Chapter Transformation

Transformation

Changing Position, shape, size, or orientation of an object on display is known as transformation.

Basic Transformation

- Basic transformation includes three transformations **Translation**, **Rotation**, and **Scaling**.
- These three transformations are known as basic transformation because with combination of these three transformations we can obtain any transformation.

Translation

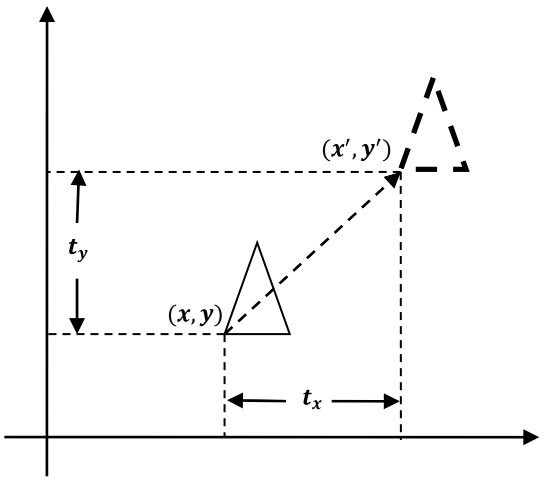


Fig. 3.1: - Translation.

- It is a transformation that used to reposition the object along the straight line path from one coordinate location to another.
- It is rigid body transformation so we need to translate whole object.
- We translate two dimensional point by adding translation distance $\boldsymbol{t_x}$ and $\boldsymbol{t_y}$ to the original coordinate position $(\boldsymbol{x}, \boldsymbol{y})$ to move at new position $(\boldsymbol{x'}, \boldsymbol{y'})$ as:
  $\boldsymbol{x' = x + t_x}$ **&** $\boldsymbol{y' = y + t_y}$
- Translation distance pair $(\boldsymbol{t_{x,}t_y})$ is called a **Translation Vector** or **Shift Vector**.
- We can represent it into single matrix equation in column vector as;
  $\boldsymbol{P' = P + T}$
  $$\begin{bmatrix}\boldsymbol{x'}\\ \boldsymbol{y'}\end{bmatrix} = \begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{bmatrix} + \begin{bmatrix}\boldsymbol{t_x}\\ \boldsymbol{t_y}\end{bmatrix}$$
- We can also represent it in row vector form as:
  $\boldsymbol{P' = P + T}$
  $[\boldsymbol{x'} \quad \boldsymbol{y'}] = [\boldsymbol{x} \quad \boldsymbol{y}] + [\boldsymbol{t_x} \quad \boldsymbol{t_y}]$
- Since column vector representation is standard mathematical notation and since many graphics package like **GKS** and **PHIGS** uses column vector we will also follow column vector representation.
- **Example**: - Translate the triangle [A (10, 10), B (15, 15), C (20, 10)] 2 unit in x direction and 1 unit in y direction.
  We know that
  $P' = P + T$
  $$P' = [P] + \begin{bmatrix}t_x\\ t_y\end{bmatrix}$$

For point (10, 10)

$A' = \begin{bmatrix}10\\10\end{bmatrix} + \begin{bmatrix}2\\1\end{bmatrix}$

$A' = \begin{bmatrix}12\\11\end{bmatrix}$

For point (15, 15)

$B' = \begin{bmatrix}15\\15\end{bmatrix} + \begin{bmatrix}2\\1\end{bmatrix}$

$B' = \begin{bmatrix}17\\16\end{bmatrix}$

For point (10, 10)

$C' = \begin{bmatrix}20\\10\end{bmatrix} + \begin{bmatrix}2\\1\end{bmatrix}$

$C' = \begin{bmatrix}22\\11\end{bmatrix}$

- Final coordinates after translation are [A′ (12, 11), B′ (17, 16), C′ (22, 11)].

Rotation

- It is a transformation that used to reposition the object along the circular path in the XY - plane.
- To generate a rotation we specify a rotation angle $\boldsymbol{\theta}$ and the position of the **Rotation Point** (**Pivot Point**) $(\boldsymbol{x_r, y_r})$ about which the object is to be rotated.
- Positive value of rotation angle defines counter clockwise rotation and negative value of rotation angle defines clockwise rotation.
- We first find the equation of rotation when pivot point is at coordinate origin$(\mathbf{0}, \mathbf{0})$.

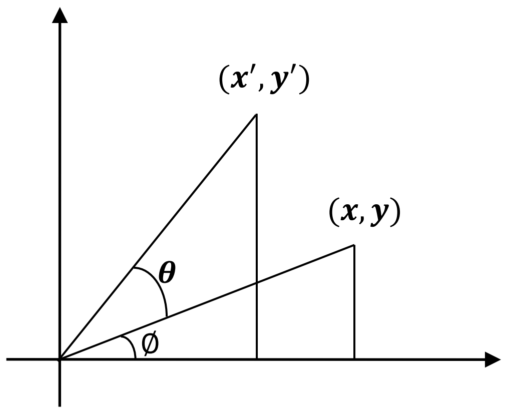


Fig. 3.2: - Rotation.

- From figure we can write.

$\boldsymbol{x = r\cos\emptyset}$

$\boldsymbol{y = r\sin\emptyset}$

and

$\boldsymbol{x' = r\cos(\theta + \emptyset) = r\cos\emptyset\cos\theta - r\sin\emptyset\sin\theta}$

$\boldsymbol{y' = r\sin(\emptyset + \theta) = r\cos\emptyset\sin\theta + r\sin\emptyset\cos\theta}$

- Now replace $\boldsymbol{r\cos\emptyset}$ with $\boldsymbol{x}$ and $\boldsymbol{r\sin\emptyset}$ with $\boldsymbol{y}$ in above equation.

$\boldsymbol{x' = x\cos\theta - y\sin\theta}$

$\boldsymbol{y' = x\sin\theta + y\cos\theta}$

- We can write it in the form of column vector matrix equation as;

$\boldsymbol{P' = R \cdot P}$

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

- Rotation about arbitrary point is illustrated in below figure.

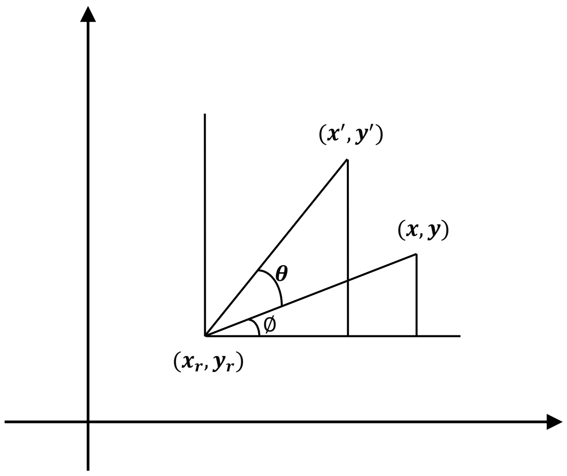


Fig. 3.3: - Rotation about pivot point.

- Transformation equation for rotation of a point about pivot point (x_r, y_r) is:
  $x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$
  $y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$
- These equations are differing from rotation about origin and its matrix representation is also different.
- Its matrix equation can be obtained by simple method that we will discuss later in this chapter.
- Rotation is also rigid body transformation so we need to rotate each point of object.
- **Example**: - Locate the new position of the triangle [A (5, 4), B (8, 3), C (8, 8)] after its rotation by 90° clockwise about the origin.
  As rotation is clockwise we will take $\theta = -90^\circ$.
  $P' = R \cdot P$
  $$P' = \begin{bmatrix} \cos(-90) & -\sin(-90) \\ \sin(-90) & \cos(-90) \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \end{bmatrix}$$
  $$P' = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \end{bmatrix}$$
  $$P' = \begin{bmatrix} 4 & 3 & 8 \\ -5 & -8 & -8 \end{bmatrix}$$
- Final coordinates after rotation are [A$'$ (4, -5), B$'$ (3, -8), C$'$ (8, -8)].

Scaling

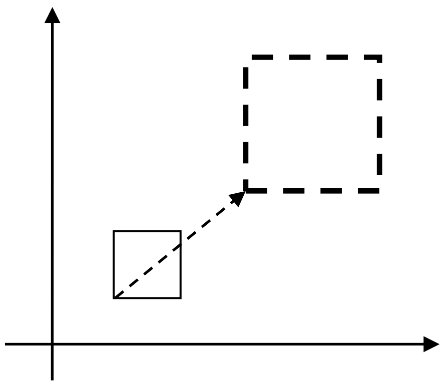


Fig. 3.4: - Scaling.

- It is a transformation that used to alter the size of an object.
- This operation is carried out by multiplying coordinate value $(\boldsymbol{x}, \boldsymbol{y})$ with scaling factor $(\boldsymbol{s_x}, \boldsymbol{s_y})$ respectively.
- So equation for scaling is given by:
  $\boldsymbol{x' = x \cdot s_x}$
  $\boldsymbol{y' = y \cdot s_y}$
- These equation can be represented in column vector matrix equation as:
  $\boldsymbol{P' = S \cdot P}$
  $$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$
- Any positive value can be assigned to $(\boldsymbol{s_x}, \boldsymbol{s_y})$.
- Values less than 1 reduce the size while values greater than 1 enlarge the size of object, and object remains unchanged when values of both factor is 1.
- Same values of $\boldsymbol{s_x}$ and $\boldsymbol{s_y}$ will produce **Uniform Scaling**. And different values of $\boldsymbol{s_x}$ and $\boldsymbol{s_y}$ will produce **Differential Scaling**.
- Objects transformed with above equation are both scale and repositioned.
- Scaling factor with value less than 1 will move object closer to origin, while scaling factor with value greater than 1 will move object away from origin.
- We can control the position of object after scaling by keeping one position fixed called **Fix point** $(\boldsymbol{x_f}, \boldsymbol{y_f})$ that point will remain unchanged after the scaling transformation.

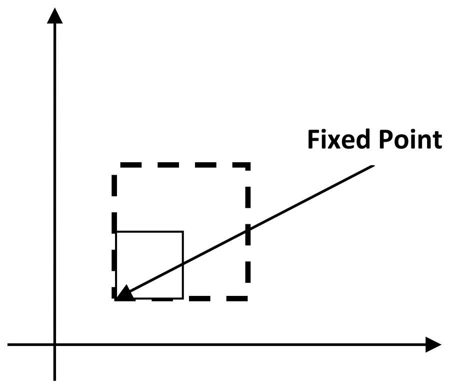


Fig. 3.5: - Fixed point scaling.

- Equation for scaling with fixed point position as $(\boldsymbol{x_f}, \boldsymbol{y_f})$ is:
  $\boldsymbol{x' = x_f + (x - x_f)s_x}$ $\boldsymbol{y' = y_f + (y - y_f)s_y}$
  $\boldsymbol{x' = x_f + xs_x - x_f s_x}$ $\boldsymbol{y' = y_f + ys_y - y_f s_y}$
  $\boldsymbol{x' = xs_x + x_f(1 - s_x)}$ $\boldsymbol{y' = ys_y + y_f(1 - s_y)}$
- Matrix equation for the same will discuss in later section.
- Polygons are scaled by applying scaling at coordinates and redrawing while other body like circle and ellipse will scale using its defining parameters. For example ellipse will scale using its semi major axis, semi minor axis and center point scaling and redrawing at that position.
- **Example**: - Consider square with left-bottom corner at (2, 2) and right-top corner at (6, 6) apply the transformation which makes its size half.
  As we want size half so value of scale factor are $s_x = 0.5, s_y = 0.5$ and Coordinates of square are [A (2, 2), B (6, 2), C (6, 6), D (2, 6)].
  $P' = S \cdot P$

$$P' = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \end{bmatrix}$$
$$P' = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \end{bmatrix}$$
$$P' = \begin{bmatrix} 1 & 3 & 3 & 1 \\ 1 & 1 & 3 & 3 \end{bmatrix}$$

- Final coordinate after scaling are [A$'$ (1, 1), B$'$ (3, 1), C$'$ (3, 3), D$'$ (1, 3)].

Matrix Representation and homogeneous coordinates

- Many graphics application involves sequence of geometric transformations.
- For example in design and picture construction application we perform Translation, Rotation, and scaling to fit the picture components into their proper positions.
- For efficient processing we will reformulate transformation sequences.
- We have matrix representation of basic transformation and we can express it in the general matrix form as:

  $\boldsymbol{P' = M_1 \cdot P + M_2}$

  Where $\boldsymbol{P}$ and $\boldsymbol{P'}$ are initial and final point position, $\boldsymbol{M_1}$ contains rotation and scaling terms and $\boldsymbol{M_2}$ contains translation al terms associated with pivot point, fixed point and reposition.
- For efficient utilization we must calculate all sequence of transformation in one step and for that reason we reformulate above equation to eliminate the matrix addition associated with translation terms in matrix $\boldsymbol{M_2}$.
- We can combine that thing by expanding 2X2 matrix representation into 3X3 matrices.
- It will allows us to convert all transformation into matrix multiplication but we need to represent vertex position $(\boldsymbol{x, y})$ with homogeneous coordinate triple $(\boldsymbol{x_h, y_h, h})$ Where $\boldsymbol{x = \frac{x_h}{h}}$, $\boldsymbol{y = \frac{y_h}{h}}$ thus we can also write triple as $(\boldsymbol{h \cdot x, h \cdot y, h})$.
- For two dimensional geometric transformation we can take value of $\boldsymbol{h}$ is any positive number so we can get infinite homogeneous representation for coordinate value $(\boldsymbol{x, y})$.
- But convenient choice is set $\boldsymbol{h = 1}$ as it is multiplicative identity, than $(\boldsymbol{x, y})$ is represented as $(\boldsymbol{x, y, 1})$.
- Expressing coordinates in homogeneous coordinates form allows us to represent all geometric transformation equations as matrix multiplication.
- Let's see each representation with $\boldsymbol{h = 1}$

  Translation

  $$\boldsymbol{P' = T_{(t_x, t_y)} \cdot P}$$
  $$\begin{bmatrix} \boldsymbol{x'} \\ \boldsymbol{y'} \\ \boldsymbol{1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{t_x} \\ \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{t_y} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{1} \end{bmatrix}$$

  NOTE: - Inverse of translation matrix is obtain by putting $-\boldsymbol{t_x}$ & $-\boldsymbol{t_y}$ instead of $\boldsymbol{t_x}$ & $\boldsymbol{t_y}$.

  Rotation

  $$\boldsymbol{P' = R_{(\theta)} \cdot P}$$
  $$\begin{bmatrix} \boldsymbol{x'} \\ \boldsymbol{y'} \\ \boldsymbol{1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\cos\theta} & \boldsymbol{-\sin\theta} & \boldsymbol{0} \\ \boldsymbol{\sin\theta} & \boldsymbol{\cos\theta} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{1} \end{bmatrix}$$

  NOTE: - Inverse of rotation matrix is obtained by replacing $\boldsymbol{\theta}$ by $-\boldsymbol{\theta}$.

  Scaling

  $$\boldsymbol{P' = S_{(s_x, s_y)} \cdot P}$$

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

NOTE: - Inverse of scaling matrix is obtained by replacing $\boldsymbol{s_x}$ & $\boldsymbol{s_y}$ by $\frac{1}{s_x}$ & $\frac{1}{s_y}$ respectively.

Composite Transformation

- We can set up a matrix for any sequence of transformations as a **composite transformation matrix** by calculating the matrix product of individual transformation.
- For column matrix representation of coordinate positions, we form composite transformations by multiplying matrices in order from right to left.

Translations

- Two successive translations are performed as:

$$\boldsymbol{P' = T(t_{x2}, t_{y2}) \cdot \{T(t_{x1}, t_{y1}) \cdot P\}}$$
$$\boldsymbol{P' = \{T(t_{x2}, t_{y2}) \cdot T(t_{x1}, t_{y1})\} \cdot P}$$
$$\boldsymbol{P' = \begin{bmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix} \cdot P}$$
$$\boldsymbol{P' = \begin{bmatrix} 1 & 0 & t_{x1} + t_{x2} \\ 0 & 1 & t_{y1} + t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot P}$$
$$\boldsymbol{P' = T(t_{x1} + t_{x2}, t_{y1} + t_{y2}) \cdot P\}}$$

Here $\boldsymbol{P'}$ and $\boldsymbol{P}$ are column vector of final and initial point coordinate respectively.

- This concept can be extended for any number of successive translations.

Example: Obtain the final coordinates after two translations on point $p(2,3)$ with translation vector $(4,\ 3)$ and $(-1,\ 2)$ respectively.

$$P' = T(t_{x1} + t_{x2}, t_{y1} + t_{y2}) \cdot P$$

$$P' = \begin{bmatrix} 1 & 0 & t_{x1} + t_{x2} \\ 0 & 1 & t_{y1} + t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot P = \begin{bmatrix} 1 & 0 & 4 + (-1) \\ 0 & 1 & 3 + 2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 8 \\ 1 \end{bmatrix}$$

Final Coordinates after translations are $p'(5,\ 8)$.

Rotations

- Two successive Rotations are performed as:

$$\boldsymbol{P' = R(\theta_2) \cdot \{R(\theta_1) \cdot P\}}$$
$$\boldsymbol{P' = \{R(\theta_2) \cdot R(\theta_1)\} \cdot P}$$
$$\boldsymbol{P' = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P}$$
$$\boldsymbol{P' = \begin{bmatrix} \cos\theta_2\cos\theta_1 - \sin\theta_2\sin\theta_1 & -\sin\theta_1\cos\theta_2 - \sin\theta_2\cos\theta_1 & 0 \\ \sin\theta_1\cos\theta_2 + \sin\theta_2\cos\theta_1 & \cos\theta_2\cos\theta_1 - \sin\theta_2\sin\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P}$$

$$\boldsymbol{P'} = \begin{bmatrix} \cos(\boldsymbol{\theta_1}+\boldsymbol{\theta_2}) & -\sin(\boldsymbol{\theta_1}+\boldsymbol{\theta_2}) & \mathbf{0} \\ \sin(\boldsymbol{\theta_1}+\boldsymbol{\theta_2}) & \cos(\boldsymbol{\theta_1}+\boldsymbol{\theta_2}) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \cdot \boldsymbol{P}$$

$$\boldsymbol{P'} = \boldsymbol{R}(\boldsymbol{\theta_1}+\boldsymbol{\theta_2}) \cdot \boldsymbol{P}$$

Here $\boldsymbol{P'}$ and $\boldsymbol{P}$ are column vector of final and initial point coordinate respectively.

- This concept can be extended for any number of successive rotations.

Example: Obtain the final coordinates after two rotations on point $p(6,9)$ with rotation angles are 30^o and 60^o respectively.

$$P' = R(\theta_1+\theta_2) \cdot P$$

$$P' = \begin{bmatrix} cos(\theta_1+\theta_2) & -sin(\theta_1+\theta_2) & 0 \\ sin(\theta_1+\theta_2) & cos(\theta_1+\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

$$P' = \begin{bmatrix} cos(30+60) & -sin(30+60) & 0 \\ sin(30+60) & cos(30+60) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

$$P' = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 6 \\ 9 \\ 1 \end{bmatrix} = \begin{bmatrix} -9 \\ 6 \\ 1 \end{bmatrix}$$

Final Coordinates after rotations are $p'(-9,\ 6)$.

Scaling

- Two successive scaling are performed as:

$$\boldsymbol{P'} = \boldsymbol{S}(\boldsymbol{s_{x2}}, \boldsymbol{s_{y2}}) \cdot \{\boldsymbol{S}(\boldsymbol{s_{x1}}, \boldsymbol{s_{y1}}) \cdot \boldsymbol{P}\}$$

$$\boldsymbol{P'} = \{\boldsymbol{S}(\boldsymbol{s_{x2}}, \boldsymbol{s_{y2}}) \cdot \boldsymbol{S}(\boldsymbol{s_{x1}}, \boldsymbol{s_{y1}})\} \cdot \boldsymbol{P}$$

$$\boldsymbol{P'} = \begin{bmatrix} \boldsymbol{s_{x2}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{s_{y2}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{s_{x1}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{s_{y1}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \cdot \boldsymbol{P}$$

$$\boldsymbol{P'} = \begin{bmatrix} \boldsymbol{s_{x1}} \cdot \boldsymbol{s_{x2}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{s_{y1}} \cdot \boldsymbol{s_{y2}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \cdot \boldsymbol{P}$$

$$\boldsymbol{P'} = \boldsymbol{S}(\boldsymbol{s_{x1}} \cdot \boldsymbol{s_{x2}}, \boldsymbol{s_{y1}} \cdot \boldsymbol{s_{y2}}) \cdot \boldsymbol{P}$$

Here $\boldsymbol{P'}$ and $\boldsymbol{P}$ are column vector of final and initial point coordinate respectively.

- This concept can be extended for any number of successive scaling.

Example: Obtain the final coordinates after two scaling on line pq $[p(2,2),\ q(8,\ 8)]$ with scaling factors are $(2,\ 2)$ and $(3,\ 3)$ respectively.

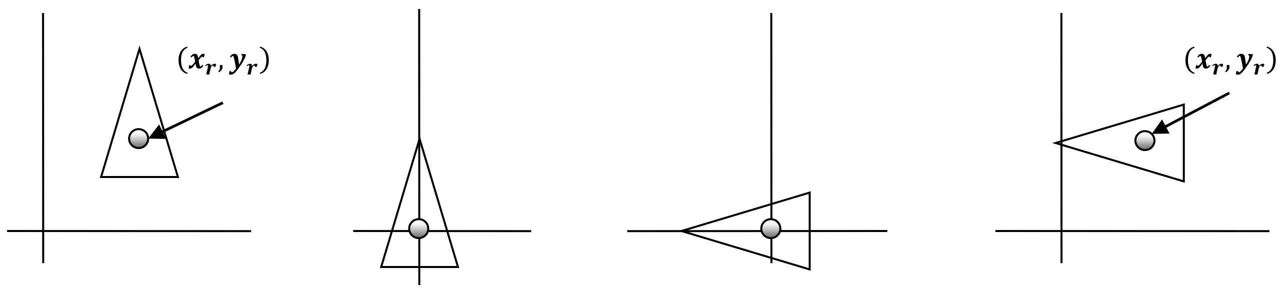
$$P' = S(s_{x1} \cdot s_{x2}, s_{y1} \cdot s_{y2}) \cdot P$$

$$P' = \begin{bmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P = \begin{bmatrix} 2 \cdot 3 & 0 & 0 \\ 0 & 2 \cdot 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

$$P' = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 8 \\ 2 & 8 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 12 & 48 \\ 12 & 48 \\ 1 & 1 \end{bmatrix}$$

Final Coordinates after rotations are $p'(12,\ 12)$ and $q'(48,\ 48)$.

General Pivot-Point Rotation



(a)
Original Position
of Object and
Pivot Point.

(b)
Translation of Object
so that Pivot Point
(x_r, y_r) is at Origin.

(c)
Rotation
about
Origin.

(d)
Translation of Object so
that Pivot Point is Return
to Position (x_r, y_r) .

Fig. 3.6: - General pivot point rotation.

- For rotating object about arbitrary point called pivot point we need to apply following sequence of transformation.
  1. Translate the object so that the pivot-point coincides with the coordinate origin.
  2. Rotate the object about the coordinate origin with specified angle.
  3. Translate the object so that the pivot-point is returned to its original position (i.e. Inverse of step-1).
- Let's find matrix equation for this

$$\boldsymbol{P'} = \boldsymbol{T}(\boldsymbol{x_r}, \boldsymbol{y_r}) \cdot [\boldsymbol{R}(\boldsymbol{\theta}) \cdot \{\boldsymbol{T}(-\boldsymbol{x_r}, -\boldsymbol{y_r}) \cdot \boldsymbol{P}\}]$$

$$\boldsymbol{P'} = \{\boldsymbol{T}(\boldsymbol{x_r}, \boldsymbol{y_r}) \cdot \boldsymbol{R}(\boldsymbol{\theta}) \cdot \boldsymbol{T}(-\boldsymbol{x_r}, -\boldsymbol{y_r})\} \cdot \boldsymbol{P}$$

$$\boldsymbol{P'} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \boldsymbol{x_r} \\ \mathbf{0} & \mathbf{1} & \boldsymbol{y_r} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{cos}\,\boldsymbol{\theta} & -\mathbf{sin}\,\boldsymbol{\theta} & \mathbf{0} \\ \mathbf{sin}\,\boldsymbol{\theta} & \mathbf{cos}\,\boldsymbol{\theta} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{1} & \mathbf{0} & -\boldsymbol{x_r} \\ \mathbf{0} & \mathbf{1} & -\boldsymbol{y_r} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \cdot \boldsymbol{P}$$

$$\boldsymbol{P'} = \begin{bmatrix} \mathbf{cos}\,\boldsymbol{\theta} & -\mathbf{sin}\,\boldsymbol{\theta} & \boldsymbol{x_r}(\mathbf{1} - \mathbf{cos}\,\boldsymbol{\theta}) + \boldsymbol{y_r}\,\mathbf{sin}\,\boldsymbol{\theta} \\ \mathbf{sin}\,\boldsymbol{\theta} & \mathbf{cos}\,\boldsymbol{\theta} & \boldsymbol{y_r}(\mathbf{1} - \mathbf{cos}\,\boldsymbol{\theta}) - \boldsymbol{x_r}\,\mathbf{sin}\,\boldsymbol{\theta} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \cdot \boldsymbol{P}$$

$$\boldsymbol{P'} = \boldsymbol{R}(\boldsymbol{x_r}, \boldsymbol{y_r}\boldsymbol{\theta}) \cdot \boldsymbol{P}$$

Here $\boldsymbol{P'}$ and $\boldsymbol{P}$ are column vector of final and initial point coordinate respectively and $(\boldsymbol{x_r}, \boldsymbol{y_r})$ are the coordinates of pivot-point.

