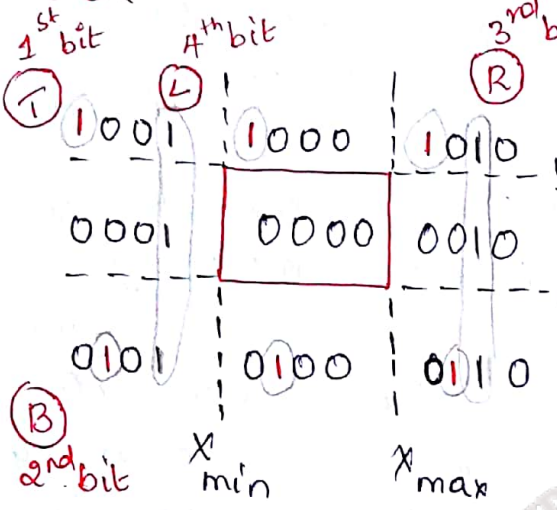
Befor clipping



After clipping

Cohen-Sutherland Line Clipping

4- bit code / T B R L
1st 2nd 3rd 4th bit



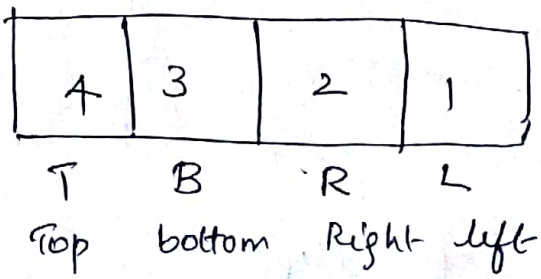
Bit Number		1	0
T	1st bit	Above top edge $y > y_{max}$	below top edge $y < y_{max}$
B	2nd bit	Below bottom edge $y < y_{min}$	Above bottom edge $y > y_{min}$
R	3rd bit	Right of Right edge $x > x_{max}$	Left of right edge $x < x_{max}$
L	4th bit	Left of left edge $x < x_{min}$	Right of left edge $x > x_{min}$

* Every line endpoint in a picture is assigned a 4-digit binary value called region code.
* Each bit position is used to indicate whether the point is inside or outside one of the clipping-window boundaries.
* Each clipping window edge divides the 2D space into 9 regions.
* Any endpoint inside the clipping widdow has the region code "0000".

Step 1

(1) Calculate differences b/w endpoint co-ordinates & clipping boundaries
w.e Determine the bit values of 2 endpoints of the line to be clipped..
To determine the bit value of any point use.

$b_1 = x - x_{min}$
$b_2 = x_{max} - x$
$b_3 = y - y_{min}$
$b_4 = y_{max} - y$



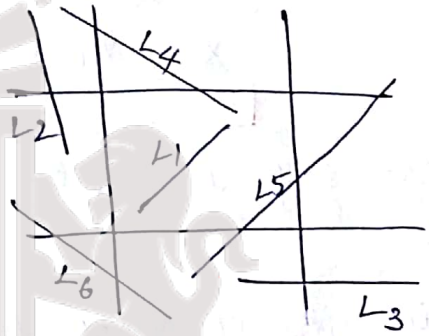
Step2:
Use these end points codes to locate the line.

Various possibilities

* If both endpoints codes are [0000], the line lies completely inside the box, no need to clip. This is the simplest case (eg. L_1).
* Any line has 1 in the same bit positions of both the endpoints, it is guaranteed to lie outside the box completely. (eg. L_2 & L_3)

(OR)

Perform AND operation for both the region codes if the result is not "0000" then line is completely outside. So line is rejected.



* Neither completely reject nor inside the box, lines L_4 & L_5 needs more processing.

Processing of lines neither completely inside or out
for lines L_4, L_5, L_6

Basic idea is to clip parts of the line in any order. (consider from top to bottom).

Step3: Compute outcodes of both endpoints to check for trivial acceptance or rejection (AND Logic).
if not so, obtain an endpoint that lies outside the box (at least one will?).

* Using the outcode, obtain the edge that is crossed by.

If the line crosses x_{min} / x_{max} then find

$$y = y_0 + m(x - x_0)$$ → line eqn

Here $x = x_{min}$ / x_{max}. the x value is set either to x_{min} or x_{max}.

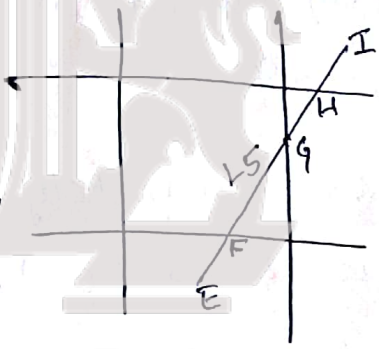
Slope of line $m = \frac{y_{end} - y_0}{x_{end} - x_0}$.

else if the line crosses y_{min} / y_{max}.

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

Here y is set either to y_{min} or y_{max}.

Obtain corresponding intersection points



* CLIP (replace the endpoint by the intersection point) w.r.t the edge
* compute the outcode for the updated endpoint & repeat the iteration, till it is 0000.

eg: Take line L_S (endpoint are E & I).

E has outcode 0100 (to be clipped w.r.t bottom edge).

so EI is clipped to FI;

outcode of F is 0000;

But outcode of I is 1010;

clip (w.r.t topedge) to get FH.

Outcode of H is 0010;

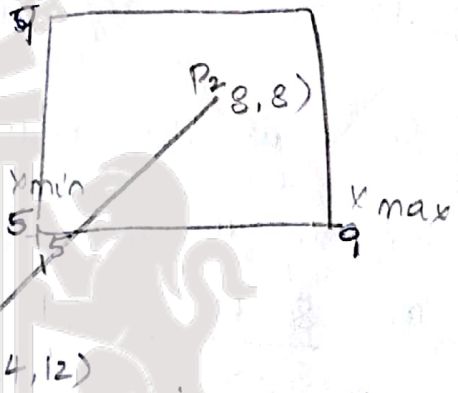
clip (with respect to right edge) to get FG.

Since outcode of G is 0000, display the final result FG.

Example Cohen-Sutherland Algo.

Consider the window size from 5 to 9, clipp the following line (4,12), (8,8)

Soln : Line $P_1(4,12)$ to $P_2(8,8)$



Region code for P_1 = 1001 (outside)

Region code for P_2 = 0000 (inside)

P_1 AND P_2

1 0 0 1
0 0 0 0
0 0 0 0 so the line is partially inside the window

As P_2 is 0000, this is inside the window, so no need to calculate new value.

P_1 is outside, so calculate the new value for P_1

$x_1 = 4, y_1 = 12, x_2 = 8, y_2 = 8.$

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{8-12}{8-4} = -1$$

Step 3 (1) : Line is intersecting x_{min} / x_{max}, find y ?

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

$$y = 12 + -1(x - 4)$$

Here line is intersecting at x_{min} $x = x_{min} \Rightarrow x = 5$

$$y = 12 + (-1)(5-4) = 12 - 1$$

$$y = 11$$

So the new point is $(x, y) = (5, 11)$

Step 4 : check with condition.

