MODULE 3

Clipping,3D Geometric Transformations, Color and Illumination Models

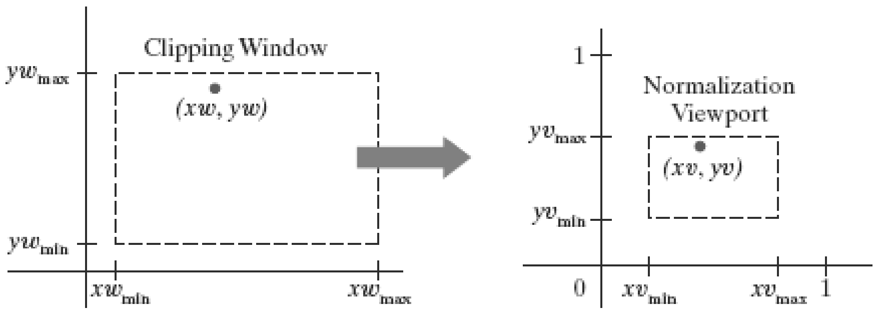
The Clipping Window

Rectangular clipping windows are easily defined by giving the coordinates of two opposite corners of each rectangle. Clipping window with any shape, size, and orientation we can choose.

Normalization and Viewport Transformations

Q.Design a transformation matrix for window to viewport transformation.

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



- To transform the world-coordinate point into the same relative position within the viewport, we require that

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}}$$

- Solving these expressions for the viewport position (x_v, y_v), we have

$$x_v = s_x x_w + t_x$$

$$y_v = s_y y_w + t_y$$

Where the scaling factors are

$$s_x = \frac{xv_{\max} - xv_{\min}}{xw_{\max} - xw_{\min}}$$

$$s_y = \frac{yv_{\max} - yv_{\min}}{yw_{\max} - yw_{\min}}$$

and the translation factors are

$$t_x = \frac{xw_{\max} xv_{\min} - xw_{\min} xv_{\max}}{xw_{\max} - xw_{\min}}$$

$$t_y = \frac{yw_{\max} yv_{\min} - yw_{\min} yv_{\max}}{yw_{\max} - yw_{\min}}$$

We could obtain the transformation from world coordinates to viewport coordinates with the following sequence:

1. Scale the clipping window to the size of the viewport using a fixed-point position of ($xw_{\min}$, $yw_{\min}$).

2. Translate ($xw_{\min}$, $yw_{\min}$) to ($xv_{\min}$, $yv_{\min}$).

The scaling transformation in step (1) can be represented with the two dimensional Matrix

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & xw_{\min}(1 - s_x) \\ 0 & s_y & yw_{\min}(1 - s_y) \\ 0 & 0 & 1 \end{bmatrix}$$

The two-dimensional matrix representation for the translation of the lower-left corner of the clipping window to the lower-left viewport corner is

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & xv_{\min} - xw_{\min} \\ 0 & 1 & yv_{\min} - yw_{\min} \\ 0 & 0 & 1 \end{bmatrix}$$

And the composite matrix representation for the transformation to the normalized viewport is

$$\mathbf{M}_{\text{window, normviewp}} = \mathbf{T} \cdot \mathbf{S} = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

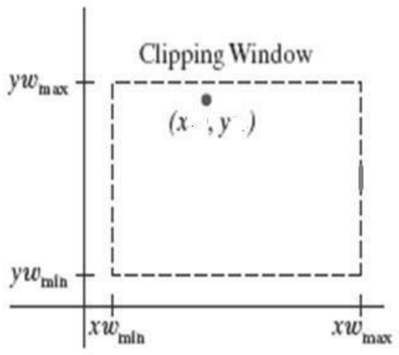
Clipping Algorithms

Any procedure that eliminates those portions of a picture that are either inside or outside a specified region of space is referred to as a **clipping algorithm** or simply **clipping.**

Different objects clipping are

1. Point clipping
2. Line clipping (straight-line segments)
3. Fill-area clipping (polygons)
4. Curve clipping
5. Text clipping

Two-Dimensional Point Clipping



For a clipping rectangle in standard position, we save a two-dimensional point **P** = (x, y) for display if the following inequalities are satisfied:

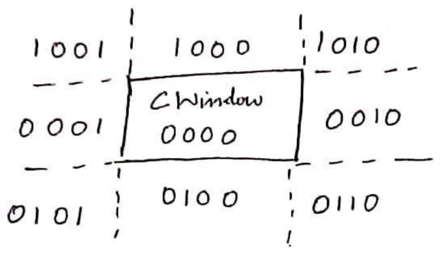
$$xw_{min} \le x \le xw_{max} \quad \text{and} \quad yw_{min} \le y \le yw_{max}$$

If any of these four inequalities is not satisfied, the point is clipped

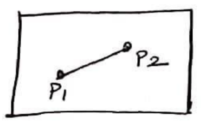
Q.What is Clipping? With the help of a suitable example explain cohen Sutherland line algorithm.

Cohen Sutherland Algorithm

- Algorithm works on Region Code
- Region code is 4 Bit Code (A B R L) Above, Below, Right, Left
  (T B R L) Top, Bottom, Right, Left

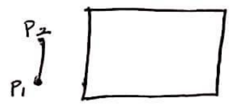


Case 1: If both endpoint Region code is zero, Completely Inside & Visible



P_1	0000	(zero)
P_2	0000	(zero)
AND	0000.	(Zero)

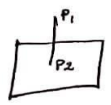
Case 2: If both endpoint Region code is Nonzero, then apply logical AND, and result is Nonzero, Completely Outside & Invisible

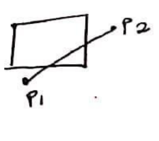


P_1 =	0001	(Nonzero)
P_2 =	0001	(Nonzero)
AND	0001	(Nonzero)

Case 3: a) If one of P_1 & P_2 is Zero } logical AND is Zero
b) Both Nonzero }

Then Find Intersection Point





Finding Intersection Points

1) Find y values of Vertical lines

Consider a line Segment (x_1, y_1) $(x_2 y_2)$ & Intersection Point (x, y)

Find Slope of $(x_1 y_1)$ & (x, y)

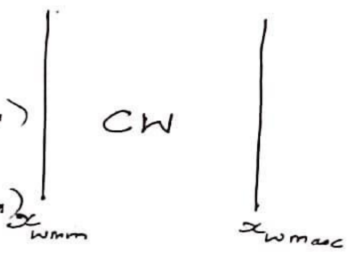
$$m = \frac{y - y_1}{x - x_1}$$

$$(y - y_1) = m(x - x_1)$$

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

$x = x_{wmin}$ (left border)

$x = x_{wmax}$ (right border)



2) Find x values of horizontal line

Find Slope of $(x_1 y_1)$ & (x, y)

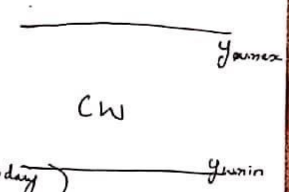
$$m = \frac{y - y_1}{x - x_1}$$

$$m(x - x_1) = y - y_1$$

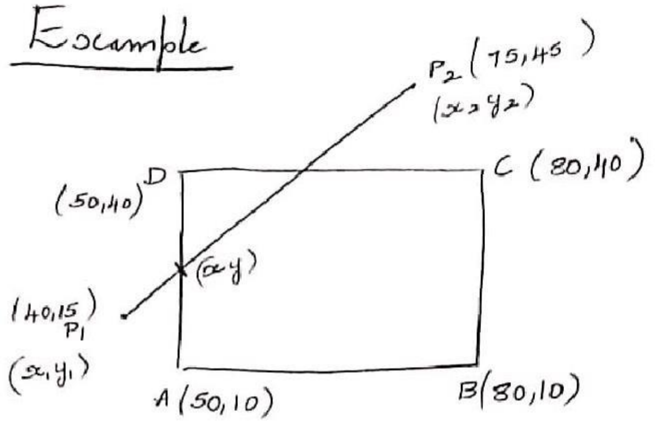
$$\boxed{x = x_1 + \frac{(y - y_1)}{m}}$$

$y = y_{wmin}$ (bottom boundary)

$y = y_{wmax}$ (upper boundary)



Example



P_1 = 0 0 0 1 (Nonzero)
P_2 = 1 0 0 0 (Nonzero)
0 0 0 0 Zero

Intersection Point (x, y)

$x_{wmin} = 50$

$x_{wmax} = 80$

$y = y_1 + m(x - x_1)$

$= 15 + 0.85(50 - 40)$

$= 15 + 0.85(10)$

$\boxed{y = 23.5}$

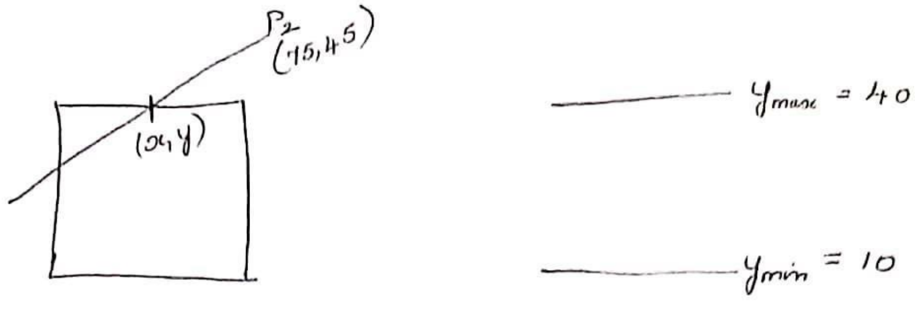
$(x,y) = \boxed{(50, 23.5)}$

P_1 (x_1, y_1) (40, 15) —— P_2 (x_2, y_2) (75, 45)