- **Example**: - Locate the new position of the triangle [A (5, 4), B (8, 3), C (8, 8)] after its rotation by 90^o clockwise about the centroid.

Pivot point is centroid of the triangle so:

$$x_r = \frac{5+8+8}{3} = 7\,, \qquad y_r = \frac{4+3+8}{3} = 5$$

As rotation is clockwise we will take $\theta = -90^{\circ}$.

$$P' = R_{(x_r, y_r, \theta)} \cdot P$$

$$P' = \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r\sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r\sin\theta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} \cos(-90) & -\sin(-90) & 7(1-\cos(-90)) + 5\sin(-90) \\ \sin(-90) & \cos(-90) & 5(1-\cos(-90)) - 7\sin(-90) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$

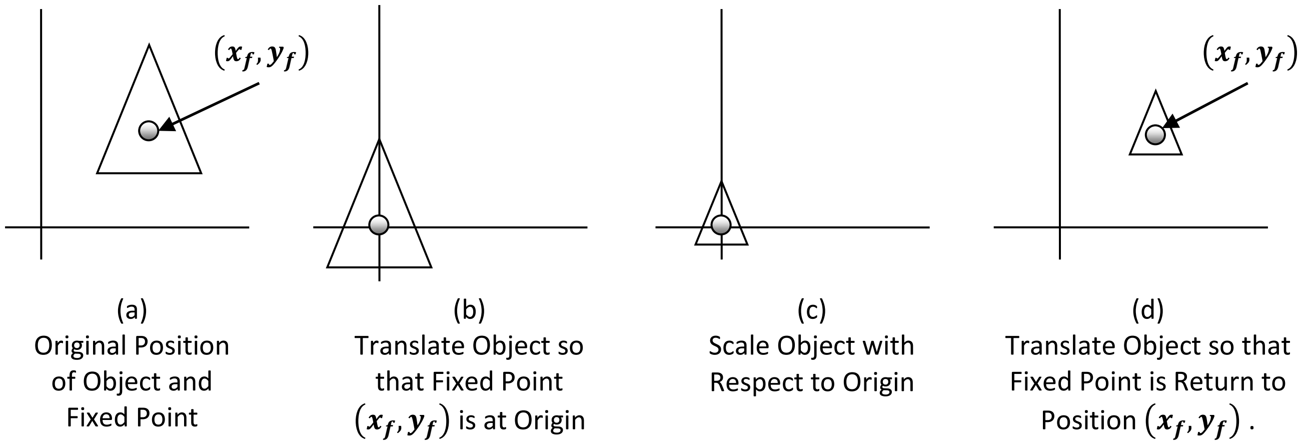
$$P' = \begin{bmatrix} 0 & 1 & 7(1-0) - 5(1) \\ -1 & 0 & 5(1-0) + 7(1) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 12 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 11 & 13 & 18 \\ 7 & 4 & 4 \\ 1 & 1 & 1 \end{bmatrix}$$

- Final coordinates after rotation are [A$'$ (11, 7), B$'$ (13, 4), C$'$ (18, 4)].

General Fixed-Point Scaling



(x_f, y_f)
(a)
Original Position of Object and Fixed Point
(b)
Translate Object so that Fixed Point (x_f, y_f) is at Origin
(c)
Scale Object with Respect to Origin
(x_f, y_f)
(d)
Translate Object so that Fixed Point is Return to Position (x_f, y_f) .

Fig. 3.7: - General fixed point scaling.

- For scaling object with position of one point called fixed point will remains same, we need to apply following sequence of transformation.
  1. Translate the object so that the fixed-point coincides with the coordinate origin.
  2. Scale the object with respect to the coordinate origin with specified scale factors.
  3. Translate the object so that the fixed-point is returned to its original position (i.e. Inverse of step-1).
- Let's find matrix equation for this

$$\boldsymbol{P' = T(x_f, y_f) \cdot [S(s_x, s_y) \cdot \{T(-x_f, -y_f) \cdot P\}]}$$

$$\boldsymbol{P' = \{T(x_f, y_f) \cdot S(s_x, s_y) \cdot T(-x_f, -y_f)\} \cdot P}$$

$$\boldsymbol{P'} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \boldsymbol{x_f} \\ \mathbf{0} & \mathbf{1} & \boldsymbol{y_f} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{s_x} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{s_y} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{1} & \mathbf{0} & \boldsymbol{-x_f} \\ \mathbf{0} & \mathbf{1} & \boldsymbol{-y_f} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \cdot \boldsymbol{P}$$

$$\boldsymbol{P'} = \begin{bmatrix} \boldsymbol{s_x} & \mathbf{0} & \boldsymbol{x_f(1 - s_x)} \\ \mathbf{0} & \boldsymbol{s_y} & \boldsymbol{y_f(1 - s_y)} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \cdot \boldsymbol{P}$$

$$\boldsymbol{P' = S(x_f, y_f, s_x, s_y) \cdot P}$$

Here $\boldsymbol{P'}$ and $\boldsymbol{P}$ are column vector of final and initial point coordinate respectively and $\boldsymbol{(x_f, y_f)}$ are the coordinates of fixed-point.

- **Example**: - Consider square with left-bottom corner at (2, 2) and right-top corner at (6, 6) apply the transformation which makes its size half such that its center remains same.

  Fixed point is center of square so:

$$x_f = 2 + \frac{6-2}{2}, \quad y_f = 2 + \frac{6-2}{2}$$

  As we want size half so value of scale factor are $s_x = 0.5, s_y = 0.5$ and Coordinates of square are [A (2, 2), B (6, 2), C (6, 6), D (2, 6)].

$$P' = S(x_f, y_f, s_x, s_y) \cdot P$$

$$P' = \begin{bmatrix} s_x & 0 & x_f(1 - s_x) \\ 0 & s_y & y_f(1 - s_y) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0.5 & 0 & 4(1-0.5) \\ 0 & 0.5 & 4(1-0.5) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$
$$P' = \begin{bmatrix} 0.5 & 0 & 2 \\ 0 & 0.5 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$
$$P' = \begin{bmatrix} 3 & 5 & 5 & 3 \\ 3 & 3 & 5 & 5 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

- Final coordinate after scaling are [A′ (3, 3), B′ (5, 3), C′ (5, 5), D′ (3, 5)]

General Scaling Directions

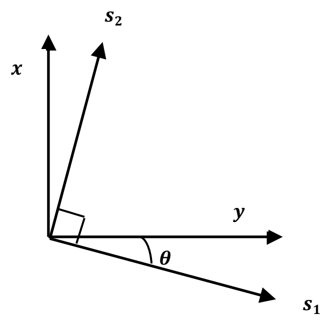


Fig. 3.8: - General scaling direction.

- Parameter $\boldsymbol{s_x}$ and $\boldsymbol{s_y}$ scale the object along $\boldsymbol{x}$ and $\boldsymbol{y}$ directions. We can scale an object in other directions by rotating the object to align the desired scaling directions with the coordinate axes before applying the scaling transformation.
- Suppose we apply scaling factor $\boldsymbol{s_1}$ and $\boldsymbol{s_2}$ in direction shown in figure than we will apply following transformations.
  1. Perform a rotation so that the direction for $\boldsymbol{s_1}$ and $\boldsymbol{s_2}$ coincide with $\boldsymbol{x}$ and $\boldsymbol{y}$ axes.
  2. Scale the object with specified scale factors.
  3. Perform opposite rotation to return points to their original orientations. (i.e. Inverse of step-1).
- Let's find matrix equation for this

$$\boldsymbol{P' = R^{-1}(\theta) \cdot [S(s_1, s_2) \cdot \{R(\theta) \cdot P\}]}$$
$$\boldsymbol{P' = \{R^{-1}(\theta) \cdot S(s_1, s_2) \cdot R(\theta)\} \cdot P}$$
$$\boldsymbol{P' = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P}$$
$$\boldsymbol{P' = \begin{bmatrix} s_1\cos^2\theta + s_2\sin^2\theta & (s_2 - s_1)\cos\theta\sin\theta & 0 \\ (s_2 - s_1)\cos\theta\sin\theta & s_1\sin^2\theta + s_2\cos^2\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P}$$

Here $\boldsymbol{P'}$ and $\boldsymbol{P}$ are column vector of final and initial point coordinate respectively and $\boldsymbol{\theta}$ is the angle between actual scaling direction and our standard coordinate axes.

Other Transformation

- Some package provides few additional transformations which are useful in certain applications. Two such transformations are reflection and shear.

Reflection

- A reflection is a transformation that produces a mirror image of an object.

- The mirror image for a two –dimensional reflection is generated relative to an **axis of reflection** by rotating the object 180^o about the reflection axis.
- Reflection gives image based on position of axis of reflection. Transformation matrix for few positions are discussed here.

<u>Transformation matrix for reflection about the line $\boldsymbol{y = 0}$, $\boldsymbol{the\ x\ axis}$.</u>

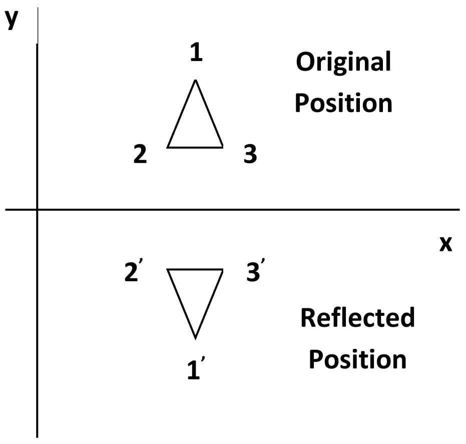


Fig. 3.9: - Reflection about x - axis.

- This transformation keeps x values are same, but flips (Change the sign) y values of coordinate positions.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

<u>Transformation matrix for reflection about the line $\boldsymbol{x = 0}$, $\boldsymbol{the\ y\ axis}$.</u>

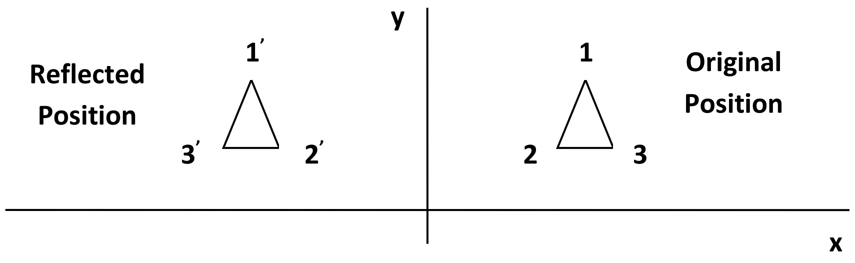


Fig. 3.10: - Reflection about y - axis.

- This transformation keeps y values are same, but flips (Change the sign) x values of coordinate positions.

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

<u>Transformation matrix for reflection about the $\boldsymbol{Origin}$.</u>

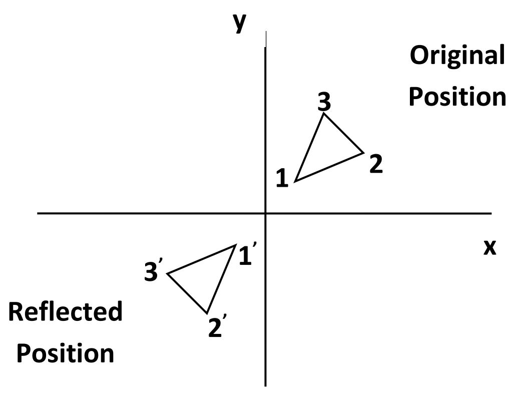


Fig. 3.11: - Reflection about origin.

- This transformation flips (Change the sign) x and y both values of coordinate positions.

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

<u>Transformation matrix for reflection about the line $\boldsymbol{x = y}$.</u>

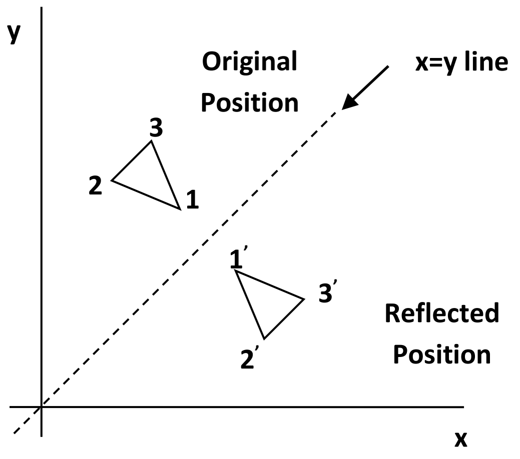


Fig. 3.12: - Reflection about x=y line.

- This transformation interchange x and y values of coordinate positions.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Transformation matrix for reflection about the line $\boldsymbol{x} = -\boldsymbol{y}$.

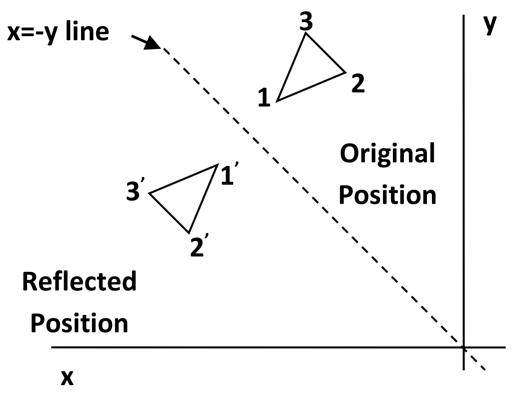


Fig. 3.12: - Reflection about x=-y line.

- This transformation interchange x and y values of coordinate positions.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- **Example**: - Find the coordinates after reflection of the triangle [A (10, 10), B (15, 15), C (20, 10)] about x axis.

$$P' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 10 & 15 & 20 \\ 10 & 15 & 10 \\ 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 10 & 15 & 20 \\ -10 & -15 & -10 \\ 1 & 1 & 1 \end{bmatrix}$$

- Final coordinate after reflection are [A′ (10, -10), B′ (15, -15), C′ (20, -10)]

Shear

- A transformation that distorts the shape of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other is called **shear**.
- Two common shearing transformations are those that shift coordinate x values and those that shift y values.

Shear in $\boldsymbol{x - direction}$.

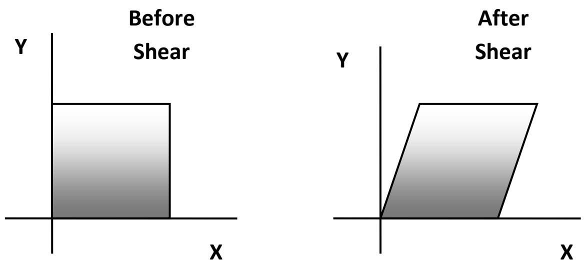


Fig. 3.13: - Shear in x-direction.

- Shear relative to $x - axis$ that is $y = 0$ line can be produced by following equation:
  $\boldsymbol{x' = x + sh_x \cdot y}$, $\boldsymbol{y' = y}$
- Transformation matrix for that is:
  $$\begin{bmatrix} \mathbf{1} & \boldsymbol{sh_x} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$
  Here $\boldsymbol{sh_x}$ is shear parameter. We can assign any real value to $\boldsymbol{sh_x}$.
- We can generate $x - direction$ shear relative to other reference line $y = y_{ref}$ with following equation:
  $\boldsymbol{x' = x + sh_x \cdot (y - y_{ref})}$, $\boldsymbol{y' = y}$
- Transformation matrix for that is:
  $$\begin{bmatrix} \mathbf{1} & \boldsymbol{sh_x} & \boldsymbol{-sh_x \cdot y_{ref}} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$
- **Example**: - Shear the unit square in x direction with shear parameter ½ relative to line $y = -1$.
  Here $y_{ref} = -1$ and $sh_x = 0.5$
  Coordinates of unit square are [A (0, 0), B (1, 0), C (1, 1), D (0, 1)].
  $$P' = \begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$
  $$P' = \begin{bmatrix} 1 & 0.5 & -0.5 \cdot (-1) \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$
  $$P' = \begin{bmatrix} 1 & 0.5 & 0.5 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$
  $$P' = \begin{bmatrix} 0.5 & 1.5 & 2 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$
- Final coordinate after shear are [A′ (0.5, 0), B′ (1.5, 0), C′ (2, 1), D′ (1, 1)]

<u>Shear in $\boldsymbol{y - direction}$.</u>

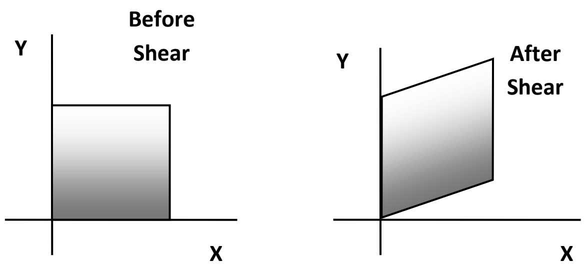


Fig. 3.14: - Shear in y-direction.

- Shear relative to $y - axis$ that is $x = 0$ line can be produced by following equation:
  $\boldsymbol{x' = x}$, $\boldsymbol{y' = y + sh_y \cdot x}$
- Transformation matrix for that is:
  $$\begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{sh_y} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$
  Here $\boldsymbol{sh_y}$ is shear parameter. We can assign any real value to $\boldsymbol{sh_y}$.

- We can generate $y - direction$ shear relative to other reference line $x = x_{ref}$ with following equation:
  $\boldsymbol{x' = x,} \qquad \boldsymbol{y' = y + sh_y \cdot (x - x_{ref})}$
- Transformation matrix for that is:
  $$\begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{sh_y} & \boldsymbol{1} & \boldsymbol{-sh_y \cdot x_{ref}} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}$$
- **Example**: - Shear the unit square in y direction with shear parameter ½ relative to line $x = -1$.
  Here $x_{ref} = -1$ and $sh_y = 0.5$
  Coordinates of unit square are [A (0, 0), B (1, 0), C (1, 1), D (0, 1)].
  $$P' = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$
  $$P' = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & -0.5 \cdot (-1) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$
  $$P' = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$
  $$P' = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0.5 & 1 & 2 & 1.5 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$
- Final coordinate after shear are [A' (0, 0.5), B' (1, 1), C' (1, 2), D' (0, 1.5)]

The Viewing Pipeline

- **Window:** Area selected in world-coordinate for display is called window. It defines what is to be viewed.
- **Viewport:** Area on a display device in which window image is display (mapped) is called viewport. It defines where to display.
- In many case window and viewport are rectangle, also other shape may be used as window and viewport.
- In general finding device coordinates of viewport from word coordinates of window is called as **viewing transformation.**
- Sometimes we consider this viewing transformation as window-to-viewport transformation but in general it involves more steps.

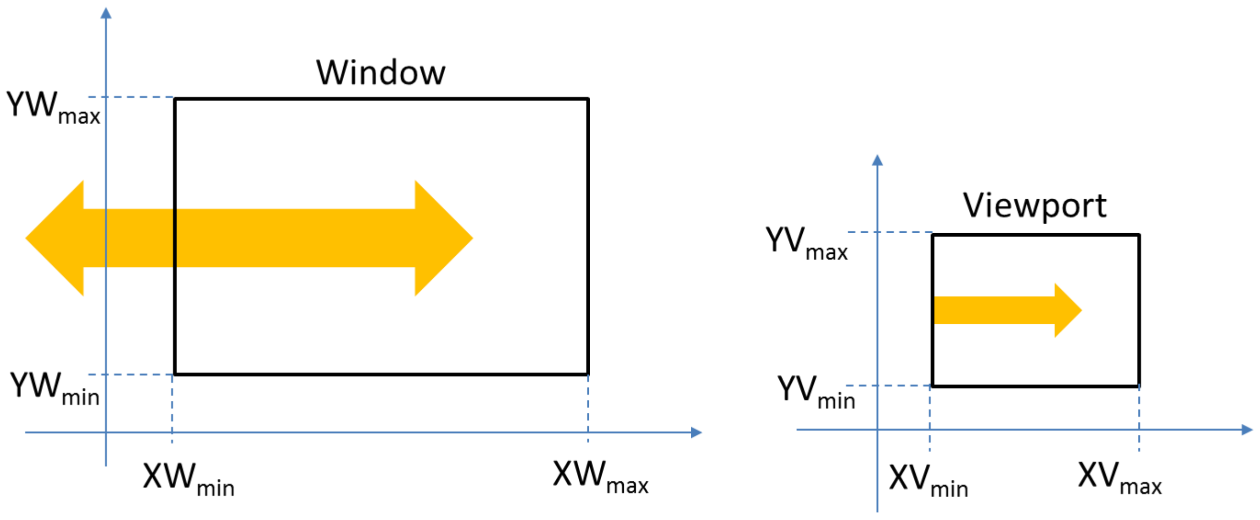


Fig. 3.1: - A viewing transformation using standard rectangles for the window and viewport.

- Now we see steps involved in viewing pipeline.

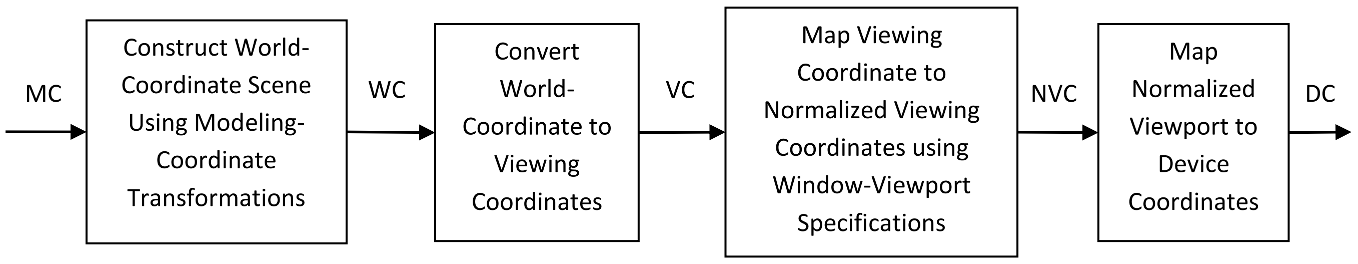

MC
Construct World-Coordinate Scene Using Modeling-Coordinate Transformations
WC
Convert World-Coordinate to Viewing Coordinates
VC
Map Viewing Coordinate to Normalized Viewing Coordinates using Window-Viewport Specifications
NVC
Map Normalized Viewport to Device Coordinates
DC

Fig. 3.2: - 2D viewing pipeline.

- As shown in figure above first of all we construct world coordinate scene using modeling coordinate transformation.
- After this we convert viewing coordinates from world coordinates using window to viewport transformation.
- Then we map viewing coordinate to normalized viewing coordinate in which we obtain values in between 0 to 1.
- At last we convert normalized viewing coordinate to device coordinate using device driver software which provide device specification.
- Finally device coordinate is used to display image on display screen.
- By changing the viewport position on screen we can see image at different place on the screen.
- By changing the size of the window and viewport we can obtain zoom in and zoom out effect as per requirement.
- Fixed size viewport and small size window gives zoom in effect, and fixed size viewport and larger window gives zoom out effect.
- View ports are generally defines with the unit square so that graphics package are more device independent which we call as normalized viewing coordinate.

Viewing Coordinate Reference Frame

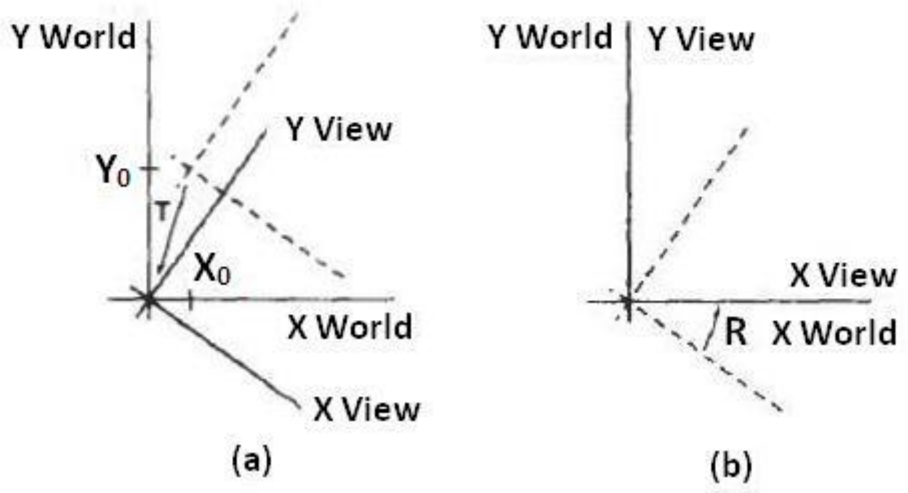


Fig. 3.3: - A viewing-coordinate frame is moved into coincidence with the world frame in two steps: (a) translate the viewing origin to the world origin, and then (b) rotate to align the axes of the two systems.

- We can obtain reference frame in any direction and at any position.
- For handling such condition first of all we translate reference frame origin to standard reference frame origin and then we rotate it to align it to standard axis.
- In this way we can adjust window in any reference frame.
- this is illustrate by following transformation matrix:

$\mathbf{M_{wc,vc} = RT}$

- Where T is translation matrix and R is rotation matrix.