$x_{min} \le x \le x_{max} \Rightarrow x_{min} \le 5 \le x_{max}$ (Yes)

$y_{min} \le y \le y_{max}$ $y_{min} \le 11 \le y_{max}$ (No).

Since x_{min} & $y_{min} = 5$

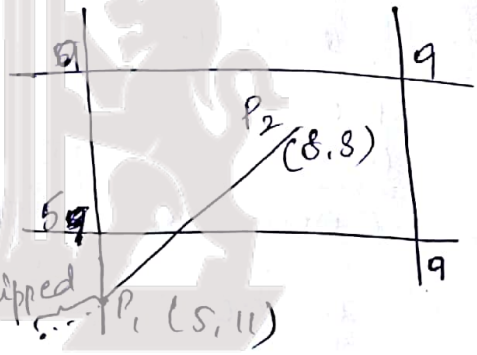
x_{max} & $y_{max} = 9$

The second condition is false, hence the ~~spell~~ step 3 is to be repeated again.

(2) So now the new line is from (5,11) to (8,8)

$x_1 = 5, y_1 = 11, x_2 = 8, y_2 = 8.$

$m = \frac{8-11}{8-5} = -1$



Now the following intersecting with y_{max}, find x.

$x = x_1 + \frac{(y - y_1)}{m}$

Here $y = y_{max} = 9$.

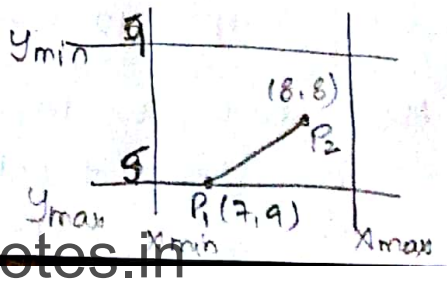
$x = 5 + \frac{(9-11)}{m}$

$x = 7$

Check for condition $x_{min} \le 7 \le x_{max}$

$y_{min} \le 9 \le y_{max}$.

Now the new line location (7,9) to (8,8)

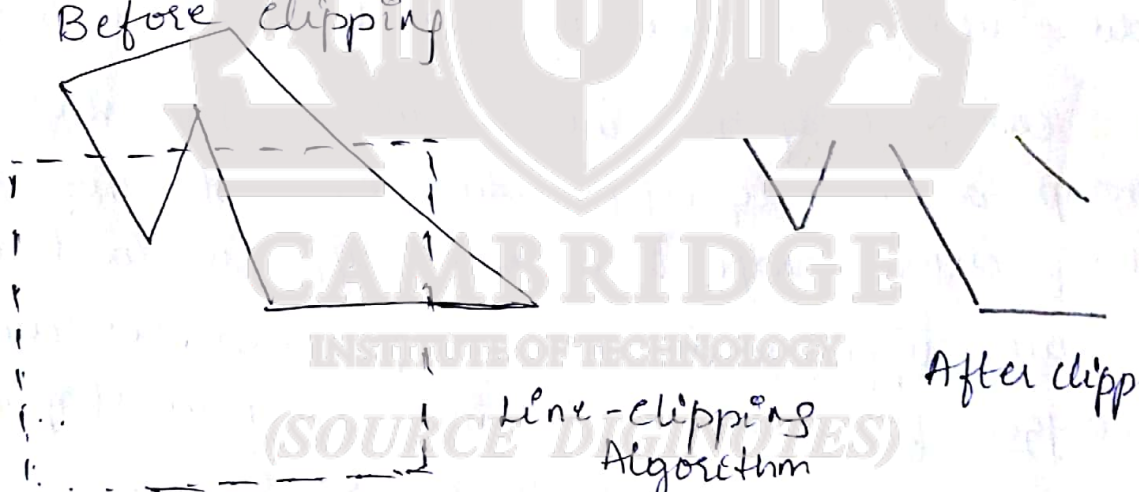


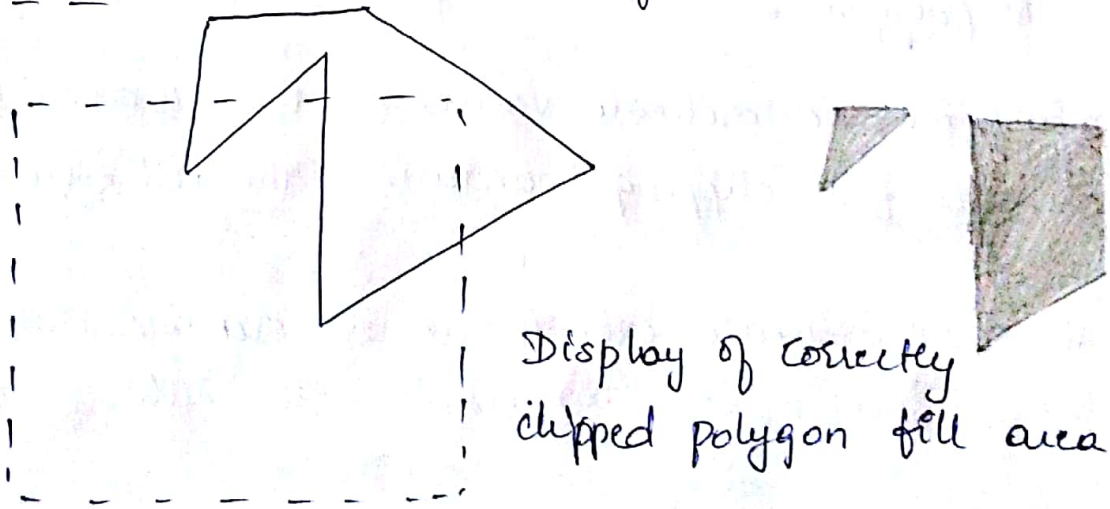
Polygon Fill-Area Clipping.

To clip a polygon fill area, it cannot directly apply a line-clipping method to the individual polygon edge (will not produce a closed polyline)

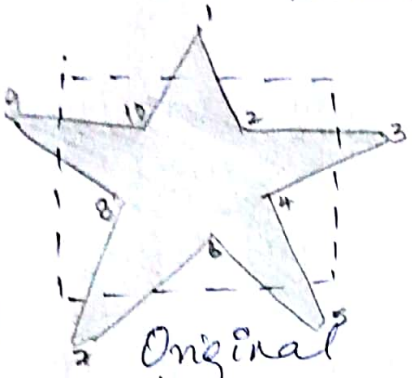
* A line clipper would often produce a disjoint set of lines with no complete information about how a closed boundary would form around the clipped fill area.

* A procedure is required that could output one or more closed polylines for the boundaries of the clipped fill area so that the polygon can be scan converted to fill the interiors with the assigned color or pattern.