$m = \dfrac{y_2 - y_1}{x_2 - x_1}$

$= \dfrac{45 - 15}{75 - 40} = \dfrac{30}{35}$

$\boxed{m = 0.85}$



(x, y)

$$x = x_1 + \frac{1}{m}(y - y_1)$$

$$= 75 + \frac{1}{0.85}(40 - 45)$$

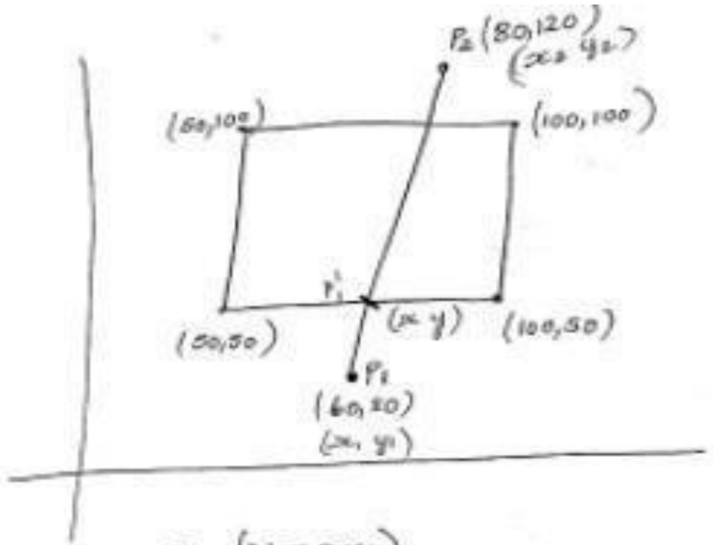
$$= 75 + (-5.882)$$

$$\boxed{x = 69.2}$$

$$\boxed{x, y = (69.2, 40)}$$

Q.Explain Cohen Sutherland line clipping, clip the lines with coordinates (x0,y0)=(60,20) and (x1,y1) =(80,120) given the window boundaries(xwmin,ywmin)=(50,50) and (xmax, ymax) = (100,100)

Soln



P_1 = 0100 (Non zero)

P_2 = 1000 (Non zero)

[illegible] = 0000 (zero) → Case 3

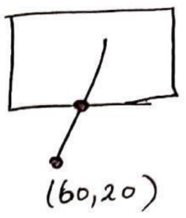
Find Intersection Point

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{120-20}{80-60} = 5$$

$$\boxed{m = 5}$$

Find $P_1'(\overset{\downarrow}{x}, y)$

We know $\boxed{\begin{array}{l} y = y_{wmin} = 50 \\ \text{Find } x \end{array}}$



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

$$= 60 + \frac{(50 - 20)}{5}$$

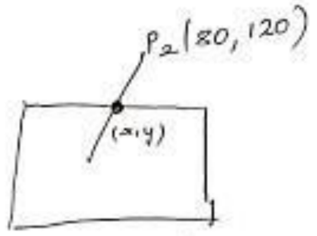
$$\boxed{x = 66}$$

$$\boxed{P_1'(x, y) = (66, 50)}$$

Find $P_2'(\overset{\downarrow}{x}\ y)$

We Know $\boxed{y = y_{max} = 100}$

Find x



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

$$= 80 + \frac{(100 - 120)}{5}$$

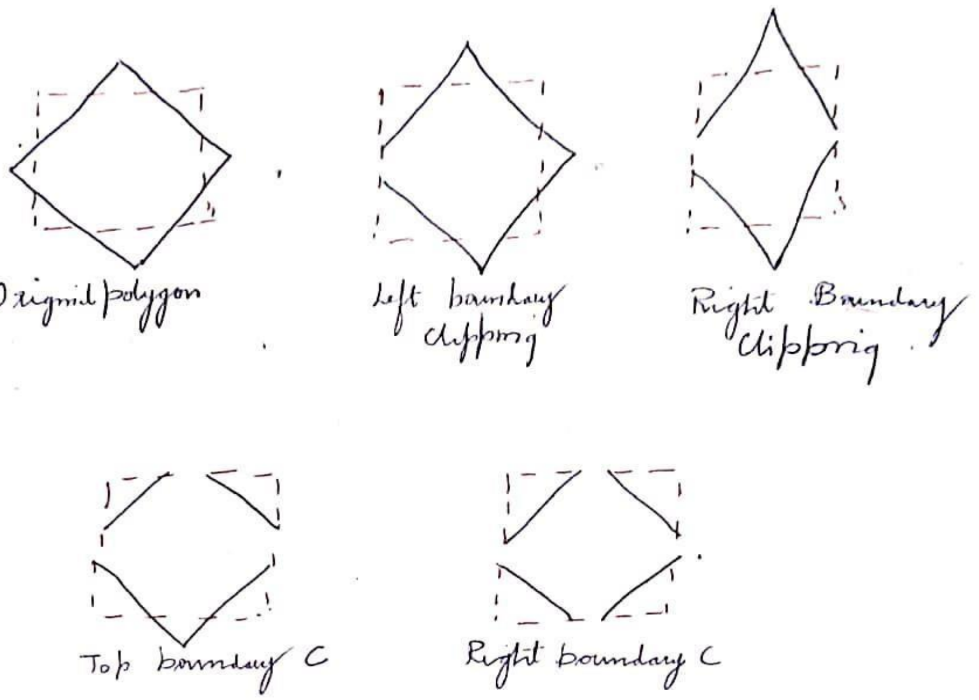
$$= 80 + \frac{(-20)}{5}$$

$$= 80 - 4$$

$$\boxed{x = 76}$$

$$\boxed{P_2'(x,y) = (76, 100)}$$

Polygon Clipping



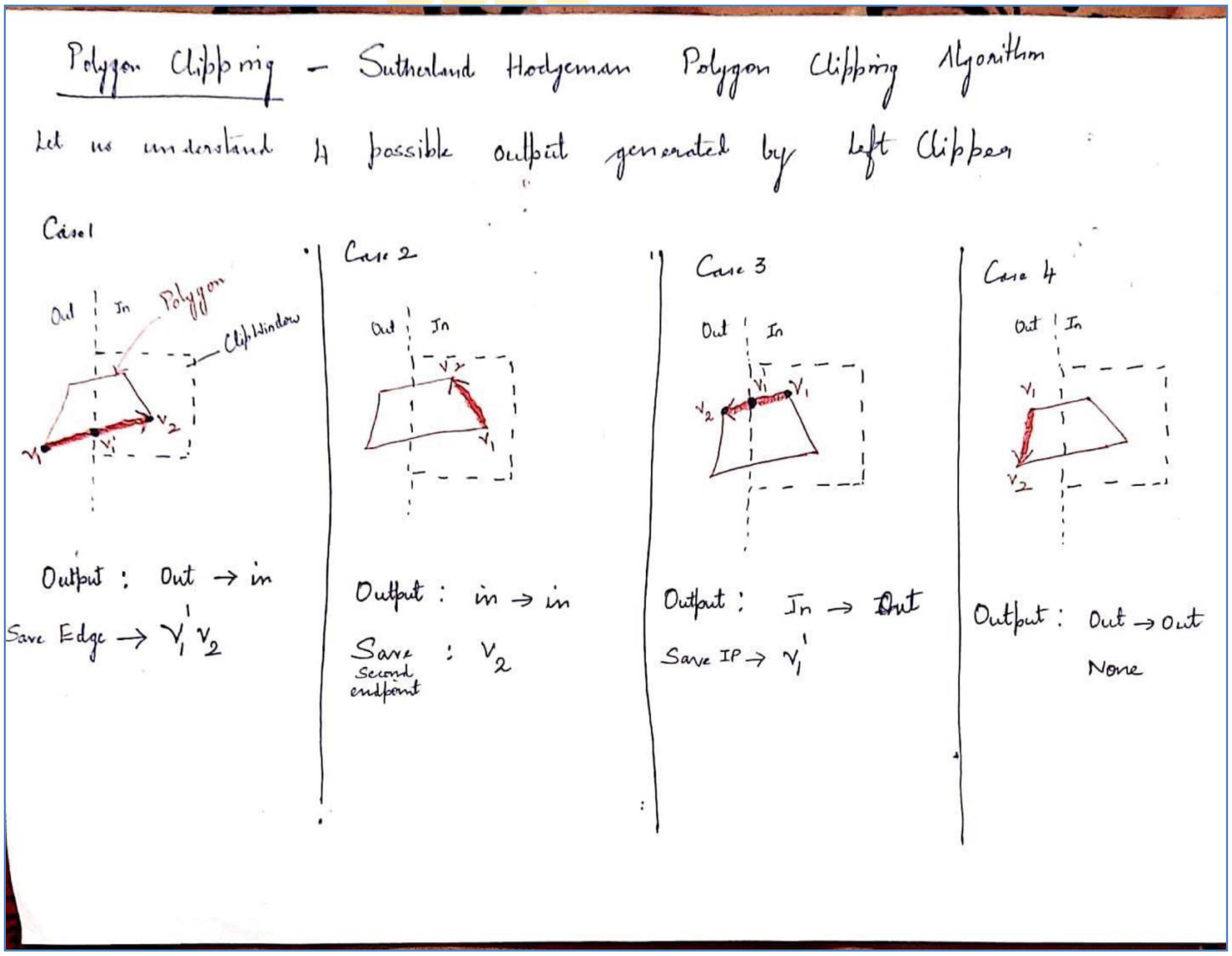
Q.What is Clipping? Explain with example Sutherland Hodgeman Polygon clipping algorithm