Window-To-Viewport Coordinate Transformation

- Mapping of window coordinate to viewport is called window to viewport transformation.
- We do this using transformation that maintains relative position of window coordinate into viewport.
- That means center coordinates in window must be remains at center position in viewport.
- We find relative position by equation as follow:

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

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

- Solving by making viewport position as subject we obtain:

$$\mathbf{x_v = x_{vmin} + (x_w - x_{wmin})s_x}$$
$$\mathbf{y_v = y_{vmin} + (y_w - y_{wmin})s_y}$$

- Where scaling factor are :

$$\mathbf{s_x = \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}}$$
$$\mathbf{s_y = \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}}}$$

- We can also map window to viewport with the set of transformation, which include following sequence of transformations:
  1. Perform a scaling transformation using a fixed-point position of (x_{Wmin},y_{wmin}) that scales the window area to the size of the viewport.
  2. Translate the scaled window area to the position of the viewport.
- For maintaining relative proportions we take ($s_x = s_y$). in case if both are not equal then we get stretched or contracted in either the x or y direction when displayed on the output device.
- Characters are handle in two different way one way is simply maintain relative position like other primitive and other is to maintain standard character size even though viewport size is enlarged or reduce.
- Number of display device can be used in application and for each we can use different window-to-viewport transformation. This mapping is called the **workstation transformation.**

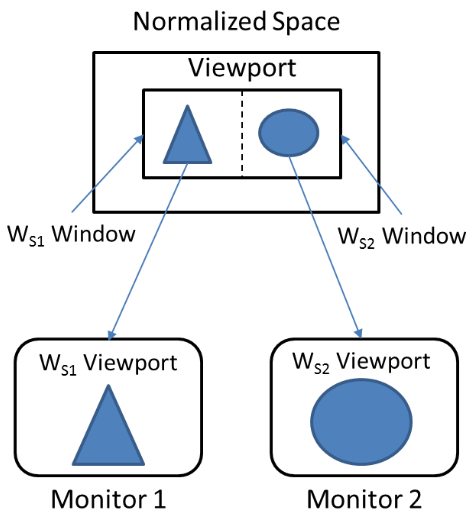


Fig. 3.4: - workstation transformation.

- As shown in figure two different displays devices are used and we map different window-to-viewport on each one.

Clipping Operations

- Generally, any procedure that identifies those portions of a picture that are either inside or outside of a specified region of space is referred to as a **clipping algorithm**, or simply **clipping**. The region against which an object is to clip is called a **clip window.**
- Clip window can be general polygon or it can be curved boundary.

Application of Clipping

- It can be used for displaying particular part of the picture on display screen.
- Identifying visible surface in 3D views.
- Antialiasing.
- Creating objects using solid-modeling procedures.
- Displaying multiple windows on same screen.
- Drawing and painting.

Point Clipping

- In point clipping we eliminate those points which are outside the clipping window and draw points which are inside the clipping window.
- Here we consider clipping window is rectangular boundary with edge (x_{wmin},x_{wmax},y_{wmin},y_{wmax}).
- So for finding wether given point is inside or outside the clipping window we use following inequality:
  $\boldsymbol{x_{wmin} \le x \le x_{wamx}}$
  $\boldsymbol{y_{wmin} \le y \le y_{wamx}}$
- If above both inequality is satisfied then the point is inside otherwise the point is outside the clipping window.

Line Clipping

- Line clipping involves several possible cases.
  1. Completely inside the clipping window.
  2. Completely outside the clipping window.
  3. Partially inside and partially outside the clipping window.

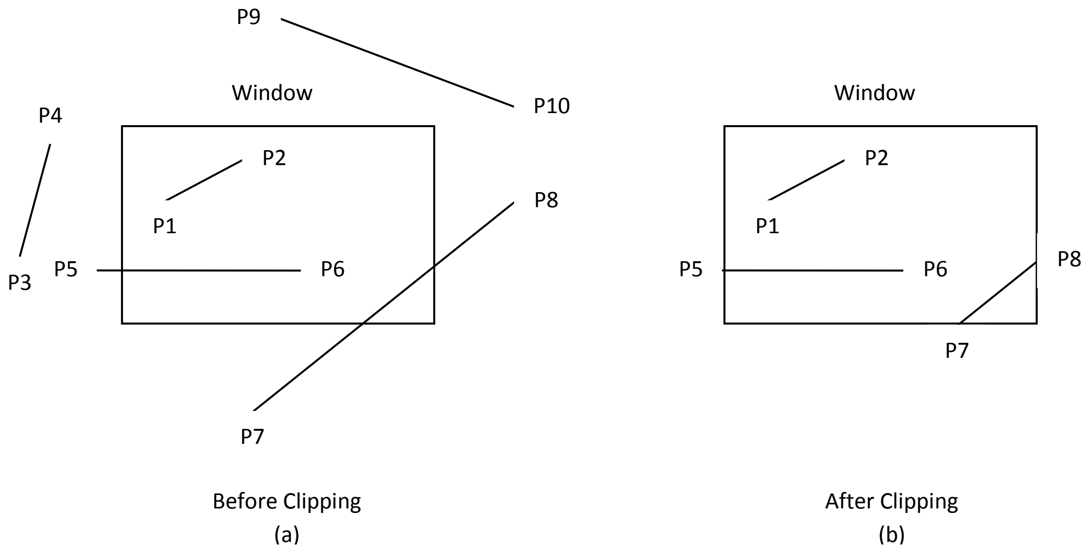


Fig. 3.5: - Line clipping against a rectangular window.

- Line which is completely inside is display completely. Line which is completely outside is eliminated from display. And for partially inside line we need to calculate intersection with window boundary and find which part is inside the clipping boundary and which part is eliminated.
- For line clipping several scientists tried different methods to solve this clipping procedure. Some of them are discuss below.

Cohen-Sutherland Line Clipping

- This is one of the oldest and most popular line-clipping procedures.

Region and Region Code

- In this we divide whole space into nine region and assign 4 bit code to each endpoint of line depending on the position where the line endpoint is located.

1001	1000	1010
0001	0000	0010
0101	0100	0110

Fig. 3.6: - Workstation transformation.

- Figure 3.6 shows code for line end point which is fall within particular area.
- Code is deriving by setting particular bit according to position of area.
  Set bit 1: For left side of clipping window.
  Set bit 2: For right side of clipping window.
  Set bit 3: For below clipping window.
  Set bit 4: For above clipping window.
- All bits as mention above are set means 1 and other are 0.

Algorithm

Step-1:

Assign region code to both endpoint of a line depending on the position where the line endpoint is located.

Step-2:

```
If both endpoint have code ‘0000’
        Then line is completely inside.
Otherwise
        Perform logical ending between this two codes.

        If result of logical ending is non-zero
                Line is completely outside the clipping window.
        Otherwise
                Calculate the intersection point with the boundary one by one.
                Divide the line into two parts from intersection point.
                Recursively call algorithm for both line segments.
```

Step-3:

Draw line segment which are completely inside and eliminate other line segment which found completely outside.

Intersection points calculation with clipping window boundary

- For intersection calculation we use line equation "$y = mx + b$".
- 'x' is constant for left and right boundary which is:
  - for left "$x = x_{wmin}$"
  - for right "$x = x_{wmax}$"
- So we calculate y coordinate of intersection for this boundary by putting values of x depending on boundary is left or right in below equation.
  $\boldsymbol{y = y_1 + m(x - x_1)}$
- 'y' coordinate is constant for top and bottom boundary which is:
  - for top "$y = y_{wmax}$"
  - for bottom "$y = y_{wmin}$"
- So we calculate x coordinate of intersection for this boundary by putting values of y depending on boundary is top or bottom in below equation.
  $\boldsymbol{x = x_1 + \frac{y - y_1}{m}}$

Liang-Barsky Line Clipping

- Line clipping approach is given by the Liang and Barsky is faster than cohen-sutherland line clipping. Which is based on analysis of the parametric equation of the line which are as below.
  $\boldsymbol{x = x_1 + u\Delta x}$
  $\boldsymbol{y = y_1 + u\Delta y}$
  Where $0 \leq u \leq 1$,$\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$.

Algorithm

```
1. Read two end points of line $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$
2. Read two corner vertices, left top and right bottom of window: $(x_{wmin}, y_{wmax})$ and $(x_{wmax}, y_{wmin})$
3. Calculate values of parameters $p_k$ and $q_k$ for $k = 1, 2, 3, 4$ such that,
   $p_1 = -\Delta x,$      $q_1 = x_1 - x_{wmin}$
   $p_2 = \Delta x,$       $q_2 = x_{wmax} - x_1$
   $p_3 = -\Delta y,$      $q_3 = y_1 - y_{wmin}$
   $p_4 = \Delta y,$       $q_4 = y_{wmax} - y_1$
4. If $p_k = 0$ for any value of $k = 1, 2, 3, 4$ then,
        Line is parallel to $k^{th}$ boundary.

        If corresponding $q_k < 0$ then,
            Line is completely outside the boundary. Therefore, discard line segment and Go to Step 10.
        Otherwise
            Check line is horizontal or vertical and accordingly check line end points with corresponding boundaries.

            If line endpoints lie within the bounded area
                Then use them to draw line.
            Otherwise
            Use boundary coordinates to draw line. And go to Step 8.
5.  For $k = 1,2,3,4$ calculate $r_k$ for nonzero values of $p_k$ and $q_k$ as follows:
    $r_k = \frac{q_k}{p_k}, for\ k = 1,2,3,4$
6.  Find $u_1\ and\ u_2$ as given below:
    $u_1 = \max\{0, r_k | where\ k\ takes\ all\ values\ for\ which\ p_k < 0\}$
    $u_2 = \min\{1, r_k | where\ k\ takes\ all\ values\ for\ which\ p_k > 0\}$
7.  If $u_1 \le u_2$ then
        Calculate endpoints of clipped line:
        $x_1' = x_1 + u_1 \Delta x$
        $y_1' = y_1 + u_1 \Delta y$
        $x_2' = x_1 + u_2 \Delta x$
        $y_2' = y_1 + u_2 \Delta y$
        Draw line $(x_1', y_1', x_2', y_2')$;
8.  Stop.
```

Advantages

1. More efficient.
2. Only requires one division to update u_1 and u_2.
3. Window intersections of line are calculated just once.

Nicholl-Lee-Nicholl Line Clipping

- By creating more regions around the clip window the NLN algorithm avoids multiple clipping of an individual line segment.
- In Cohen-Sutherlan line clipping sometimes multiple calculation of intersection point of a line is done before actual window boundary intersection or line is completely rejected.
- These multiple intersection calculation is avoided in NLN line clipping procedure.
- NLN line clipping perform the fewer comparisons and divisions so it is more efficient.
- But NLN line clipping cannot be extended for three dimensions while Cohen-Sutherland and Liang-Barsky algorithm can be easily extended for three dimensions.
- For given line we find first point falls in which region out of nine region shown in figure below but three region shown in figure by putting point are only considered and if point falls in other region than we transfer that point in one of the three region.

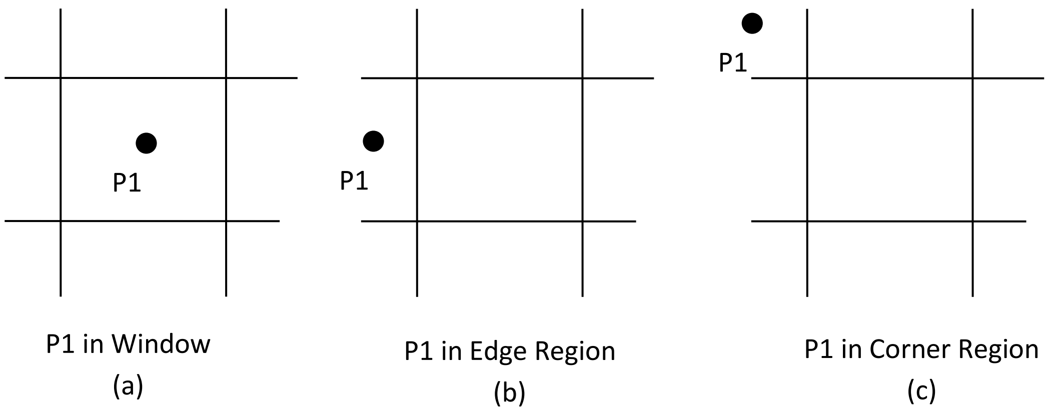


Fig. 3.7: - Three possible position for a line endpoint p1 in the NLN line-clipping algorithm.

- We can also extend this procedure for all nine regions.
- Now for p1 is inside the window we divide whole area in following region:

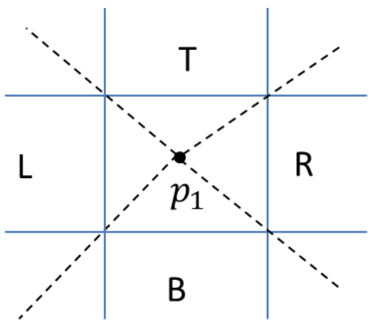


Fig. 3.8: - Clipping region when p1 is inside the window.

- Now for p1 is in edge region we divide whole area in following region:

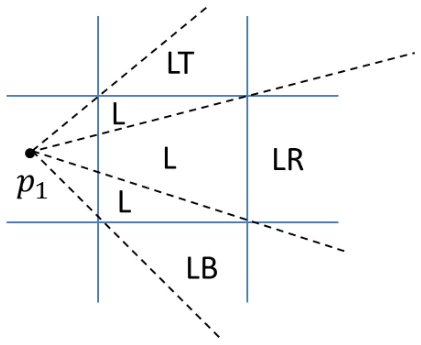


Fig. 3.9: - Clipping region when p1 is in edge region.

- Now for p1 is in corner region we divide whole area in following region:

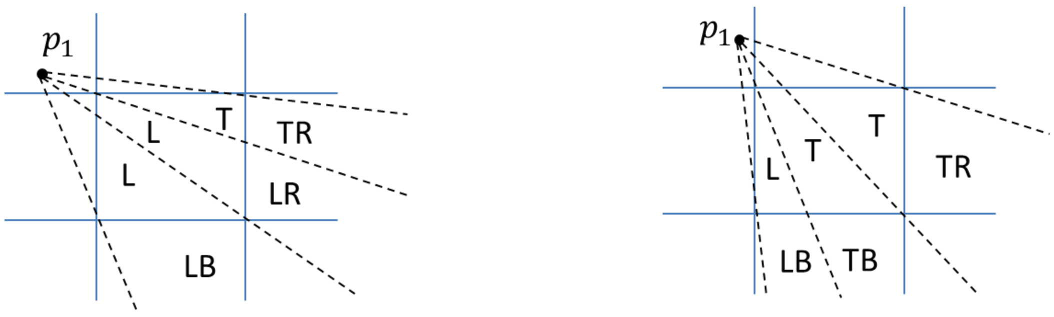


Fig. 3.10: - Two possible sets of clipping region when p1 is in corner region.

- Regions are name in such a way that name in which region p2 falls is gives the window edge which intersects the line.
- For example region LT says that line need to clip at left and top boundary.
- For finding that in which region line $\boldsymbol{p_1p_2}$ falls we compare the slope of the line to the slope of the boundaries:

  $\boldsymbol{slope\ \overline{p_1p_{B1}}} < \boldsymbol{slope\ \overline{p_1p_2}} < slope\ \overline{\boldsymbol{p_1p_{B2}}}$

  Where $\overline{\boldsymbol{p_1p_{B1}}}$ and $\overline{\boldsymbol{p_1p_{B2}}}$ are boundary lines.
- For example p1 is in edge region and for checking whether p2 is in region LT we use following equation.

  $\boldsymbol{slope\ \overline{p_1p_{TR}}} < \boldsymbol{slope\ \overline{p_1p_2}} < slope\ \overline{\boldsymbol{p_1p_{TL}}}$

  $$\frac{\boldsymbol{y_T - y_1}}{\boldsymbol{x_R - x_1}} < \frac{\boldsymbol{y_2 - y_1}}{\boldsymbol{x_2 - x_1}} < \frac{\boldsymbol{y_T - y_1}}{\boldsymbol{x_L - x_1}}$$
- After checking slope condition we need to check weather it crossing zero, one or two edges.
- This can be done by comparing coordinates of p_2 with coordinates of window boundary.
- For left and right boundary we compare x coordinates and for top and bottom boundary we compare y coordinates.
- If line is not fall in any defined region than clip entire line.

- Otherwise calculate intersection.
- After finding region we calculate intersection point using parametric equation which are:
- $\boldsymbol{x = x_1 + (x_2 - x_1)u}$
- $\boldsymbol{y = y_1 + (y_2 - y_1)u}$
- For left or right boundary $x = x_l \ or \ x_r$ respectively, with $u = (x_{l/r} - x1)/\ (x2 - x1)$, so that y can be obtain from parametric equation as below:
- $\boldsymbol{y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x_L - x_1)}$
- Keep the portion which is inside and clip the rest.

Polygon Clipping

- For polygon clipping we need to modify the line clipping procedure because in line clipping we need to consider about only line segment while in polygon clipping we need to consider the area and the new boundary of the polygon after clipping.

Sutherland-Hodgeman Polygon Clipping

- For correctly clip a polygon we process the polygon boundary as a whole against each window edge.
- This is done by whole polygon vertices against each clip rectangle boundary one by one.
- Beginning with the initial set of polygon vertices we first clip against the left boundary and produce new sequence of vertices.
- Then that new set of vertices is clipped against the right boundary clipper, a bottom boundary clipper and a top boundary clipper, as shown in figure below.

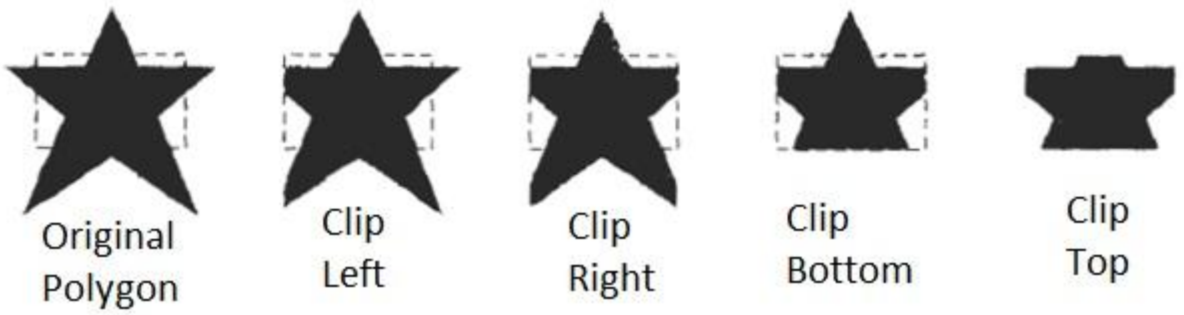


Fig. 3.11: - Clipping a polygon against successive window boundaries.

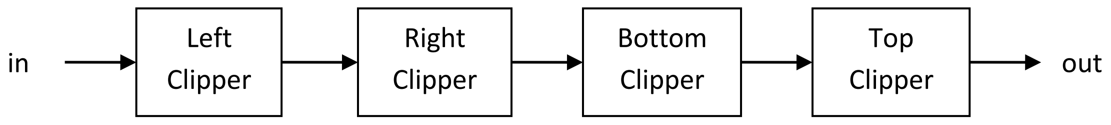


Fig. 3.12: - Processing the vertices of the polygon through boundary clipper.

- There are four possible cases when processing vertices in sequence around the perimeter of a polygon.

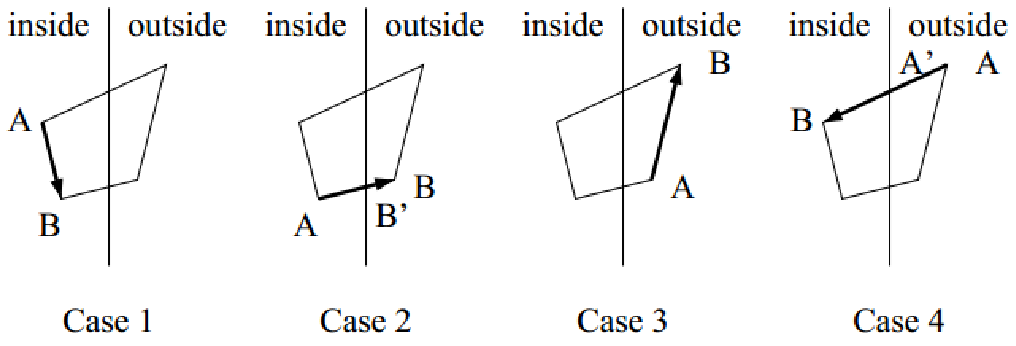


Fig. 3.13: - Clipping a polygon against successive window boundaries.

- As shown in case 1: if both vertices are inside the window we add only second vertices to output list.
- In case 2: if first vertices is inside the boundary and second vertices is outside the boundary only the edge intersection with the window boundary is added to the output vertex list.
- In case 3: if both vertices are outside the window boundary nothing is added to window boundary.
- In case 4: first vertex is outside and second vertex is inside the boundary, then adds both intersection point with window boundary, and second vertex to the output list.
- When polygon clipping is done against one boundary then we clip against next window boundary.
- We illustrate this method by simple example.

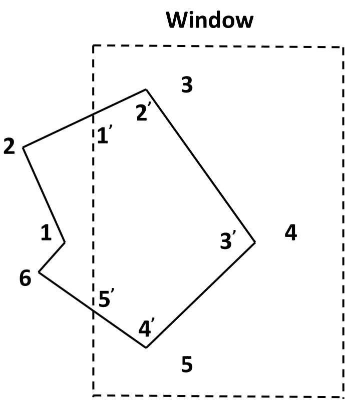


Fig. 3.14: - Clipping a polygon against left window boundaries.

- As shown in figure above we clip against left boundary vertices 1 and 2 are found to be on the outside of the boundary. Then we move to vertex 3, which is inside, we calculate the intersection and add both intersection point and vertex 3 to output list.
- Then we move to vertex 4 in which vertex 3 and 4 both are inside so we add vertex 4 to output list, similarly from 4 to 5 we add 5 to output list, then from 5 to 6 we move inside to outside so we add intersection pint to output list and finally 6 to 1 both vertex are outside the window so we does not add anything.
- Convex polygons are correctly clipped by the Sutherland-Hodgeman algorithm but concave polygons may be displayed with extraneous lines.
- For overcome this problem we have one possible solution is to divide polygon into numbers of small convex polygon and then process one by one.
- Another approach is to use Weiler-Atherton algorithm.

Weiler-Atherton Polygon Clipping

- In this algorithm vertex processing procedure for window boundary is modified so that concave polygon also clip correctly.
- This can be applied for arbitrary polygon clipping regions as it is developed for visible surface identification.
- Main idea of this algorithm is instead of always proceeding around the polygon edges as vertices are processed we sometimes need to follow the window boundaries.
- Other procedure is similar to Sutherland-Hodgeman algorithm.
- For clockwise processing of polygon vertices we use the following rules:
    - For an outside to inside pair of vertices, follow the polygon boundary.
    - For an inside to outside pair of vertices, follow the window boundary in a clockwise direction.
- We illustrate it with example:

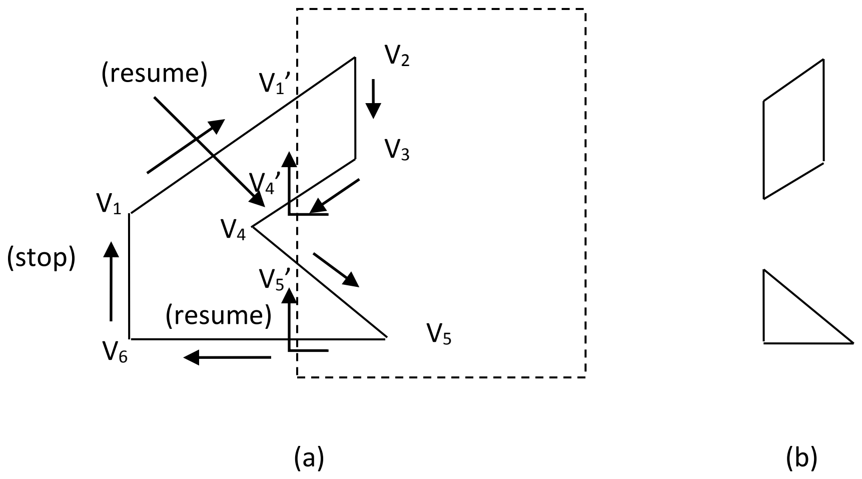


Fig. 3.14: - Clipping a concave polygon (a) with the Weiler-Atherton algorithm generates the two se

- As shown in figure we start from v1 and move clockwise towards v2 and add intersection point and next point to output list by following polygon boundary, then from v2 to v3 we add v3 to output list.
- From v3 to v4 we calculate intersection point and add to output list and follow window boundary.
- Similarly from v4 to v5 we add intersection point and next point and follow the polygon boundary, next we move v5 to v6 and add intersection point and follow the window boundary, and finally v6 to v1 is outside so no need to add anything.
- This way we get two separate polygon section after clipping.

Chapter 3 D Modeling

Three Dimensional Display Methods

Parallel Projection

- This method generates view from solid object by projecting parallel lines onto the display plane.
- By changing viewing position we can get different views of 3D object onto 2D display screen.