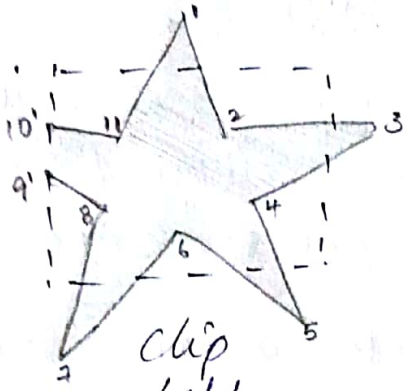




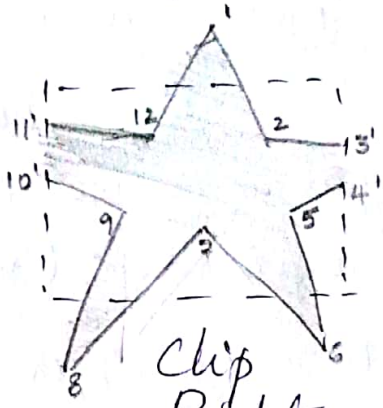
Vertex number with "'" are the new list of vertices added



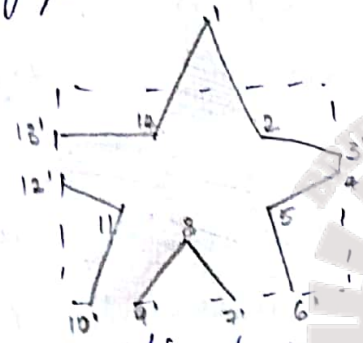
Original Polygon



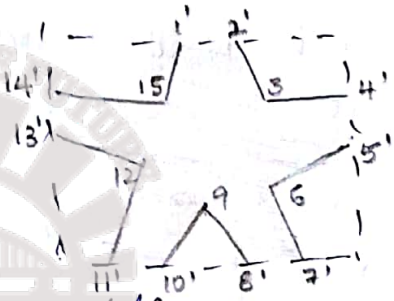
Clip left



Clip Right



Clip bottom

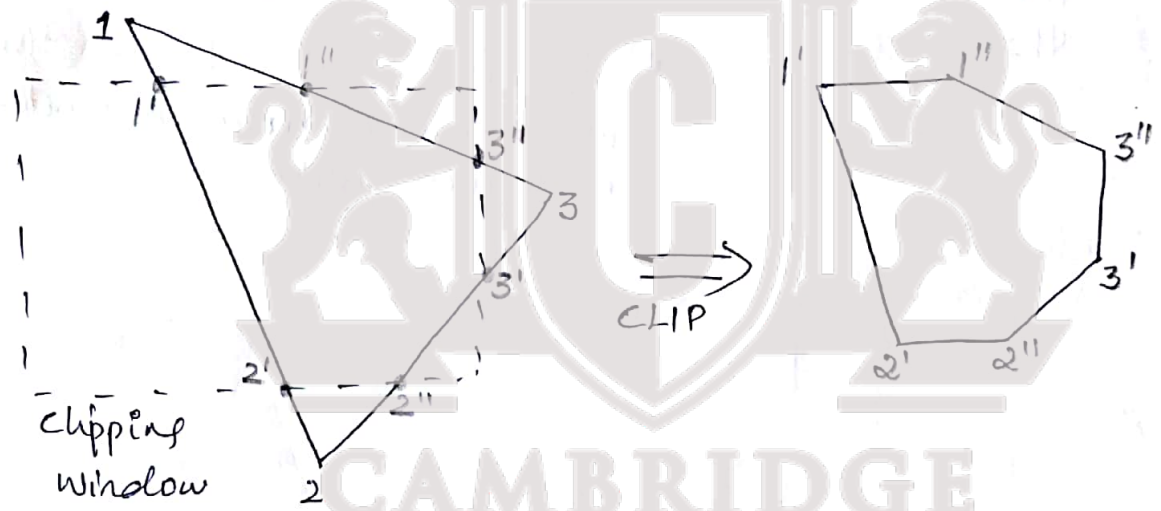


Clip Top.

Count of No. of vertices in the polygon after each clip.

* Polygon fill area is processed against borders of a clipping window using same general approach as in line clipping.
* The end points of the line segment are processed through a line-clipping procedure by constructing a new set of clipped endpoints at each clipping window boundary.
* As each clipping window edge is processed, we can clip a polygon fill area by determining the new shape for the polygon as shown in figure above.
* If min & max co-ordinate values for the fill area are inside all four clipping boundaries, the fill area is saved.
* If these co-ordinate extents are all outside any one of the clipping window borders, we eliminate the polygon.

* When the polygon fill area cannot be identified as being completely inside or completely outside the clipping window, then locate the polygon intersection positions with the clipping boundaries.
* One way to implement is to create a new vertex list at each clipping boundary, & then pass this new vertex list to the next boundary clipper
* The o/p of final clipping stage is the vertex list for the clipped polygon.

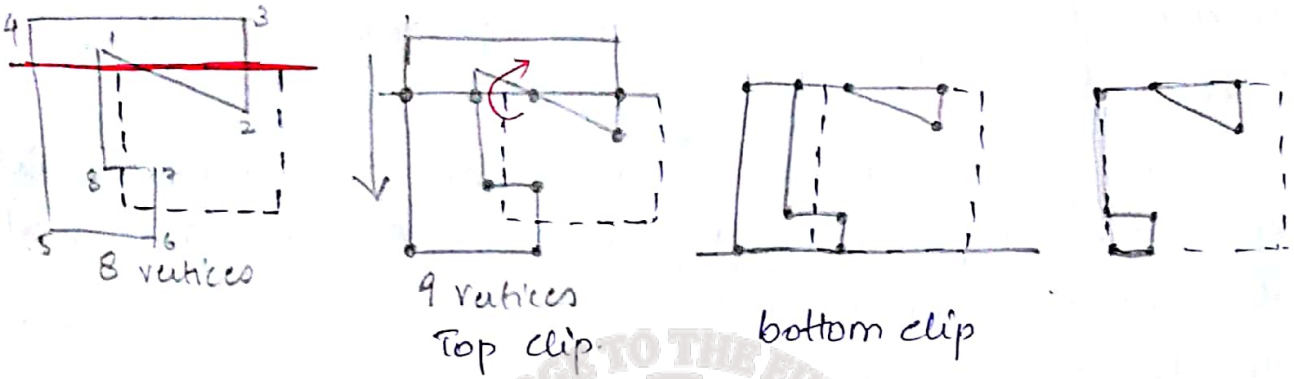


Sutherland-Hodgman Polygon Clipping

developed by Sutherland and Hodgman

* The method sends the polygon vertices through each clipping stage so that a single clipped vertex can be immediately passed to the next stage thus eliminating the need for an output set of vertices at each clipping stage.
* Final output is list of vertices that describes the edges of clipped polygon fill area.

In case of multiple components - it cannot correctly generate 2 o/p polygons.