Clipping: Any procedure that eliminates those portions of a picture that are either inside or outside a specified region of space

Sutherland Hodgeman Polygon clipping algorithm

Send pair of endpoints for each successive polygon line segment through the series of clippers.
Four possible cases:

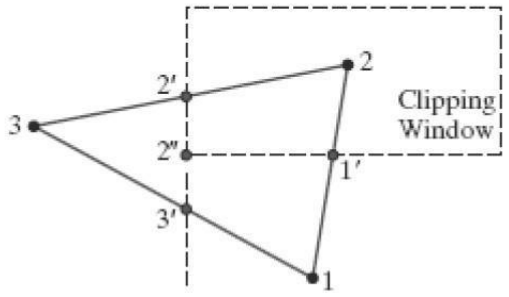
1. If the first input vertex is outside this clipping-window border and the second vertex is inside, both the intersection point of the polygon edge with the window border and the second vertex are sent to the next clipper
2. If both input vertices are inside this clipping-window border, only the second vertex is sent to the next clipper
3. If the first vertex is inside and the second vertex is outside, only the polygon edge intersection position with the clipping-window border is sent to the next clipper
4. If both input vertices are outside this clipping-window border, no vertices are sent to the next clipper

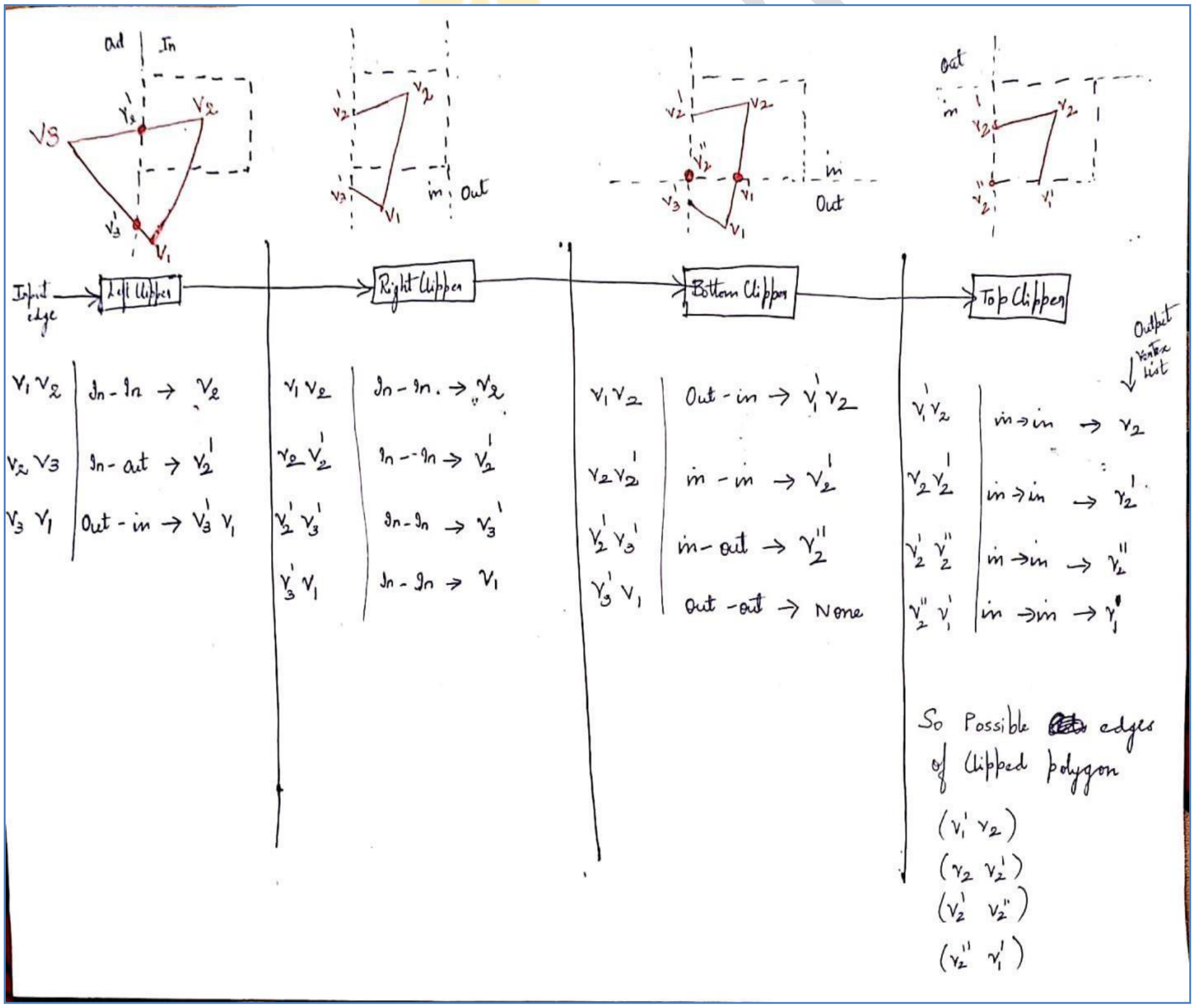


Polygon Clipping – Sutherland Hodgeman Polygon Clipping Algorithm

Let us understand 4 possible output generated by Left Clipper

Example for Sutherland Hodgeman Polygon clipping algorithm





V_1V_2	In - In → V_2
V_2V_3	In - out → V_2'
V_3V_1	Out - in → $V_3'V_1$

V_1V_2	In - In. → V_2
V_2V_2'	In -- In → V_2'
$V_2'V_3'$	In - In → V_3'
$V_3'V_1$	In - In → V_1

V_1V_2	Out - in → $V_1'V_2$
V_2V_2'	in - in → V_2'
$V_2'V_3'$	in - out → V_2''
$V_3'V_1$	Out - out → None

$V_1'V_2$	in → in → V_2
V_2V_2'	in → in → V_2'
$V_2'V_2''$	in → in → V_2''
$V_2''V_1'$	in → in → V_1'

So Possible edges of Clipped polygon

$(V_1'\ V_2)$

$(V_2\ V_2')$

$(V_2'\ V_2'')$

$(V_2''\ V_1')$

3DGeometric Transformations

Three-Dimensional Geometric Transformations

Q.With the help of a suitable diagram explain basic 3D Geometric transformation techniques and give the transformation matrix.

The most common 3D transformations are:

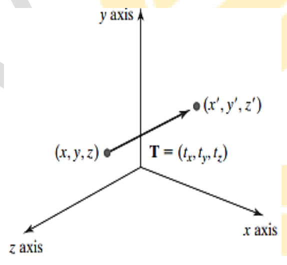
Translation
Rotation
Scaling

Three-Dimensional Translation

- A position P=(x, y, z) in three-dimensional space is translated to a location P'=(x',y',z') by adding translation distances tx, ty, and tz to the Cartesian coordinates of P.

$$x' = x + t_x, \qquad y' = y + t_y, \qquad z' = z + t_z$$

- 3D translation is given in Figure



- 3D translation operations can be represented in matrix format. The coordinate positions, P and P', are represented in homogeneous coordinates with four-element column matrices, and the translation operator T is a 4×4 matrix:

$$\mathbf{P'} = \mathbf{T} \cdot \mathbf{P}$$

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

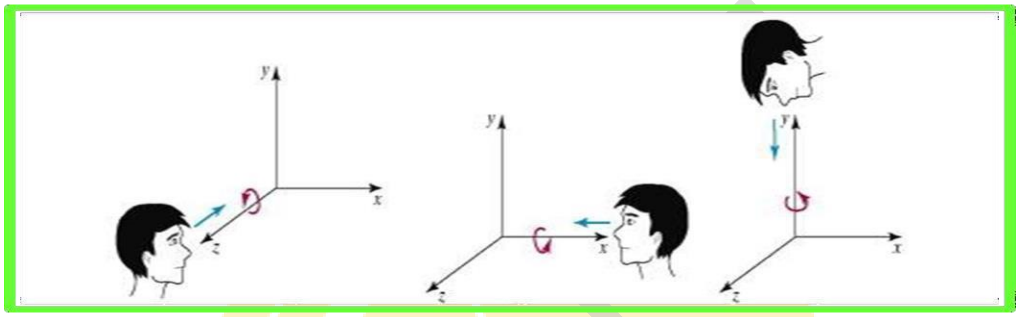
Three-Dimensional Rotation

- Rotate an object about any axis in space.

