33. Demonstrate Reflection of an object w.r.t the straight line y=x → 79
34. Explain the reflection and shearing. → 80
35. Explain with example ,vector method for splitting a polygon → 84
36. Describe OpenGL polygon fill area function with example → 87
37. Write a note on
    a. fill style b. color blended fill region → 89
38. Write a OpenGL program to rotate a triangle using composite matrix calculation. → 91
39. What are homogeneous coordinates? Write the matrix representation for translation, rotation and scaling. → 92
40. What is raster operation? Explain the raster methods for geometric transformation. → 94
41. Write a note on
    a. OpenGL fillpattern function.
    b. OpenGL texture and interpolation pattern.
    c. OpenGL wire frame methods.
    d. OpenGL front face function. → 96
42. Explain the composite 2D translation, Rotation and scaling. → 98
43. Explain the 2D OpenGL geometric transformations. → 100
44. Write the steps for rotation about pivot point and scaling about fixed point. → 102
45. Briefly explain Inverse transformation, composite transformation. → 104
46. Explain the OpenGL matrix operations and Matrix stacks. → 108
47. Explain the OpenGL 2D viewing functions. → 109
48. Translate a square with the following coordinate by 2units in both directions → 111
    A(0,0),B(2,0),C(2,2),D(0,2)
49. Rotate a triangle at A(0,0),B(6,0),C(3,3) by 90degrre about origin and fixed point (3,3) both → 112
    Anticlockwise and clockwise direction.
50. What are the polygon classifications? How to identify a convex polygon? Illustrate how to split a Concave polygon. → 115
51. What is stitching effect? How does OpenGL deals with it. → 117

MODULE 3

52. Imagine a 3 D cube object with rotation axis projected onto the Z-axis defined by the vector u. Rotate it and find the final rotation matrix R. Show all the 5 steps involved in it with 7 series of operations. → 119
53. Demonstrate the 3D Translation and Reflection with Homogenous coordinates. → 122
54. Demonstrate the 3D Scaling and Shearing with Homogenous coordinates. → 124
55. Explain the ambient light, diffuse reflection and specular reflection with equations. → 126
56. Explain OpenGL 3D Viewing Functions. → 129
57. Imagine you have a 3D object in front of you. Illustrate how to Normalize the transformation for an Orthogonal Projection? → 131
58. What is clipping and clipping window. → 133
59. Map the clipping window into a normalized viewport. → 138
60. Explain specular refection. → 142
61. Explain the 3D coordinate axis-Rotation → 143
62. Map the clipping window into a Normalized square. → 145
63. Explain the Cohen-Sutherland line-clipping algorithm. → 149
64. With neat diagram, illustrate Sutherland-Hodgeman polygon clipping algorithm. → 152
65. What is quaternion? Explain the quaternion methods for 3D rotations. → 156

66. What is affine transformation? → 159
67. List the 3D OpenGL geometric transformations. → 168
68. What is color model? Explain the RGB color model. → 171
69. Explain the CMY and CMYK color models. → 173
70. What is light source? Explain the types of light source. → 175
71. Explain the PHONG model. → 182

MODULE 4

72. What is projection plane, parallel and perspective projection? → 184
73. What is depth cueing? → 188
74. Explain the 3D viewing pipeline with diagram. → 189
75. Explain the transformation from world to viewing coordinates. → 190
76. Explain the orthogonal projections. → 192
77. Explain the perspective projection transformation coordinates. → 194
78. Explain the OpenGL 3D viewing functions. → 197
79. Classify the visible surface detection algorithms. → 200
80. Explain the back-face detection algorithm. → 201
81. Explain the z-buffer/depth-buffer algorithm. → 203
82. Explain the OpenGL visibility detection functions. → 206
83. Explain in detail, Oblique and Symmetric perspective projection frustum. → 209
84. Explain vanishing points for perspective projections. → 212
85. Explain briefly the following: → 214
    a) Projections
    b) Depth Cueing
    c) Identifying visible lines and surfaces
    d) Surface rendering
    e) Exploded and cutaway views
    f) 3D and stereoscopic viewing
86. Explain viewup vector and uvn viewing coordinate reference frame → 220
87. Write short notes on axonometric and isometric orthogonal projections → 222.
88. Explain OpenGL functions with respect to: → 223
    a. Viewing Transformation functions
    b. Orthogonal Projection functions
    c. Symmetric Perspective Projection functions
    d. General Perspective Projection functions
    e. Viewport and Display Window
89. Imagine you have a 3D object in front of you. Illustrate how to Normalize the transformation for an Orthogonal Projection? → 227

MODULE 5

90. Explain how an event driven input can be performed for → 229
    (a) window events (b) pointing devices
91. Explain how an event driven input can be programmed for a keyboard device. → 232
92. List out any four characteristics of good interactive program. → 234
93. What are the major characteristics that describe the logical behavior of an input device? → 236
94. Explain how OpenGL provides the functionality of each of the classes of logical input devices. → 238

Shankar R
Asst Professor,
CSE, BMSIT&M

52 Imagine a 3D cube object with rotation axis that projected onto z-axis defined by vector u. Rotate it and find the final rotation matrix 'R'. Show all 5 steps involved in it with 7 series of operation.

A When an object is to be rotated about an axis that is not parallel to one of co-ordinate axis, we need to accomplish the required rotation in five steps given that is it projected onto z-axis. (defined by vector u).

Let us assume that rotation axis is defined by two points P_1 and P_2. The components of rotation-axis vector,

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

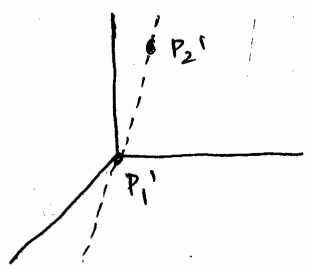
unit rotation-axis vector u is, $u = \frac{V}{|V|} = (a, b, c)$.

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

Step ①: Translate the object so that rotation axis passes through co-ordinate origin.

Considering counter-clockwise rotation viewing along the axis from P_2 to P_1, the translation matrix is given as

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



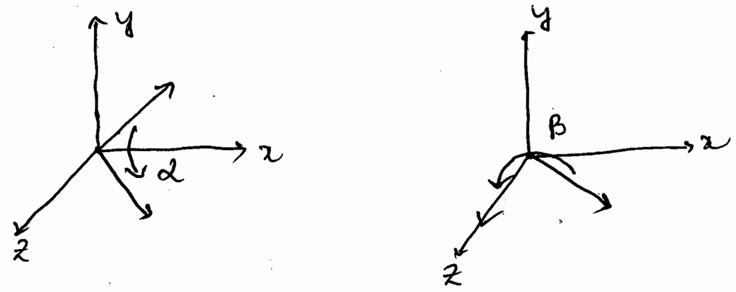
Shankar R
Asst Professor,
CSE, BMSIT&M

Step ②:- Rotate the object so that axis of rotation coincides with one of the co-ordinate axis.

It can be done in 2 steps.

(a) → rotate about x-axis get the vector 'u' in xz plane

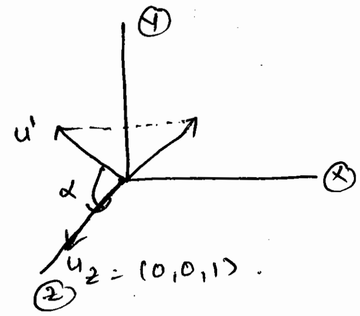
(b) → rotate about y-axis get it to coincide with z-axis.



⇒ rotation angle 'α' can be determined by dot product of 'u'.

$$\cos\alpha = \frac{u'.u_z}{|u'||u_z|} = \frac{c}{d}$$

FIGURE (a)



$$d = \sqrt{b^2+c^2}$$

apply cross product

$$u' \times u_z = u_x |u'||u_z| \sin\alpha \quad —①$$

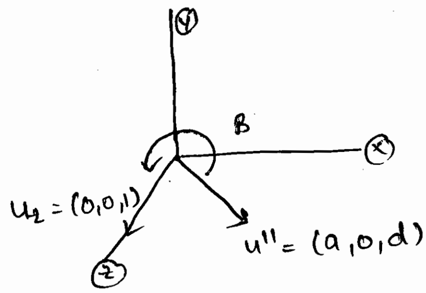
$$u' \times u_z = u_x . b \quad —②$$

equate ① & ②

$$d \sin\alpha = b$$

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

FIGURE (b)



So,

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

Shankar R
Asst Professor,
CSE, BMSIT&M

Now, to rotate about y-axis, to coincide about as in figure (b).

Consider,

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

here, $|u_z| = |u''| = 1$

$$u'' \times u_z = u_y |u''| |u_z| \sin\beta \quad -③$$

$$u'' \times u_z = u_y \cdot (-a) \quad -④$$

equating ③ & ④

$$\sin\beta = -a$$

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

Step ③ :- Perform the specified rotation about the selected co-ordinate axis.

The rotation angle (θ) can now be applied as

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

Step ④ :- Apply inverse rotations to bring the rotation axis back to its original orientation. Apply $R_x^{-1}(\alpha)$ & $R_y^{-1}(\beta)$.

Step ⑤ :- Apply inverse translations and the rotation matrix for 3D rotation can be written as

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

Shankar R
Asst Professor,
CSE, BMSIT&M

(53) Demonstrate the 3D Translation and Reflection with Homogeneous Co-ordinates.

(A) A position $P = (x, y, z)$ in three-dimensional space is translated to a location $P'(x', y', z')$ by adding translation distances t_x, t_y, t_z to the cartesian co-ordinates of P.

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

We can express in 4x4 Matrix form as:

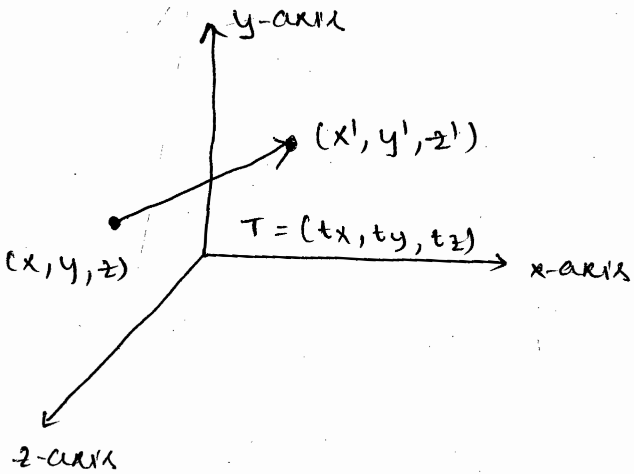
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

(or)

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

An object is translated in 3 dimensions by transforming each of the defining co-ordinate positions for the object, then reconstructing the object at new location.