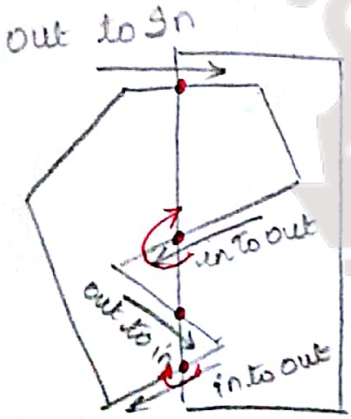


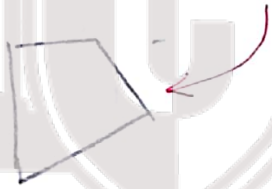
Now the o/p is one pixel line joining

rather than

For this kind of problem

1) For out → in pair, follow the polygon boundary
2) For in → out, follow the window boundary in clock or anticlock wise.





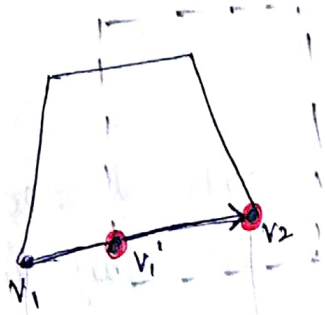
4 possible cases

case 1: first edge endpoint is outside the clipping boundary & second endpoint is inside.

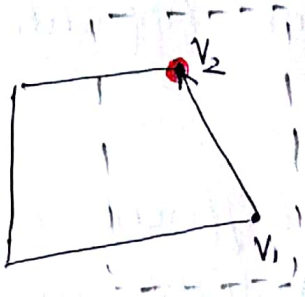
case 2: Both endpoints are inside the clipping boundary.

case 3: first endpoint is inside the clipping boundary & second endpoint is outside

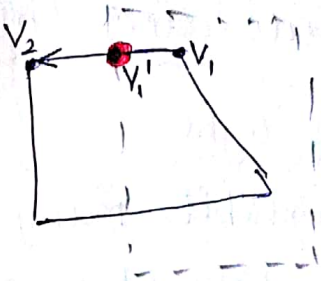
case 4: Both endpoints are outside the clipping boundary.



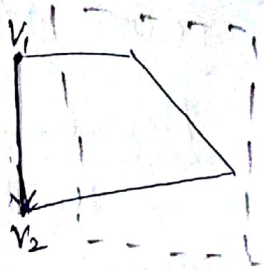
out → in
case 1
Output : V_1', V_2



in → in
case 2
Output : V_2



in → out
case 3
output : V_1'



out → out
case 4
output : none

1. In case 1, both the intersection point of the polygon and window border & the second vertex that is inside are sent to the next clipper.
2. In case 2, as both the vertices are inside the clipping window, only the second vertex (V_2 position) is sent to the next clipper
3. In case 3, only the polygon edge-intersection position with the window border (marked in red) are sent to the next clipper
4. In case 4, both the vertices are outside, hence no vertices are sent to the next clipper.

Example to be reffered from textbook.

2.5 COLOR :

A visible color can be characterized by a function $C(\lambda)$ that occupies wavelengths from about 350 - 780nm, as shown in the fig. The value for a given wavelength λ in the visible spectrum gives the intensity of that wavelength in the color.

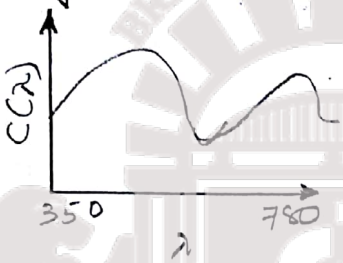


Fig: A color distribution

⇒ The human visual system has three types of cones responsible for color vision. Hence, our brains do not receive the entire distribution $C(\lambda)$ for a given color but rather three values - the **tristimulus values** - that are the responses of the three types of cones to the color. This reduction of a color to three values leads to the **basic tenet of three-color theory**.

*** If two colors produce the same tristimulus values, then they are visually indistinguishable ***

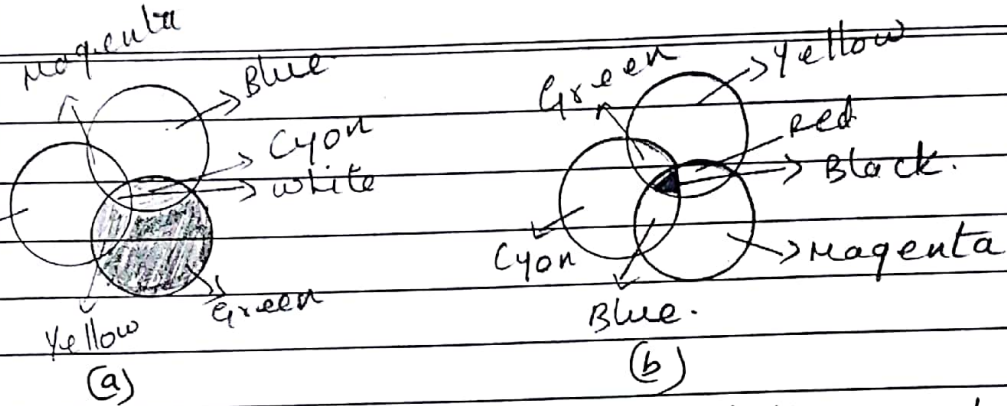


Fig: Color formation. (a) Additive color (b) Subtractive color

i) The CRT is one example of **additive color** where the primary colors add together to give the perceived color. Other examples of additive color include projectors and slide (+ve) film. In such systems the primaries are usually red, green, and blue.

With additive color, primaries add light to an initially black display yielding the desired color.

ii) For processes such as commercial printing and painting, a **subtractive color model** is more appropriate.

Here we start with a white surface such as a sheet of paper. Colored pigments remove color components from light that is striking the surface.

In subtractive system, the primaries are usually the **complimentary colors**: cyan, magenta, and yellow (CMY) - as shown in the fig.

2.5.1 RGB Color

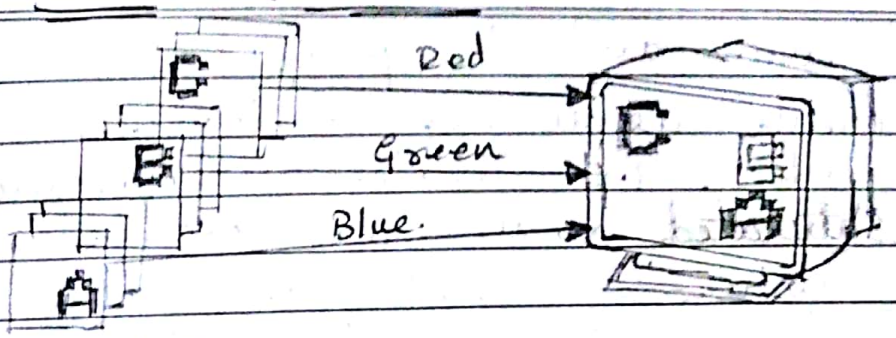


Fig: RGB color

=> In a three-primary-color, additive-color RGB system, there are conceptually separate buffers for red, green, and blue images. Each pixel has separate red, green, and blue components that correspond to locations in memory. as shown in the above fig.

=> In a typical system, there might be a 1280 x 1024 array of pixels, and each pixel might consist of 24 bits (3 bytes); 1 byte for each of red, green, and blue.

=> With present commodity graphics cards having from 128MB to 512MB of memory, there is no longer a problem of storing and displaying the contents of the frame buffer at video rates.