  - Easiest rotation axes to handle are those that are parallel to the Cartesian-coordinate axes.

  - Also can use combinations of coordinate-axis rotations (along with appropriate translations) to specify a rotation about any other line in space.

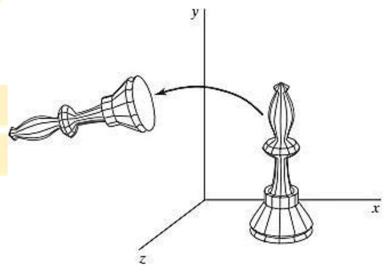
- Positive rotation angles produce counterclockwise rotations about a coordinate axis (for negative direction along that coordinate axis)



- The 3D z-axis rotation equations are easily extended to three dimensions, as follows:

Along z axis:

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



Where, θ ❼ the rotation angle about the z axis
❼ z-coordinate values are unchanged by this transformation

- 3D Rotation operations can be represented in matrix format. In homogeneous-coordinate form, the three-dimensional *z*-axis rotation equations are

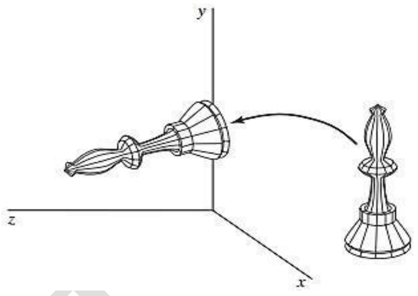
$$\mathbf{P'} = \mathbf{R}_z(\theta) \cdot \mathbf{P}$$

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

- Similarly along x axis and y axis

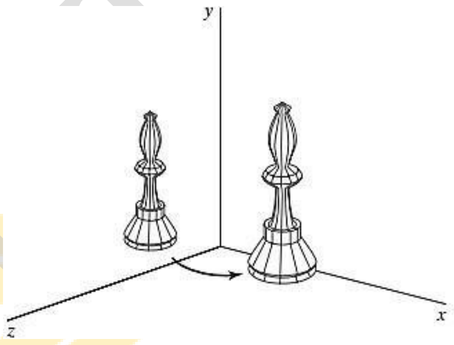
Along x axis

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



Along y axis

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



Three-Dimensional Scaling

- The matrix expression for the three-dimensional scaling transformation of a position P=(x, y, z)relative to the coordinate origin is a simple extension of 3D scaling. Include the parameter for z-coordinate scaling in the transformation matrix:

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

- The three-dimensional scaling transformation for a point position can be represented as

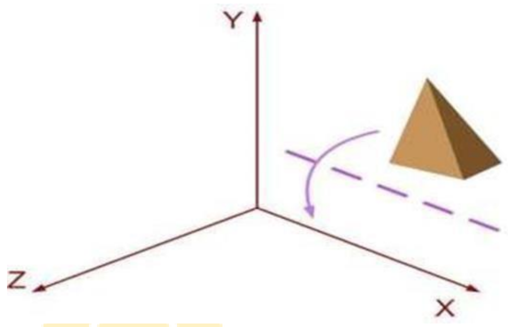
$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P}$$

where scaling parameters *sx*, *sy*, and *sz* are assigned any positive values.

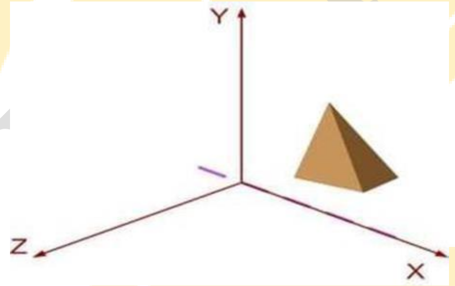
- Explicit expressions for the scaling transformation relative to the origin are

Q.Design transformation matrix to rotate an 3D Object about an axis that is parallel to one of the coordinate axes.

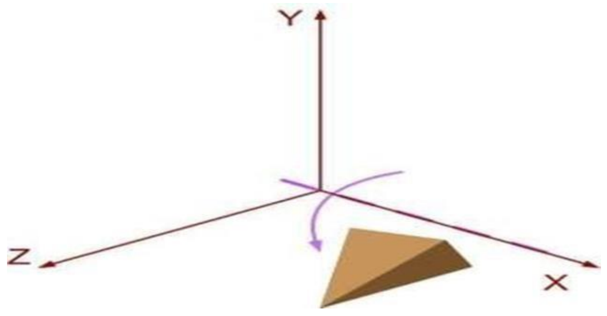
A rotation matrix for any axis that does not coincide with a coordinate axis(as shown in below first figure) can be set up as a composite transformation involving combinations of translations and the coordinate-axis rotations the following transformation sequence is used:



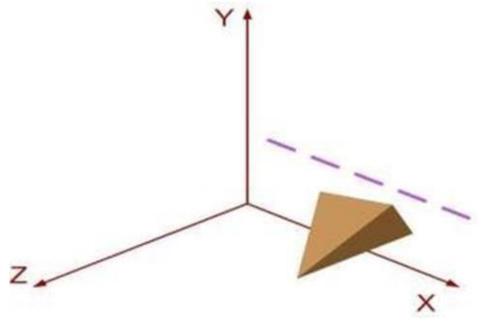
1. **Translate the object so that the rotation axis coincides with the parallel coordinate axis.**



2. **Perform the specified rotation about that axis.**



3. Translate the object so that the rotation axis is moved back to its original position.



A coordinate position **P** is transformed with the sequence shown in this figure as

$$\mathbf{P}' = \mathbf{T}^{-1} \cdot \mathbf{R}_x(\theta) \cdot \mathbf{T} \cdot \mathbf{P}$$

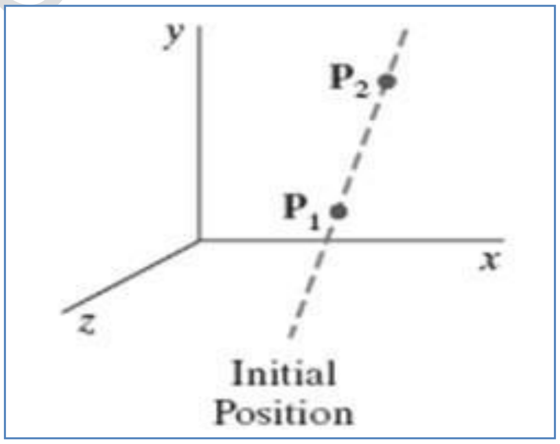
Where the composite rotation matrix for the transformation is

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

Q.Obtain the matrix representation for rotation of a object about an arbitrary axis

Rotation about arbitrary axis that is not parallel to one of the coordinate axes, we can accomplish the required rotation in five steps:

Consider the Initial position of object in the below figure



Components of the rotation-axis vector are then computed as

$$\mathbf{V} = \mathbf{P2} - \mathbf{P1}$$
$$= (x_2 - x_1,\ y_2 - y_1,\ z_2 - z_1)$$

The unit rotation-axis vector **u** is

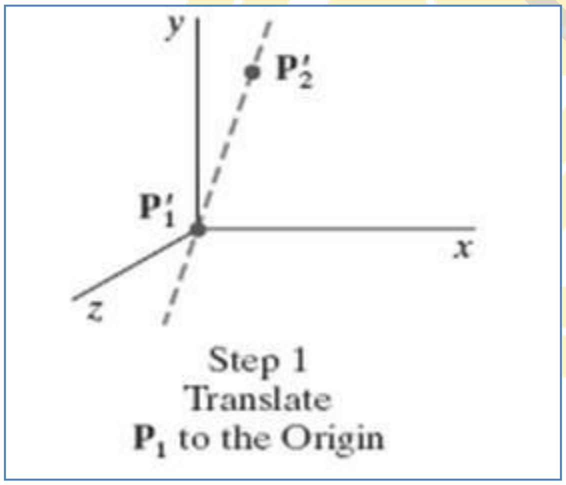
$$\mathbf{u} = \frac{\mathbf{V}}{|\mathbf{V}|} = (a, b, c)$$

Where the components *a*, *b*, and *c* are the direction cosines for the rotation axis

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

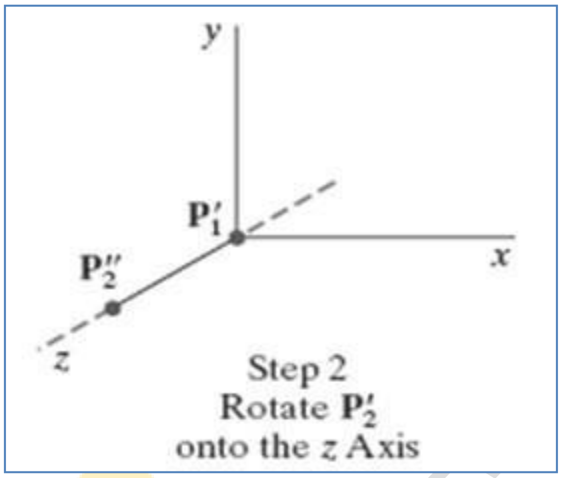
Rotation about arbitrary axis that is not parallel to one of the coordinate axes, we can accomplish the required rotation in five steps:

1. Translate the object so that the rotation axis passes through the coordinate origin.



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

2. Rotate the object so that the axis of rotation coincides with one of the coordinate axes.



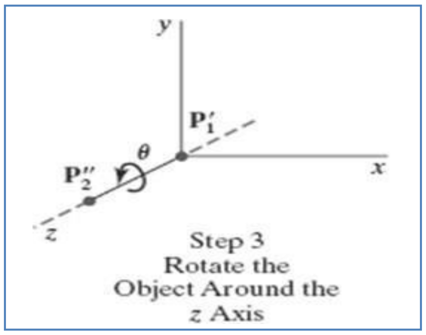
Rotation of u around the x axis into the x z plane is accomplished by rotating u' (the projection of u in the y z plane) through angle α onto the z axis.