Consider, moving a co-ordinate position with translation vector $T = (t_x, t_y, t_z)$

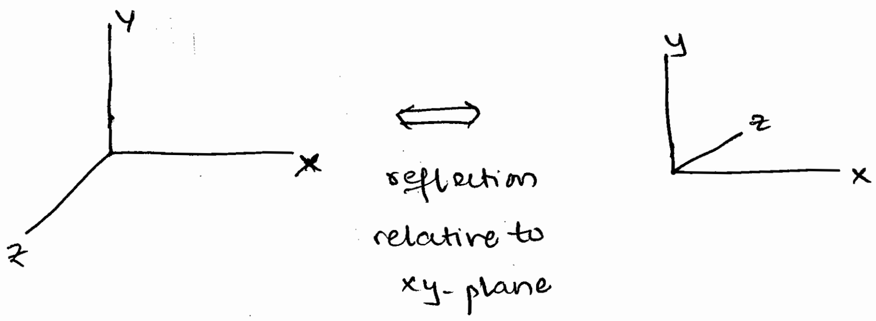


Shankar R
Asst Professor,
CSE, BMSIT&M

3-D REFLECTION:-

It can be performed relative to a selected reflection axis or with respect to a reflection plane. Reflections given ~~the~~ to a given axis equivalent to 180° rotations about that axis. When a reflection plane is a co-ordinate plane, we can think of transformation as a conversion between left handed frame and right handed frame.

An example of reflection that converts co-ordinate specifications from right-handed system to left-handed system as shown below



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

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

Transformation matrices for inverting x-coordinates or y co-ordinates are defined similarly, as reflections relative to the yz-plane or the xz-plane respectively.

Shankar R
Asst Professor,
CSE, BMSIT&M

54.) Demonstrate the 3D Scaling and Shearing with Homogenous Coordinates.

⇒ Scaling:

The matrix expression for the 3-D scaling transformation of a position $P = (x, y, z)$ relative to the co-ordinate origin is a simple extension of 2-D scaling. We just include the parameter of z-co-ordinate scaling in transformation matrix:

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

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

$$P' = S \cdot P$$

where Scaling parameters S_x, S_y, S_z are assigned any positive values.

Explicit expressions for the scaling transformation relative to origin are

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

We can construct a scaling transformation with respect to any selected fixed position (x_f, y_f, z_f) using the following sequence:

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

The matrix representation for an arbitrary fixed-point scaling can be expressed as the concatenation of these translate-scale-translate transformations:

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

Shearing:

These transformations can be used to modify object shapes, just as in two-dimensional applications. They are also applied in 3-D viewing transformations for perspective projections. For 3-D, 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:

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

Shearing parameters sh_{zx} and sh_{zy} can be assigned any real values. The effect of this transformation matrix is to alter the values for the x and y co-ordinates by an amount that is proportional to the distance from z_{ref}, while leaving the z-co-ordinate unchanged.

Shankar R
Asst Professor,
CSE, BMSIT&M

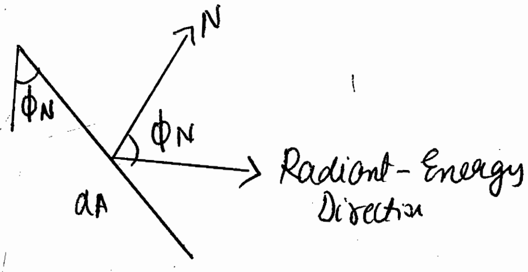
55.) Explain the ambient light, diffuse reflection and specular reflection with equations.

⇒

Ambient light:-

In 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 only monochromatic lighting effects, such as shades of grey, we designate the level for the ambient light in a scene with an intensity parameter I_a. Each surface in the scene is then illuminated with this background light. Reflections produced by ambient-light illuminations are simply a form of diffuse reflection, and they are independent of the viewing direction and spatial orientation of a surface. However, the amount of the incident ambient light that is reflected depends on surface optical properties, which determine how much of the incident energy is reflected and how much is absorbed.



Radiant energy from a surface area element dA in direction ϕ_N relative to the surface normal direction is proportional to $\cos\phi_N$.

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 referred to as Lambertian reflectors, because the reflected radiant light energy from any point on the surface is calculated with Lambert's cosine law, which 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 $\cos\phi_N$.

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

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

$$= \text{constant}$$

The ambient contribution to the diffuse reflection at any point on a surface is

$$I_{ambdiff} = K_d I_a.$$

The amount of incident light on a surface from a source with intensity I_l is

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

Diffuse reflections from a light source with intensity I_l

$$I_{l,diff} = K_d I_{l,incident}$$
$$= K_d I_l \cos\theta.$$

At any surface position, we can denote the unit normal vector as N and the unit direction vector to a point source as L, as in fig. Then, $\cos\theta = N \cdot L$ and the diffuse reflection equation for single point-source illumination at a surface position can be expressed in the form.

$$I_{l,diff} = \begin{cases} K_d I_l (N \cdot L), & \text{if } N \cdot L > 0 \\ 0.0 & \text{if } N \cdot L \leq 0 \end{cases}$$

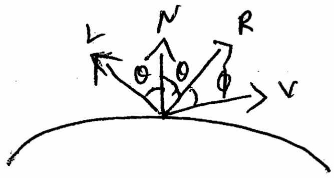
Using parameter K_a, we can write the total diffuse-reflection equation for a single point source as.

$$I_{diff} = \begin{cases} K_a I_a + K_d I_l (N \cdot L), & \text{if } N \cdot L > 0 \\ K_a I_a, & \text{if } N \cdot L \leq 0 \end{cases} \quad \text{where } L = \frac{P_{source} - P_{surf}}{|P_{source} - P_{surf}|}$$

Shankar R
Asst Professor,
CSE, BMSIT&M

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. Figure shows the specular reflection direction for a position on an illuminated surface. 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. In the figure, R represents the unit vector in the direction of ideal specular reflection, L is the unit vector directed toward the point light source, V is the unit vector pointing to the viewer from the selected surface position. Angle ϕ is the viewing angle relative to the specular-reflection direction R.



The intensity of the specular reflection due to a point light source at a surface position with the calculation is:

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

Shankar R
Asst Professor,
CSE, BMSIT&M

5b) Explain OpenGL 3D Viewing Functions

When we designate the viewing parameters in OpenGL, a matrix is formed & concatenated with the current model view matrix. Consequently, this viewing matrix is combined with any geometric transformations we may have also specified. This composite matrix is then applied to transform object descriptions in world coordinates to viewing coordinates.

glMatrixMode (GL_MODELVIEW);

Viewing parameters are specified with the following GLU function, which is in the OpenGL utility library because it invokes the translation & rotation routines in the basic OpenGL library.

gluLookAt ($x_0, y_0, z_0, x_{ref}, y_{ref}, z_{ref}, V_x, V_y, V_z$);

Values for all parameters in this function to be assigned double-precision, floating-point values. This function designates the origin of the viewing reference frame as the world-coordinate position $P_0 = (x_0, y_0, z_0)$, the reference position as $P_{ref} = (x_{ref}, y_{ref}, z_{ref})$, & the view up vector as $V = (V_x, V_y, V_z)$.

The positive z_{view} axis for the viewing frame is in the direction $N = P_0 - P_{ref}$, and the unit axis vectors for the viewing reference frame are calculated with Equations 1.

Shankar R
Asst Professor,
CSE, BMSIT&M

Because the viewing direction is along the $-Z_{view}$ axis, this reference position P_{ref} is referred to as "look at point". This usually taken to be some position in the center of the scene. And we can think of the reference position as the point at which we want to aim a camera that is located at the viewing origin. The up orientation for the camera is designated with vector V, which is adjusted to a direction perpendicular to N.

If we do not invoke gluLookAt function, the default OpenGL viewing parameters are

$$P_0 = (0, 0, 0)$$
$$P_{ref} = (0, 0, -1)$$
$$V = (0, 1, 0)$$

For these default values, the viewing reference frame is the same as the world frame, with the viewing direction along the -ve Z_{world} axis.

In many applications, we can conveniently use the default values for the viewing parameters.

Shankar R
Asst Professor,
CSE, BMSIT&M

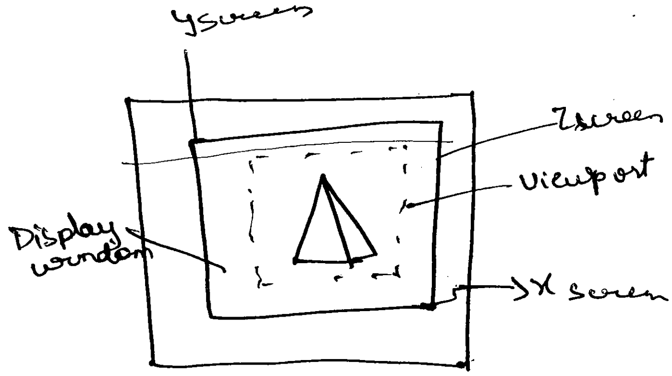
57) Imagine you have a 3D object in front of you. Illustrate how to Normalize the transformation for an orthogonal projection?

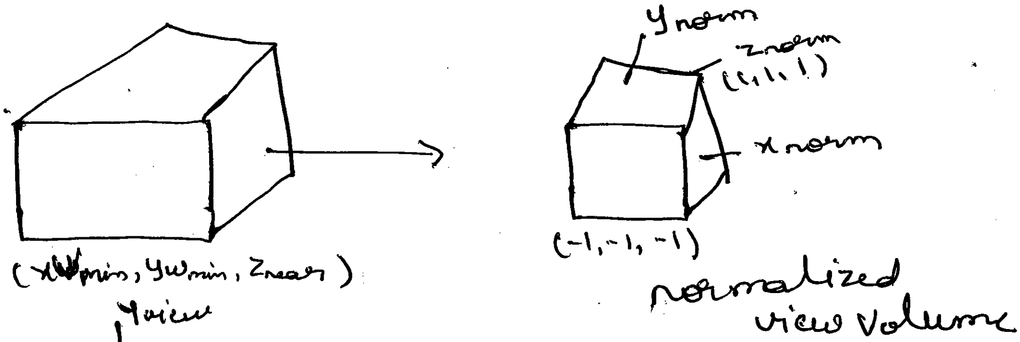
Using an orthogonal transfer of coordinate positions onto the view plane, we obtain the projected position of any spatial point (x, y, z) as simply (x, y). Thus, once we have established the limits for the view volume, coordinate descriptions inside this rectangular parallelepiped are the projection coordinates, & they can be mapped into a normalized view volume without any further projection processing. Some graphics packages use a unit cube for this normalized view volume, with each of the x, y & z coordinates normalized in the range from 0 to 1. Another normalization - transformation approach is to use a symmetric cube, with coordinates in the range from -1 to 1.