⇒ A natural technique is to use the color cube and to specify color components as numbers between 0.0 and 1.0, where 1.0 denotes the maximum (or **saturated** value), and 0.0 denotes a zero value of that primary.

⇒ In OpenGL, we use the color cube as follows.

Ex:- To draw in red, we issue the following function call:

```
glColor3f(1.0, 0.0, 0.0);
```

The execution of this function will set the current drawing color to red.

⇒ Using 3f in glVertex function, conveys that we are using a three-color (RGB) model and that the values of the components are given as floats in C

⇒ Now, we shall be interested in a four-color (RGBA) system. The fourth color (A, or **alpha**) also is stored in the frame buffer as are the RGB values; it can be set with four-dimensional versions of the color functions.

*** uses for alpha, such as for creating fog effects or combining images. ***

⇒ **Alpha** value will be treated by OpenGL as either an **opacity** or **transparency** value

refer last page

⇒ Opacity values can range from fully transparent (A=0.0) to fully opaque (A=1.0).

⇒ One of the first tasks that we must do in a program is to clear an area of the screen - a drawing window - in which to display our output.

We also must clear this window whenever we want to draw a new frame. By using the four-dimensional (RGBA) color system:

The function call
glClearColor (1.0, 1.0, 1.0, 1.0);
defines an RGB-color clearing color that is white, because the first three components are set to 1.0, and is opaque, because the alpha component is 1.0.

Hence the function glClear to make the window on the screen solid and white.

COLOR MODELS

(Refer Color image proces... I & II] NPTEL

12.1 Properties of Light:

Light exhibits many different characteristics and we describe the properties of light in different ways in different contexts.

→ Electromagnetic Spectrum:

In physical terms, color is electromagnetic radiation within a narrow frequency band.

Some of the other frequency groups in the electromagnetic spectrum are referred to as radio waves, microwaves, infrared waves, and X-rays.

Below figure shows the approximate frequency ranges for these various aspects of electromagnetic radiation.

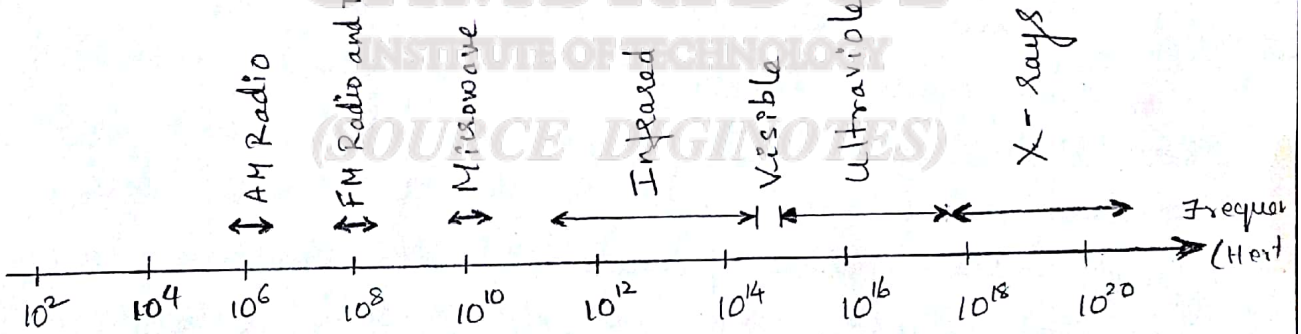


Fig: Electromagnetic spectrum

Note:

Each frequency value within the the visible region of the electromagnetic spectrum corresponds to a distinct spectral color.

At the low frequency end are the red colors, and at the high-frequency end are the violet colors. Actually, the human eye is sensitive to some frequencies into the infrared and ultraviolet bands.

Spectral color range from shades of red through orange and yellow, at the low-frequency end, to shades of green, blue, and violet at the high end.

A light source such as the sun or a standard household light bulb emits all frequencies within the visible range to produce white light.

When white light is incident upon an opaque object, some frequencies are reflected and some are absorbed.

The combination of frequencies present in the reflected light determines what we perceive as the color of the object. If low frequencies are predominent in the reflected light, the object is described as red.

In this case, we say that the perceived light has a **dominent frequency** or (dominent wavelength) at the red end of a spectrum. The dominant frequency is also called the **hue**, or simply the **color** of light

⇒ Physcological characteristics of color

Other properties besides frequency are needed to characterize our perception of light.

When we view a source of light, our eyes respond to the color (or dominant frequency) and two other basic sensations

→ One of these we call the **brightness**, which corresponds to the ~~light~~ total light energy and can be quantified as the luminance of the light. [Section 10.3]

→ The third perceived characteristics is called the purity, or the **saturation**, of the light. Purity describes how close a light appears to be to a pure spectral color, such as red.

Pastels and Pale colors have low purity and they appear to be nearly white.

Another term, **chromaticity**, is used to refer collectively the two properties describing color characteristics: purity and dominant frequency (hue).

12.2 COLOR MODELS

Any method for explaining the properties or behavior of color within some particular context is called a color model.

Primary Colors:

When we combine the light from two or more sources with different dominant frequencies, we can vary the amount (intensity) of light from each source to generate a range of additional colors.

This represents one method for forming a color model.

The hues (color) that we choose for the sources are called the **primary colors**, and the **color gamut** for the model is the set of all colors that we can produce from the primary colors.

Two primaries that produce white are referred to as **complementary colors**.

Ex:- of complementary color pairs are red and cyan, green and magenta, and blue and yellow.

No finite set of real primary colors can be combined to produce all possible visible colors

A mixture of one or two of the primaries with the fourth color can be used to match some combination of the remaining primaries.

⇒ Intuitive Color Concepts:

An Artist creates a color painting by mixing color pigments with white and black pigments to form the various shades, tints, and tones in the scene.

Starting with the pigments for a "pure color" ("pure hue") the artist adds a black pigment to produce different **shades** of that color. The more black pigment, the darker the shade.

Similarly different **tints** of the color are obtained by adding a white pigment to the original color, making it lighter as more white is added.

Tones of the color are produced by adding both black and white pigments.

It is generally much easier to think of creating a pastel red color by adding white to pure red and producing a dark blue color by adding black to pure blue.

Therefore, graphics packages providing color palettes to a user often employ two or more color models.

12.4. RGB Color Model

According to the tristimulus theory of vision, our eyes perceive color through the stimulation of three visual pigments in the cones of the retina.

By comparing intensities in a light source, we perceive the color of the light.

This theory of vision is the basis for displaying color output on a video monitor using the three primaries red, green, and blue, which is referred to as the **RGB color Model**

We can represent this model using the unit cube defined on R, G, B axes as shown in the fig.