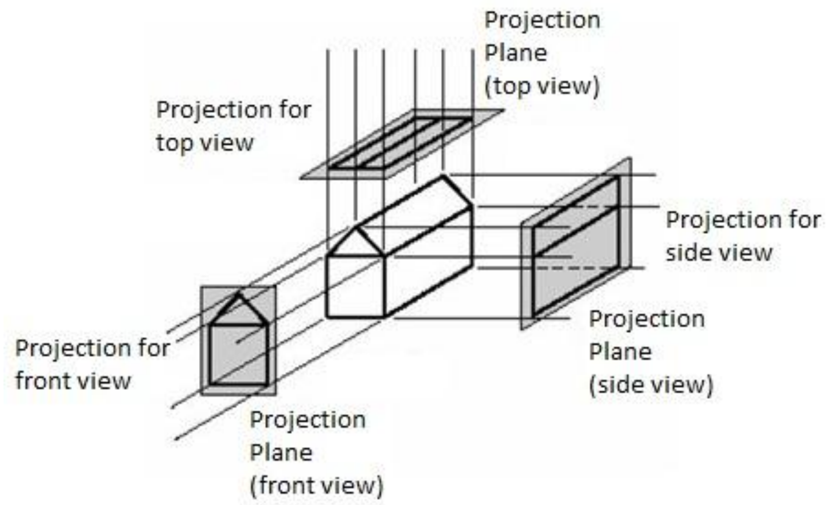

Projection Plane (top view)
Projection for top view
Projection for side view
Projection Plane (side view)
Projection for front view
Projection Plane (front view)

Fig. 4.1: - different views object by changing viewing plane position.

- Above figure shows different views of objects.
- This technique is used in Engineering & Architecture drawing to represent an object with a set of views that maintain relative properties of the object e.g.:- orthographic projection.

Perspective projection

- This method generating view of 3D object by projecting point on the display plane along converging paths.

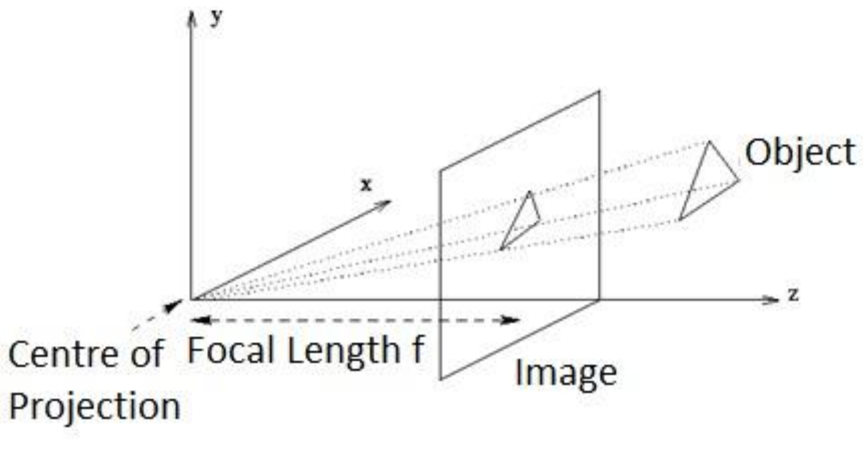


Fig. 4.2: - perspective projection

- This will display object smaller when it is away from the view plane and of nearly same size when closer to view plane.
- It will produce more realistic view as it is the way our eye is forming image.

Depth cueing

- Many times depth information is important so that we can identify for a particular viewing direction which are the front surfaces and which are the back surfaces of display object.

- Simple method to do this is depth cueing in which assign higher intensity to closer object & lower intensity to the far objects.
- Depth cuing is applied by choosing maximum and minimum intensity values and a range of distance over which the intensities are to vary.
- Another application is to modeling effect of atmosphere.

Visible line and surface Identification

- In this method we first identify visible lines or surfaces by some method.
- Then display visible lines with highlighting or with some different color.
- Other way is to display hidden lines with dashed lines or simply not display hidden lines.
- But not drawing hidden lines will loss the some information.
- Similar method we can apply for the surface also by displaying shaded surface or color surface.
- Some visible surface algorithm establishes visibility pixel by pixel across the view plane. Other determines visibility of object surface as a whole.

Surface Rendering

- More realistic image is produce by setting surface intensity according to light reflect from that surface & the characteristics of that surface.
- It will give more intensity to the shiny surface and less to dull surface.
- It also applies high intensity where light is more & less where light falls is less.

Exploded and Cutaway views

- Many times internal structure of the object is need to store. For ex., in machine drawing internal assembly is important.
- For displaying such views it will remove (cutaway) upper cover of body so that internal part's can be visible.

Three dimensional stereoscopic views

- This method display using computer generated scenes.
- It may display object by three dimensional views.
- The graphics monitor which are display three dimensional scenes are devised using a technique that reflects a CRT image from a vibrating flexible mirror.

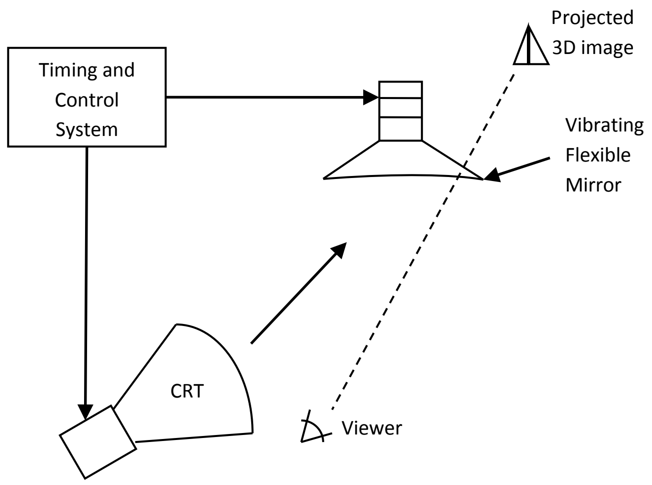


Fig. 4.3: - 3D display system uses a vibrating mirror.

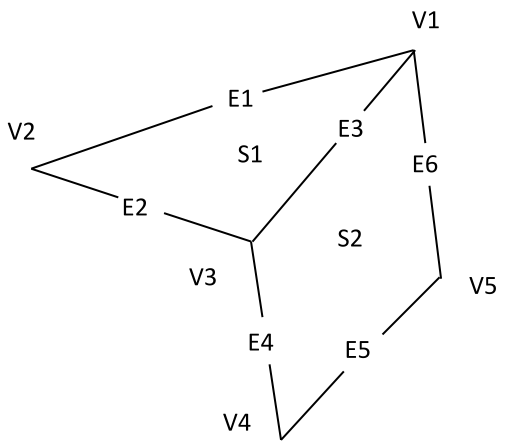
- Vibrating mirror changes its focal length due to vibration which is synchronized with the display of an object on CRT.
- The each point on the object is reflected from the mirror into spatial position corresponding to distance of that point from a viewing position.
- Very good example of this system is GENISCO SPACE GRAPH system, which use vibrating mirror to project 3D objects into a 25 cm by 25 cm by 25 cm volume. This system is also capable to show 2D cross section at different depth.
- Another way is stereoscopic views.
- Stereoscopic views does not produce three dimensional images, but it produce 3D effects by presenting different view to each eye of an observer so that it appears to have depth.
- To obtain this we first need to obtain two views of object generated from viewing direction corresponding to each eye.
- We can contract the two views as computer generated scenes with different viewing positions or we can use stereo camera pair to photograph some object or scene.
- When we see simultaneously both the view as left view with left eye and right view with right eye then two views is merge and produce image which appears to have depth.
- One way to produce stereoscopic effect is to display each of the two views with raster system on alternate refresh cycles.
- The screen is viewed through glasses with each lance design such a way that it act as a rapidly alternating shutter that is synchronized to block out one of the views.

Polygon Surfaces

- A polygonal surface can be thought of as a surface composed of polygonal faces.
- The most commonly used boundary representation for a three dimensional object is a set of polygon surfaces that enclose the object interior

Polygon Tables

- Representation of vertex coordinates, edges and other property of polygon into table form is called polygon table.
- Polygon data tables can be organized into two groups: geometric table and attributes table.
- Geometric table contains vertex coordinate and the other parameter which specify geometry of polygon.
- Attributes table stores other information like Color, transparency etc.
- Convenient way to represent geometric table into three different table namely vertex table, edge table, and polygon table.



Vertex Table	
V1:	X1, Y1, Z1
V2:	X2, Y2, Z2
V3:	X3, Y3, Z3
V4:	X4, Y4, Z4
V5:	X5, Y5, Z5

Edge Table	
E1:	V1, V2
E2:	V2, V3
E3:	V3, V1
E4:	V3, V4
E5:	V4, V5
E6:	V5, V1

Polygon Surface Table	
S1:	E1, E2, E3
S2:	E3, E4, E5, E6

Fig. 4.4: - Geometric Data Table representation.

- Vertex table stores each vertex included into polygon.
- Edge table stores each edge with the two endpoint vertex pointers back to vertex table.
- Polygon table stores each surface of polygon with edge pointer for the each edge of the surface.
- This three table representation stores each vertex one time only and similarly each edge is also one time. So it will avoid representation of storing common vertex and edge so it is memory efficient method.
- Another method to represent with two table vertex table and polygon table but it is inefficient as it will store common edge multiple times.
- Since tables have many entries for large number of polygon we need to check for consistency as it may be possible that errors may occurs during input.
- For dealing with that we add extra information into tables. For example figure below shows edge table of above example with information of the surface in which particular edge is present.

E1:	V1, V2, S1
E2:	V2, V3, S1
E3:	V3, V1, S1, S2
E4:	V3, V4, S2
E5:	V4, V5, S2
E6:	V5, V1, S2

Fig. 4.5: - Edge table of above example with extra information as surface pointer.

- Now if any surface entry in polygon table will find edge in edge table it will verify whether this edge is of particular surface's edge or not if not it will detect errors and may be correct if sufficient information is added.

Plane Equations

- For producing display of 3D objects we must process the input data representation for the object through several procedures.
- For this processing we sometimes need to find orientation and it can be obtained by vertex coordinate values and the equation of polygon plane.
- Equation of plane is given as
  $Ax + By + Cz + D = 0$
- Where (x, y, z) is any point on the plane and A, B, C, D are constants by solving three plane equation for three non collinear points. And solve simultaneous equation for ratio A/D, B/D, and C/D as follows
  $\frac{A}{D}x_1 + \frac{B}{D}y_1 + \frac{C}{D}z_1 = -1$
  $\frac{A}{D}x_2 + \frac{B}{D}y_2 + \frac{C}{D}z_2 = -1$
  $\frac{A}{D}x_3 + \frac{B}{D}y_3 + \frac{C}{D}z_3 = -1$
- Solving by determinant
  $A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \quad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix} \quad C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \quad D = -\begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$
- By expanding a determinant we get
  $A = y_1(z_2 - z_3) + y_2(z_3 - z_1) + y_3(z_1 - z_2)$
  $B = z_1(x_2 - x_3) + z_2(x_3 - x_1) + z_3(x_1 - x_2)$
  $C = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$
  $D = -x_1(y_2z_3 - y_3z_2) - x_2(y_3z_1 - y_1z_3) - x_3(y_1z_2 - y_2z_1)$
- This values of A, B, C, D are then store in polygon data structure with other polygon data.
- Orientation of plane is described with normal vector to the plane.

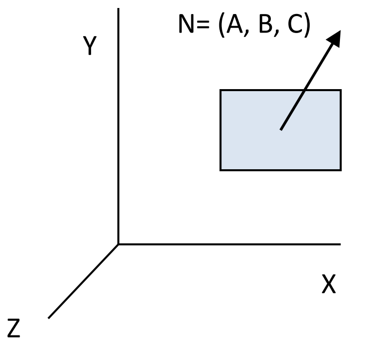


Fig. 4.6: - the vector N normal to the surface.

- Here N= (A,B,C) where A, B, C are the plane coefficient.
- When we are dealing with the polygon surfaces that enclose object interior we define the side of the faces towards object interior is as inside face and outward side as outside face.
- We can calculate normal vector N for any particular surface by cross product of two vectors in counter clockwise direction in right handed system then.
  $N = (v2 - v1)X(v3 - v1)$
- Now N gives values A, B, C for that plane and D can be obtained by putting these values in plane equation for one of the vertices and solving for D.
- Using plane equation in vector form we can obtain D as
  $N \cdot P = -D$
- Plane equation is also used to find position of any point compare to plane surface as follows

If $Ax + By + Cz + D \neq 0$ the point (x,y,z) is not on that plane.
If $Ax + By + Cz + D < 0$ the point (x,y,z) is inside the surface.
If $Ax + By + Cz + D > 0$ the point (x,y,z) is outside the surface.

- These equation are valid for right handed system provides plane parameter A, B, C, and D were calculated using vertices selected in a counter clockwise order when viewing the surface in an outside to inside direction.

Polygon Meshes

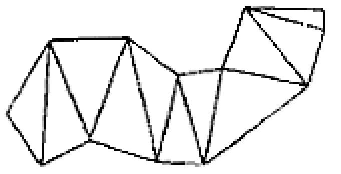


Fig. 4.7: -A triangle strip formed with 11 triangle connecting 13 vertices

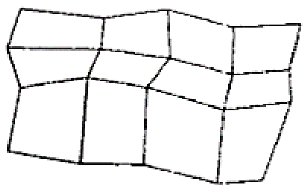


Fig. 4.8: -A quadrilateral mesh containing 12 quadrilaterals constructed from a 5 by 4 input vertex array

- Polygon mesh is a collection of edges, vertices and faces that defines the shape of the polyhedral object in 3D computer graphics and solid modeling.
- An edge can be shared by two or more polygons and vertex is shared by at least two edges.
- Polygon mesh is represented in following ways
  - Explicit representation
  - Pointer to vertex list
  - Pointer to edge list

Explicit Representation

- In explicit representation each polygon stores all the vertices in order in the memory as,
  $P = \left(\left((x_1, y_1, z_1), (x_2, y_2, z_2)\right), \dots, \left((x_m, y_m, z_m), (x_n, y_n, z_n)\right)\right)$
- It process fast but requires more memory for storing.

Pointer to Vertex list

- In this method each vertex stores in vertex list and then polygon contains pointer to the required vertex.
  $V = ((x_1, y_1, z_1), (x_2, y_2, z_2), \dots, (x_n, y_n, z_n))$
- And now polygon of vertices 3, 4, 5 is represented as $P = ((v_3, v_4), (v_4, v_5), (v_5, v_3))$.
- It is considerably space saving but common edges is difficult to find.

Pointer to Edge List

- In this polygon have pointers to the edge list and edge list have pointer to vertex list for each edge two vertex pointer is required which points in vertex list.
  $V = ((x_1, y_1, z_1), (x_2, y_2, z_2), \dots, (x_n, y_n, z_n))$
  $E = ((v_1, v_2), (v_2, v_3), \dots, (v_n, v_m))$
  $P = (E_1, E_2, E_n)$
- This approach is more memory efficient and easy to find common edges.

Spline Representations

- Spline is flexible strip used to produce a smooth curve through a designated set of points.
- Several small weights are attached to spline to hold in particular position.
- Spline curve is a curve drawn with this method.

- The term spline curve now referred to any composite curve formed with polynomial sections satisfying specified continuity condition at the boundary of the pieces.
- A spline surface can be described with two sets of orthogonal spline curves.

Interpolation and approximation splines

- We specify spline curve by giving a set of coordinate positions called control points. This indicates the general shape of the curve.
- **Interpolation Spline**: - When curve section passes through each control point, the curve is said to interpolate the set of control points and that spline is known as Interpolation Spline.

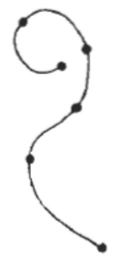


Fig. 4.9: -interpolation spline.

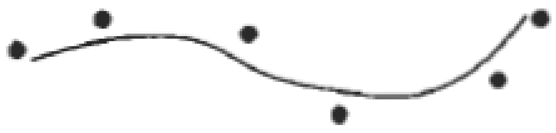


Fig. 4.10: -Approximation spline.

- **Approximation Spline**: - When curve section follows general control point path without necessarily passing through any control point, the resulting curve is said to approximate the set of control points and that curve is known as Approximation Spline.
- Spline curve can be modified by selecting different control point position.
- We can apply transformation on the curve according to need like translation scaling etc.
- The convex polygon boundary that encloses a set of control points is called **convex hull**.

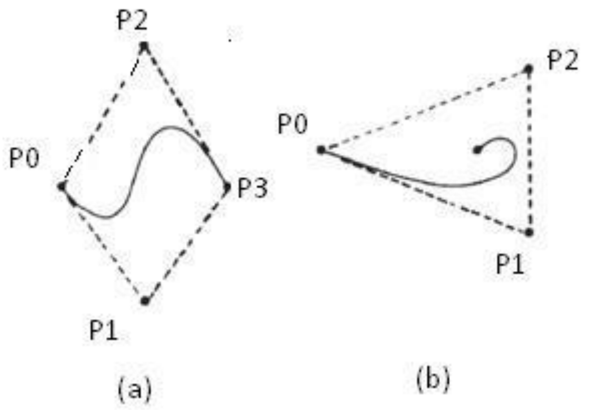


Fig. 4.11: -convex hull shapes for two sets of control points.

- A poly line connecting the sequence of control points for an approximation spline is usually displayed to remind a designer of the control point ordering. This set of connected line segment is often referred as control graph of the curve.
- Control graph is also referred as control polygon or characteristic polygon.

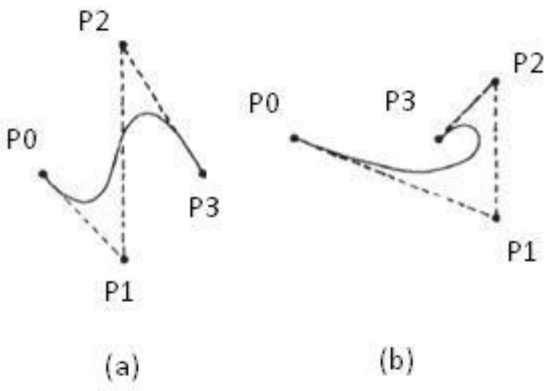


Fig. 4.12: -Control-graph shapes for two different sets of control points.

Parametric continuity condition

- For smooth transition from one curve section on to next curve section we put various continuity conditions at connection points.
- Let parametric coordinate functions as
  $x = x(u),\ y = y(u),\ z = z(u) \qquad\qquad \because u_1 \ll u \ll u_2$
- Then **zero order parametric continuity (c^0)** means simply curves meets i.e. last point of first curve section & first points of second curve section are same.
- **First order parametric continuity (c^1)** means first parametric derivatives are same for both curve section at intersection points.
- **Second order parametric continuity (c^2)** means both the first & second parametric derivative of two curve section are same at intersection.
- Higher order parametric continuity is can be obtain similarly.

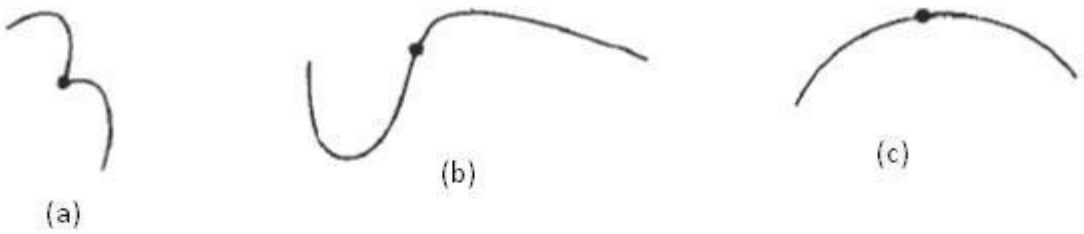


Fig. 4.13: - Piecewise construction of a curve by joining two curve segments uses different orders of continuity: (a) zero-order continuity only, (b) first-order continuity, and (c) second-order continuity.

- First order continuity is often sufficient for general application but some graphics package like cad requires second order continuity for accuracy.

Geometric continuity condition

- Another method for joining two successive curve sections is to specify condition for geometric continuity.
- **Zero order geometric continuity (g^0)** is same as parametric zero order continuity that two curve section meets.
- **First order geometric continuity (g^1)** means that the parametric first derivatives are proportional at the intersection of two successive sections but does not necessary Its magnitude will be equal.
- **Second order geometric continuity (g^2)** means that the both parametric first & second derivatives are proportional at the intersection of two successive sections but does not necessarily magnitude will be equal.

Cubic Spline Interpolation Methods

- Cubic splines are mostly used for representing path of moving object or existing object shape or drawing.
- Sometimes it also used for design the object shapes.
- Cubic spline gives reasonable computation on as compared to higher order spline and more stable compare to lower order polynomial spline. So it is often used for modeling curve shape.
- Cubic interpolation splines obtained by fitting the input points with piecewise cubic polynomial curve that passes through every control point.

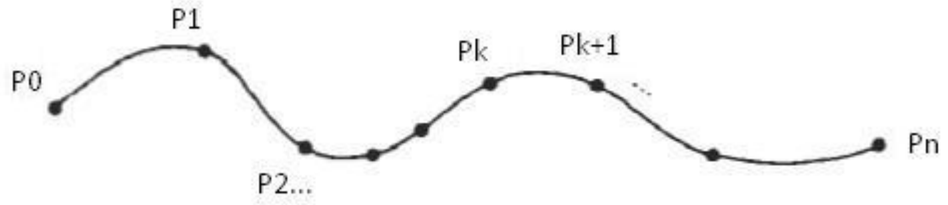


Fig. 4.14: -A piecewise continuous cubic-spline interpolation of n+1 control points.

$p_k = (x_k, y_k, z_k)$ Where, k=0, 1, 2, 3 ..., n

- Parametric cubic polynomial for this curve is given by
  $x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$
  $y(u) = a_y u^3 + b_y u^2 + c_y u + d_y$
  $z(u) = a_z u^3 + b_z u^2 + c_z u + d_z$
  $where(0 \leq u \leq 1)$
- For above equation we need to determine for constant a, b, c and d the polynomial representation for each of n curve section.
- This is obtained by settling proper boundary condition at the joints.
- Now we will see common method for settling this condition.

Natural Cubic Splines

- Natural cubic spline is a mathematical representation of the original drafting spline.
- We consider that curve is in c^2 continuity means first and second parametric derivatives of adjacent curve section are same at control point.
- For the ''n+1'' control point we have n curve section and 4n polynomial constants to find.
- For all interior control points we have four boundary conditions. The two curve section on either side of control point must have same first & second order derivative at the control points and each curve passes through that control points.
- We get other two condition as p_0 (first control points) starting & p_n(last control point) is end point of the curve.
- We still required two conditions for obtaining coefficient values.
- One approach is to setup second derivative at p_0 & p_n to be 0. Another approach is to add one extra dummy point at each end. I.e. we add p_{-1} & p_{n+1} then all original control points are interior and we get 4n boundary condition.
- Although it is mathematical model it has major disadvantage is with change in the control point entire curve is changed.
- So it is not allowed for local control and we cannot modify part of the curve.

Hermit Interpolation

- It is named after French mathematician Charles hermit

- It is an interpolating piecewise cubic polynomial with specified tangent at each control points.
- It is adjusted locally because each curve section is depends on it's end points only.
- Parametric cubic point function for any curve section is then given by:
  $p(0) = p_k$
  $p(1) = p_{k+1}$
  $p'(0) = dp_k$
  $p''(1) = dp_{k+1}$
  Where dp_k & dp_{k+1} are values of parametric derivatives at point p_k & p_{k+1} respectively.
- Vector equation of cubic spline is:
  $p(u) = au^3 + bu^2 + cu + d$
- Where x component of p is
- $x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$ and similarly y & z components
- Matrix form of above equation is
  $$P(u) = [u^3\ u^2\ u\ 1]\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$
- Now derivatives of p(u) is $p'(u)=3au^2+2bu+c+0$
- Matrix form of p'(u) is
  $$P'(u) = [3u^2\ 2u\ 1\ 0]\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$
- Now substitute end point value of u as 0 & 1 in above equation & combine all four parametric equations in matrix form:
  $$\begin{bmatrix} p_k \\ p_{k+1} \\ dp_k \\ dp_{k+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$
- Now solving it for polynomial co efficient
  $$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} p_k \\ p_{k+1} \\ dp_k \\ dp_{k+1} \end{bmatrix}$$
  $$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = M_H\begin{bmatrix} p_k \\ p_{k+1} \\ dp_k \\ dp_{k+1} \end{bmatrix}$$
- Now Put value of above equation in equation of $p(u)$
  $$p(u) = [u^3\ u^2\ u\ 1]\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} p_k \\ p_{k+1} \\ dp_k \\ dp_{k+1} \end{bmatrix}$$
  $$p(u) = [2u^3 - 3u^2 + 1 \quad -2u^3 + 3u^2 \quad u^3 - 2u^2 + u \quad u^3 - u^2]\begin{bmatrix} p_k \\ p_{k+1} \\ dp_k \\ dp_{k+1} \end{bmatrix}$$
  $p(u) = p_k(2u^3 - 3u^2 + 1) + p_{k+1}(-2u^3 + 3u^2) + dp_k(u^3 - 2u^2 + u) + dp_{k+1}(u^3 - u^2)$
  $p(u) = p_k H_0(\mathrm{u}) + p_{k+1} H_1(\mathrm{u}) + dp_k H_2(\mathrm{u}) + dp_{k+1} H_3(\mathrm{u})$
  Where $H_k(\mathrm{u})$ for k=0 , 1 , 2 , 3 are referred to as blending functions because that blend the boundary constraint values for curve section.