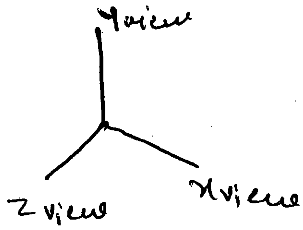
Because screen coordinates are often specified in a left handed reference frame, normalized coordinates also are often specified in a left handed system. This allows positive distances in the viewing direction to be directly interpreted as distances from the screen. Thus, we can convert projection coordinates into positions will then be transformed to left-handed screen coordinates.

To illustrate the normalisation transformation, we assume that the orthogonal-projection view volume is to be mapped into the symmetric

Shankar R
Asst Professor,
CSE, BMSIT&M







Normalisation cube within left handed Reference frame. Transforming the rectangular-parallelepiped view volume to a normalized cube is similar to converting the clipping window into normalised symmetric square. The normalisation transformation for the orthogonal view volume is

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

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

Shankar R
Asst Professor,
CSE, BMSIT&M

(58) What is clipping and clipping window.

Clipping:-

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

The region against which an object is clipped is called a clip window.

Types of clipping:-

1. Point clipping
2. Line clipping
3. Polygon clipping
4. Curve clipping
5. Text clipping

Point Clipping:-

clipping Individual points:-

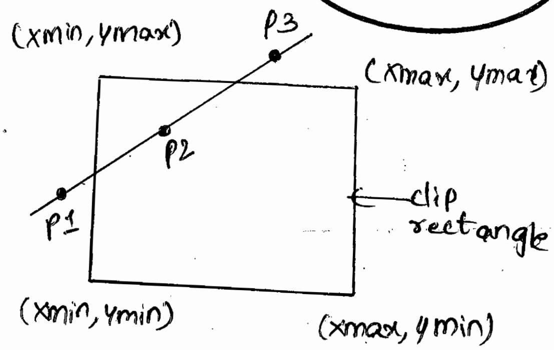
The x coordinate boundaries of the clipping rectangle are X_{min} and X_{max}, and the y coordinate boundaries are y_{min} and y_{max}, then the following inequalities must be statisfied for a point at (x, y) to be inside the clipping rectangle:

$$X_{min} < x < X_{max} \quad \text{and} \quad y_{min} < y < y_{max}$$

Shankar R
Asst Professor,
CSE, BMSIT&M

If any of the four inequalities does not hold, the point is outside the clipping rectangle.

- P1 point clipped away
- P2 point is visible
- P3 point clipped away.



Line clipping:-

Cohen-Suderland Algorithm:-

Algorithm:-

1. given a line segment with endpoint $P1 = (x1, y1)$ and $P2 = (x2, y2)$
2. compute the 4-bit codes for each endpoint.

* If both codes are 0000, line lies completely inside the window: pass the endpoints to the draw routine.

* If both codes have a 1 in the same bit position, the line lies outside the window. It can be trivially rejected.

3. If a line cannot be trivially accepted or rejected, at least one of the two end points must lie outside the window and the line segment crosses a window edge. This line must be clipped at the window edge before being passed to the drawing routine.
4. Examine one of the endpoints, say $P1 = (x1, y1)$

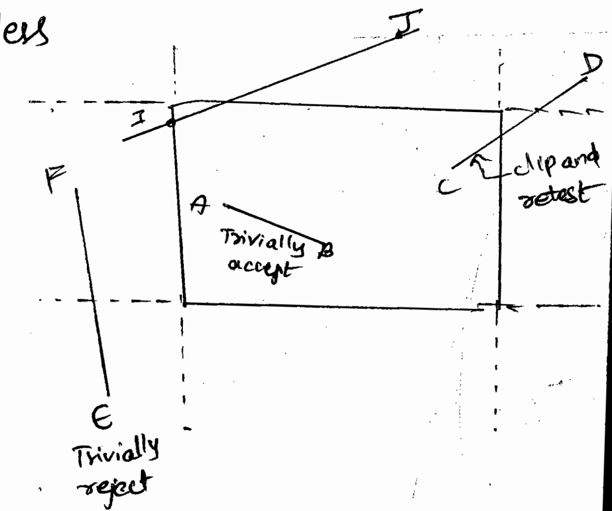
   Read P1's 4-bit code in order: Left to right, Bottom to Top.
5. When a set bit (1) is found, compute the intersection I of the corresponding window edge with the line from P1 to P2.

Replace P1 with I and repeat the algorithm.

Basic algorithm:-

→ Accept lines that have both endpoints inside the region.

→ Reject lines that have both endpoints less than x_{min} or y_{min} or greater than x_{max} or y_{max}.



→ clip the remaining lines at a region boundary and repeat the previous steps on the clipped line segments.

→ Assign a 4-bit code to each endpoint c_0, c_1 based on its position:

1001	1000	1010
0001	0000	0010
0101	0100	0110

- 1^{st} bit (1000) : if $y > y_{max}$
- 2^{nd} bit (0100) : if $y < y_{min}$
- 3^{rd} bit (0010) : if $x > x_{max}$
- 4^{th} bit (0001) : if $x < x_{min}$

→ Test using bitwise functions

if $c_0 \mid c_1 = 0000$
  accept (draw)

else if $c_0 \& c_1 \neq 0000$
  reject (don't draw)

else
  clip and retest

Shankar R
Asst Professor,
CSE, BMSIT&M

Intersection algorithm:

if $c_0 \neq 0000$ then $c = c_0$

else $c = c1$;

$dx = x_1 - x_0$; $dy = y_1 - y_0$

if $c \,\&\, 1000$

$x = x_0 + dx * (y_{max} - y_0) / dy$; $y = y_{max}$

else if $c \,\&\, 0100$

$x = x_0 + dx * (y_{min} - y_0) / dy$; $y = y_{min}$

else if $c \,\&\, 0010$

$y = y_0 + dy * (x_{max} - x_0) / dx$; $x = x_{max}$

else

$y = y_0 + dy * (x_{min} - x_0) / dx$; $x = x_{min}$

if $c = c_0$

$x_0 = x$, $y_0 = y$;

else

$x_1 = x$; $y_1 = y$;

Polygon clipping:

Basic idea:

consider each edge of the viewport individually. clip the polygon against the edge equation. After doing all planes, the polygon is fully clipped.

There are four clips. They are:

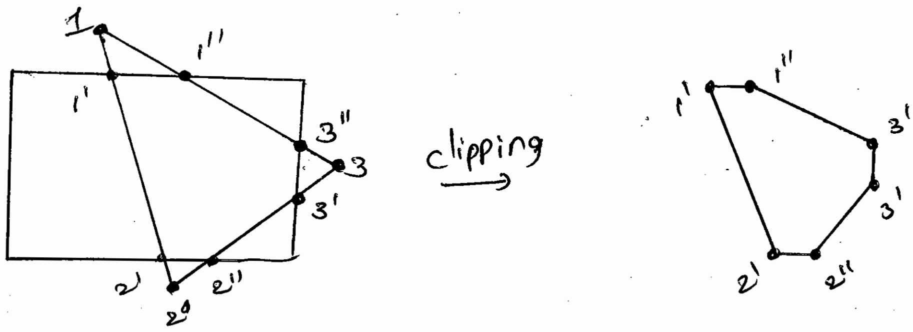
(1) Top clip

(2) Bottom clip

(3) Right clip

Shankar R
Asst Professor,
CSE, BMSIT&M

(4) Left clip



Pseudo code:-

case 1 : wholly inside visible region - save end point.

case 2 : Exit visible region - save the intersection.

case 3 : wholly outside visible region - save nothing.

case 4 : Enter visible region - save intersection and endpoint.

Shankar R
Asst Professor,
CSE, BMSIT&M

59. Map the clipping window into a Normalized viewport.

→ * First a viewport is defined with normalized co-ordinates values between 0 and 1

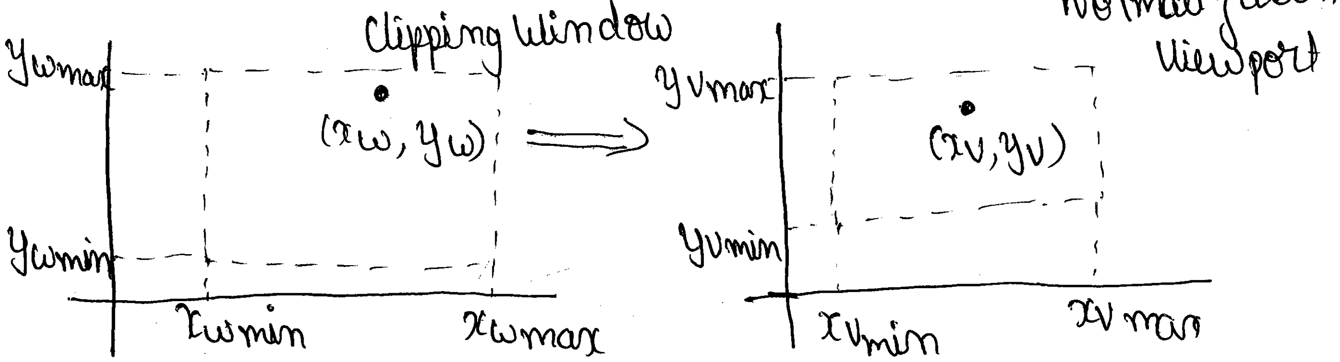
* Same relative placement of a point is maintained in the viewport as it had in clipping window i.e the relative position will not change.

* i.e based on the position we need to find to find relative viewport co-ordinates based on window co-ordinates.