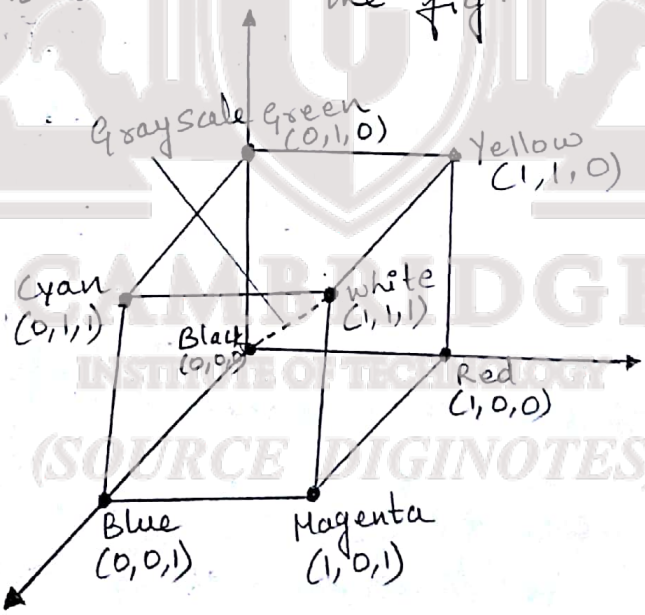


Fig: RGB color model

Any color within the unit cube can be described as an additive combination of the three primary colors.

The origin represents black and the diagonally opposite vertex, with coordinates (1,1,1) is white.

Vertices of the cube on the axes represent the primary colors, and the remaining vertices are the complementary color points for each of the primary colors.

The RGB color scheme is an additive model.

Each color point within the unit cube can be represented as a weighed vector sum of the primary colors, using unit vectors **R**, **G**, and **B**:

$$C(\lambda) = (R, G, B) = R\mathbf{R} + G\mathbf{G} + B\mathbf{B}$$

where parameters R, G and B are assigned values in the range from 0 to 1.0.

For Ex:- ① The magenta vertex is obtained by adding maximum red and blue values to produce the triple (1, 0, 1) and ② white at (1, 1, 1) is the sum of the maximum values for red, green and blue.

③ Shades of gray are represented along the main diagonal of the cube from the origin (black) to the white vertex.

Points along ~~side~~ this diagonal have equal contributions from each primary color, and a gray shade halfway b/w black and white is represented as (0.5, 0.5, 0.5).

Example for RGB color model is TV, computer monitor, where the color displays on the black screen

12.6 The CMY AND CMYK Color Models

Hard copy devices, such as printers and plotters, produce a color picture by coating a paper with color pigments.

We see the color patterns on the paper by reflected light, which is a subtractive process.

The CMY Parameters :→

A subtractive color model can be formed with the three primary colors cyan, magenta and yellow.

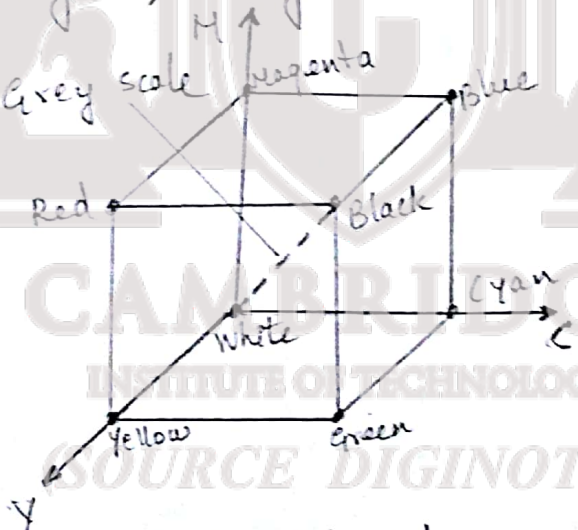


Fig: CMY color model

Positions within the unit cube are described by subtracting the specified amounts of the primary colors from white.

Cyan can be described as a combination of green and blue.

There

Therefore when white light is reflected from cyan colored ink, the reflected light contains only the green and blue components, and the red component is absorbed or subtracted by the ink.

Similarly magenta ink subtracts the green component from incident light, and yellow subtracts the blue component.

A unit cube representation for the CMY model is illustrated in the above figure.

In the CMY model, the spatial position (1, 1, 1) represents black, because all components of the incident light are subtracted. The origin represents white light.

Equal amounts of each of the primary colors produce shades of gray along the main diagonal of the cube.

A combination of cyan and magenta ink produces blue light, because the red and green components of the incident light are absorbed. etc. Explain the cube

The CMY printing process often uses a collection of four ink dots, which are arranged in a close pattern somewhat as an RGB monitor uses three phosphor dots.

Thus, in practice, the CMY color model is referred to as the **CMYK model**, where K is the black color parameter.

One ink dot is used for each of the primary colors (cyan, magenta, and yellow), and one ink dot is black.

Illumination models

An illumination model, also called a **lighting model** (and sometimes referred to as a shading model), is used to calculate the color of an illuminated position on the surface of an object

lighting Model
Shading Model
light source
an object as light source & light reflector
properties → position, color, direction

10.1 LIGHT SOURCES

⇒ Any object that is emitting radiant energy is a **light source** that contributes to the lighting effects for other object in a Scene.

⇒ In some applications, however, we may want to create an object that is both a light source and a light reflector.

For Ex:- A plastic globe surrounding a light bulb both emits and reflects light from the surface of the globe.

⇒ A light source can be defined with a number of properties. We can specify its position, the color of the emitted light, the emission direction, and its shape.

If the source is also to be a light-reflecting surface, we need to give its reflectivity properties.

⇒ In addition, we could set up a light source that emits different colors in different directions.

For Ex:- we could define a light source that emits a red light on one side and a green light on the other side.

⇒ We assign light-emitting properties using a single value for each of the RGB color components, which we can describe as the amount, or the "intensity", of the color component.

i) Point Light Sources:

The simplest model for an object that is emitting radiant energy is a **point light source** with a single color, specified with three RGB components.

We define a point source for a scene by giving its position and the color of the emitted light, As shown in fig below.

Fig: Diverging ray paths from a point light source

In this the light rays are generated along radially diverging paths from the single color source position.

ii) ~~Infinitely Distant light Sources~~:

This light-source model is a reasonable approximation for sources whose dimensions are small compared to the size of objects in the scene.

ii) Infinitely Distant light Sources :

A large light source, such as the sun, that is very far from a scene can also be approximated as a point emitter, but there is little variation in its directional effects.

Sun
little variation in direction
constant - light path from distant light source

The light path from a distant light source by assigning to any position in the scene is nearly constant, as illustrated in the below fig.