- Shape of the four hermit blending function is given below.

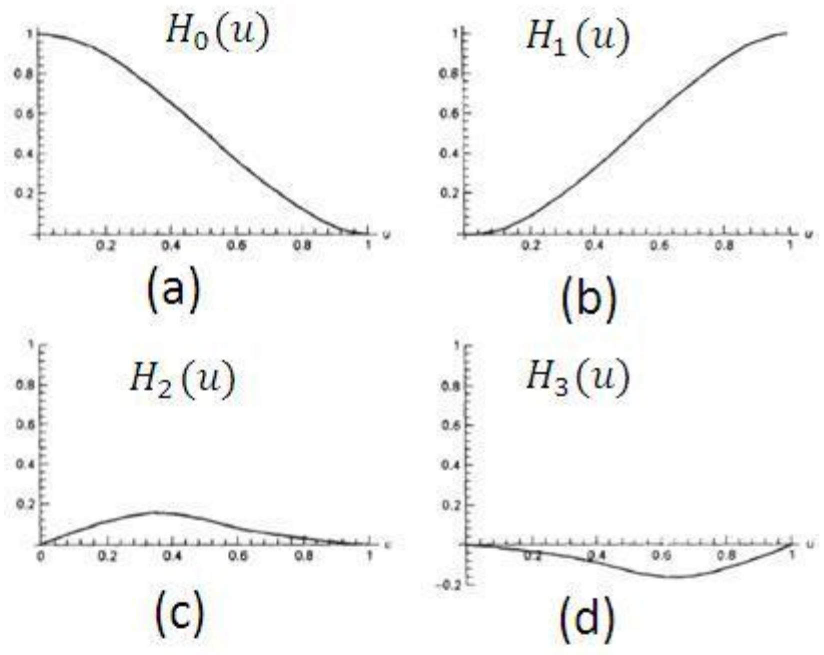


Fig. 4.15: -the hermit blending functions.

- Hermit curves are used in digitizing application where we input the approximate curve slope means DP_k & DP_{k+1}.
- But in application where this input is difficult to approximate at that place we cannot use hermit curve.

Cardinal Splines

- As like hermit spline cardinal splines also interpolating piecewise cubics with specified endpoint tangents at the boundary of each section.
- But in this spline we need not have to input the values of endpoint tangents.
- In cardinal spline values of slope at control point is calculated from two immediate neighbor control points.
- It's spline section is completely specified by the 4-control points.

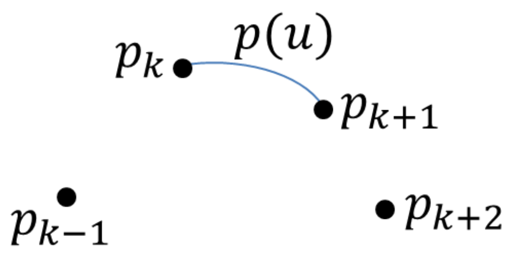


Fig. 4.16: -parametric point function p(u) for a cardinal spline section between control points p_k and p_{k+1}.

- The middle two are two endpoints of curve section and other two are used to calculate slope of endpoints.
- Now parametric equation for cardinal spline is:

$$p(0) = p_k$$
$$p(1) = p_{k+1}$$
$$p'(0) = \frac{1}{2}(1-t)(p_{k+1} - p_{k-1})$$
$$p'(1) = \frac{1}{2}(1-t)(p_{k+2} - p_k)$$

Where parameter t is called **tension** parameter since it controls how loosely or tightly the cardinal spline fit the control points.

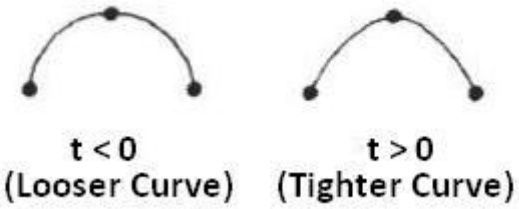


Fig. 4.17: -Effect of the tension parameter on the shape of a cardinal spline section.

- When t = 0 this class of curve is referred to as **catmull-rom spline** or **overhauser splines.**
- Using similar method like hermit we can obtain:

$$p(u) = [u^3 \quad u^2 \quad u \quad 1] \cdot M_c \cdot \begin{bmatrix} p_{k-1} \\ p_k \\ p_{k+1} \\ p_{k+2} \end{bmatrix}$$

- Where the cardinal matrix is

$$M_c = \begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

- With $s = (1-t)/2$
- Put value of M$_c$ in equation of p(u)

$$p(u) = [u^3 \quad u^2 \quad u \quad 1] \cdot \begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_{k-1} \\ p_k \\ p_{k+1} \\ p_{k+2} \end{bmatrix}$$

$$p(u) = [-su^3 + 2su^2 - su \quad (2-s)u^3 + (s-3)u^2 + 1 \quad (s-2)u^3 + (3-s)u^2 + su \quad su^3 - su^2] \cdot \begin{bmatrix} p_{k-1} \\ p_k \\ p_{k+1} \\ p_{k+2} \end{bmatrix}$$

$$p(u) = p_{k-1}(-su^3 + 2su^2 - su) + p_k\big((2-s)u^3 + (s-3)u^2 + 1\big) + p_{k+1}\big((s-2)u^3 + (3-s)u^2 + su\big) + p_{k+2}(su^3 - su^2)$$

$$p(u) = p_{k-1}CAR_0(u) + p_kCAR_1(u) + p_{k+1}CAR_2(u) + p_{k+2}CAR_3(u)$$

Where polynomial $CAR_k(u)\ for\ k = 0,1,2,3$ are the cardinals blending functions.

- Figure below shows this blending function shape for t = 0.

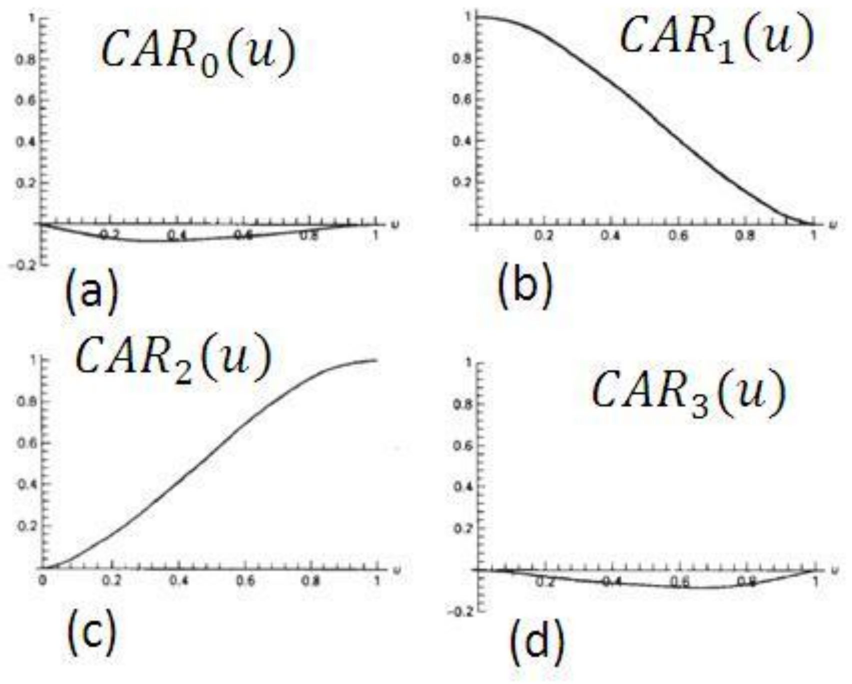


Fig. 4.18: -The cardinal blending function for t=0 and s=0.5.

Kochanek-Bartels spline

- It is extension of cardinal spline
- Two additional parameters are introduced into the constraint equation for defining kochanek-Bartels spline to provide more flexibility in adjusting the shape of curve section.
- For this parametric equations are as follows:

  $p(0) = p_k$

  $p(1) = p_{k+1}$

  $p'(0) = \frac{1}{2}(1-t)[(1+b)(1-c)(p_k - p_{k-1}) + (1-b)(1+c)(p_{k+1} - p_k)]$

  $p'(1) = \frac{1}{2}(1-t)[(1+b)(1+c)(p_{k+1} - p_k) + (1-b)(1-c)(p_{k+2} - p_{k+1})]$

  Where 't' is tension parameter same as used in cardinal spline.
- B is **bias parameter** and C is the **continuity parameter.**
- In this spline parametric derivatives may not be continuous across section boundaries.
- Bias B is used to adjust the amount that the curve bends at each end of section.

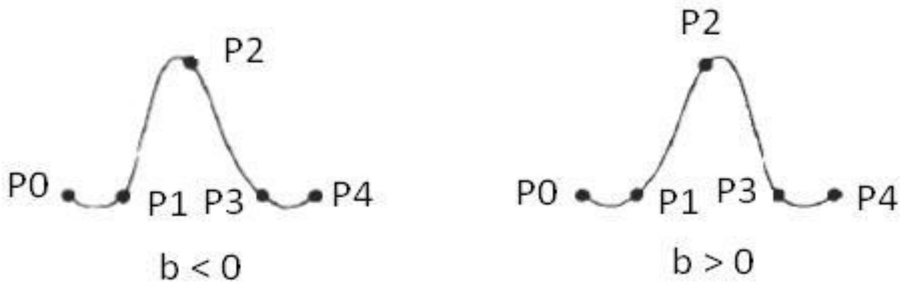


Fig. 4.19: -Effect of bias parameter on the shape of a Kochanek-Bartels spline section.

- Parameter c is used to controls continuity of the tangent vectors across the boundaries of section. If C is nonzero there is discontinuity in the slope of the curve across section boundaries.

- It is used in animation paths in particular abrupt change in motion which is simulated with nonzero values for parameter C.

Bezier Curves and Surfaces

- It is developed by French engineer Pierre Bezier for the Renault automobile bodies.
- It has number of properties and easy to implement so it is widely available in various CAD and graphics package.

Bezier Curves

- Bezier curve section can be fitted to any number of control points.
- Number of control points and their relative position gives degree of the Bezier polynomials.
- With the interpolation spline Bezier curve can be specified with boundary condition or blending function.
- Most convenient method is to specify Bezier curve with blending function.
- Consider we are given n+1 control point position from p_0 to p_n where $p_k = (x_k, y_k, z_k)$.
- This is blended to gives position vector p(u) which gives path of the approximate Bezier curve is:

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

  Where $BEZ_{k,n}(u) = C(n,k)u^k(1-u)^{n-k}$

  And $C(n,k) = {n!}/{k!\,(n-k)!}$
- We can also solve Bezier blending function by recursion as follow:

  $BEZ_{k,n}(u) = (1-u)BEZ_{k,n-1}(u) + uBEZ_{k-1,n-1}(u) \qquad n > k \ge 1$

  Here $BEZ_{k,k}(u) = u^k$ and $BEZ_{0,k}(u) = (1-u)^k$
- Parametric equation from vector equation can be obtain as follows.

  $$x(u) = \sum_{k=0}^{n} x_k BEZ_{k,n}(u)$$

  $$y(u) = \sum_{k=0}^{n} y_k BEZ_{k,n}(u)$$

  $$z(u) = \sum_{k=0}^{n} z_k BEZ_{k,n}(u)$$
- Bezier curve is a polynomial of degree one less than the number of control points.
- Below figure shows some possible curve shapes by selecting various control point.

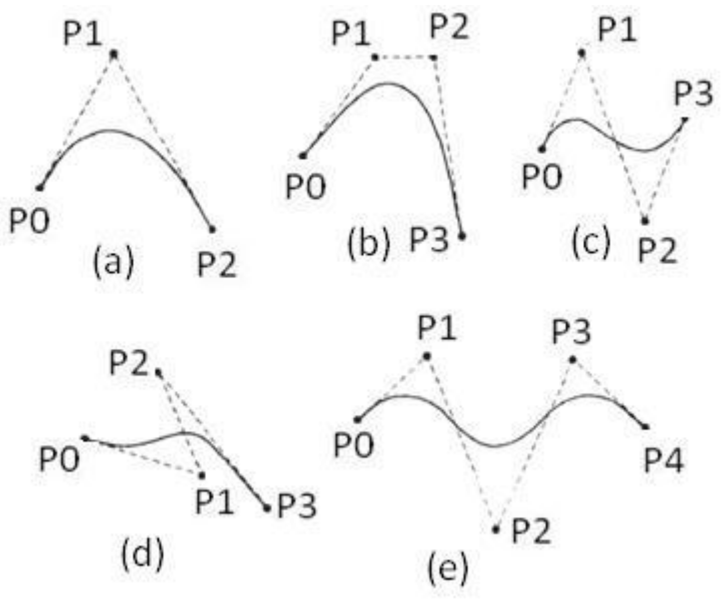


Fig. 4.20: -Example of 2D Bezier curves generated by different number of control points.

- Efficient method for determining coordinate positions along a Bezier curve can be set up using recursive calculation
- For example successive binomial coefficients can be calculated as

$$C(n,k) = \frac{n-k+1}{k} C(n,k-1) \qquad n \geq k$$

Properties of Bezier curves

- It always passes through first control point i.e. p(0) = p_0
- It always passes through last control point i.e. p(1) = p_n
- Parametric first order derivatives of a Bezier curve at the endpoints can be obtain from control point coordinates as:

$$p'(0) = -np_0 + np_1$$
$$p'(1) = -np_{n-1} + np_n$$

- Parametric second order derivatives of endpoints are also obtained by control point coordinates as:

$$p''(0) = n(n-1)[(p_2 - p_1) - (p_1 - p_0)]$$
$$p''(1) = n(n-1)[(p_{n-2} - p_{n-1}) - (p_{n-1} - p_n)]$$

- Bezier curve always lies within the convex hull of the control points.
- Bezier blending function is always positive.
- Sum of all Bezier blending function is always 1.

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

- So any curve position is simply the weighted sum of the control point positions.
- Bezier curve smoothly follows the control points without erratic oscillations.

Design Technique Using Bezier Curves

- For obtaining closed Bezier curve we specify first and last control point at same position.

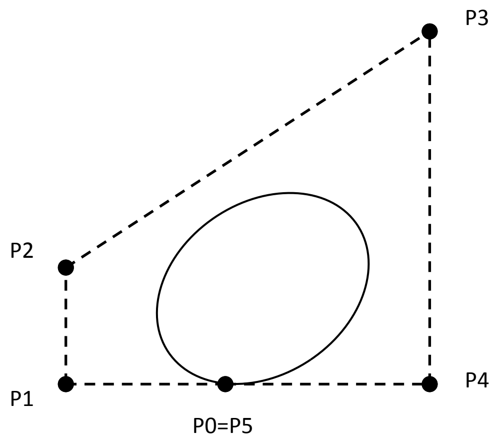


Fig. 4.21: -A closed Bezier Curve generated by specifying the first and last control points at the same location.

- If we specify multiple control point at same position it will get more weight and curve is pull towards that position.

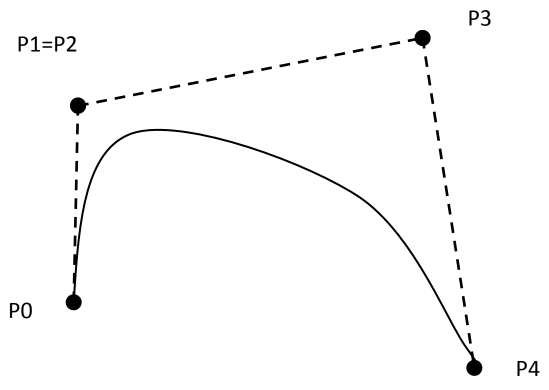


Fig. 4.22: -A Bezier curve can be made to pass closer to a given coordinate position by assigning multiple control point at that position.

- Bezier curve can be fitted for any number of control points but it requires higher order polynomial calculation.
- Complicated Bezier curve can be generated by dividing whole curve into several lower order polynomial curves. So we can get better control over the shape of small region.
- Since Bezier curve passes through first and last control point it is easy to join two curve sections with zero order parametric continuity (C^0).
- For first order continuity we put end point of first curve and start point of second curve at same position and last two points of first curve and first two point of second curve is collinear. And second control point of second curve is at position
  $p_n + (p_n - p_{n-1})$
- So that control points are equally spaced.

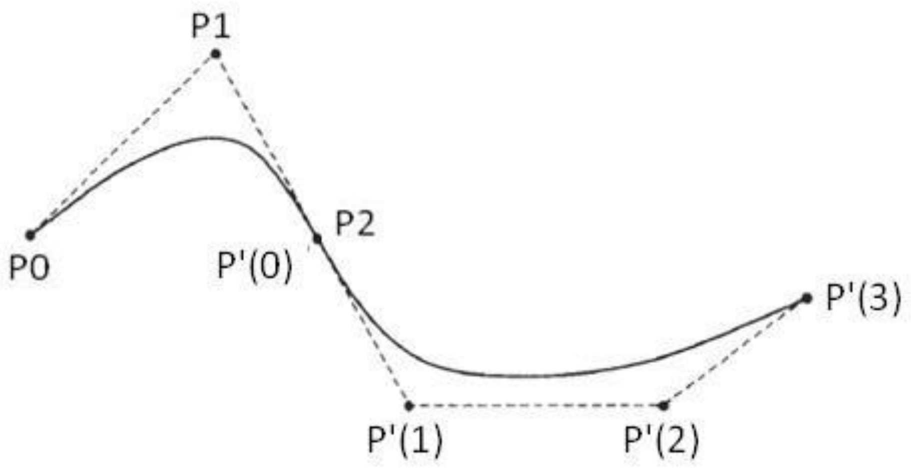


Fig. 4.23: -Zero and first order continuous curve by putting control point at proper place.

- Similarly for second order continuity the third control point of second curve in terms of position of the last three control points of first curve section as
  $p_{n-2} + 4(p_n - p_{n-1})$
- C^2 continuity can be unnecessary restrictive especially for cubic curve we left only one control point for adjust the shape of the curve.

Cubic Bezier Curves

- Many graphics package provides only cubic spline function because this gives reasonable design flexibility in average calculation.
- Cubic Bezier curves are generated using 4 control points.
- 4 blending function obtained by substituting n=3
  $BEZ_{0,3}(u) = (1-u)^3$
  $BEZ_{1,3}(u) = 3u(1-u)^2$
  $BEZ_{2,3}(u) = 3u^2(1-u)$
  $BEZ_{3,3}(u) = u^3$
- Plots of this Bezier blending function are shown in figure below

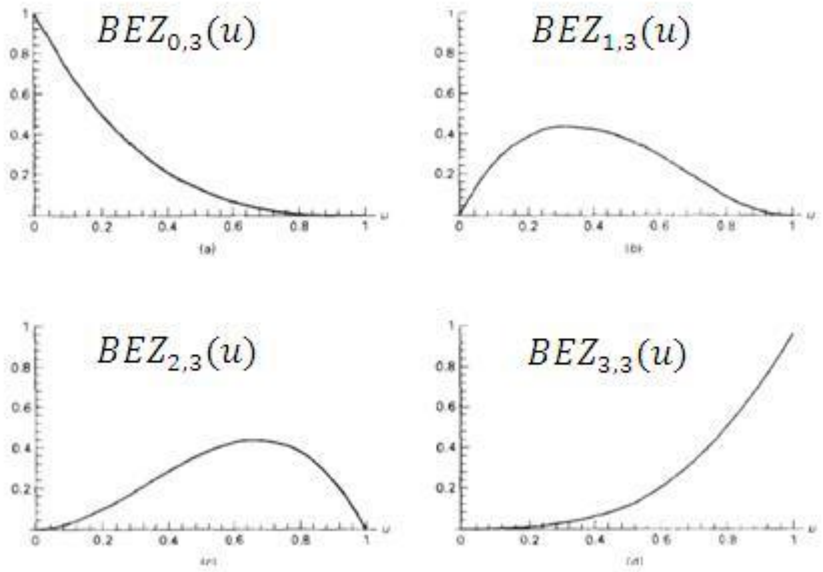


Fig. 4.24: -Four Bezier blending function for cubic curve.

- The form of blending functions determines how control points affect the shape of the curve for values of parameter u over the range from 0 to 1.
  At u = 0 $BEZ_{0,3}(u)$ is only nonzero blending function with values 1.
  At u = 1 $BEZ_{3,3}(u)$ is only nonzero blending function with values 1.
- So the cubic Bezier curve is always pass through p_0 and p_3.
- Other blending function is affecting the shape of the curve in intermediate values of parameter u.
- $BEZ_{1,3}(u)$ is maximum at $u = {}^{1}/_{3}$and $BEZ_{2,3}(u)$ is maximum at $u = {}^{2}/_{3}$
- Blending function is always nonzero over the entire range of u so it is not allowed for local control of the curve shape.
- At end point positions parametric first order derivatives are :
  $p'(0) = 3(p_1 - p_0)$
  $p'(1) = 3(p_3 - p_2)$
- And second order parametric derivatives are.
  $p''(0) = 6(p_0 - 2p_1 + p_2)$
  $p''(1) = 6(p_1 - 2p_2 + p_3)$
- This expression can be used to construct piecewise curve with C^1 and C^2 continuity.
- Now we represent polynomial expression for blending function in matrix form:

$$p(u) = [u^3 \quad u^2 \quad u \quad 1] \cdot M_{BEZ} \cdot \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

$$M_{BEZ} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

- We can add additional parameter like tension and bias as we did with the interpolating spline.

Bezier Surfaces

- Two sets of orthogonal Bezier curves can be used to design an object surface by an input mesh of control points.
- By taking Cartesian product of Bezier blending function we obtain parametric vector function as:

$$p(u,v) = \sum_{j=0}^{m} \sum_{k=0}^{n} p_{j,k} BEZ_{j,m}(v) BEZ_{k,n}(u)$$

- $p_{j,k}$ Specifying the location of the (m+1) by (n+1) control points.
- Figure below shows Bezier surfaces plot, control points are connected by dashed line and curve is represented by solid lines.

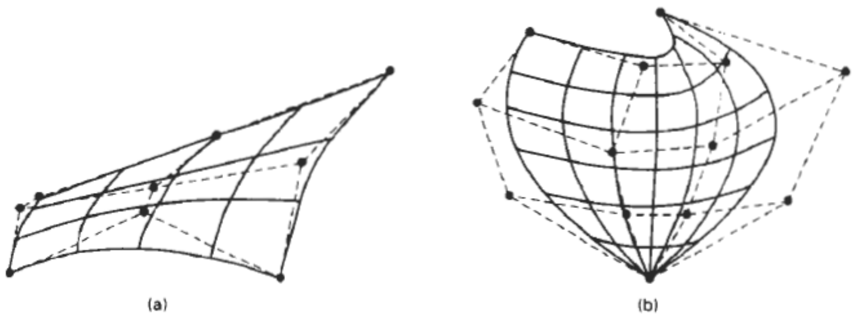


Fig. 4.25: -Bezier surfaces constructed for (a) m=3, n=3, and (b) m=4, n=4. Dashed line connects the control points.

- Each curve of constant u is plotted by varying v over interval 0 to 1. And similarly we can plot for constant v.
- Bezier surfaces have same properties as Bezier curve, so it can be used in interactive design application.
- For each surface patch we first select mesh of control point XY and then select elevation in Z direction.
- We can put two or more surfaces together and form required surfaces using method similar to curve section joining with continuity C^0, C^1, and C^2 as per need.

B-Spline Curves and Surfaces

- B-Spline is most widely used approximation spline.
- It has two advantage over Bezier spline
  1. Degree of a B-Spline polynomial can be set independently of the number of control points (with certain limitation).
  2. B-Spline allows local control.
- Disadvantage of B-Spline curve is more complex then Bezier spline

B-Spline Curves

- General expression for B-Spline curve in terms of blending function is given by:

  $$p(u) = \sum_{k=0}^{n} p_k B_{k,d}(u) \qquad u_{min} \le u \le u_{max}, 2 \le d \le n+1$$

  Where p_k is input set of control points.
- The range of parameter u is now depends on how we choose the B-Spline parameters.
- B-Spline blending function $B_{k,d}$ are polynomials of degree d-1 , where d can be any value in between 2 to n+1.
- We can set d=1 but then curve is only point plot.
- By defining blending function for subintervals of whole range we can achieve local control.
- Blending function of B-Spline is solved by Cox-deBoor recursion formulas as follows.

  $$B_{k,1}(u) = \begin{cases} 1 & if\ u_k \le u \le u_{k+1} \\ 0 & otherwise \end{cases}$$

  $$B_{k,d}(u) = \frac{u - u_k}{u_{k+d-1} - u_k} B_{k,d-1}(u) + \frac{u_{k+d} - u}{u_{k+d} - u_{k+1}} B_{k+1,d-1}(u)$$
- The selected set of subinterval endpoints u_j is reffered to as a **knot vector**.
- We can set any value as a subinterval end point but it must follow $u_j \le u_{j+1}$
- Values of u_{min} and u_{max} depends on number of control points, degree d, and knot vector.
- Figure below shows local control

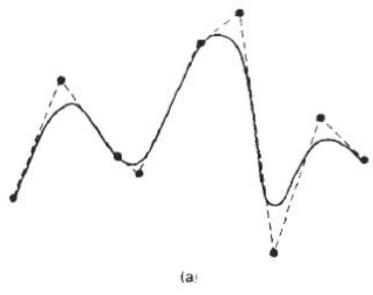


Fig. 4.26: -Local modification of B-Spline curve.