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

$$\frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}} = \frac{x_w - x_{wmin}}{x_{wmax} - x_{wmin}}$$

$$\frac{y_v - y_{vmin}}{y_{vmax} - y_{vmin}} = \frac{y_w - y_{wmin}}{y_{wmax} - y_{wmin}}$$



Scaling parameter 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 co-ordinate 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

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

$$\left[x_v = x_{vmin} + (x_w - x_{wmin}) S_x\right]$$

$|||^{le}$

$$\left[y_v = y_{vmin} + (y_w - y_{wmin}) S_y\right]$$

$$x_v = x_{vmin} + S_x x_w - S_x x_{wmin}$$

$$= S_x x_w - S_x x_{wmin} + x_{vmin}$$ substitute S_x

$$= S_x x_w - \frac{x_{vmax} x_{wmin} + x_{vmin} x_{wmin}}{x_{wmax} - x_{wmin}} + x_{vmin}$$

multiply & take common

$$= S_x . x_w - \frac{x_{vmax} x_{wmin} + \cancel{x_{vmin} x_{wmin}} + x_{vmin} x_{wmax} - \cancel{x_{vmin} x_{wmin}}}{x_{wmax} - x_{wmin}}$$

$$= S_x . x_w + \frac{x_{wmax} x_{vmin} - x_{vmax} x_{wmin}}{x_{wmax} - x_{wmin}}$$

Shankar R
Asst Professor,
CSE, BMSIT&M

$$[X_v = S_x x_w + t_x]$$

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

Also. $$Y_v = Y_{vmin} + (Y_w - Y_{wmin}) S_y$$

$$Y_v = Y_{vmin} + S_y \cdot Y_w - S_y \cdot Y_{wmin}$$

$$= S_y \cdot Y_w - \left[\frac{Y_{vmax} - Y_{vmin}}{Y_{wmax} - Y_{wmin}}\right] \cdot Y_{wmin} + Y_{vmin}$$

$$= S_y \cdot Y_w - Y_{wmin} \frac{Y_{vmax} + Y_{vmin}}{Y_{wmax} - Y_{wmin}} Y_{wmin} + Y_{vmin}$$

$$= S_y \cdot Y_w - \frac{Y_{wmin} Y_{vmax} + Y_{vmin} Y_{wmin} + Y_{vmin} Y_{wmax} - Y_{vmin} \cdot Y_{wmin}}{Y_{wmax} - Y_{wmin}}$$

$$= S_y Y_w + \frac{Y_{wmax} \cdot Y_{vmin} - Y_{vmax} Y_{wmin}}{Y_{wmax} - Y_{wmin}}$$

$$[Y_v = S_y \cdot Y_w + t_y]$$

where $$t_y = \frac{Y_{wmax} \cdot Y_{vmin} - Y_{vmax} \cdot Y_{wmin}}{Y_{wmax} - Y_{wmin}}$$

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 transportation in ① can be represented

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

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

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

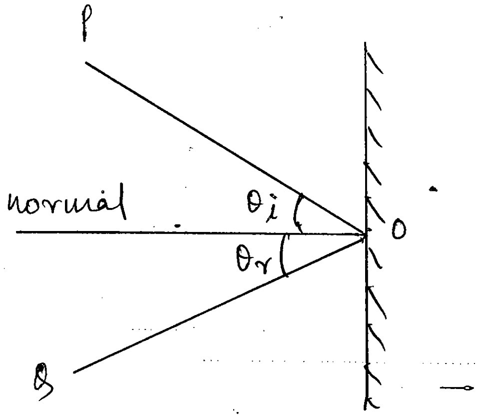
https://shankarrajagopal.github.io/

Shankar R
Asst Professor,
CSE, BMSIT&M

60

Que. Explain specular reflection.

Sol^n:



Specular reflection, also known as regular reflection, is the mirror-like reflection of waves, such as light, from a surface. In this process, each incident ray is reflected at the same angle to the surface normal as the incident ray, but on the opposing side of the surface normal as the incident ray. The result is that an image reflected by the surface is reproduced in mirror-like fashion.

Reflection off of smooth surfaces such as mirrors or a calm body of water leads to a type of reflection known as specular reflection.

One application pertains to the relative difficulty of night driving on a wet asphalt roadway compared to a dry asphalt.

Shankar R
Asst Professor,
CSE, BMSIT&M

61

Que. Explain the 3-D coordinate axis-rotation.

Soln. The 2-D z-axis rotation equations are easily extended to three dimensions.

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

Parameter θ specifies the rotation angle about the z-axis and z-coordinate values are unchanged by this transformation.

The homogeneous coordinate form is

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

which we can write as

$$P' = R_z(\theta) \cdot P$$

Transformation eq^ns for rotations about the other two coordinate axes can be obtained with a cyclic permutation of parameters x, y, z.

$$x \to y \to z \to x$$

Shankar R
Asst Professor,
CSE, BMSIT&M

Substituting permutations, we get eq^ns for x-axis rotation.

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

Rotation of an object around x-axis can be obtained similarly.

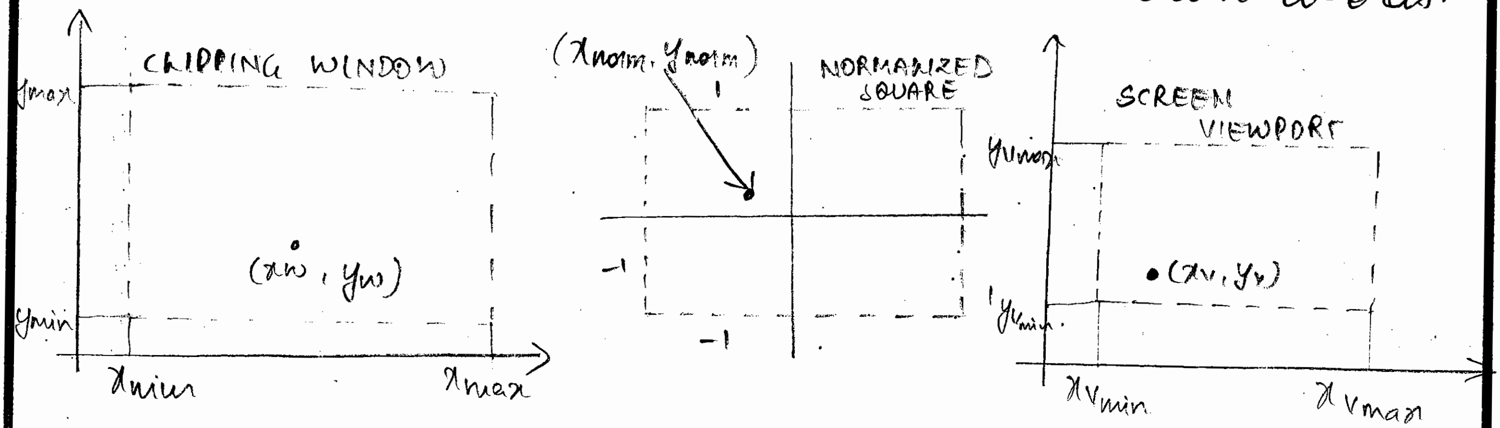
A cyclic permutation of coordinates gives us the transformation equations for y-axis rotation.

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

Shankar R
Asst Professor,
CSE, BMSIT&M

62) Map the clipping window into a normalized square.

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



Normalized cordinates are in the range from -1 to 1 clipping window algorithms are standardized such that objects outside the boundaries $x = \pm 1$ & $y = \pm 1$ are detected and 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 coordinates position (x_{norm}, y_{norm}), then to a screen coordinate position (x_v, y_v) in a view port.

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

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

we also know

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

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

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

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

substitute s_x, & s_y, t_x & t_y in composite matrix

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

Shankar R
Asst Professor,
CSE, BMSIT&M

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} and y_{wmax}

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

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

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

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

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

substitute s_x, s_y, t_x & t_y in composite matrix

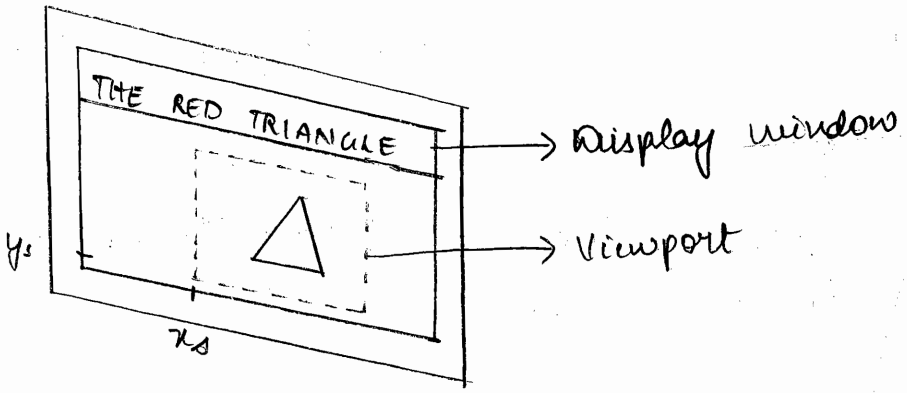
$$M_{normsquare,\, viewport} = \begin{bmatrix} \frac{x_{vmax} - x_{vmin}}{2} & 0 & \frac{x_{vmin} + x_{vmax}}{2} \\ 0 & \frac{y_{vmax} - y_{vmin}}{2} & \frac{y_{vmin} + y_{vmax}}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

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

https://shankarrajagopal.github.io/

be same as the clipping window, if not objects may be stretched or contracted in the x or y direction.

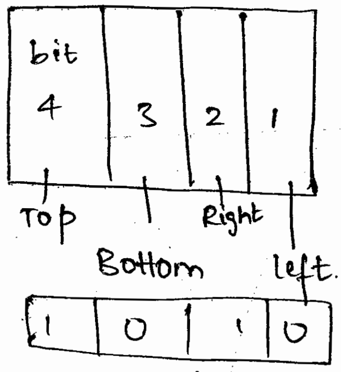


Viewport at coordinate position (x_s, y_s) within a display window.

Shankar R
Asst Professor,
CSE, BMSIT&M

(63) Explain the Cohen-Sutherland line-clipping alg.

Every line endpoint in a picture is assigned a four-digit binary value, called a region code, and each bit position is used to indicate whether the point is inside or outside of one of the clipping window boundaries. There are 9 regions:

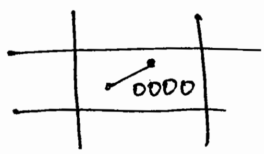


→ Top-right
Similarly, the rest of the 8 regions.