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

Rotation of unit vector u" (vector u after rotation into the x z plane) about the y axis. Positive rotation angle β aligns u" with vector uz .

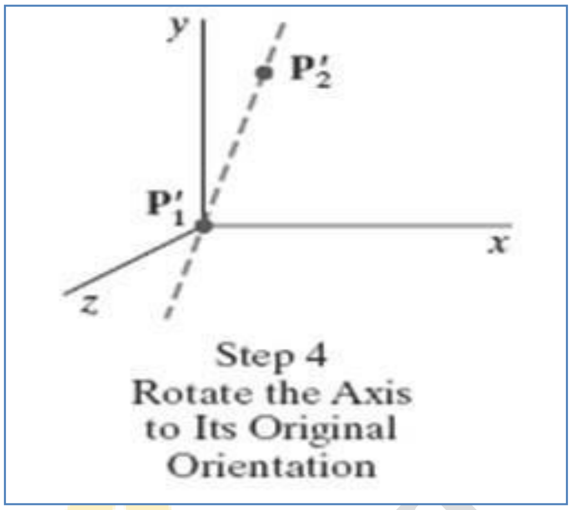
$$R_y(\beta) = \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. Perform the specified rotation about the selected coordinate axis.

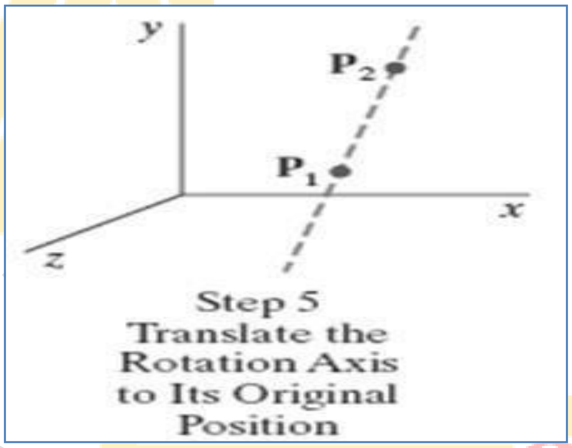


$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4. Apply inverse rotations to bring the rotation axis back to its original orientation.



5. Apply the inverse translation to bring the rotation axis back to its original spatial position.



The transformation matrix for rotation about an arbitrary axis can then be expressed as the composition of these seven individual transformations:

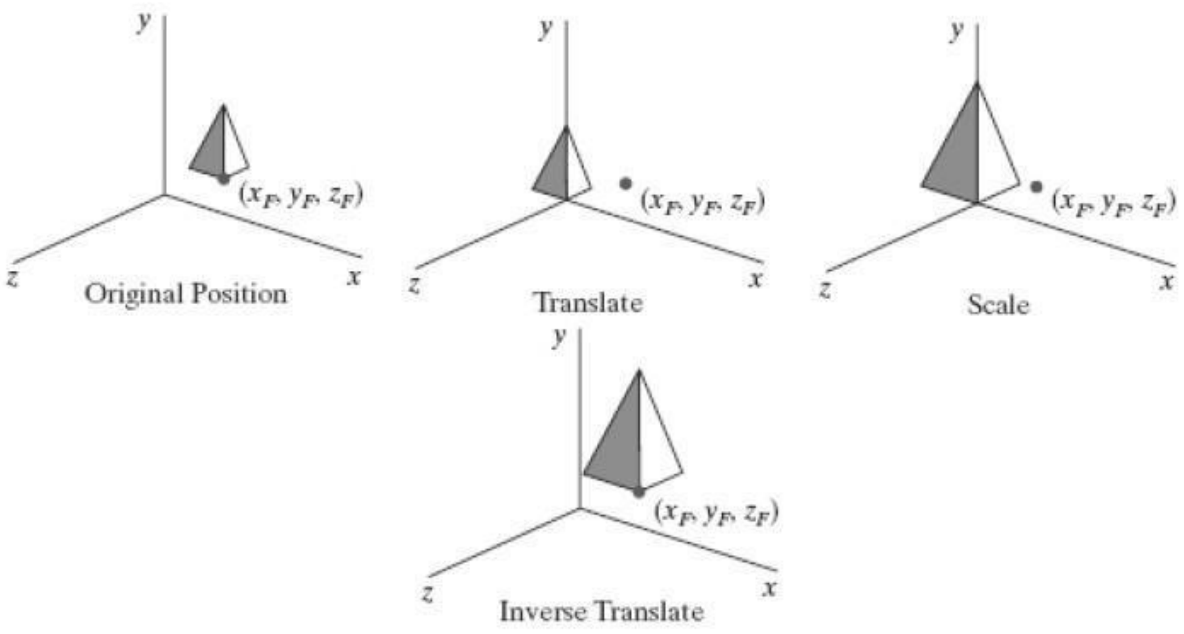
$$\boxed{R(\theta) = T^{-1} \cdot R_x^{-1}(\alpha) \cdot R_y^{-1}(\beta) \cdot R_z(\theta) \cdot R_y(\beta) \cdot R_x(\alpha) \cdot T}$$

Q.Obtain Scaling transformation matrix with respect to any selected fixed position.

Scaling transformation with respect to any selected fixed position (x_f , y_f , z_f) using the following transformation sequence:

1. Translate the fixed point to the origin.
2. Apply the scaling transformation relative to the coordinate origin
3. Translate the fixed point back to its original position.

This sequence of transformations is demonstrated



$$\mathbf{T}(x_f, y_f, z_f) \cdot \mathbf{S}(s_x, s_y, s_z) \cdot \mathbf{T}(-x_f, -y_f, -z_f) = \begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Other Three-Dimensional Transformations

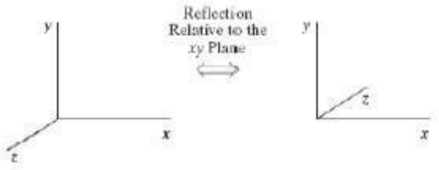
Q.Explain the 3D reflection and shearing.

Three-Dimensional Reflections

A reflection in a three-dimensional space can be performed relative to a selected *reflection axis* or with respect to a *reflection plane*.

Reflections with respect to a plane are similar; when the reflection plane is a coordinate plane (x_y, x_z, or y_z), we can think of the transformation as a 180◦ rotation in four-dimensional space with a conversion between a left-handed frame and a right-handed frame

An example of a reflection that converts coordinate specifications froma right handed system to a left-handed system is shown below



The matrix representation for this reflection relative to the *xy* plane is

$$M_{zreflect} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Three-Dimensional Shears

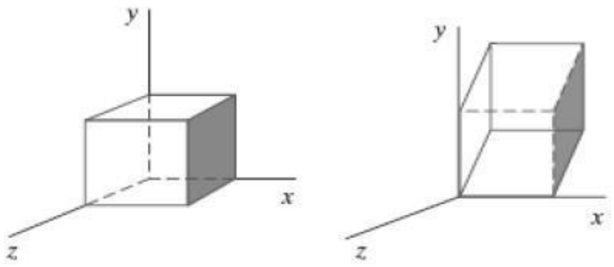
These transformations can be used to modify object shapes.

For three-dimensional we can also generate shears relative to the z axis.

A general z-axis shearing transformation relative to a selected reference position is produced with the following matrix:

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

The Below figure shows the shear transformation of a cube



Affine Transformations

Q.Explain the meaning of Affine transformation

Affine transformations (in two dimensions, three dimensions, or higher dimensions) have the general properties that parallel lines are transformed into parallel lines, and finite points map to finite points.

Translation, rotation, scaling, reflection,andshear are examples of affine transformations.

Another example of an affine transformation is the conversion of coordinate descriptions for a scene from one reference system to another because this transformation can be described as a combination of translation and rotation

A coordinate 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**

OpenGL Geometric-Transformation Functions

OpenGL Matrix Stacks

glMatrixMode:

used to select the modelview composite transformation matrix as the target of subsequent OpenGL transformation calls

four modes: modelview, projection, texture, and color

the top matrix on each stack is called the "current matrix".

for that mode. the **modelview matrix stack** is the 4 × 4 composite matrix that combines the viewing transformations and the various geometric transformations that we want to apply to a scene.

OpenGL supports a modelview stack depth of at least 32,

glGetIntegerv (GL_MAX_MODELVIEW_STACK_DEPTH, stackSize);

determine the number of positions available in the modelview stack for a particular implementation of OpenGL.