- B-Spline allows adding or removing control points in the curve without changing the degree of curve.
- B-Spline curve lies within the convex hull of at most d+1 control points so that B-Spline is tightly bound to input positions.

- For any u in between u_{d-1} to u_{n+1}, sum of all blending function is 1 i.e. $\sum_{k=0}^{n} B_{k,d}(u) = 1$
- There are three general classification for knot vectors:
  - Uniform
  - Open uniform
  - Non uniform

Properties of B-Spline Curves

- It has degree d-1 and continuity C^{d-2} over range of u.
- For n+1 control point we have n+1 blending function.
- Each blending function $B_{k,d}(u)$ is defined over d subintervals of the total range of u, starting at knot value u_k.
- The range of u is divided into n+d subintervals by the n+d+1 values specified in the knot vector.
- With knot values labeled as $\{u_0, u_1, \dots, u_{n+d}\}$ the resulting B-Spline curve is defined only in interval from knot values u_{d-1} up to knot values u_{n+1}
- Each spline section is influenced by d control points.
- Any one control point can affect at most d curve section.

Uniform Periodic B-Spline

- When spacing between knot values is constant, the resulting curve is called a uniform B-Spline.
- For example $\{0.0, 0.1, 0.2, \dots, 1.0\}$ or $\{0,1,2,3,4,5,6,7\}$
- Uniform B-Spline have periodic blending function. So for given values of n and d all blending function has same shape. And each successive blending function is simply a shifted version of previous function.
  $B_{k,d}(u) = B_{k+1,d}(u + \Delta u) = B_{k+2,d}(u + 2\Delta u)$
  Where Δu is interval between adjacent knot vectors.

Cubic Periodic B-Spline

- It commonly used in many graphics packages.
- It is particularly useful for generating closed curve.
- If any three consecutive control points are identical the curve passes through that coordinate position.
- Here for cubic curve d = 4 and n = 3 knot vector spans d+n+1 =4+3+1=8 so it is {0,1,2,3,4,5,6,7}
- Now boundary conditions for cubic B-Spline curve is obtain from equation.

$$p(u) = \sum_{k=0}^{n} p_k B_{k,d}(u) \qquad u_{min} \le u \le u_{max}, 2 \le d \le n+1$$

- That are

$$p(0) = \frac{1}{6}(p_0 + 4p_1 + p_2)$$

$$p(1) = \frac{1}{6}(p_1 + 4p_2 + p_3)$$

$$p'(0) = \frac{1}{2}(p_2 - p_0)$$

$$p'(1) = \frac{1}{2}(p_3 - p_1)$$

- Matrix formulation for a cubic periodic B-Splines with the four control points can then be written as

$$p(u) = [u^3 \quad u^2 \quad u \quad 1] \cdot M_B \cdot \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

Where

$$M_B = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

- We can also modify the B-Spline equation to include a tension parameter t.
- The periodic cubic B-Spline with tension matrix then has the form:

$$M_{Bt} = \frac{1}{6}\begin{bmatrix} -t & 12-9t & 9t-12 & t \\ 3t & 12t-18 & 18-15t & 0 \\ -3t & 0 & 3t & 0 \\ t & 6-2t & t & 0 \end{bmatrix}$$

When t = 1 $M_{Bt} = M_B$

- We can obtain cubic B-Spline blending function for parametric range from 0 to 1 by converting matrix representation into polynomial form for t = 1 we have

$$B_{0,3}(u) = \frac{1}{6}(1-u)^3$$

$$B_{1,3}(u) = \frac{1}{6}(3u^3 - 6u^2 + 4)$$

$$B_{2,3}(u) = \frac{1}{6}(-3u^3 + 3u^2 + 3u + 1)$$

$$B_{3,3}(u) = \frac{1}{6}u^3$$

Open Uniform B-Splines

- This class is cross between uniform B-Spline and non uniform B-Splines.
- Sometimes it is treated as a special type of uniform B-Spline, and sometimes as non uniform B-Spline
- For open uniform B-Spline (open B-Spline) the knot spacing is uniform except at the ends where knot values are repeated d times.
- For example {0,0,1,2,3,3} for d=2 and n=3, and {0,0,0,0,1,2,2,2,2} for d=4 and n=4.
- For any values of parameter d and n we can generate an open uniform knot vector with integer values using the calculations as follow:

$$u_j = \begin{cases} 0, & for\ 0 \le j < d \\ j-d+1 & for\ d \le j \le n \\ n-d+2 & for\ j > n \end{cases}$$

Where $0 \le j \le n+d$

- Open uniform B-Spline is similar to Bezier spline if we take d=n+1 it will reduce to Bezier spline as all knot values are either 0 or 1.
- For example cubic open uniform B-Spline with d=4 have knot vector {0,0,0,0,1,1,1,1}.
- Open uniform B-Spline curve passes through first and last control points.
- Also slope at each end is parallel to line joining two adjacent control points at that end.
- So geometric condition for matching curve sections are same as for Bezier curves.
- For closed curve we specify first and last control point at the same position.

Non Uniform B-Spline

- For this class of spline we can specify any values and interval for knot vector.
- For example {0,1,2,3,3,4}, and {0,0,1,2,2,3,4}
- It will give more flexible shape of curves. Each blending function have different shape when plots and different intervals.
- By increasing knot multiplicity we produce variation in curve shape and also introduce discontinuities.

- Multiple knot value also reduces continuity by 1 for each repeat of particular value.
- We can solve non uniform B-Spline using similar method as we used in uniform B-Spline.
- For set of n+1 control point we set degree d and knot values.
- Then using the recurrence relations we can obtain blending function or evaluate curve position directly for display of the curve.

B-Spline Surfaces

- B-Spline surface formation is also similar to Bezier splines orthogonal set of curves are used and for connecting two surface we use same method which is used in Bezier surfaces.
- Vector equation of B-Spline surface is given by cartesion product of B-Spline blending functions:

$$p(u,v) = \sum_{k1=0}^{n1} \sum_{k2=0}^{n2} p_{k1,k2} B_{k1,d1}(u) B_{k2,d2}(v)$$

- Where $p_{k1,k2}$ specify control point position.
- It has same properties as B-Spline curve.

Chapter 3D Translation

3D Translation

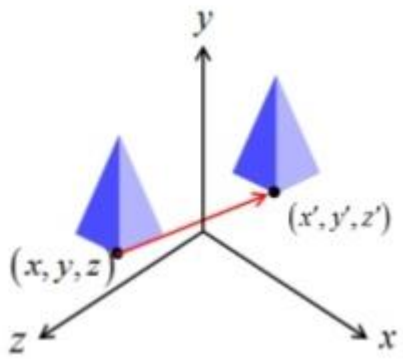


Fig. 5.1: - 3D Translation.

- Similar to 2D translation, which used 3x3 matrices, 3D translation use 4X4 matrices (X, Y, Z, h).
- In 3D translation point (X, Y, Z) is to be translated by amount tx, ty and tz to location (X', Y', Z').

  $\boldsymbol{x' = x + tx}$

  $\boldsymbol{y' = y + ty}$

  $\boldsymbol{z' = z + tz}$
- Let's see matrix equation

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

  $$\begin{bmatrix}\boldsymbol{x'}\\\boldsymbol{y'}\\\boldsymbol{z'}\\\boldsymbol{1}\end{bmatrix} = \begin{bmatrix}\boldsymbol{1}&\boldsymbol{0}&\boldsymbol{0}&\boldsymbol{t_x}\\\boldsymbol{0}&\boldsymbol{1}&\boldsymbol{0}&\boldsymbol{t_y}\\\boldsymbol{0}&\boldsymbol{0}&\boldsymbol{1}&\boldsymbol{t_z}\\\boldsymbol{0}&\boldsymbol{0}&\boldsymbol{0}&\boldsymbol{1}\end{bmatrix} \cdot \begin{bmatrix}\boldsymbol{x}\\\boldsymbol{y}\\\boldsymbol{z}\\\boldsymbol{1}\end{bmatrix}$$
- Example : - Translate the given point P (10,10,10) into 3D space with translation factor T (10,20,5).

  $P' = T \cdot P$

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

  $$\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix} = \begin{bmatrix}1&0&0&10\\0&1&0&20\\0&0&1&5\\0&0&0&1\end{bmatrix} \cdot \begin{bmatrix}10\\10\\10\\1\end{bmatrix}$$

  $$\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix} = \begin{bmatrix}20\\30\\15\\1\end{bmatrix}$$

  Final coordinate after translation is P' (20, 30, 15).

Rotation

- For 3D rotation we need to pick an axis to rotate about.
- The most common choices are the X-axis, the Y-axis, and the Z-axis

Coordinate-Axes Rotations

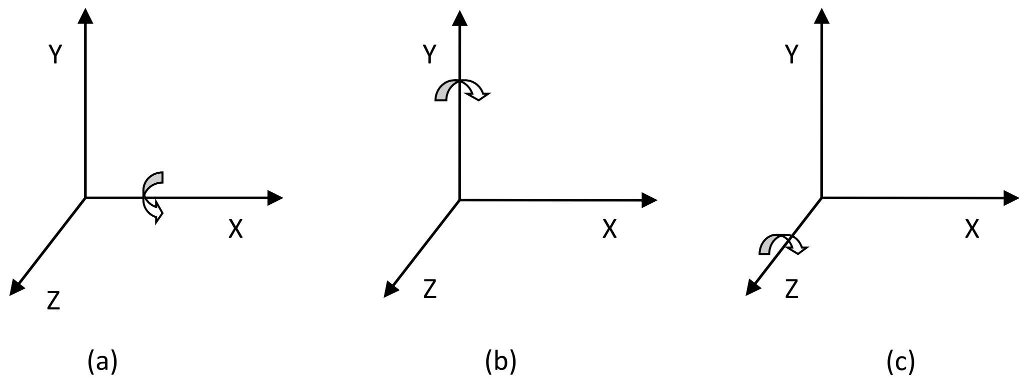


Fig. 5.2: - 3D Rotations.

Z-Axis Rotation

- Two dimension rotation equations can be easily convert into 3D Z-axis rotation equations.
- Rotation about z axis we leave z coordinate unchanged.

  $\boldsymbol{x' = x\cos\theta - y\sin\theta}$

  $\boldsymbol{y' = x\sin\theta + y\cos\theta}$

  $\boldsymbol{z' = z}$

  Where Parameter $\boldsymbol{\theta}$ specify rotation angle.
- Matrix equation is written as:

  $\boldsymbol{P' = R_z(\theta)\cdot P}$

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

X-Axis Rotation

- Transformation equation for x-axis is obtain from equation of z-axis rotation by replacing cyclically as shown here

  $\boldsymbol{x \to y \to z \to x}$
- Rotation about x axis we leave x coordinate unchanged.

  $\boldsymbol{y' = y\cos\theta - z\sin\theta}$

  $\boldsymbol{z' = y\sin\theta + z\cos\theta}$

  $\boldsymbol{x' = x}$

  Where Parameter $\boldsymbol{\theta}$ specify rotation angle.
- Matrix equation is written as:

  $\boldsymbol{P' = R_x(\theta)\cdot P}$

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

Y-Axis Rotation

- Transformation equation for y-axis is obtain from equation of x-axis rotation by replacing cyclically as shown here

  $\boldsymbol{x \to y \to z \to x}$

- Rotation about y axis we leave y coordinate unchanged.
  $\boldsymbol{z' = z\cos\theta - x\sin\theta}$
  $\boldsymbol{x' = z\sin\theta + x\cos\theta}$
  $\boldsymbol{y' = y}$
  Where Parameter $\boldsymbol{\theta}$ specify rotation angle.
- Matrix equation is written as:
  $\boldsymbol{P' = R_y(\theta)\cdot P}$
  $$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
- Example: - Rotate the point P(5,5,5) 90° about Z axis.
  $P' = R_z(\theta)\cdot P$
  $$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
  $$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos 90 & -\sin 90 & 0 & 0 \\ \sin 90 & \cos 90 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
  $$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ 5 \\ 5 \\ 1 \end{bmatrix}$$
  $$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} -5 \\ 5 \\ 5 \\ 1 \end{bmatrix}$$
  Final coordinate after rotation is P′ (-5, 5, 5).

General 3D Rotations when rotation axis is parallel to one of the standard axis

- Three steps require to complete such rotation
  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.
- This can be represented in equation form as:
  $\boldsymbol{P' = T^{-1}\cdot R(\theta)\cdot T\cdot P}$

General 3D Rotations when rotation axis is inclined in arbitrary direction

- When object is to be rotated about an axis that is not parallel to one of the coordinate axes, we need rotations to align the axis with a selected coordinate axis and to bring the axis back to its original orientation.
- Five steps require to complete such rotation.
  1. Translate the object so that the rotation axis passes through the coordinate origin.
  2. Rotate the object so that the axis of rotation coincides with one of the coordinate axes.
  3. Perform the specified rotation about that coordinate axis.
  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 position.
- We can transform rotation axis onto any of the three coordinate axes. The Z-axis is a reasonable choice.

- We are given line in the form of two end points P1 (x1,y1,z1), and P2 (x2,y2,z2).
- We will see procedure step by step.

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

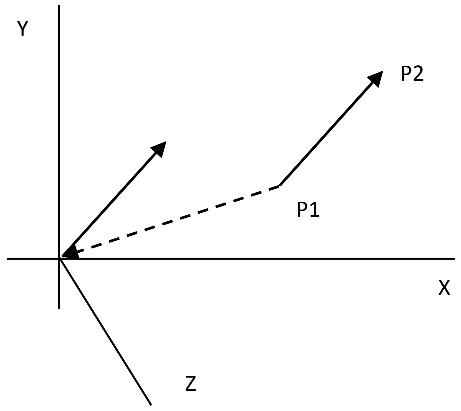


Fig. 5.3: - Translation of vector V.

- For translation of step one we will bring first end point at origin and transformation matrix for the same is as below

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

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

- This task can be completed by two rotations first rotation about x-axis and second rotation about y-axis.
- But here we do not know rotation angle so we will use dot product and vector product.
- Lets write rotation axis in vector form.

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

- Unit vector along rotation axis is obtained by dividing vector by its magnitude.

$$u = \frac{V}{|V|} = \left(\frac{x_2 - x_1}{|V|}, \frac{y_2 - y_1}{|V|}, \frac{z_2 - z_1}{|V|}\right) = (a, b, c)$$

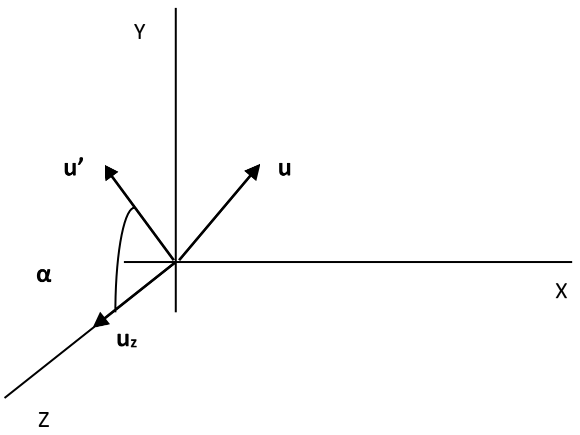


Fig. 5.4: - Projection of u on YZ-Plane.

- Now we need cosine and sin value of angle between unit vector '**u**' and XZ plane and for that we will take projection of u on YZ-plane say '**u'**' and then find dot product and cross product of '**u'**' and '**u**$_z$' .
- Coordinate of '**u'**' is (0,b,c) as we will take projection on YZ-plane x value is zero.

$$u' \cdot u_z = |u'||u_z| \cos \alpha$$

$$\cos\alpha = \frac{u' \cdot u_z}{|u'||u_z|} = \frac{(0,b,c)(0,0,1)}{\sqrt{b^2+c^2}} = \frac{c}{d} \quad \textit{where } d = \sqrt{b^2+c^2}$$

And

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

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

Comparing magnitude

$|u'||u_z| \sin\alpha = b$

$\sqrt{b^2+c^2} \cdot (1) \sin\alpha = b$

$d \sin\alpha = b$

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

- Now we have **$\sin\alpha$** and **$\cos\alpha$** so we will write matrix for rotation about X-axis.

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

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

- After performing above rotation '**u**' will rotated into '**u''**' in XZ-plane with coordinates (a, 0, √(b^2+c^2)). As we know rotation about x axis will leave x coordinate unchanged, '**u''**' is in XZ=plane so y coordinate is zero, and z component is same as magnitude of '**u'**'.
- Now rotate '**u''**' about Y-axis so that it coincides with Z-axis.

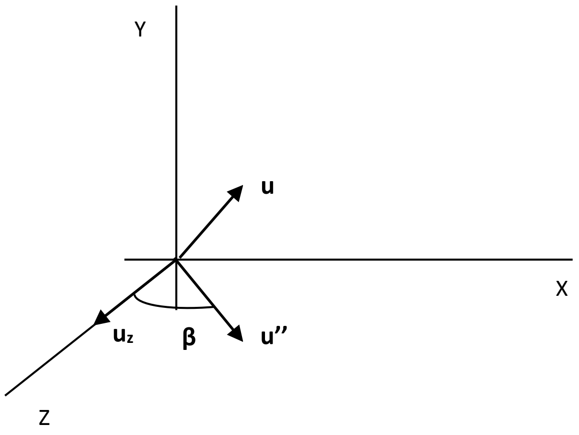


Fig. 5.5: - Rotation of u about X-axis.

- For that we repeat above procedure between '**u''**' and '**u**z' to find matrix for rotation about Y-axis.

$u'' \cdot u_z = |u''||u_z| \cos\beta$

$$\cos\beta = \frac{u' \cdot u_z}{|u'||u_z|} = \frac{\left(a, 0, \sqrt{b^2+c^2}\right)(0,0,1)}{1} = \sqrt{b^2+c^2} = d \quad \textit{where } d = \sqrt{b^2+c^2}$$

And

$u'' \times u_z = u_y|u''||u_z| \sin\beta = u_y \cdot (-a)$

$u_y|u''||u_z| \sin\beta = u_y \cdot (-a)$

Comparing magnitude

$|u''||u_z| \sin\beta = (-a)$

$(1)\ \sin\beta = -a$

$\sin\beta = -a$

- Now we have $\sin\beta$ and $\cos\beta$ so we will write matrix for rotation about Y-axis.

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

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

- Now by combining both rotation we can coincides rotation axis with Z-axis

3) **Perform the specified rotation about that coordinate axis.**

- As we know we align rotation axis with Z axis so now matrix for rotation about z axis

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

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

- This step is inverse of step number 2.

5) **Apply the inverse translation to bring the rotation axis back to its original position.**
6) This step is inverse of step number 1.

So finally sequence of transformation for general 3D rotation is

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

Scaling

- It is used to resize the object in 3D space.
- We can apply uniform as well as non uniform scaling by selecting proper scaling factor.
- Scaling in 3D is similar to scaling in 2D. Only one extra coordinate need to consider into it.

Coordinate Axes Scaling

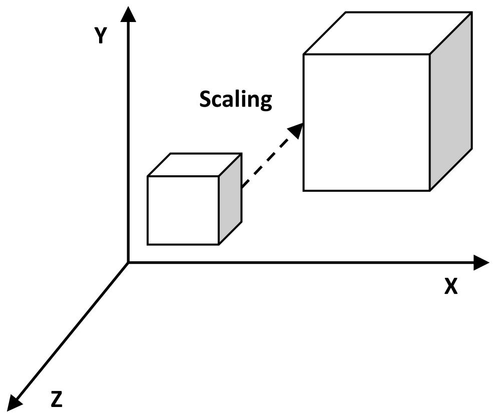


Fig. 5.6: - 3D Scaling.

- Simple coordinate axis scaling can be performed as below.

$P' = S \cdot P$

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

- Example: - Scale the line AB with coordinates (10,20,10) and (20,30,30) respectively with scale factor S(3,2,4).

  $P' = S \cdot P$

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

$$\begin{bmatrix} A_x' & B_x' \\ A_y' & B_y' \\ A_z' & B_z' \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 10 & 20 \\ 20 & 30 \\ 10 & 30 \\ 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} A_x' & B_x' \\ A_y' & B_y' \\ A_z' & B_z' \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 30 & 60 \\ 40 & 60 \\ 40 & 120 \\ 1 & 1 \end{bmatrix}$$

  Final coordinates after scaling are A′ (30, 40, 40) and B′ (60, 60, 120).

Fixed Point Scaling

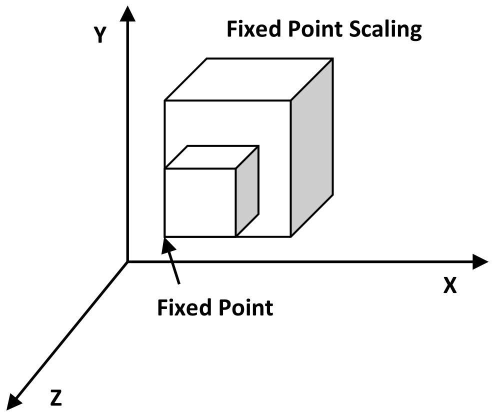


Fig. 5.7: - 3D Fixed point scaling.

- Fixed point scaling is used when we require scaling of object but particular point must be at its original position.
- Fixed point scaling matrix can be obtained in three step procedure.
  1. Translate the fixed point to the origin.
  2. Scale the object relative to the coordinate origin using coordinate axes scaling.
  3. Translate the fixed point back to its original position.
- Let's see its equation.

$$\boldsymbol{P' = T(x_f, y_f, z_f) \cdot S(s_x, s_y, s_z) \cdot T(-x_f, -y_f, -z_f) \cdot P}$$

$$\boldsymbol{P'} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \boldsymbol{x_f} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \boldsymbol{y_f} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \boldsymbol{z_f} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{s_x} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{s_y} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{s_z} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \boldsymbol{-x_f} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \boldsymbol{-y_f} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \boldsymbol{-z_f} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \cdot \boldsymbol{P}$$

$$P' = \begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot P$$

Other Transformations

Reflections

- Reflection means mirror image produced when mirror is placed at require position.
- When mirror is placed in XY-plane we obtain coordinates of image by just changing the sign of z coordinate.
- Transformation matrix for reflection about XY-plane is given below.

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

- Similarly Transformation matrix for reflection about YZ-plane is.

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

- Similarly Transformation matrix for reflection about XZ-plane is.

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

Shears

- Shearing transformation can be used to modify object shapes.
- They are also useful in 3D viewing for obtaining general projection transformations.
- Here we use shear parameter '**a**' and '**b**'
- Shear matrix for Z-axis is given below

$$SH_z = \begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Similarly Shear matrix for X-axis is.

$$SH_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ a & 1 & 0 & 0 \\ b & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Similarly Shear matrix for X-axis is.

$$SH_y = \begin{bmatrix} 1 & a & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & b & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Viewing Pipeline

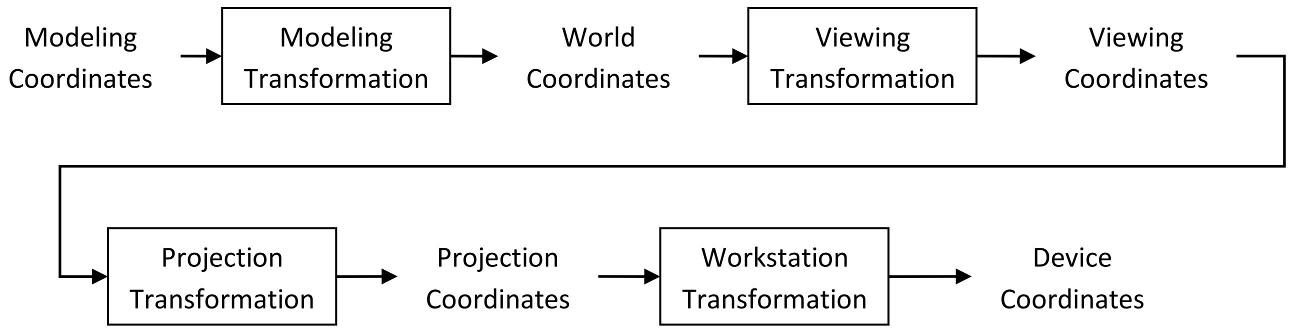


Fig. 5.8: - General 3D viewing pipeline.

- Steps involved in 3D pipeline are similar to the process of taking a photograph.
- As shown in figure that initially we have modeling coordinate of any object which we want to display on the screen.
- By applying modeling transformation we convert modeling coordinates to world coordinates which gives which part or portion is to be display.
- Then by applying viewing transformation we obtain the viewing coordinate which is fitted in viewing coordinate reference frame.
- Then in case of three dimensional objects we have three dimensions coordinate but we need to display that object on two dimensional screens so we apply projection transformation on it which gives projection coordinate.
- Finally projection coordinate is converted into device coordinate by applying workstation transformation which gives coordinates which is specific to particular device.

Viewing Co-ordinates.

- Generating a view of an object is similar to photographing the object.
- We can take photograph from any side with any angle & orientation of camera.
- Similarly we can specify viewing coordinate in ordinary direction.

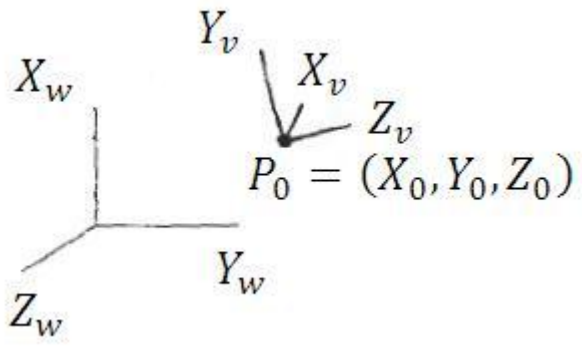


Fig. 5.9: -A right handed viewing coordinate system, with axes Xv, Yv, and Zv, relative to a world-coordinate scene.