1001	1000	1010
0001	0000 (clipping window)	0010
0101	0100	0110

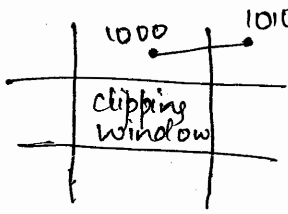
Once we have established region codes for all line endpoints, we can quickly determine which lines are completely contained within clip window & which are clearly outside.

When the OR operation between 2 endpoints region codes for a line segment is false (0000), the line is inside the clipping window. example:



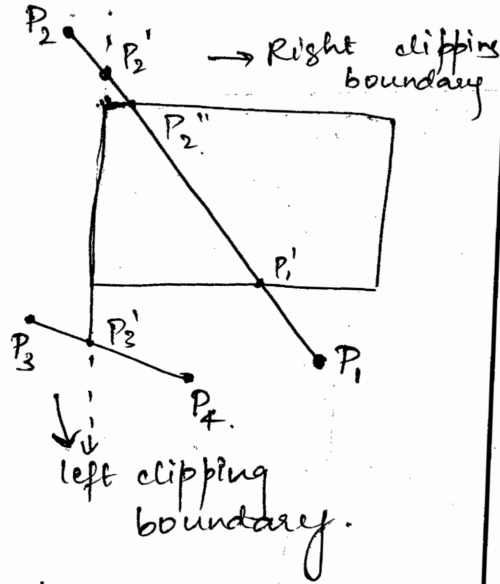
(0000)||(0000)
⇓
(0000)
false.

When the AND operation between 2 endpoints region codes for a line is true (not 0000), the line is completely outside the clipping window & can be eliminated. Example:



(1000) & (1010)
⇓
1000
(true).

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



The region codes for P_1 & P_2 are 0100 and 1001. Thus, P_1 is inside the left clipping boundary & P_2 is outside.

Therefore, we calculate the intersection P_2' and clip off the section P_2 to P_2'. The remaining portion of the line is inside the right border line, so we next check the ~~next~~ bottom border. P_1 is below the bottom clipping edge an P_2' is above it, so we find the intersection at this boundary, P_1'. Therefore P_1' to P_1 is clipped-off. Proceed to top-edge window. we determine the intersection to be P_2'' & P_2' to P_2'' is clipped off.

For line P_3 to P_4, we find that point P_3 is outside the left boundary & P_4 is inside. Therefore, the intersection is P_3' & P_3 to P_3' is clipped off.

By checking region codes of P_3' & P_4, we find the remainder of the line is below the clipping window & can be eliminated. To determine a boundary intersection for a line segment, we can use the slope-intercept form of line equation. For a line with end point coordinates (x_0, y_0) & (x_{end}, y_{end}),

Shankar R
Asst Professor,
CSE, BMSIT&M

The y co-ordinates of the intersection point with vertical clipping border line can be obtained by

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

where x is either xw_{min} (or) xw_{max}, and slope is

$$m = (y_{end} - y_0) / (x_{end} - x_0).$$

∴ for intersection with horizontal border, the x co-ordinate is

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

Shankar R
Asst Professor,
CSE, BMSIT&M

64) With a neat diagram, illustrate Sutherland - Hodgeman polygon clipping algorithm

Ans An efficient method for clipping a convex polygon fill area, developed by Sutherland-Hodgeman, is to send the polygon vertices through each clipping stage so that a single clipped vertex can be immediately passed to the next stage. This eliminates the need for an output set of vertices at each clipping stage, and it allows the boundary-clipping routines to be implemented in parallel. The final output is a list of vertices that describe the edges of the clipped polygon fill area.

Because the Sutherland Hodgman algorithm produces only one list of output vertices, it cannot correctly generate the two output polygons in fig(1b) that is the result of clipping the concave polygon shown in fig(1a). However more processing steps can be added to the algorithm to allow it to produce multiple output vertex lists, so that general concave polygon clipping can be accomodated.

The general strategy in this algorithm is to send the pair of end points for each successive polygon line segment through the series of clippers (left, right, bottom, top). There are four possible cases that need to be considered when processing a polygon edge against one of the clipping boundries. One possibility is that both the clipping points are inside the clipping boundary, other possiblities could have both end points outside the clipping window or one end point is inside and other end point is outside the clipping window.

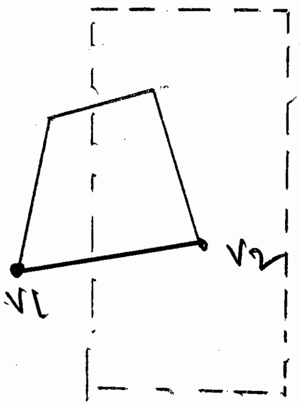
As each successive pair of end points is passed to one of the four clippers, an output is generated for the next clipper according to the results of the following tests:

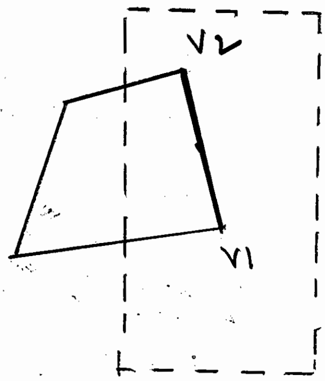
→ If the first input vertex is outside and the second vertex is inside the clipping window, both the the intersection point and the second vertex is are sent to the next clipper.

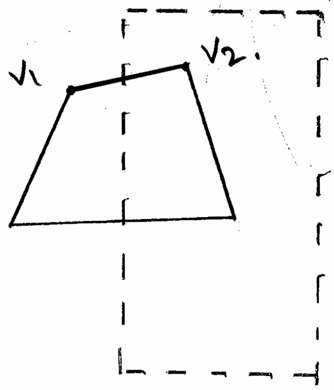
→ If both input vertices are inside this clipping-window border, only the second vertex is sent to the next clipper.

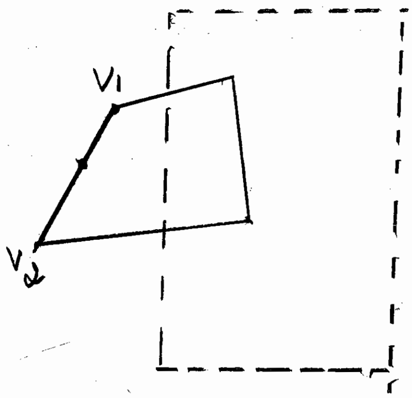
3) If the first vertex is inside the clipping window border and the second vertex is outside, only the polygon edge intersection point with the clipping window is sent to the next clipper.

4) If both input vertices are outside this clipping window, no vertices are sent to the next clipper.

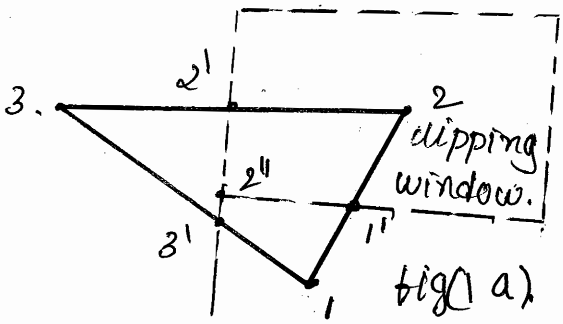






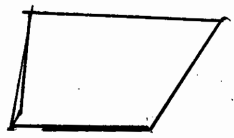


Shankar R
Asst Professor,
CSE, BMSIT&M



fig(a)

After clipping.



fig(b)

Left clipper → Right clipper → Bottom clipper → Top clipper.

Left clipper:

[1, 2] (in-in) → 2
[2, 3] (in-out) → 2
[3, 1] (out-in) → 3', 1

Right clipper:

[2, 2'] : (in-in) - 2'
[2', 3'] : (in-in) - 3'
[3', 1] : (in-in) - 1
[1, 2] : (in-in) - 2

Bottom clipper:

[2', 3'] : (in-out) - 2''
[3', 1] : (out-out)
[1, 2] : (out-in) - 1', 2
[2, 2'] : (in-in) → 2'

Top clipper:

[2'', 1'] : (in-in) → 1'
[1', 2] : (in-in) → 2
[2, 2'] : (in-in) → 2'
[2', 2''] : (in-in) - 2'

Shankar R
Asst Professor,
CSE, BMSIT&M

65) What is Quaternion? Explain the Quaternion methods for 3D Rotation?

→ A more efficient method for generating a rotation about an arbitrarily selected axis is to use a quaternion representation for the rotation transformation.

→ Quaternions, which are extensions of 2D complex numbers, are useful in a number of computer-graphics procedures, including the generation of fractal objects.

→ One way to characterise a Quaternion is an ordered pair, consisting of a scalar part and a vector part:

$$q = (s, v)$$

→ A rotation about any axis passing through the coordinate origin is accomplished by first setting up a unit Quaternion with the scalar & vector parts as follows.

Shankar R
Asst Professor,
CSE, BMSIT&M

Any point position P that is to be rotated by this Quaternion can be represented in Quaternion rotation as.

$$P = (0, p)$$

→ Rotation of the point is then carried out with the Quaternion operation

$$P' = qPq^{-1}$$

where $q^{-1} = (s, -v)$ inverse of q

→ This transformation produces the foll new Quaternion.

$$P' = (0, p')$$

→ The 2nd term in this ordered pair is in rotated point position p', which is evaluated with vector dot and cross product as

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

→ designating the components of the vector part of q as $v = (a, b, c)$

Shankar R
Asst Professor,
CSE, BMSIT&M

we obtain the elements for the composite rotation matrix

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

→ Using the following trigonometric identities to simplify the trignometric identities.

$$\cos^2\theta/2 - \sin^2\theta/2 = 1 - 2\sin^2\theta/2 = \cos\theta,$$

$$2\cos\theta/2 \sin\theta/2 = \sin\theta$$

we can rewrite as.

$M_R(\theta) =$

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

Shankar R
Asst Professor,
CSE, BMSIT&M

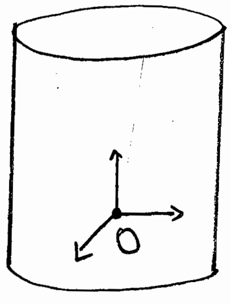
66. What is affine transformation?