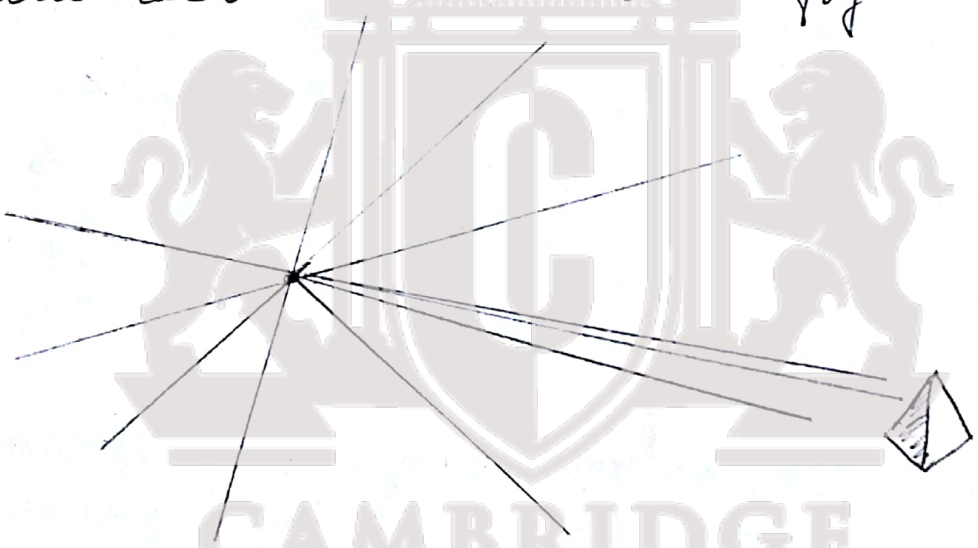


Fig: Light rays from an infinitely distant light source illuminate an object along nearly parallel light paths.

iii) Radial Intensity Attenuation:

As radiant energy from a light source travels outwards through space, its amplitude at any distance d_i from the source is attenuated by the factor $1/d_i^2$.

This means that a surface close to the light source is attenuated by the factor receives a higher incident light intensity from that source than a more distant surface.

Therefore, to produce realistic lighting effects, we should take this intensity attenuation into account. Otherwise all surfaces are illuminated with the same intensity from a light source, and undesirable display effects can result.

iii) Directional Light Sources and Spot light effects

A local light source can easily be modified to produce a directional, or spotlight beam of light.

One way to set up a directional light source is to assign it a vector direction and an angular limit θ_l measured from that vector direction, in addition to its position and color.

This defines a conical region of space with the light-source vector direction along the axis of the cone as shown in the fig below

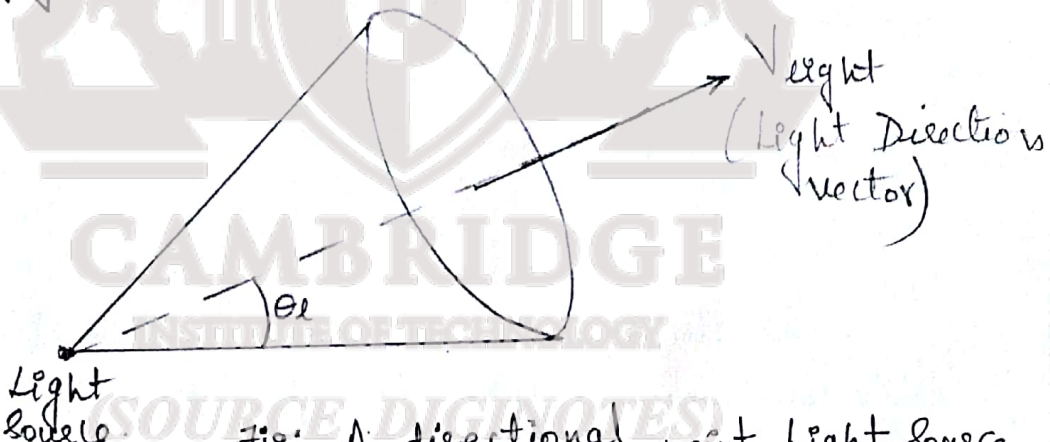


Fig: A directional point light source

=> The unit light-direction vector defines the axis of a light cone, and angle θ_l defines the angular extent of the circular cone

We can denote V_{light} as the unit vector in the light-source direction and V_{obj} as the unit vector in the direction from the light position to an object position. Then.

$$V_{obj} \cdot V_{light} = \cos\alpha$$

where

⇒ Angle α is the angular distance of the object from the light direction vector.

⇒ If we restrict the angular extent of any light cone so that $0° < \theta_l \leq 90°$, then the object is within the spotlight if $\cos\alpha > \cos\theta_l$ as shown in fig below

⇒ But if $V_{obj} \cdot V_{light} < \cos\theta_l$, the object is outside the light cone

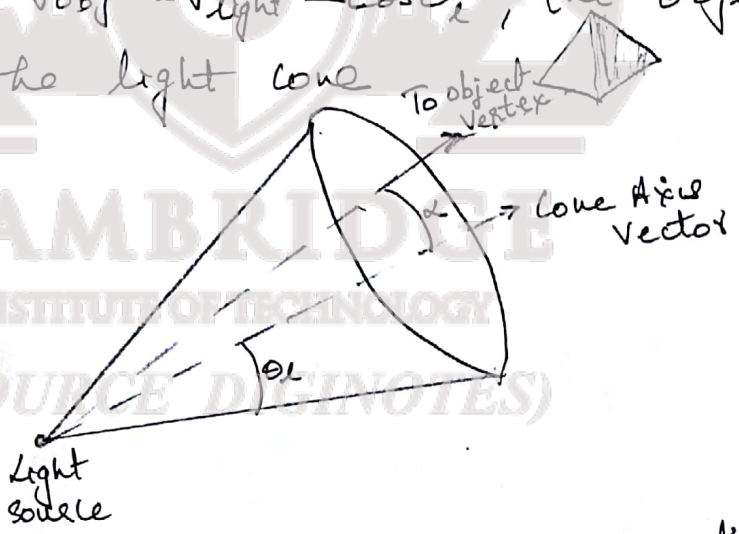


Fig: An object illuminated by a directional point light source.

iv) Angular Intensity Attenuation

For a directional light source, we can attenuate the light intensity angular about the source as well as radially out from the point-source position.

This allow us to simulate a cone of light that is most intense along the axis of the cone, with the intensity decreasing as we move farther from the cone axis.

10.3 BASIC ILLUMINATION MODELS

Accurate surface lighting models compute the results of interactions between incident radiant energy and the material composition of an object.

The empirical model described in this section produces reasonably good results, and it is implemented in most graphics systems.

⇒ Light-emitting objects in a basic illumination model are generally limited to point sources.

However, many graphics packages provide additional functions for dealing with directional lighting (spot lights) and extended light sources.

Ambient Light:

In our basic illumination model, we can incorporate background lighting by setting a general brightness level for a scene.

This produces a uniform ambient lighting that is the same for all objects, and it approximates the global diffuse reflections from the various illuminated surfaces.

=> Assuming that we are describing monochromatic lighting effects, such as shades of grey, we designate the level for the ambient light in a scene with and intensity parameter I_a. Each surface in the scene is then illuminated ~~are simply a form of diffuse~~ with this background light.

=> Reflections produced by ambient-light illumination are simply a form of diffuse reflection, and they are independent of the viewing direction and the spatial orientation of a surface.

Diffuse Reflection:

We can model diffuse reflections from a surface by assuming that the incident light is scattered with equal intensity in all directions, independent of the viewing position. Such surfaces are called **ideal diffuse reflectors**.

They are also often referred to as **Lambertian reflectors**, because the reflected radiant light energy from any point on the surface is calculated with **Lambert's cosine law**.