Specifying the view plan

- We decide view for a scene by first establishing viewing coordinate system, also referred as view reference coordinate system.
- Then projection plane is setup in perpendicular direction to Zv axis.

- Then projections positions in the scene are transferred to viewing coordinate then viewing coordinate are projected onto the view plane.
- The origin of our viewing coordinate system is called view reference point.
- View reference point is often chosen to be close to or on the surface as same object scene. We can also choose other point also.
- Next we select positive direction for the viewing Zv axis and the orientation of the view plane by specifying the view plane normal vector N.
- Finally we choose the up direction for the view by specifying a vector V called the view up vector. Which specify orientation of camera.
- View up vector is generally selected perpendicular to normal vector but we can select any angle between V & N.
- By fixing view reference point and changing direction of normal vector N we get different views of same object this is illustrated by figure below.

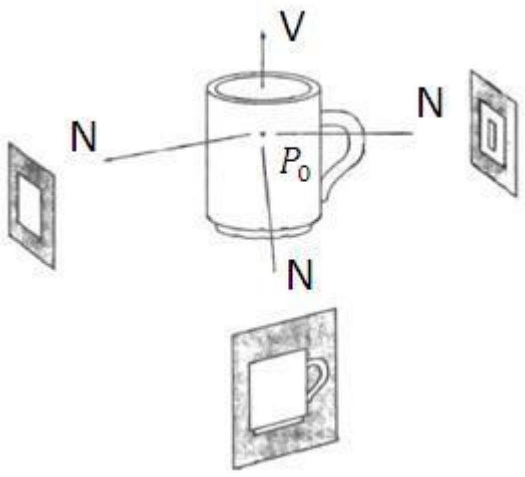


Fig. 5.10: -Viewing scene from different direction with a fixed view-reference point.

Transformation from world to viewing coordinates

- Before taking projection of view plane object description is need to transfer from world to viewing coordinate.
- It is same as transformation that superimposes viewing coordinate system to world coordinate system.
- It requires following basic transformation.
- 1) Translate view reference point to the origin of the world coordinate system.
- 2) Apply rotation to align

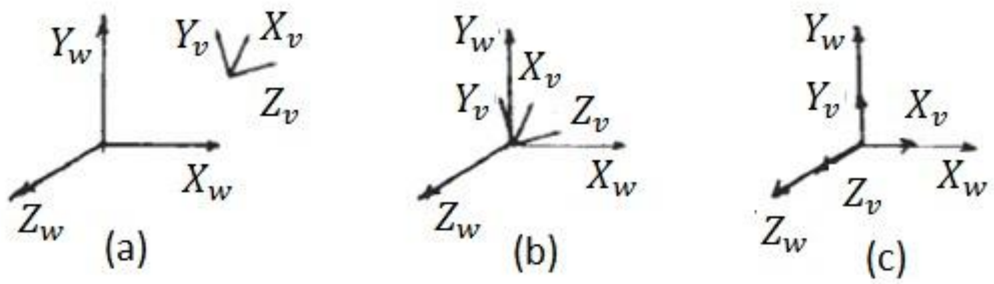


Fig. 5.11: - Aligning a viewing system with the world-coordinate axes using a sequence of translate-rotate transformations.

- As shown in figure the steps of transformation

- Consider view reference point in world coordinate system is at position (x_0, y_0, z_0)than for align view reference point to world origin we perform translation with matrix:
- $T = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- Now we require rotation sequence up-to three coordinate axis rotations depending upon direction we choose for N.
- In general case N is at arbitrary direction then we can align it with word coordinate axes by rotation sequence $Rz \cdot Ry \cdot Rx$.
- Another method for generating the rotation transformation matrix is to calculate unit *uvn* vectors and from the composite rotation matrix directly.
- Here

$$n = \frac{N}{|N|} = (n_1, n_2, n_3)$$
$$u = \frac{V \times N}{|V \times N|} = (u_1, u_2, u_3)$$
$$v = n \times u = (v_1, v_2, v_3)$$

- This method also automatically adjusts the direction for *u* so that *v* is perpendicular to *n*.
- Than composite rotation matrix for the viewing transformation is then:

$$R = \begin{bmatrix} u_1 & u_2 & u_3 & 0 \\ v_1 & v_2 & v_3 & 0 \\ n_1 & n_2 & n_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- This aligns *u* to Xw axis, *v* to Yw axis and *n* to Zw axis.
- Finally composite matrix for world to viewing coordinate transformation is given by:
- $M_{wc,vc} = R \cdot T$
- This transformation is applied to object's coordinate to transfer them to the viewing reference frame.

Projections

- Once world-coordinate descriptions of the objects in a scene are converted to viewing coordinates, we can project the three-dimensional objects onto the two-dimensional view plane.
- Process of converting three-dimensional coordinates into two-dimensional scene is known as **projection**.
- There are two projection methods namely.
  1. Parallel Projection.
  2. Perspective Projection.
- Lets discuss each one.

Parallel Projections

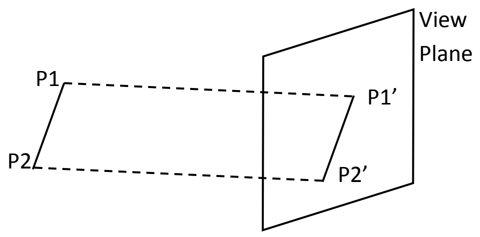


Fig. 5.12: - Parallel projection.

- In a parallel projection, coordinate positions are transformed to the view plane along parallel lines, as shown in the, example of above Figure.
- We can specify a parallel projection with a projection vector that defines the direction for the projection lines.
- It is further divide into two types.
    1. Orthographic parallel projection.
    2. Oblique parallel projection.

Orthographic parallel projection

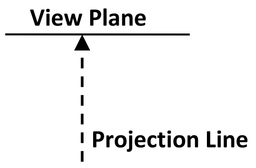


Fig. 5.13: - Orthographic parallel projection.

- When the projection lines are perpendicular to the view plane, we have an orthographic parallel projection.
- Orthographic projections are most often used to produce the front, side, and top views of an object, as shown in Fig.

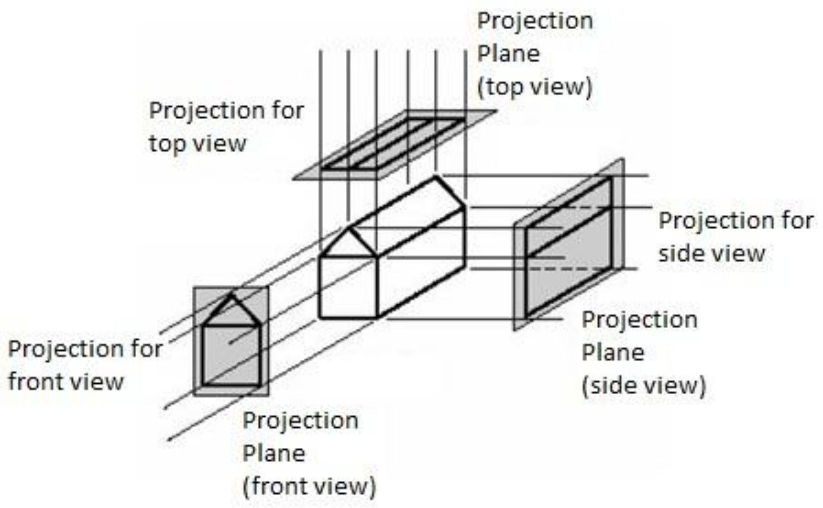


Fig. 5.14: - Orthographic parallel projection.

- Engineering and architectural drawings commonly use orthographic projections, because lengths and angles are accurately depicted and can be measure from the drawings.
- We can also form orthographic projections that display more than one face of an object. Such view are called **axonometric orthographic projections**. Very good example of it is **Isometric** projection.
- Transformation equations for an orthographic parallel projection are straight forward.
- If the view plane is placed at position z_{vp} along the z_v axis, then any point (x, y, z) in viewing coordinates is transformed to projection coordinates as

  $x_p = x, \qquad y_p = y$

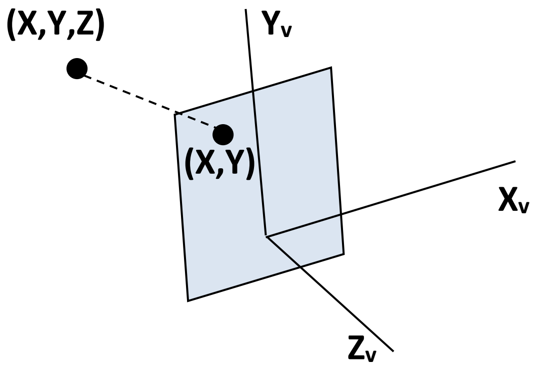


Fig. 5.15: - Orthographic parallel projection.

- Where the original z-coordinate value is preserved for the depth information needed in depth cueing and visible-surface determination procedures.

Oblique parallel projection.

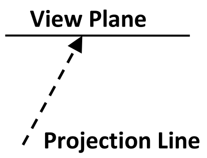


Fig. 5.16: - Oblique parallel projection.

- An oblique projection is obtained by projecting points along parallel lines that are not perpendicular to the projection plane.
- Coordinate of oblique parallel projection can be obtained as below.

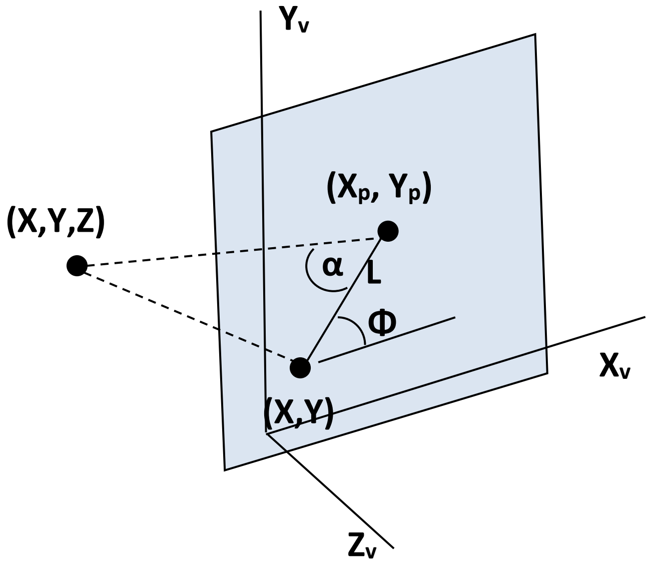


Fig. 5.17: - Oblique parallel projection.

- As shown in the figure (X,Y,Z) is a point of which we are taking oblique projection (Xp,Yp) on the view plane and point (X,Y) on view plane is orthographic projection of (X,Y,Z).
- Now from figure using trigonometric rules we can write

$$x_p = x + L\cos\emptyset$$
$$y_p = y + L\sin\emptyset$$

- Length L depends on the angle α and the z coordinate of the point to be projected:

$$\tan\alpha = \frac{Z}{L}$$
$$L = \frac{Z}{\tan\alpha}$$
$$L = ZL_1, \quad Where\ L_1 = \frac{1}{\tan\alpha}$$

- Now put the value of L in projection equation.

$$x_p = x + ZL_1\cos\emptyset$$
$$y_p = y + ZL_1\sin\emptyset$$

- Now we will write transformation matrix for this equation.

$$M_{parallel} = \begin{bmatrix} 1 & 0 & L_1\cos\emptyset & 0 \\ 0 & 1 & L_1\sin\emptyset & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- This equation can be used for any parallel projection. For orthographic projection L_1=0 and so whole term which is multiply with z component is zero.
- When value of $\tan\alpha = 1$ projection is known as **Cavalier projection**.
- When value of $\tan\alpha = 2$ projection is known as **Cabinet projection**.

Perspective Projection

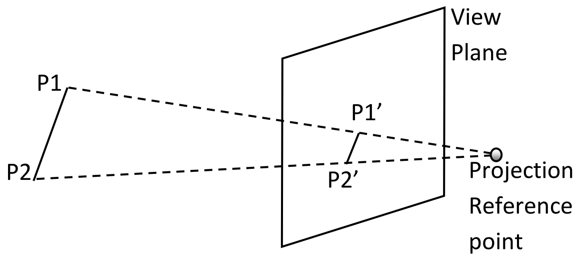


Fig. 5.18: - Perspective projection.

- In perspective projection object positions are transformed to the view plane along lines that converge to a point called the **projection reference point** (or **center of projection** or **vanishing point**).

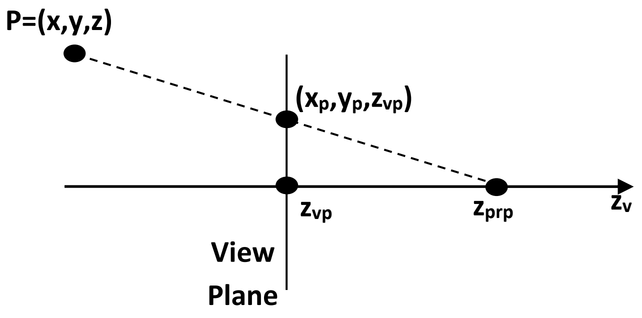


Fig. 5.19: - Perspective projection.

- Suppose we set the projection reference point at position z_{prp} along the z_v axis, and we place the view plane at z_{vp} as shown in Figure above. We can write equations describing coordinate positions along this perspective projection line in parametric form as

$$x' = x - xu$$
$$y' = y - yu$$
$$z' = z - (z - z_{prp})u$$

- Here parameter u takes the value from 0 to 1, which is depends on the position of object, view plane, and projection reference point.
- For obtaining value of u we will put z'=z_{vp} and solve equation of z'.

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

- Now substituting value of u in equation of x' and y' we will obtain.

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

- Using three dimensional homogeneous-coordinate representations, we can write the perspective projection transformation matrix form as.

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -z_{vp}/d_p & z_{vp}(z_{prp}/d_p) \\ 0 & 0 & -1/d_p & z_{prp}/d_p \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- In this representation, the homogeneous factor is.

$$h = \frac{z_{prp} - z}{d_p}\ and$$
$$x_p = x_h/h\ \ and\ y_p = y_h/h$$

- There are number of special cases for the perspective transformation equations.
- If view plane is taken to be uv plane, then $z_{vp} = 0$ and the projection coordinates are.

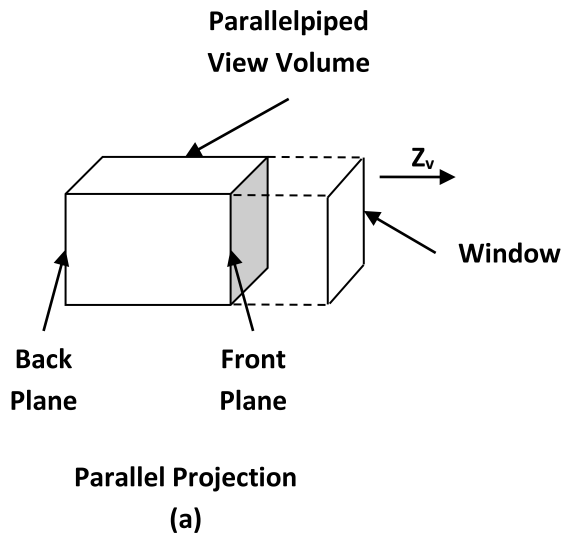
$$x_p = x\left(\frac{z_{prp}}{z_{prp} - z}\right) = x\left(\frac{1}{1 - z/z_{prp}}\right)$$
$$y_p = y\left(\frac{z_{prp}}{z_{prp} - z}\right) = y\left(\frac{1}{1 - z/z_{prp}}\right)$$

- If we take projection reference point at origin than $z_{prp} = 0$ and the projection coordinates are.

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

- The vanishing point for any set of lines that are parallel to one of the principal axes of an object is referred to as a principal vanishing point
- We control the number of principal vanishing points (one, two, or three) with the orientation of the projection plane, and perspective projections are accordingly classified as one-point, two-point, or three-point projections.
- The number of principal vanishing points in a projection is determined by the number of principal axes intersecting the view plane.

View Volumes and General Projection Transformations



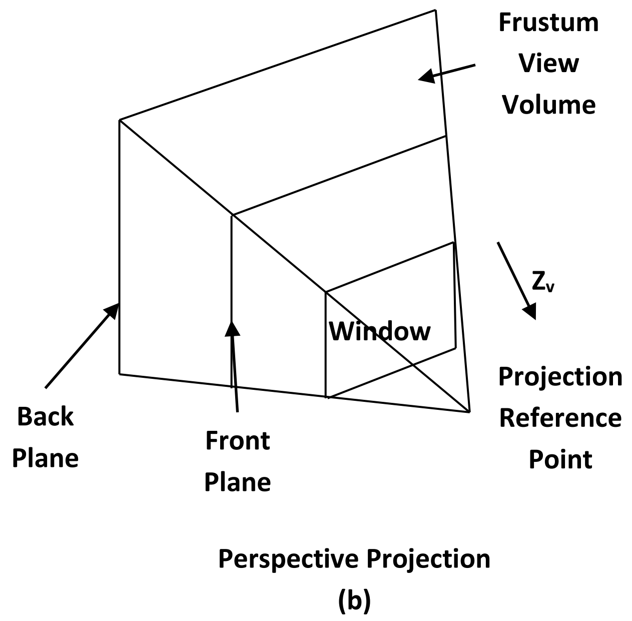


Fig. 5.20: - View volume of parallel and perspective projection.

- Based on view window we can generate different image of the same scene.
- Volume which is appears on the display is known as view volume.
- Given the specification of the view window, we can set up a view volume using the window boundaries.
- Only those objects within the view volume will appear in the generated display on an output device; all others are clipped from the display.
- The size of the view volume depends on the size of the window, while the shape of the view volume depends on the type of projection to be used to generate the display.
- A finite view volume is obtained by limiting the extent of the volume in the z_v direction.
- This is done by specifying positions for one or two additional boundary planes. These z_v-boundary planes are referred to as the **front plane** and **back plane**, or the **near plane** and the **far plane**, of the viewing volume.
- Orthographic parallel projections are not affected by view-plane positioning, because the projection lines are perpendicular to the view plane regardless of its location.
- Oblique projections may be affected by view-plane positioning, depending on how the projection direction is to be specified.

General Parallel-Projection Transformation

- Here we will obtain transformation matrix for parallel projection which is applicable to both orthographic as well as oblique projection.

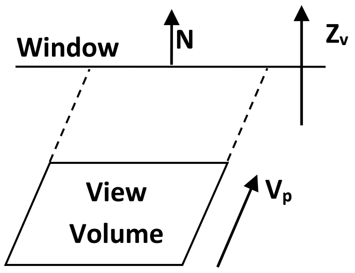


Fig. 5.21: - General parallel projection.

- As shown on figure parallel projection is specified with a projection vector from the projection reference point to the view window.
- Now we will apply shear transformation so that view volume will convert into regular parallelepiped and projection vector will become parallel to normal vector N.

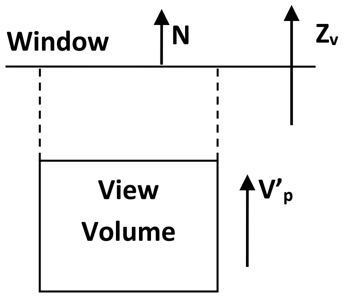


Fig. 5.22: - Shear operation in General parallel projection.

- Let's consider projection vector $\boldsymbol{V_p = (p_x, p_y, p_z)}$.
- We need to determine the elements of a shear matrix that will align the projection vector $\boldsymbol{V_p}$ with the view plane normal vector **N**. This transformation can be expressed as

$$\boldsymbol{V_p' = M_{parallel} \cdot V_p}$$

$$\boldsymbol{V_p' = \begin{bmatrix} 0 \\ 0 \\ p_z \\ 1 \end{bmatrix}}$$

- where $\boldsymbol{M_{parallel}}$ is equivalent to the parallel projection matrix and represents a z-axis shear of the form

$$\boldsymbol{M_{parallel} = \begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}$$

- Now from above equation we can write

$$\boldsymbol{\begin{bmatrix} 0 \\ 0 \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}}$$

- From matrix we can write.

$$\boldsymbol{0 = p_x + ap_z}$$
$$\boldsymbol{0 = p_y + bp_z}$$

So

$$\boldsymbol{a = \frac{-p_x}{p_z}}, \quad \boldsymbol{b = \frac{-p_y}{p_z}}$$

- Thus, we have the general parallel-projection matrix in terms of the elements of the projection vector as

$$M_{parallel} = \begin{bmatrix} 1 & 0 & \frac{-p_x}{p_z} & 0 \\ 0 & 1 & \frac{-p_y}{p_z} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- For an orthographic parallel projection, $p_x = p_y = 0$, and is the identity matrix.

General Perspective-Projection Transformations

- The projection reference point can be located at any position in the viewing system, except on the view plane or between the front and back clipping planes.

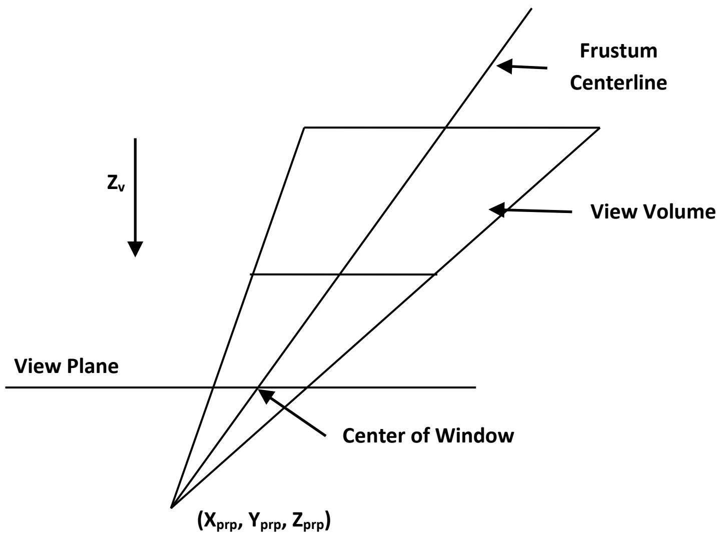


Fig. 5.23: - General perspective projection.

- We can obtain the general perspective-projection transformation with the following two operations:
  1. Shear the view volume so that the center line of the frustum is perpendicular to the view plane.
  2. Scale the view volume with a scaling factor that depends on 1/z .
- A shear operation to align a general perspective view volume with the projection window is shown in Figure.

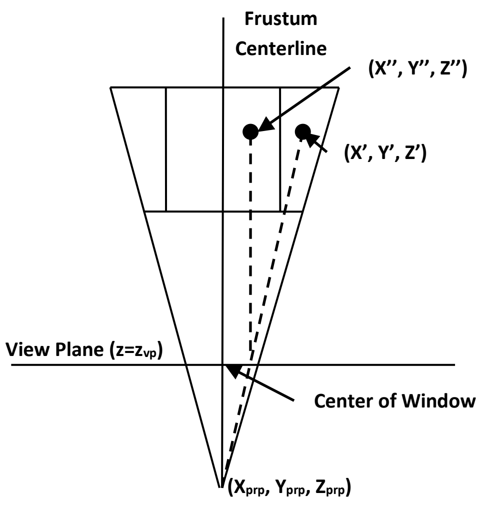


Fig. 5.24: - Shear and scaling operation in general perspective projection.

- With the projection reference point at a general position (X_{prp}, Y_{prp}, Z_{prp}) the transformation involves a combination z-axis shear and a translation:

  $$\boldsymbol{M_{shear}} = \begin{bmatrix} 1 & 0 & a & -az_{prp} \\ 0 & 1 & b & -bz_{prp} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  Where the shear parameters are

  $$a = -\frac{x_{prp} - (xw_{min} + xw_{max})/2}{z_{prp}}$$

  $$b = -\frac{y_{prp} - \frac{yw_{min}+yw_{max}}{2}}{z_{prp}}$$

- Points within the view volume are transformed by this operation as

  $$x' = x + a(z - z_{prp})$$
  $$y' = y + b(z - z_{prp})$$
  $$z' = z$$

- After shear we apply scaling operation.

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

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

- Homogeneous matrix for this transformation is:

  $$\boldsymbol{M_{scale}} = \begin{bmatrix} 1 & 0 & \frac{-x_{prp}}{z_{prp} - z_{vp}} & \frac{x_{prp}z_{vp}}{z_{prp} - z_{vp}} \\ 0 & 1 & \frac{-y_{prp}}{z_{prp} - z_{vp}} & \frac{y_{prp}z_{vp}}{z_{prp} - z_{vp}} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{-1}{z_{prp} - z_{vp}} & \frac{z_{prp}}{z_{prp} - z_{vp}} \end{bmatrix}$$

- Therefore the general perspective-projection transformation is obtained by equation:

  $$\boldsymbol{M_{perspective}} = \boldsymbol{M_{scale}} \cdot \boldsymbol{M_{shear}}$$

Chapter Z buffer and Surface Detection

Classification of Visible-Surface Detection Algorithms

- It is broadly divided into two parts
    - Object-Space methods
    - Image-Space methods
- Object space method compares objects and parts of objects to each other within the scene definition to determine which surface is visible.
- In image space algorithm visibility is decided point by point at each pixel position on the projection plane.