A co-ordinate transformation of the form

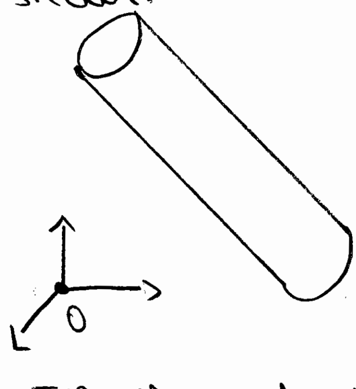
$$x' = a_{xx}x + a_{xy}y + a_{xz}z + b_x$$
$$y' = a_{yx}x + a_{yy}y + a_{yz} + b_y$$
$$z' = a_{zx}x + a_{zy}y + a_{zz}z + b_z$$

is called an affine transformation. Each of the transformed co-ordinates x', y' and z' is a linear function of the original co-ordinates x, y and z. Parameters a_{ij} and b_k are constants determined by the transformation type.

The figure below shows an example of what we mean. On the left, a cylinder has been built in a convenient place, and to a convenient size. Because of the requirements of a scene, it is first scaled to be longer and thinner than its original design, rotated to a desired orientation in space, and then moved to a desired position. The set of operations providing for all such transformations, are known as the affine transforms. The affines include translations and all linear transformations, like scale, rotate, and shear.



Original cylinder model



Transformed cylinder. It has been scaled, rotated & translated.

→ Affine Transformations.

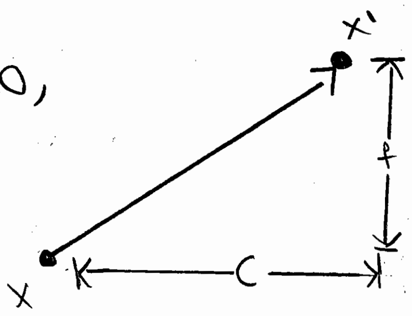
Let us first examine the affine transforms in 2D space where it is easy to illustrate them with diagrams, Then later we will look at the affines in 3D.

Consider a point $x = (x, y)$. Affine transformations of x are all transforms that can be written

$$x' = \begin{bmatrix} ax + by + c \\ dx + ey + f \end{bmatrix},$$

where a through f are scalars.

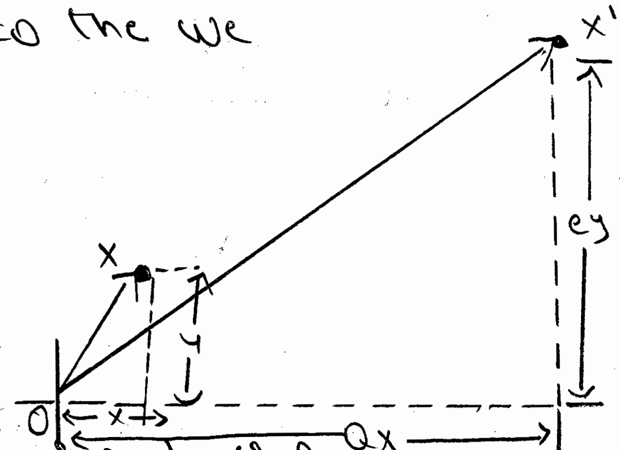
For example, if $a, e = 1$, and $b, d = 0$, then we have a pure translation



$$x' = \begin{bmatrix} x + c \\ y + f \end{bmatrix},$$

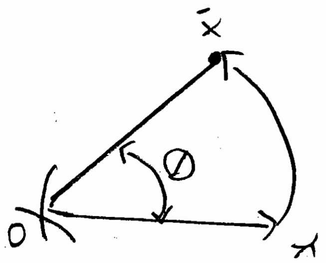
If $b, d = 0$ and $c, f = 0$ then we have a pure scale.

$$x' = \begin{bmatrix} ax \\ ey \end{bmatrix}$$



And, if $a, e = \cos\theta$, $b = -\sin\theta$, $d = \sin\theta$, and $c, f = 0$, Then we have a pure rotation about the origin

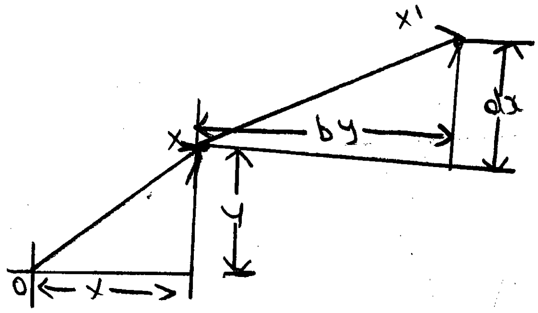
$$x' = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$$



Shankar R
Asst Professor,
CSE, BMSIT&M

Finally if a, e = 1, and c, f = 0 we have the shear transforms

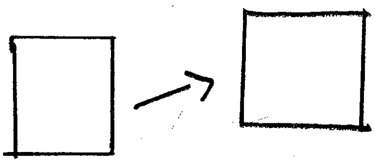
$$x' = \begin{bmatrix} x + by \\ y + dx \end{bmatrix}.$$



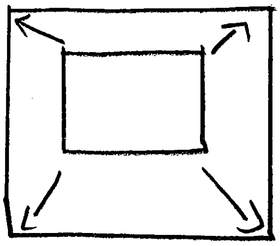
In summary, we have the four basic affine transformations shown in the figure below:

- Translate moves a set of points a fixed distance in x and y,
- Scales scales a set of points up or down in the x and y directions,
- Rotate rotates a set of points about the origin,
- Shear offsets a set of points a distance proportional to their x and y co-ordinates.

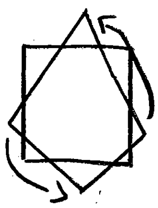
Note that only shear and scale change the shape determined by a set of points.



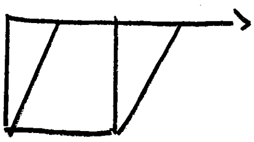
Translate



Scale



Rotate.



Shear

Matrix Representation of the Linear Transformations.

The affine transforms scale, rotate and shear are actually linear transforms and can be represented by a matrix multiplication of a point represented as a vector,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ax + by \\ dx + ey \end{bmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix},$$

or $x' = Mx$, where M is the matrix.

One very nice feature of the matrix representation is that we can use it to factor a complex transform into a set of simpler transforms. For example, suppose we want to scale an object up to a new size, shear the object to a new shape, and finally rotate the object. Let S be the scale matrix, H be the shear matrix and R be the rotation matrix.

$$x' = R(H(Sx))$$

defines a sequence of three transforms: 1st - scale, 2nd - shear, 3rd - rotate. Because matrix multiplication is associative, we can remove the parentheses and multiply the three matrices together, giving a new matrix $M = RHS$. Now we can rewrite out transform

$$x' = (RHS)x = Mx.$$

In matrix form, we can catlog the linear transform as

scale: $\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$, Rotate: $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$; shear: $\begin{bmatrix} 1 & h_x \\ h_y & 1 \end{bmatrix}$,

Shankar R
Asst Professor,
CSE, BMSIT&M

where s_x and s_y scale the x and y coordinates of a point, θ is an angle of counterclockwise rotation around the origin, h_x is a horizontal shear factor, and h_y is a vertical shear factor.

→ Homogeneous co-ordinates

Since the matrix form is so handy for building up complex transforms from simple ones, it would be very useful to be able to represent all of the affine transforms by matrices. The problem is that translation is not a linear transform. The way out of this dilemma is to turn the 2D problem into a 3D problem, but in homogeneous coordinates.

We first take all of our points $x=(x,y)$, express them as 2D vectors $\begin{bmatrix} x \\ y \end{bmatrix}$ and make these into 3D vectors with identical 3rd co-ordinates set to 1:

$$\begin{bmatrix} x \\ y \end{bmatrix} \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

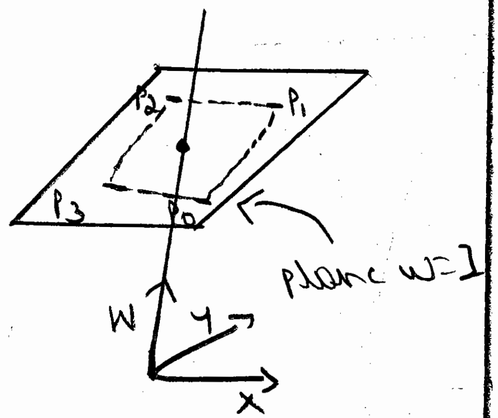
By convention, we call this coordinate the w co-ordinate, to distinguish it from the usual 3D z coordinate. We also extend our 2D matrices to 3D homogeneous form by appending an extra row and column, giving.

$$\text{scale}: \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \text{Rotate}: \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \text{shear}: \begin{bmatrix} 1 & h_x & 0 \\ h_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Note what happens when we multiply our 3D homogeneous matrices by 3D homogeneous vectors:

$$\begin{bmatrix} a & b & 0 \\ d & e & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax+by \\ dx+ey \\ 1 \end{bmatrix}$$

This is the same result as in 2D, with the exception of the extra w coordinate, which remains 1. All we have really done is to place all of our 2D points on the plane w=1 in 3D space, and now we do all the operations on this plane. Really, the operations are still 2D operations.



But, the magic happens when we place the translation parameters c and f in the matrix in the 3rd column:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax+by+c \\ dx+ey+f \\ 1 \end{bmatrix}$$

we can now do translations as linear operations in homogeneous coordinates! so, we can add a final matrix to our catalog:

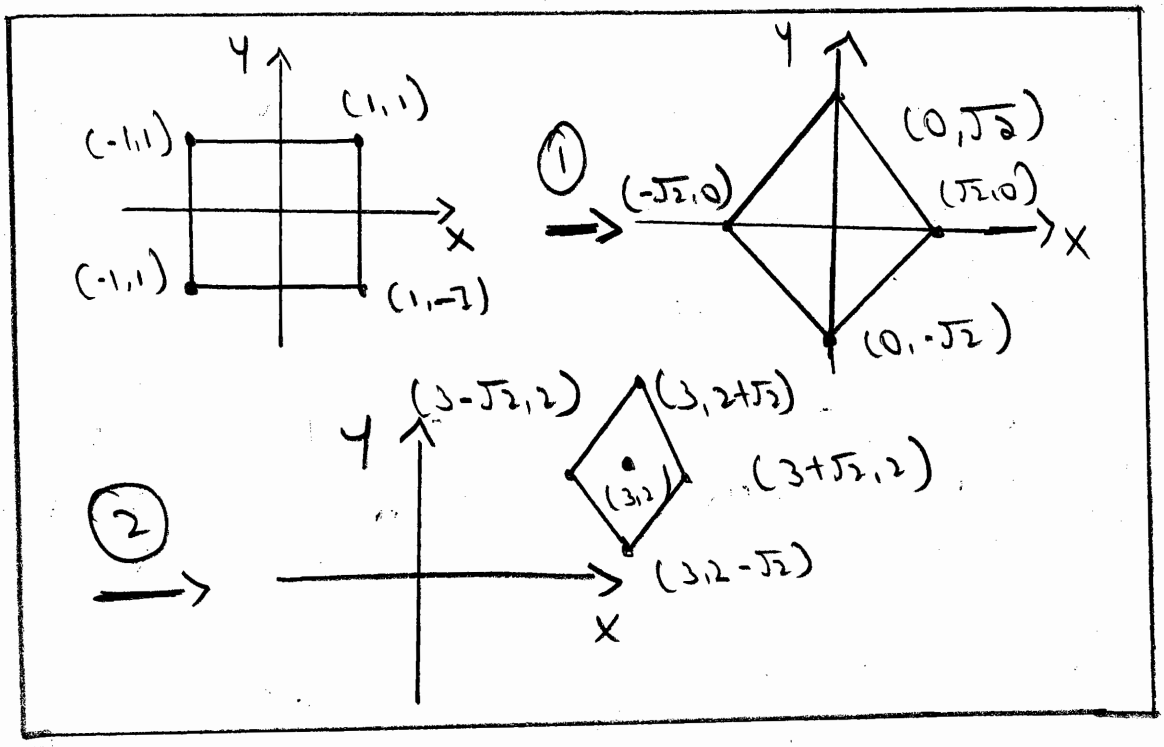
$$\text{Translate}: \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix}$$

where Δx is the translation in the x direction and Δy is the translate in the y direction. The astute reader will see the trick behind the magic - 2D translation is now being expressed as a shear in 3D space.

Now, suppose we have a 2x2 square centered at the origin and we want to first rotate the square by 45°.

Shankar R
Asst Professor,
CSE, BMSIT&M

about its center and then move the square so its center is at (3,2). we can do this in two steps, as shown in the diagram to the right.



In matrix form:

$$H = T_{(3,2)} R_{45^\circ} = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & 0 \\ \sin 45^\circ & \cos 45^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & 3 \\ \sin 45^\circ & \cos 45^\circ & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 3 \\ \sqrt{2}/2 & \sqrt{2}/2 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

Note that

$$H \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 + \frac{3}{\sqrt{2}} \\ 1 \end{bmatrix}, \text{ and } H \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3-\sqrt{2} \\ 2 \\ 1 \end{bmatrix}$$

verifying that we get the same result shown in the figure.

→ 3D Form of the Affine Transformations

Now, we can extend all of these ideas to 3D in the following.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

The extra (4th) coordinate is again called the w co-ordinate.

I. Use matrices to represent the 3D affine transforms in homogeneous form. The following matrices constitute the basic affine transforms in 3D, expressed in homogeneous form:

Translate: $\begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix}$, scale: $\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and shear: $\begin{bmatrix} 1 & h_{xy} & h_{xz} & 0 \\ h_{yx} & 1 & h_{yz} & 0 \\ h_{zx} & h_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

In addition, there are three basic rotations in 3D,

Rotation about x axis: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Rotation about the y axis: $\begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

and Rotation about the z axis: $\begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

The rotations, specified in this way, determine an amount of rotation about each of the individual axes of the coordinate system. The angles θx, θy, and θz of rotation about the three axes are called the Euler angles. They can be used to describe an off-axis rotation, by combining Euler angle rotations via matrix multiplication, Note, however, that the order of rotation affects the end result, so besides specifying Euler angles, an order of rotation must be specified. In general, affine transformations are associative but are not commutative, so the order in which operations are done is highly important. One can see this for rotations by computing the product $R\theta x$, $R\theta y$, $R\theta z$, and comparing with the result obtained by the product $R\theta z$, $R\theta y$, $R\theta x$.

Shankar R
Asst Professor,
CSE, BMSIT&M

67. List the 3D OpenGL geometric transformations.

The 3D OpenGL geometric transformations are.

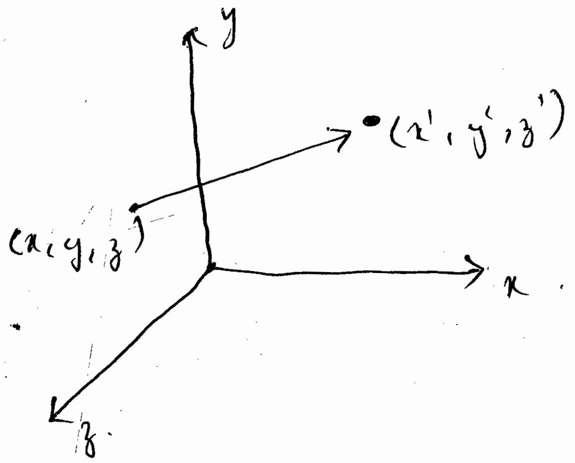
(i) Three - Dimensional Translation
(ii) Three - Dimensional Rotation
(iii) Three - Dimensional scaling.

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 and t_z to the cartesian coordinates of P

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

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

$$P' = T \cdot P$$



Shankar R
Asst Professor,
CSE, BMSIT&M

Three - Dimensional Rotation.

The 2D z-axis rotation equations are easily extended to three dimensions, as follows:

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

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

$$P' = R_z(\theta) \cdot P$$

$$x \rightarrow y \rightarrow z \rightarrow x$$

$$y' = y\cos\theta - z\sin\theta$$
$$z' = y\sin\theta + z\cos\theta \quad \longrightarrow \text{x-axis rotation.}$$
$$x' = x.$$

y-axis rotation

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

Three Dimensional scaling.

$$P' = S \cdot P$$

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

Shankar R
Asst Professor,
CSE, BMSIT&M

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

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

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

Shankar R
Asst Professor,
CSE, BMSIT&M

68. What is color model? Explain the RGB color model.

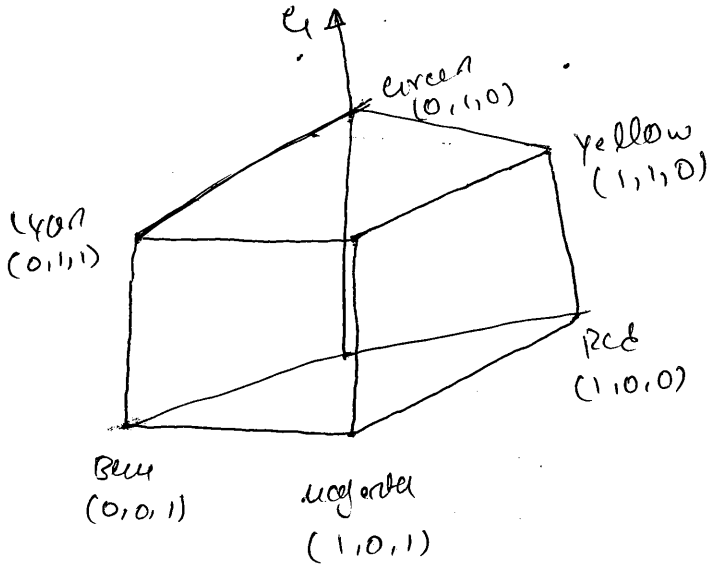
Any method for explaining the properties or behavior of color within some particular context is called a color model.

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 630nm (red), another has its peak sensitivity at about 530nm (green) and the third pigment is most receptive to light with a wavelength of about 450nm (blue). This theory of vision is the basis for displaying color output on a video monitor using the three primaries red, green and blue.

We can represent this model using the unit color cube defined on R, G and B axes.

Shankar R
Asst Professor,
CSE, BMSIT&M



As with XYZ color system, the RGB color scheme is an additive model. Each color point within the unit cube can represent as a weighted vector sum of the primary colors, using unit vectors R, G and B

$$C(\lambda) = (R, G, B) = RR + GG + BB$$

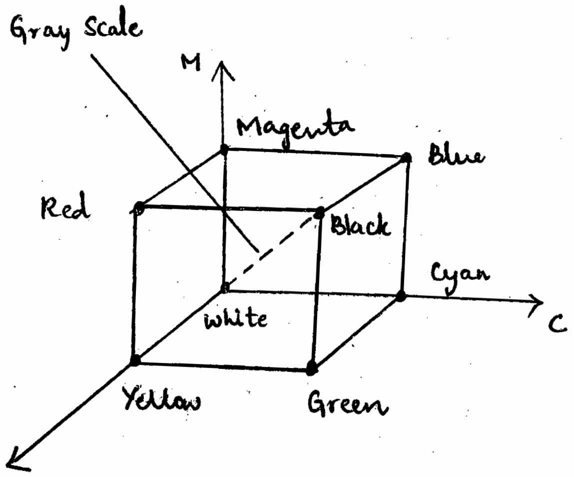
where parameters R, G and B are assigned values in the range from 0 to 1.0 for example.

White is the maximum sum of all three value (1,1,1)

Gray scale color can be represented halfway between white and black (0.5, 0.5, 0.5)

Shankar R.
Asst Professor,
CSE, BMSIT&M

69. Explain the CMY and CMYK color models.



A subtractive color model can be formed with the 3 primary colors cyan, magenta and yellow. When white light is reflected from cyan colored ink the reflected light contains only the green and blue components, and the red components are absorbed. In the CMY model the spacial position (1,1,1) represent black because all components of incident light are subtracted. The Origin represent white light. Equal amounts of each of the primary colors produces shades of grey along the cube's main diagonal. A combination of cyan and magenta ink produces blue light. similarly a combination of cyan and yellow produce green light and combination of magenta and yellow yields red light.

The CMY printing process often uses a collection of 4 ink dots, which ~~are~~ are arranged in a close pattern. In practise the CMY model is referred to as the CMYK model,

Shankar R
Asst Professor,
CSE, BMSIT&M

where k is the black color parameter. A black dot is included because the reflected light from the cyan, magenta and yellow inks typically produce only shades of grey. Some plotters produces different color combination by spraying the ink for the 3 primary colors over each other and allowing them to mix before they dry. for black and white and grey scale ~~paint~~ printing, only the black ink is used.