It returns a single integer value to array **stackSize**

other OpenGL symbolic constants: GL_MAX_PROJECTION_STACK_DEPTH, GL_MAX_TEXTURE_STACK_DEPTH, or GL_MAX_COLOR_STACK_DEPTH.

We can also find out how many matrices are currently in the stack with

glGetIntegerv (GL_MODELVIEW_STACK_DEPTH, numMats);

We have two functions available in OpenGL for processing the matrices in a stack

glPushMatrix ( );

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

glPopMatrix ( );

which destroys the matrix at the top of the stack, and the second matrix in the stack becomes the current matrix

Illumination and Color

Illumination Models

Q.Explain the different types of light sources supported by OpenGL

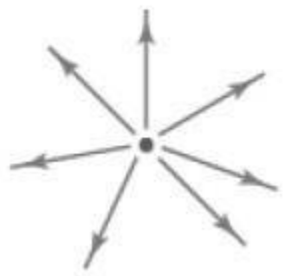
A light source can be defined with a number of properties. We can specify its position, the color of the emitted light, the emission direction, and its shape.

Different Light Sources

1. Point Light Sources
2. Infinitely Distant Light Sources
3. Directional Light Sources and Spotlight Effects
4. Extended Light Sources and the Warn Model

1.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

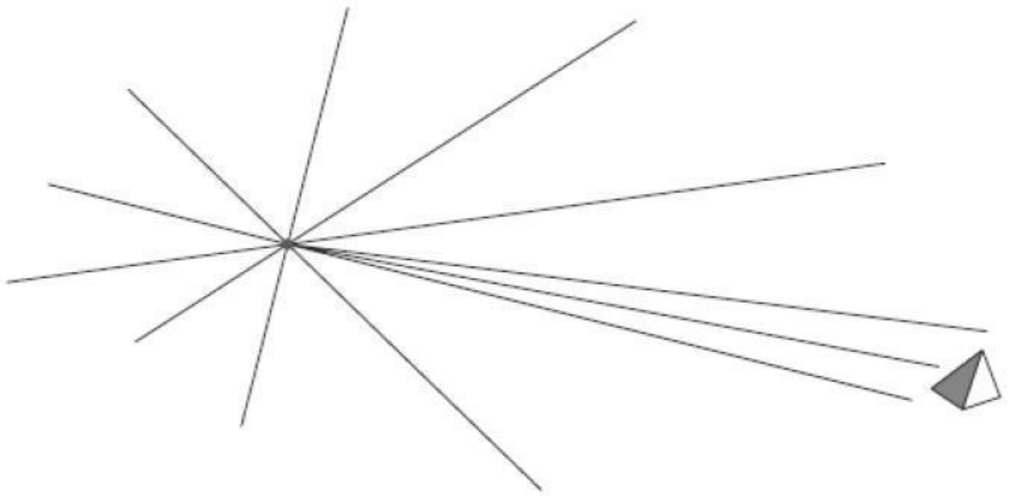


A point source for a scene by giving its position and the color of the emitted light. light rays are generated along radially diverging paths from the single-color source position. This light-source model is a reasonable approximation for sources whose dimensions are small compared to the size of objects in the scene

2. 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.

The light path from a distant light source to any position in the scene is nearly constant



We can simulate an infinitely distant light source by assigning it a color value and a fixed direction for the light rays emanating from the source.

The vector for the emission direction and the light-source color are needed in the illumination calculations, but not the position of the source.

3. Directional Light Sources and Spotlight Effects

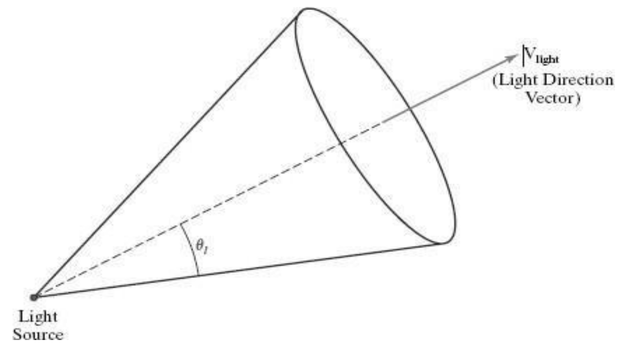
A local light source can be modified easily to produce a directional, or spotlight, beam of light.

If an object is outside the directional limits of the light source, we exclude it from illumination by that source

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

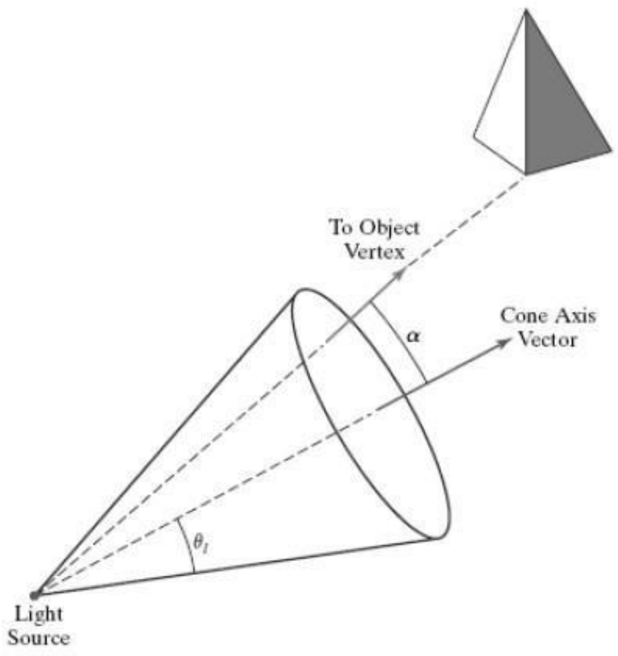
We can denote $\mathbf{V}_{light}$ as the unit vector in the light-source direction and $\mathbf{V}_{obj}$ as the unit vector in the direction from the light position to an object position.

Then $\mathbf{V}_{obj} \cdot \mathbf{V}_{light} = \cos\alpha$



where angle α is the angular distance of the object from the light direction vector.

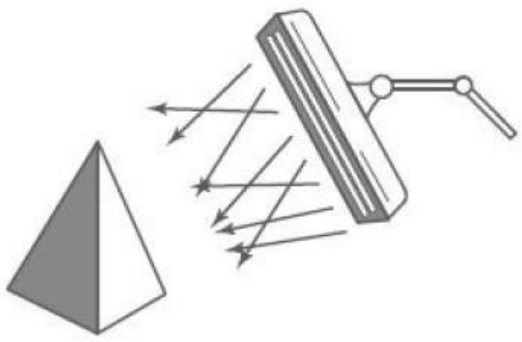
If we restrict the angular extent of any light cone so that $0^\circ < \theta_l \leq 90^\circ$, then the object is within the spotlight if $\cos\alpha \geq \cos\theta_l$, as shown



. If $\mathbf{V}_{obj} \cdot \mathbf{V}_{light} < \cos\theta_l$, however, the object is outside the light cone.

4. Extended Light Sources and the Warn Model

When we want to include a large light source at a position close to the objects in a scene, such as the long neon lamp, we can approximate it as a lightemitting surface



One way to do this is to model the light surface as a grid of directional point emitters.
We can set the direction for the point sources so that objects behind the light-emitting surface are not illuminated.
We could also include other controls to restrict the direction of the emitted light near the edges of the source

The **Warn model** provides a method for producing studio lighting effects using sets of point emitters with various parameters to simulate the barn doors, flaps, and spotlighting controls employed by photographers.
Spotlighting is achieved with the cone of light discussed earlier, and the flaps and barn doors provide additional directional control

Q. Explain basic illumination models

These below models are used to calculate the intensity of light that is reflected at a given point on a surface

- Ambient lighting
- Diffuse reflection
- Specular reflection

Ambient lighting

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

- The reflected intensity I_{amb} of any point on the surface is:

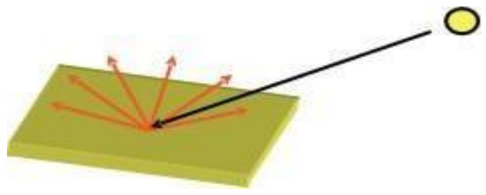
$$\boxed{I_{amb} = K_a I_a}$$

I_a - ambient light intensity
$K_a \in [0,1]$ - surface ambient reflectivity

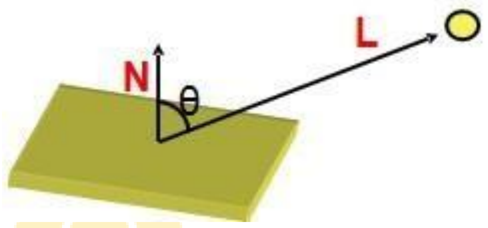
- In principle I_a and K_a are functions of color, so we have I^R_{amb}, I^G_{amb} and I^B_{amb}