Back-Face Detection

- Back-Face Detection is simple and fast object –space method.
- It identifies back faces of polygon based on the inside-outside tests.
- A point (x, y, z) is inside if Ax + By + Cz + d < 0 where A, B, C, and D are constants and this equation is nothing but equation of polygon surface.
- We can simplify test by taking normal vector N= (A, B, C) of polygon surface and vector V in viewing direction from eye as shown in figure

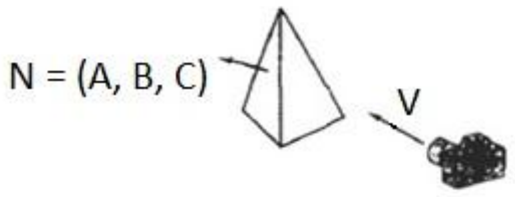


Fig. 6.1:- vector V in the viewing direction and back-face normal vector N of a polyhedron.

- Then we check condition if $V \cdot N > 0$ then polygon is back face.
- If we convert object description in projection coordinates and our viewing direction is parallel to z_v then v= (0,0,v_z) and
  $V \cdot N = V_z C.$
- So now we only need to check sign of C.
- In right handed viewing system V is along negative z_v axis. And in that case
  If C<0 the polygon is backface.
- Also we cannot see any face for which C=0.
- So in general for right handed system
  If $C \leq 0$ polygon is back face.
- Similar method can be used for left handed system.
- In left handed system V is along the positive Z direction and polygon is back face if $C \geq 0$.
- For a single convex polyhedron such as the pyramid by examining parameter C for the different plane we identify back faces.
- So far the scene contains only non overlapping convex polyhedral, back face method works properly.
- For other object such as concave polyhedron as shown in figure below we need to do more tests for determining back face.

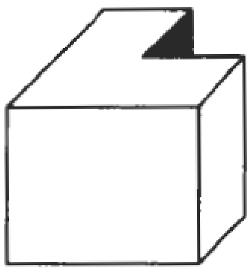


Fig. 6.2:-view of a concave polyhedron with one face partially hidden by other faces.

Depth Buffer Method/ Z Buffer Method

Algorithm

- Initialize the depth buffer and refresh buffer so that for all buffer positions***(x,*** y),
    - $depth(x, y) = 0, \qquad refresh(x, y) = I_{backgnd}$
- For each position on each polygon surface, compare depth values to previously stored values in the depth buffer to determine visibility.
- Calculate the depth z for each *(x,* y) position on the polygon.
- If z > depth(x, y), then set
    - $depth(x, y) = z, \qquad refresh(x, y) = I_{surf}(x, y)$
- Where $I_{backgnd}$ **is** the value for the background intensity, and $I_{surf}(x, y)$ is the projected intensity value for the surface at pixel position (x,y). After all surfaces have been processed, the depth buffer contains depth values for the visible surfaces and the refresh buffer contains the corresponding intensity values for those surfaces.

- It is image space approach.
- It compares surface depth at each pixel position on the projection plane.
- It is also referred to as z-buffer method since generally depth is measured in z-direction.
- Each surface of the scene is process separately one point at a time across the surface.

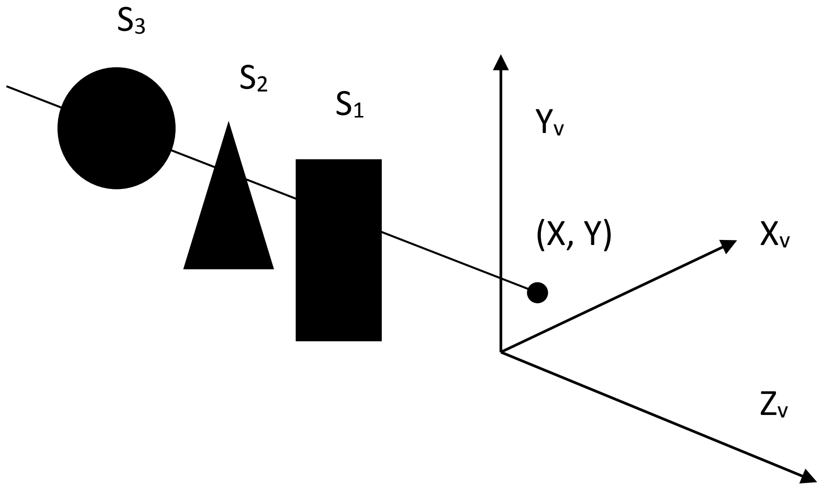


Fig. 6.3:- At view plane position (x, y), surface s_1 has smallest depth from the view plane and so is visible at that position.

- We are starting with pixel position of view plane and for particular surface of object.
- If we take orthographic projection of any point (x,y,z) of the surface on the view plane we get two dimension coordinate (x,y) for that point to display.
- Here we are taking (x.y) position on plan and find particular surface is at how much depth.
- We can implement depth buffer algorithm in normalized coordinates so that z values range from 0 at the back clipping plane to zmax at the front clipping plane.

- Zmax value can be 1 for unit cube or the largest value.
- Here two buffers are required. A depth buffer to store depth value of each (x,y) position and refresh buffer to store corresponding intensity values.
- Initially depth buffer value is 0 and refresh buffer value is intensity of background.
- Each surface of polygon is then process one by one scanline at a time.
- Calculate the z values at each (x,y) pixel position.
- If calculated depth value is greater than the value stored in depth buffer it is replaced with new calculated values and store intensity of that point into refresh buffer at (x,y) position.
- Depth values are calculated from plane equation $Ax + By + Cz + D = 0$ as:
  $$z = \frac{-Ax - By - D}{C}$$

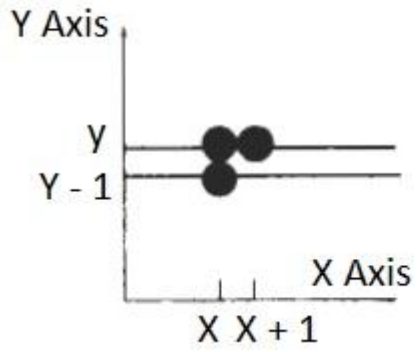


Fig. 6.4:-From position (x,y) on a scan line, the next position across the line has coordinates (x+1,y), and the position immediately below on the next line has coordinates (x,y-1).

- For horizontal line next pixel's z values can be calculated by putting x'=x+1 in above equation.
  $$z' = \frac{-A(x+1) - By - D}{C}$$
  $$z' = z - \frac{A}{C}$$
- Similarly for vertical line pixel below the current pixel has y'=y-1 so it's z values can be calculated as follows.
  $$z' = \frac{-Ax - B(y-1) - D}{C}$$
  $$z' = z + \frac{B}{C}$$
- If we are moving along polygon boundary then it will improve performance by eliminating extra calculation.
- For this if we move top to bottom along polygon boundary we get x'=x-1/m and y'=y-1, so z value is obtain as follows.
  $$z' = \frac{-A(x - 1/m) - B(y-1) - D}{C}$$
  $$z' = z + \frac{A/_m + B}{C}$$
- Alternately we can use midpoint method to find the z values.

Light source

- When we see any object we see reflected light from that object. Total reflected light is the sum of contribution from all sources and reflected light from other object that falls on the object.
- So that the surface which is not directly exposed to light may also visible if nearby object is illuminated.

- The simplest model for light source is **point source.** Rays from the source then follows radial diverging paths from the source position.

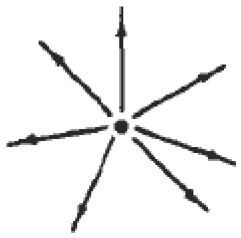


Fig. 6.5:- Diverging ray paths from a point light source.

- This light source model is reasonable approximation for source whose size is small compared to the size of object or may be at sufficient distance so that we can see it as point source. For example sun can be taken as point source on earth.
- A nearby source such as the long fluorescent light is more accurately modelled as a **distributed light source.**
- In this case the illumination effects cannot be approximated with point source because the area of the source is not small compare to the size of object.
- When light is falls on the surface the part of the light is reflected and part of the light is absorbed. Amount of reflected and absorbed light is depends on the property of the object surface. For example shiny surface reflect more light while dull surface reflect less light.

Basic Illumination Models/ Shading Model/ Lighting Model

- These models give simple and fast method for calculating the intensities of light for various reflections.

Ambient Light

- This is a simple way to model combination of light reflection from various surfaces to produce a uniform illumination called **ambient light,** or **background light.**
- Ambient light has no directional properties. The amount of ambient light incident on all the surfaces and object are constant in all direction.
- If consider that ambient light of intensity I_a and each surface is illuminate with I_a intensity then resulting reflected light is constant for all the surfaces.

Diffuse Reflection

- When some intensity of light is falls on object surface and that surface reflect light in all the direction in equal amount then the resulting reflection is called **diffuse reflection.**
- Ambient light reflection is approximation of global diffuse lighting effects.
- Diffuse reflections are constant over each surface independent of our viewing direction.
- Amount of reflected light is depend on the parameter K_d, the **diffuse reflection coefficient** or **diffuse reflectivity.**
- K_d is assign value in between 0 and 1 depending on reflecting property. Shiny surface reflect more light so K_d is assign larger value while dull surface assign small value.
- If surface is exposed to only ambient light we calculate ambient diffuse reflection as:
  $I_{ambdiff} = K_d I_a$
  Where $I_a$the ambient light is falls on the surface.
- Practically most of times each object is illuminated by one light source so now we discuss diffuse reflection intensity for point source.

- We assume that the diffuse reflection from source are scattered with equal intensity in all directions, independent of the viewing direction such a surface are sometimes referred as **ideal diffuse reflector** or **lambertian reflector.**
- This is modelled by **lambert's cosine law.** this law states that the radiant energy from any small surface area dA in any direction $\emptyset_n$ relative to surface normal is proportional to $cos\emptyset_n$.

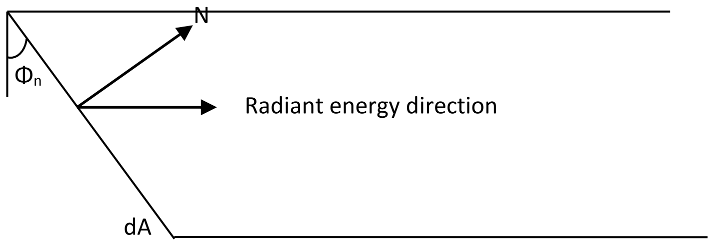


Fig. 6.6:- Radiant energy from a surface area dA in direction Φ_n relative to the surface normal direction.

- As shown reflected light intensity is does not depends on viewing direction so for lambertian reflection, the intensity of light is same in all viewing direction.
- Even though there is equal light distribution in all direction from perfect reflector the brightness of a surface does depend on the orientation of the surface relative to light source.
- As the angle between surface normal and incidence light direction increases light falls on the surface is decreases

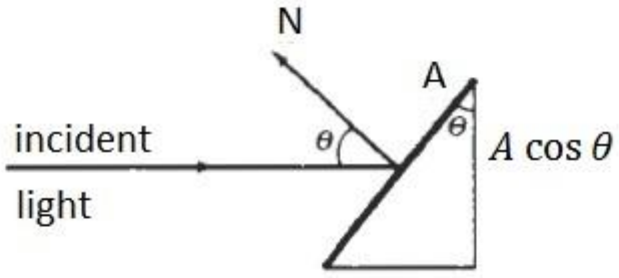


Fig. 6.7:- An illuminated area projected perpendicular to the path of the incoming light rays.

- If we denote the **angle of incidence** between the incoming light and surface normal asθ, then the projected area of a surface patch perpendicular to the light direction is proportional to $cos\theta$.
- If I_l is the intensity of the point light source, then the diffuse reflection equation for a point on the surface can be written as
  $I_{l,diff} = K_d I_l cos\theta$
- Surface is illuminated by a point source only if the angle of incidence is in the range 0^0 to 90^0 other than this value of θ light source is behind the surface.

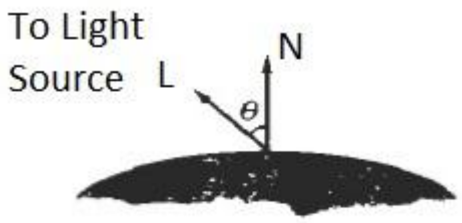


Fig. 6.8:-Angle of incidence θ between the unit light-source direction vector L and the unit surface normal N.

- As shown in figure **N** is the unit normal vector to surface and **L** is unit vector in direction of light source then we can take dot product of this to is:
  $N \cdot L = \cos\theta$
  And

$I_{l,diff} = K_d I_l (N \cdot L)$

- Now in practical ambient light and light source both are present and so total diffuse reflection is given by:
  $I_{diff} = K_a I_a + K_d I_l (N \cdot L)$
- Here for ambient reflection coefficient K_a is used in many graphics package so here we use K_a instead of K_d.

Specular Reflection and the Phong Model.

- When we look at an illuminated shiny surface, such as polished metal we see a highlight, or bright spot, at certain viewing directions. This phenomenon is called **specular reflection,** is the result of total, or near total reflection of the incident light in a concentrated region around the **specular reflection angle.**

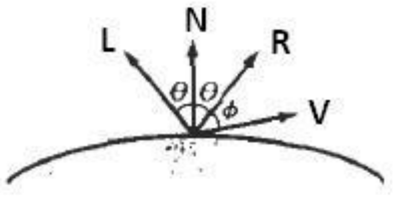


Fig. 6.9:-Specular reflection angle equals angle of incidence θ.

- Figure shows specular reflection direction at a point on the illuminated surface. The specular reflection angle equals the angle of the incident light.
- Here we use R as unit vector in direction of reflection L is unit vector point towards light vector N is unit normal vector and V is unit vector in viewing direction.
- Objects other than ideal reflectors exhibits specular reflection over a finite range of viewing positions around vector R. Shiny surface have a narrow specular reflection range and dull surface have wide specular reflection range.
- **By phong specular reflection model** or simply **phong model** sets the intensity of specular reflection proportional to $cos^{ns}\emptyset$. Angle $\emptyset$ varies in between 0^0 to 90^0.
- Values assigned to **specular reflection parameter** ns is determined by the type of surface that we want to display. A shiny surface assigned ns values large nearly 100 and dull surface assigned small nearly 1.
- Intensity of specular reflection depends on the material properties of the surface and the angle of incidence as well as **specular reflection coefficient, $w(\theta)$ for each surfaces.**
- Then specular reflection is given by:
  $I_{spec} = w(\theta) I_l cos^{ns}\emptyset$
  Where I_l is the intensity of light source and $\emptyset$ is angle between viewing direction V and specular reflection direction R.
- Since $\emptyset$ is angle between two unit vector V and R we can put $cos\emptyset = V \cdot R$.
- And also for many surfaces $w(\theta)$ is constant so we take specular reflection constant as K_s so equation becomes.
  $I_{spec} = K_s I_l (V \cdot R)^{ns}$
- Vector r is calculated in terms of vector L and N as shown in figure

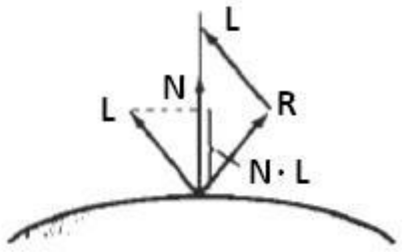


Fig. 6.10:- Calculation of vector R by considering projection onto the direction of the normal vector N.

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

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

- Somewhat simplified phong model is to calculate between half way vectors H and use product of H and N instead of V and R.
- Here H is calculated as follow:

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

Combined Diffuse and Specular Reflections With Multiple Light Sources

- For a single point light source we can combined both diffuse and specular reflection by adding intensity due to both reflection as follows:

  $I = I_{diff} + I_{spec}$

  $I = K_a I_a + K_d I_l (N \cdot L) + K_s I_l (N \cdot H)^{ns}$
- And for multiple source we can extend this equation as follow:

  $I = K_a I_a + \sum_{i=1}^{n} I_l [K_d (N \cdot L) + K_s (N \cdot H)^{ns}]$

Properties of Light

- Light is an electromagnetic wave. Visible light is have narrow band in electromagnetic spectrum nearly 400nm to 700nm light is visible and other bands not visible by human eye.

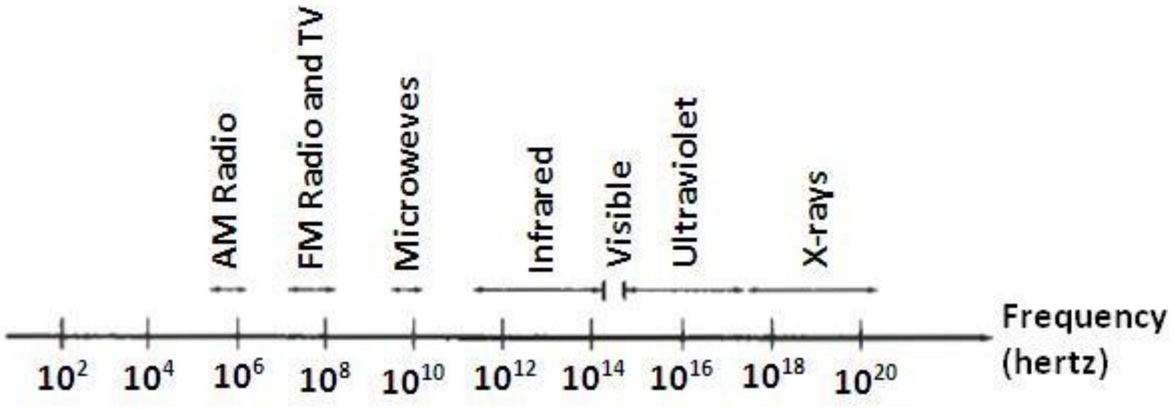


Fig. 6.11:- Electromagnetic spectrum.

- Electromagnetic spectrum shown in figure shows other waves are present in spectrum like microwave infrared etc.
- Frequency value from 4.3 X 10^14 hertz (red) to 7.5 X 10^14 (violet) is visible renge.
- We can specify different color by frequency f or by wavelength λ of the wave.
- We can find relation between f and λ as follows:

  $c = \lambda f$
- Frequency is constant for all the material but speed of the light and wavelength are material dependent.

- For producing white light source emits all visible frequency light.
- Reflected light have some frequency and some are absorbed by the light. This frequency reflected back is decide the color we see and this frequency is called as **dominant frequency (hue)** and corresponding reflected wavelength is called **dominant wavelength.**
- Other property are **purity** and **brightness.** Brightness is perceived intensity of light. Intensity is the radiant energy emitted per unit time, per unit solid angle and per unit projected area of the source.
- **Purity** or **saturation** of the light describes how washed out or how "pure" the color of the light appears.
- Dominant frequency and purity both collectively refers as **chromaticity.**
- If two color source combined to produce white light they are called **complementary color** of each other. For example red and cyan are complementary color.
- Typical color models that are uses to describe combination of light in terms of dominant frequency use three colors to obtain reasonable wide range of colors, called the **color gamut** for that model.
- Two or three colors are used to obtain other colors in the range are called **primary colors.**

XYZ Color Model

- The set of CIE primaries is generally referred to as XYZ or (X, Y, Z) color model.

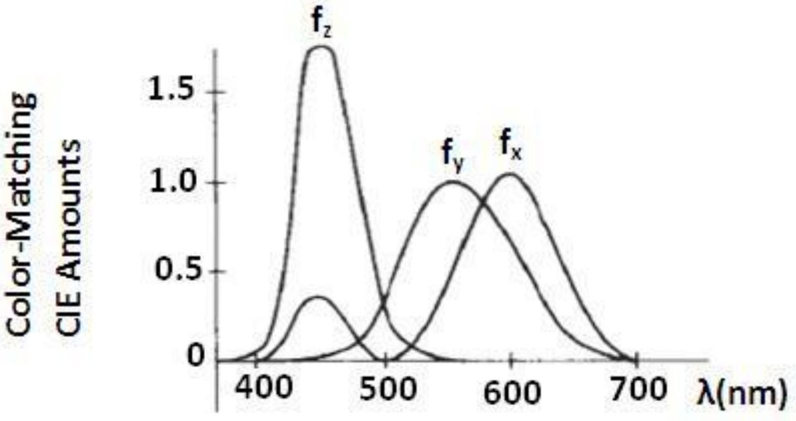


Fig. 6.12:- Amount of CIE primaries needed to display spectral colors.

- X, Y, Z represents vectors in a three dimensional, additive color space.
- Any color C_λ is a combination of three primary colors as
  $C_\lambda = XX + YY + ZZ$
  Where X, Y, Z, is the amount of standard primary need to combine for obtaining color C_λ.
- If we normalize it then.
  $x = \frac{X}{X+Y+Z} \quad y = \frac{Y}{X+Y+Z} \quad z = \frac{Z}{X+Y+Z}$
  With $x + y + z = 1$
- Now we can represent any color with x,y only as z we can find z=1-x-y.
- X, and y are called chromaticity values because they depends only on hue and purity.
- Now if we specify colors with only x, and y values we cannot find amount X, Y, and Z.
- So we specify color with x, y, and Y and rest CIE amount is calculated as:
  $X = \frac{x}{y}Y \quad Z = \frac{z}{y}Y$
  Where z=1-x-y

RGB Color Model

- Based on tristimulus theory of vision our eye perceives color through stimulate one of three visual pigments in the cones of the retina.

- These visual pigments have peak sensitivity at red, green and blue color.
- So combining these three colors we can obtain wide range of color this concept is used in RGB color model.

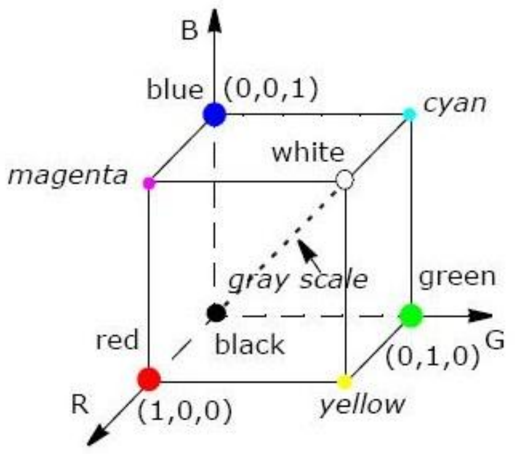


Fig. 6.13:- The RGB color model.

- As shown in figure this model is represented as unit cube.
- Origin represent black color and vertex (1,1,1) is white.
- Vertex of the cube on the axis represents primary color R, G, and B.
- In XYZ color model any color intensity is obtained by addition of primary color.
  $C_\lambda = RR + GG + BB$
- Where R, G, and B is amount of corresponding primary color
- Since it is bounded in between unit cube it's values is very in between 0 to 1 and represented as triplets (R,G,B). For example magenta color is represented with (1,0,1).
- Shades of gray are represented along the main diagonal of cube from black to white vertex.
- For half way gray scale we use triplets (0.5,0.5,0.5).

YIQ Color Model

- As we know RGB monitors requires separates signals for red, green, and blue component of an image but television monitors uses single composite signals.
- For this composite signal NTSC use YIQ color model.
- Here parameter Y is represented as luminance (brightness) while chromaticity information (hue and purity) is specified into I and Q parameter.
- Combination of all red, green, and blue intensities are chosen for Y so black and white television monitors only use signal Y for brightness.
- So largest bandwidth (about 4 MHz) is assigned to Y information signal.
- Parameter I contain orange-cyan hue information that provides the flash-tone shading, and occupies a bandwidth approximately 1.5 MHz.
- Parameter Q carries green-magenta hue information in a bandwidth of about 0.6 MHz.
- An RGB signal can be converted to a television signal using encoder which converts RGB to YIQ values.
- This conversion by transformation is given by:
  $$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.144 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.528 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$
- Similarly reverse of this is performed by decoder and by transformation using inverse of above matrix as.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.956 & 0.620 \\ 1.000 & -0.272 & -0.647 \\ 1.000 & -1.108 & 1.705 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

CMY Color Model

- A color model CMY is used for hardcopy devices as we produce picture by coating a paper with color pigments, we see the color by reflected light a subtractive process.
- When white light is reflected from cyan colored ink the reflected light must have no red component that is red light is absorbed or subtracted by the ink.
- Similarly magenta is subtracting green component.
- Unit cube for CMY model is shown in figure below.

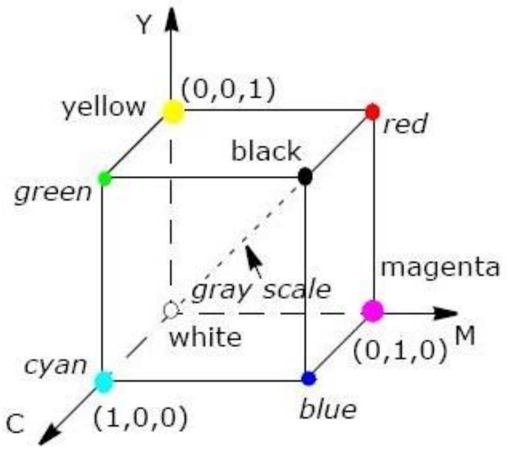


Fig. 6.14:- The CMY color model.

- Point (1,1,1) represents black because all components are subtracts and origin represents white light.
- Gray can be produce among main diagonal by using all three color in equal amount.
- Printing process often use CMY model generates a color points with a collection of four ink dots, one for each primary color C, M, and Y and one dot is black.
- Conversion of RGB to CMY is done by:

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

- And similarly reverse is done by:

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