Shankar R
Asst Professor,
CSE, BMSIT&M

7.0 What is Light Source? Explain the different types of light Source? ①

Light Source :-

⇒ An object that is emitting radiant energy is called as light Source that contribute to the lighting effects for other objects

⇒ We can model light sources with a Variety of shapes & characteristics

⇒ It is defined with number of properties by specify its position, colour, direction & shape

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

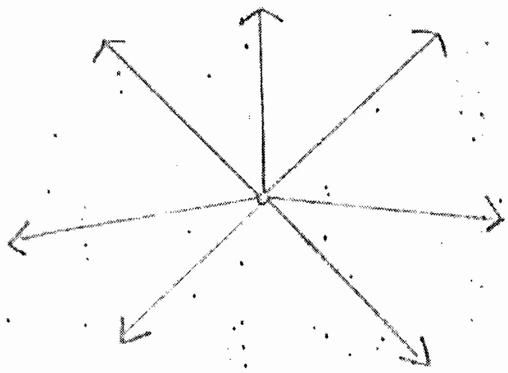
The Different types of Light Sources are :-

1) Point Light Source

→ The Simplest model for an object that is emitting radiant energy is a Point light Source with a Single color, specified with 3 RGB Components.

Shankar R
Asst Professor,
CSE, BMSIT&M

9



=) A Point Source for a Scene by giving its Position & the color of the emitted light.

=) The Rays are generated along radially diverging path from the Single Color Source Position.

=) This light-Source model is reasonable approximation for Sources whose dimension are small compared to the Size of object in that Scene.

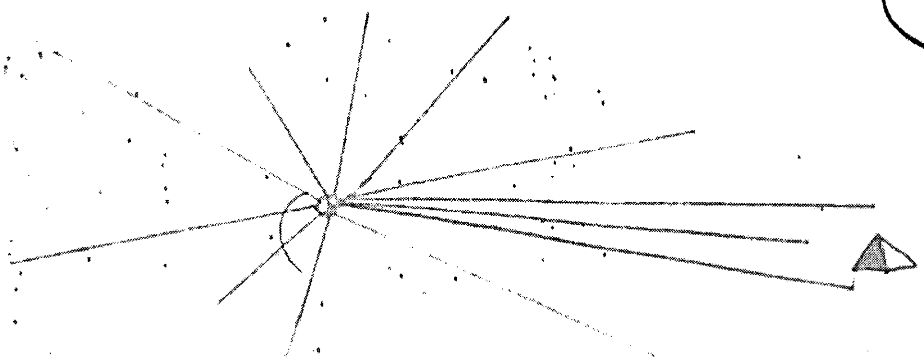
2) Infinitely Distant Light Sources

=) 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.

Shankar R
Asst Professor,
CSE, BMSIT&M

3



=) We can Simulate an infinetly distant light source by assigning it a color value & a fixed direction for the light ray from the Source

=) The Vector for the emission direction & the light source color are needed in the illumination calculation; but not position of Source

3) Radial Intensity Attenuation :-

=) As radiant energy from a light source travels outwards through space, its amplitude at any distance d_1 from source is attenuated by the factor $1/d^2$ a surface close to the light source receives a higher incident light intensity from that source than more distant Surface.

Shankar R
Asst. Professor,
CSE, BMSIT&M

⇒ However using an attenuation factor of $1/d_l^2$ with a point source does not always produce realistic pictures (4)

⇒ The factor $1/d_l^2$ tends to produce too much intensity variation for objects that are close to the light source.

⇒ We can attenuate light intensities with an inverse quadratic function of d_l that includes a linear term.

$$f_{radiatten}(d_l) = \frac{1}{a_0 + a_1 d_l + a_2 d_l^2}$$

⇒ The numerical values for the co-efficient a_0, a_1, & a_2 can then be adjusted to produce optimal attenuation effects.

⇒ We cannot apply intensity attenuation calculation 1 to a point source at "infinity" because the distance of the light source.

$$f_{l,radatten} = \begin{cases} 1.0, & \text{if source is infinity} \\ \dfrac{1}{a_0 + a_1 d_l + a_2 d_l^2}, & \text{if source is local} \end{cases}$$

Shankar R
Asst Professor,
CSE, BMSIT&M

5

4) Directional light Source & Spotlight effect

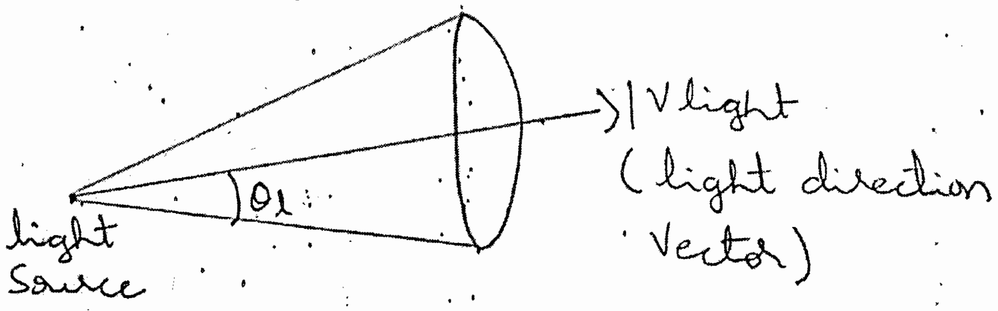
=) A local light source can be modified easily to produce a directional or spotlight beam of light.

=) If an object is outside the directional limit 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 & an angular limit θ_l.

=) We can denote V_{light} as the unit vector in the light source direction & V_{obj} as the unit vector in the direction from the light position to an object position.

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

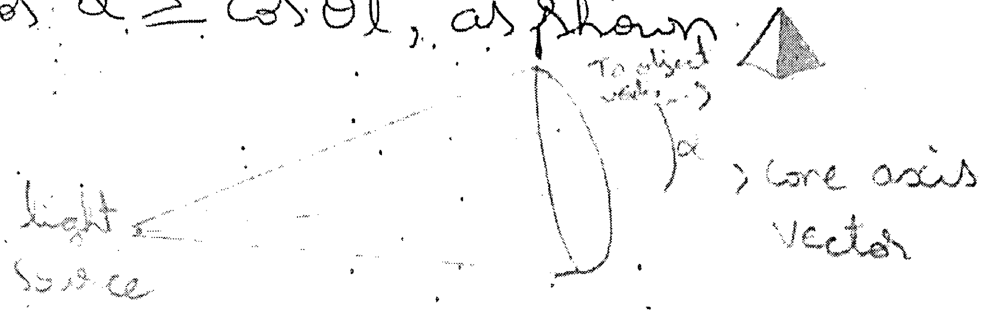


where angle α is the angular distance of the object from the light direction vector

Shankar R
Asst Professor,
CSE, BMSIT&M

(6)

=) 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 $\cos \alpha \geq \cos \theta l$, as shown



=) If $V_{obj} \cdot V_{light} < \cos \theta l$, however the object is outside the light cone.

(5)

Angular Intensity Attenuation :-

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

=) This allows intensity decreasing as we move farther from the cone axis.

=) A commonly used angular intensity attenuation function for a directional light source is $f_{angatten}(\phi) = \cos^{al} \phi$, $0° \leq \phi \leq \theta$

Shankar R
Asst Professor,
CSE, BMSIT&M (7)

⇒ Where the attenuation exponent a_l is assigned some positive value & angle ϕ is measured from the cone axis.

⇒ The greater the value for the attenuation exponent a_l, the smaller the value of the angular intensity function for a given value of angle $\phi > 0°$.

⇒ There is no angular attenuation if the light source is not directional

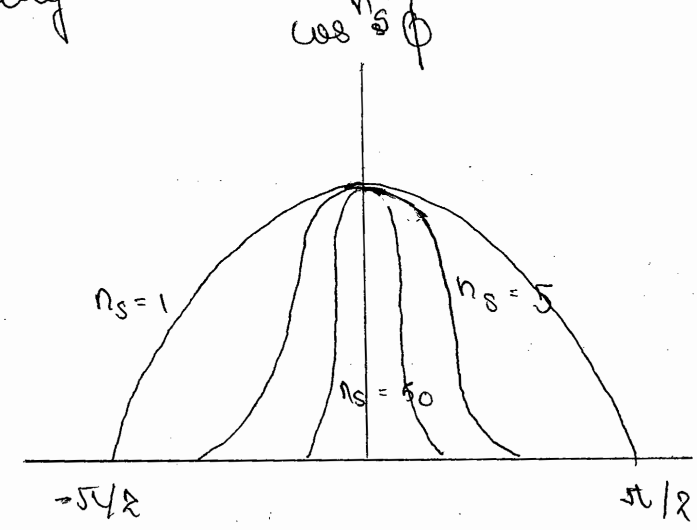
⇒ We can express the general equation for angular attenuation as :-

$$f_l.angatten = \begin{cases} 1.0, & \text{if source is not a spot light} \\ 0.0, & \text{if } V_{obj} \cdot V_{light} = \cos\alpha < \cos\theta_l \\ (V_{obj} \cdot V_{light})^{a_l}, & \text{otherwise} \end{cases}$$

Shankar R
Asst Professor,
CSE, BMSIT&M

71) Explain the PHONG Model.

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



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

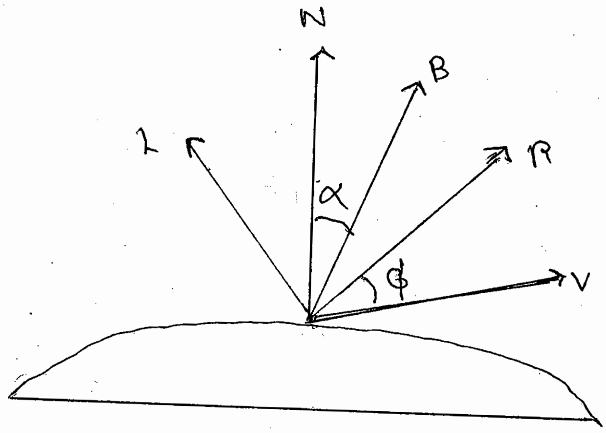
$$I_{l,specular} = W(\theta)\, I_l \cos^{n_s}\phi$$

$0 \le W(\theta) \le 1$ is called specular reflection coefficient

Shankar R
Asst Professor,
CSE, BMSIT&M

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

For most opaque materials specular-reflection co-efficient is nearly constant k_s



$$I_{specular} = \begin{cases} k_s I_l (V \cdot R)^{n_s}, & V \cdot R > 0 \text{ and } N \cdot L > 0 \\ 0.0, & \text{otherwise} \end{cases}$$

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

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

Efficient computation : $H = \frac{L+V}{|L+V|}$

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

H is the direction yielding maximum specular reflection in viewing direction V if the surface normal N would coincide with H

If V is coplanar with R and L (and hence with N too) $\alpha = \phi/2$