Diffuse reflection

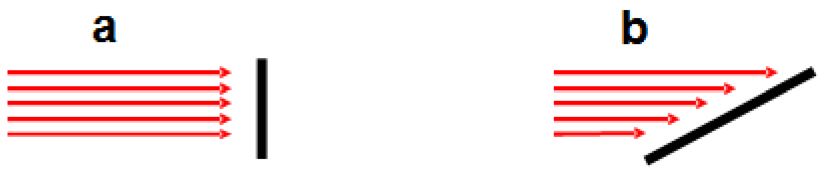
- The incident light on the surface is scattered with equal intensity in all directions, independent of the viewing position. Such surfaces are called **ideal diffuse reflectors**
- They are also referred to as **Lambertian reflectors,** because the reflected radiant light energy fromany point on the surface is calculated with **Lambert's cosine law.**
- Diffuse (Lambertian) surfaces are rough or grainy, like clay, soil, fabric
- The surface appears equally bright from all viewing directions



- The brightness at each point is proportional to cos(θ)



- Brightness is proportional to cos(θ) because a surface (a) perpendicular to the light direction is more illuminated than a surface (b) at an oblique angle



- The reflected intensity I_{diff} of a point on the surface is:

$$\boxed{I_{diff} = K_d I_p \cos(\theta) = K_d I_p (N \cdot L)}$$

I_p - the point light intensity. May appear as attenuated source $f_{att}(r)I_P$

$K_d \in [0,1]$ - the surface diffuse reflectivity

N - the surface normal

L - the light direction

NOTE: If N and L have unitary length: $\cos(\theta) = N \cdot L$

- Commonly, there are two types of light sources:
  - A background ambient light
  - A point light source

- The equation that combines the two models is:

$$\boxed{I = I_{diff} + I_{amb} = K_d I_p N \cdot L + K_a I_a}$$

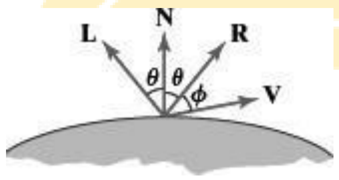
Specular reflection

Q.Explain Specular Reflection and Phong model

Q.Describe phone lighting model

Specular reflection

- The bright spot, or specular reflection, that we can see on a shiny surface is the result of total, or near total, reflection of the incident light in a concentrated region around the specular-reflection angle.
- The below figure shows the specular reflection direction for a position on an illuminated surface



N represents: unit normal surface vector The specular reflection angle equals the angle of the incident light, with the two angles measured on opposite sides of the unit normal surface vector **N**

Rrepresents the unit vector in the direction of ideal specular reflection,

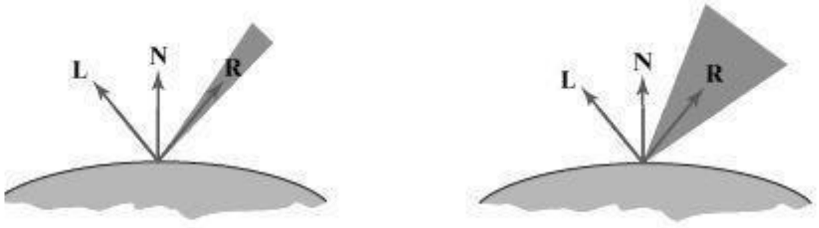
L is the unit vector directed toward the point light source, and

Vis the unit vector pointing to the viewer fromthe selected surface position.

Angle φ is the viewing angle relative to the specular-reflection direction **R**

Phong model

- An empirical model for calculating the specular reflection range, developed by Phong Bui Tuong and called the Phong specular-reflection model or simply the Phon G model, sets the intensity of specular reflection proportional to $\cos^{ns} \varphi$
- Angle φ can be assigned values in the range 0◦ to 90◦, so that cos φ varies from 0 to 1.0.
- The value assigned to the specular-reflection exponent *ns* is determined by the type of surface that we want to display.
    - A very shiny surface is modeled with a large value for *ns* (say, 100 or more)
    - Smaller values (down to 1) are used for duller surfaces.
    - For a perfect reflector, *ns* is infinite.
    - For a rough surface, such as chalk or cinderblock, *ns* is assigned a value near 1.



Shiny Surface (Large n_s)   Dull Surface (Small n_s)

- We can approximately model monochromatic specular intensity variations using a specular-reflection coefficient, W(θ), for each surface.
- W(θ) tends to increase as the angle of incidence increases.
- At θ = 90◦, all the incident light is reflected (W(θ) = 1)
- Using the spectral-reflection function W(θ), we can write the Phong specular-reflection model as.

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

where I_l is the intensity of the light source, and φ is the viewing angle relative to the specular-reflection direction **R**.

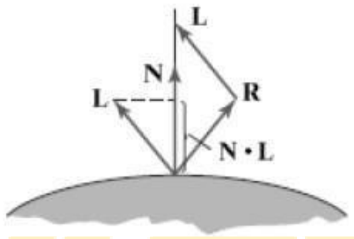
Because**V**and **R** are unit vectors in the viewing and specular-reflection directions, we can calculate the value of cos φ with the dot product **V·R**.

In addition, no specular effects are generated for the display of a surface if **V** and **L** are on the same side of the normal vector **N** or if the light source is behind the surface

We can determine the intensity of the specular reflection due to a point light source at a surface position with the calculation

$$I_{l,\text{spec}} = \begin{cases} k_s I_l (\mathbf{V}\cdot\mathbf{R})^{n_s}, & \text{if } \mathbf{V}\cdot\mathbf{R} > 0 \quad \text{and} \quad \mathbf{N}\cdot\mathbf{L} > 0 \\ 0.0, & \text{if } \mathbf{V}\cdot\mathbf{R} \le 0 \quad \text{or} \quad \mathbf{N}\cdot\mathbf{L} \le 0 \end{cases}$$

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



The projection of **L** onto the direction of the normal vector has a magnitude equal to the dot product**N·L**, which is also equal to the magnitude of the projection of unit vector **R** onto the direction of **N**.

Therefore, from this diagram, we see that

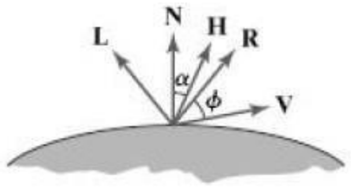
$$\mathbf{R} + \mathbf{L} = (2\mathbf{N}\cdot\mathbf{L})\mathbf{N}$$

and the specular-reflection vector is obtained as

$$\mathbf{R} = (2\mathbf{N}\cdot\mathbf{L})\mathbf{N} - \mathbf{L}$$

A somewhat simplified Phong model is obtained using the **halfway vector H** between **L** and **V** to calculate the range of specular reflections.

If we replace **V·R** in the Phong model with the dot product**N·H**, this simply replaces the empirical cos φ calculation with the empirical cos α calculation



The halfway vector is obtained as

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

For nonplanar surfaces, **N·H** requires less computation than **V·R** because the calculation of **R** at each surface point involves the variable vector **N**.

OpenGL Illumination Functions

Q.Explain OpenGL illumination functions

OpenGL Point Light-Source Function

glLight* (lightName, lightProperty, propertyValue);

A suffix code of **i** or **f** is appended to the function name, depending on the data type of the property value

lightName: GL_LIGHT0, GL_LIGHT1, GL_LIGHT2, . . . , GL_LIGHT7

lightProperty: must be assigned one of the OpenGL symbolic property constants

glEnable (lightName);→ turn on that light with the command

glEnable (GL_LIGHTING);→ activate the OpenGL lighting routines

Specifying an OpenGL Light-Source Position and Type

GL_POSITION:

specifies light-source position

this symbolic constant is used to set two light-source properties at the same time: the light-source position and the *light-source type*

Specifying OpenGL Light-Source Colors

Unlike an actual light source, an OpenGL light has three different color properties the symbolic color-property constants **GL_AMBIENT, GL_DIFFUSE,** and **GL_SPECULAR**

Example:

```
GLfloat blackColor [ ] = {0.0, 0.0, 0.0, 1.0};
GLfloat whiteColor [ ] = {1.0, 1.0, 1.0, 1.0};
glLightfv (GL_LIGHT3, GL_AMBIENT, blackColor);
glLightfv (GL_LIGHT3, GL_DIFFUSE, whiteColor);
glLightfv (GL_LIGHT3, GL_SPECULAR, whiteColor);
```

Specifying Radial-Intensity Attenuation Coefficients

For an OpenGL Light Source we could assign the radial-attenuation coefficient values as

```
glLightf (GL_LIGHT6, GL_CONSTANT_ATTENUATION, 1.5);
glLightf (GL_LIGHT6, GL_LINEAR_ATTENUATION, 0.75);
glLightf (GL_LIGHT6, GL_QUADRATIC_ATTENUATION, 0.4);
```

OpenGL Directional Light Sources (Spotlights)

There are three OpenGL property constants for directional effects: **GL_SPOT_DIRECTION, GL_SPOT_CUTOFF**, and **GL_SPOT_EXPONENT**

```
GLfloat dirVector [ ] = {1.0, 0.0, 0.0};
glLightfv (GL_LIGHT3, GL_SPOT_DIRECTION, dirVector);
glLightf (GL_LIGHT3, GL_SPOT_CUTOFF, 30.0);
glLightf (GL_LIGHT3, GL_SPOT_EXPONENT, 2.5);
```

OpenGL Global Lighting Parameters

glLightModel* (paramName, paramValue);

We append a suffix code of i or f, depending on the data type of the parameter value.