For Lambertian reflection, the intensity of light is the same over all viewing directions.

Assuming that every surface is to be treated as an ideal diffuse reflector (Lambertian), we can set a parameter k_d for each surface that determines the fraction of the incident light that is to be scattered as diffuse reflections. This parameter is called the **diffuse-reflection coefficient** or the **diffuse reflectivity**.

→ The diffuse reflection in any direction is then a constant, which is equal to the incident light intensity

multiplied by the diffuse-reflection co-efficient.

⇒ For the background lighting effects, we can assume that every surface is fully illuminated by the ambient light I_a that we assigned to the scene.

Therefore, the ambient contribution to the diffuse reflection at any point on a surface is simply.

k-material

$$I_{ambdiff} = k_d I_a$$

k_d - diffuse reflection co-efficient, I_a - ambient light intensity.

Lobertian Surface (ex: dull, matte surfaces, snow, chalk, projection & movie screens surfaces, uniformly painted walls.

Lambart's cosine law: The law states that the amount of radiant energy coming from any small surface area dA in a direction ϕ_N relative to the surface normal is proportional to $\cos\phi_N$.

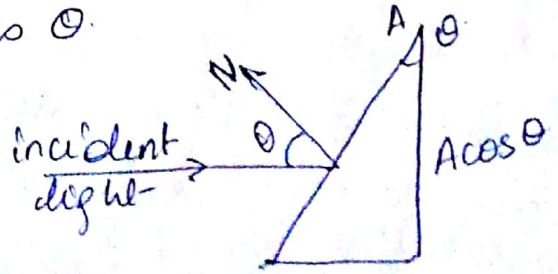
$$\text{Intensity} = \frac{\text{radiant energy per unit time}}{\text{projected area}}$$

$$\propto \frac{\cos\phi_N}{dA\cos\phi_N}$$

$$= \text{constant}$$

angle of incidence - between the incoming light direction & surface normal as θ.

$$I_{l,diff} = k_d \cdot I_{l,incident}$$
$$= I_l \cos\theta \cdot k_d$$

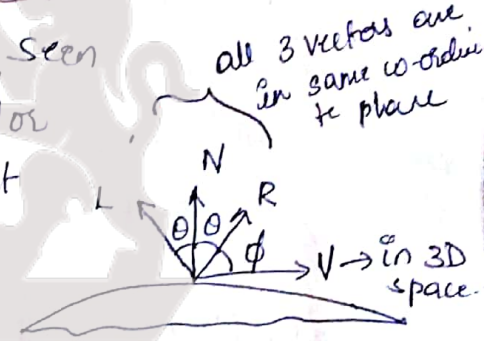


Combining ambient & point-source intensity calculations to obtain an expression for total diffuse reflection at a surface position

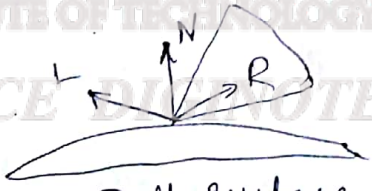
$$I_{diff} = \begin{cases} k_a I_a + k_d I_l (N \cdot L) & \text{if } N \cdot L > 0 \\ k_a I_a & \text{if } N \cdot L \leq 0 \end{cases}$$

Specular Reflection and Phong Model

The bright spot or specular reflection seen on shiny surface is a result of total or near total reflection of incident light in concentrated region around the specular-reflection angle (equal to angle of incident light)



Shiny surface (large n_s)



Dull surface (small n_s)

$n_s \rightarrow$ specular reflection exponent

$L, V \rightarrow \frac{1}{2}(h)$

Intensity of specular reflection depends on material properties of surface & angle of incidence.

$W(\theta) \rightarrow$ specular reflection co-efficient.

$I_l \rightarrow$ intensity of light source, ϕ is viewing angle relative to specular-reflection direction R

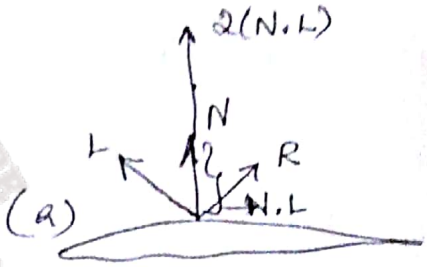
$$I_{l,spec} = W(\theta) \cdot I_l \cos^{n_s}\phi$$

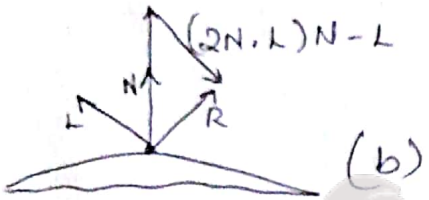
The direction for R, the reflection vector, can be computed from the directions for vectors L & N

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

and Specular-reflection vector is obtained as

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

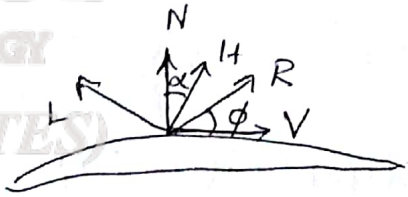




A simplified Phong model is obtained using halfway vector H between L and V to calculate the range of Specular reflection.

if we replace V.R in phong model with dot-product N.H, the halfway vector is obtained

$$H = \frac{L+V}{|L+V|}$$



OpenGL Matrix Stacks

- ModelView, projection, texture & color (4 modes) can be selected with glMatrixMode function.

OpenGL maintains a matrix stack

- initially each stack maintains/contains only identity matrix
- top of the stack is called "current matrix"
- As we perform viewing & geometric transformations, the top of modelview matrix stack is 4×4 composite matrix combining viewing & various geometric transformations
- modelview stack depth of 32 to save composite matrics for each created ~~with~~ by multiple views

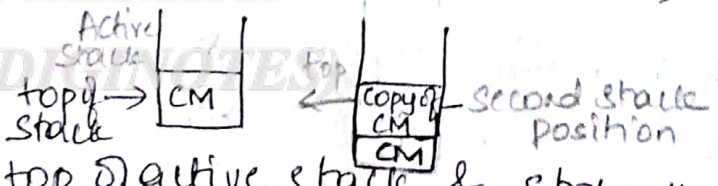
```
glGetIntegerv(GL_MAX_MODELVIEW_STACK_DEPTH, stackSize);
```

Other 3 stacks depth are of 2.

To find how many matrices are in stack currently

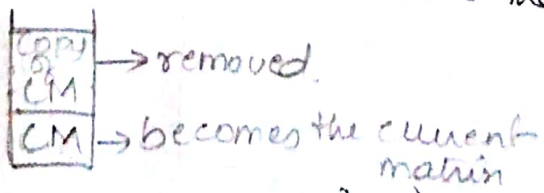
```
glGetIntegerv(GL_MODELVIEW_STACK_DEPTH, numMats);
```

* glPushMatrix();



copy the current matrix at top of active stack & store that copy in second stack position

* glPopMatrix();



destroys the matrix at top of stack & second matrix in the stack becomes the current matrix.