In addition, for vector data, we append the suffix code v.

Parameter paramName is assigned an OpenGL symbolic constant that identifies the global property to be set, and parameter paramValue is assigned a single value or set of values.

```
globalAmbient [ ] = {0.0, 0.0, 0.3, 1.0);
glLightModelfv (GL_LIGHT_MODEL_AMBIENT, globalAmbient);
```

glLightModeli (GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);

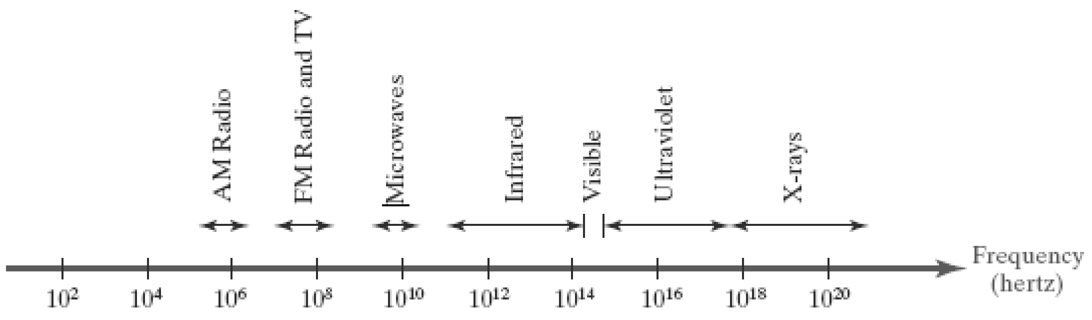
turn off this default and use the actual viewing position (which is the viewing-coordinate origin) to calculate V

Color Models

Properties of Light

The Electromagnetic Spectrum

- 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. The frequency is shown below



- Each frequency value within the visible region of the electromagnetic spectrum corresponds to a distinct **spectral color.**
- At the low-frequency end (approximately 3.8×1014 hertz) are the red colors, and at the high-frequency end (approximately 7.9 × 1014 hertz) are the violet colors.
- In the wave model of electromagnetic radiation, light can be described as oscillating transverse electric and magnetic fields propagating through space.
- The electric and magnetic fields are oscillating in directions that are perpendicular to each other and to the direction of propagation.
- For one spectral color (a monochromatic wave), the wavelength and frequency are inversely proportional to each other, with the proportionality constant as the speed of light (*c*):

$$c = \lambda f$$

- When white light is incident upon an opaque object, some frequencies are reflected and some are absorbed.
- If low frequencies are predominant in the reflected light, the object is described as red. In this case, we say that the perceived light has a dominant frequency (or dominant wavelength) at the red end of the spectrum.

- The dominant frequency is also called the **hue,** or simply the **color,** of the light.

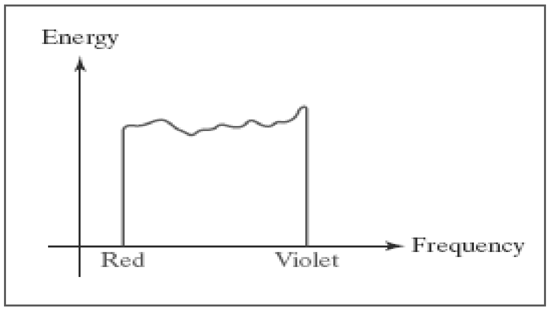
Psychological Characteristics of Color

Other properties besides frequency are needed to characterize our perception of Light

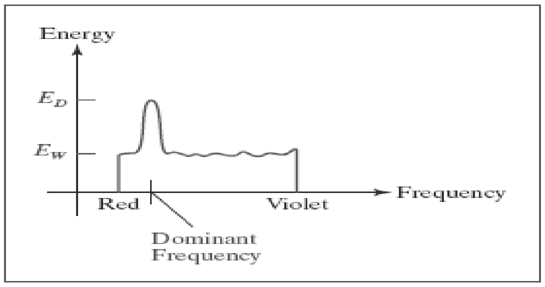
- **Brightness:** which corresponds to the total light energy and can be quantified as the luminance of the light.
- **Purity**, or the **saturation** of the light: Purity describes how close a light appears to be to a pure spectral color, such as red.
- **Chromaticity**, is used to refer collectively to the two properties describing color characteristics: purity and dominant frequency (hue).

We can calculate the brightness of the source as the area under the curve, which gives the total energy density emitted.

Purity (saturation) depends on the difference between ED and EW. Below figure shows Energy distribution for a white light source



- Below figure shows, Energy distribution for a light source with a dominant frequency near the red end of the frequency range.



Color Models

Q.Define color model. With neat diagram explain RGB and CYM color model

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

Primary Colors

- The hues 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.
- Examples of complementary color pairs are red and cyan, green and magenta, and blue and yellow

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 pigment for a "pure color" ("pure hue"), the artist adds a black pigment to produce different shades of that color.
- Tones of the color are produced by adding both black and white pigments.

Q.With the help of a suitable diagram, explain RGB and CMY color models.

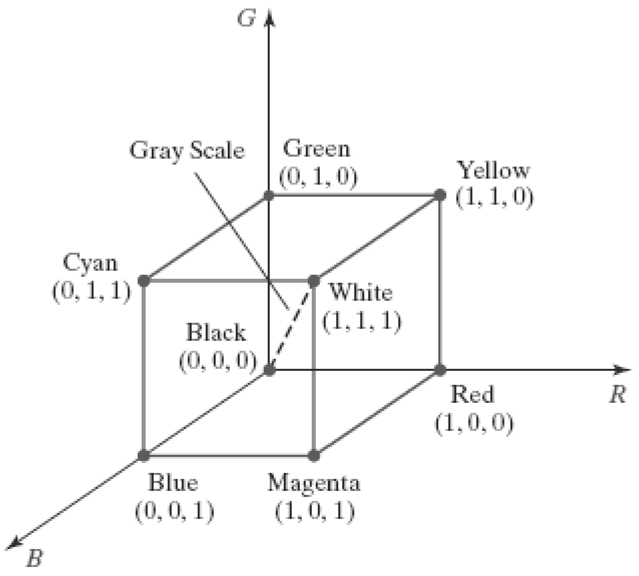
The RGB Color Model

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

One of the pigments is most sensitive to light with a wavelength of about 630 nm (red), another has its peak sensitivity at about 530 nm (green), and the third pigment is most receptive to light with a wavelength of about 450 nm (blue).

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*, and *B* axes, as shown in Figure



The origin represents black and the diagonally opposite vertex, with coordinates (1, 1, 1), is white the RGB color scheme is an additive model.

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

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

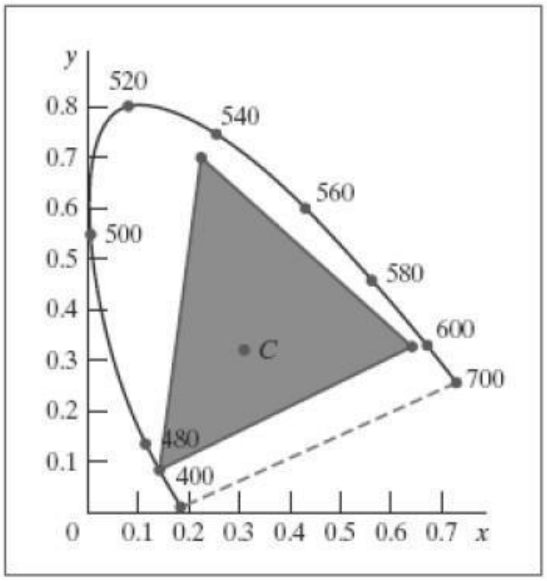
where parameters R, G, and B are assigned values in the range from 0 to 1.0

Chromaticity coordinates for the National Television System Committee (NTSC) standard RGB phosphors are listed in Table

RGB (x, y) Chromaticity Coordinates

	NTSC Standard	CIE Model	Approx. Color Monitor Values
R	(0.670, 0.330)	(0.735, 0.265)	(0.628, 0.346)
G	(0.210, 0.710)	(0.274, 0.717)	(0.268, 0.588)
B	(0.140, 0.080)	(0.167, 0.009)	(0.150, 0.070)

Below figure shows the approximate color gamut for the NTSC standard RGB primaries

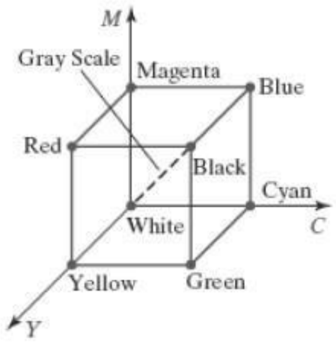


The CMY and CMYK Color Models

The CMY Parameters

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

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

Similarly, a combination of cyan and yellow ink produces green light, and a combination of magenta and yellow ink yields red light.

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

Q.Explain the transformation between CMY and RGB color space

We can express the conversion from an RGB representation to a CMY representation using the following matrix transformation:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Where the white point in RGB space is represented as the unit column vector.

And we convert from a CMY color representation to an RGB representation using the matrix transformation

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$