Unit 5
Three-Dimensional Graphics

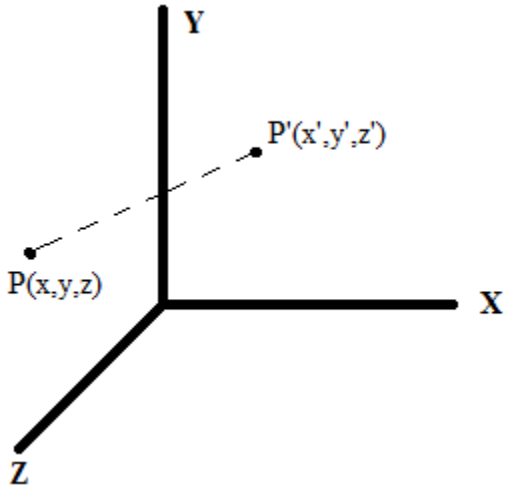
3D Transformations

→ Matrix used in 3D transformation is of order 4×4.

→ Homogeneous co-ordinate for 3D is $[x, y, z, 1]$.

Translation / Shifting

A point can be translated in 3D by adding translation coordinate (tx,ty,tz) to the original coordinate P (x, y, z) to get the new coordinate P' (x', y', z').



Translation matrix:

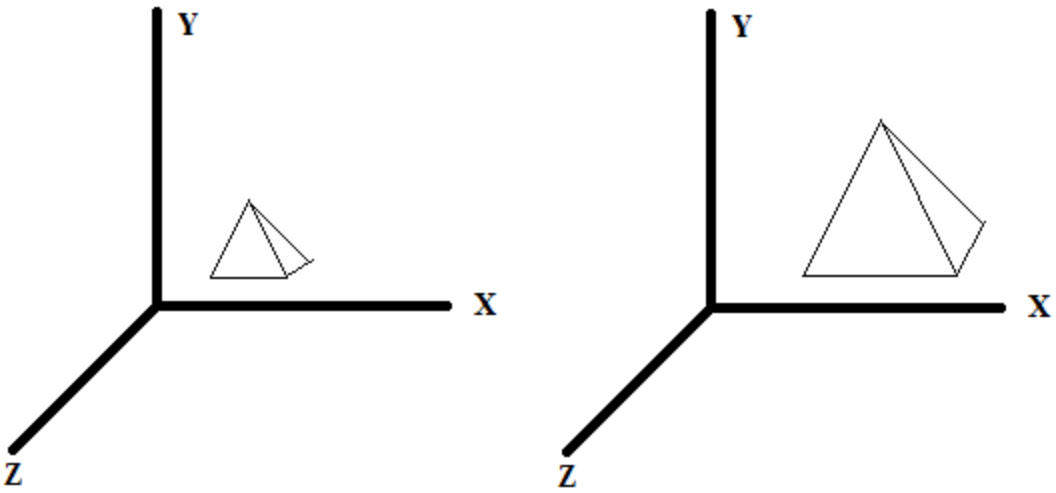
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

i.e. $P' = T(t_x, t_y, t_z).P$

Scaling

→ Changes size of an object relative to the co-ordinate origin.

→ If transformation parameters are not equal, then object gets destroyed. We can preserve the original shape of an object with uniform scaling$(s_x = s_y = s_z)$.



Transformation equations for scaling at origin:

$x' = x.s_x$

$y' = y.s_y$

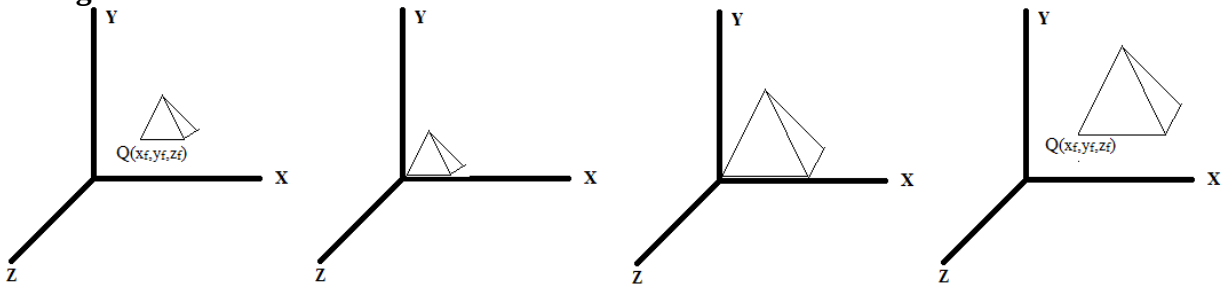
$z' = z.s_z$

where s_x, s_y, s_z are scaling factors which scale object in x, y, z direction respectively.

Scaling Matrix:

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

Scaling about Fixed-Point



Scaling with respect to a selected fixed position (x_f, y_f, z_f) can be represented with the following transformation sequence:

→ Translate the fixed point to the origin.

→ Scale the object relative to the co-ordinate origin using equation:

$x' = x.s_x$

$y' = y.s_y$

$z' = z.s_z$

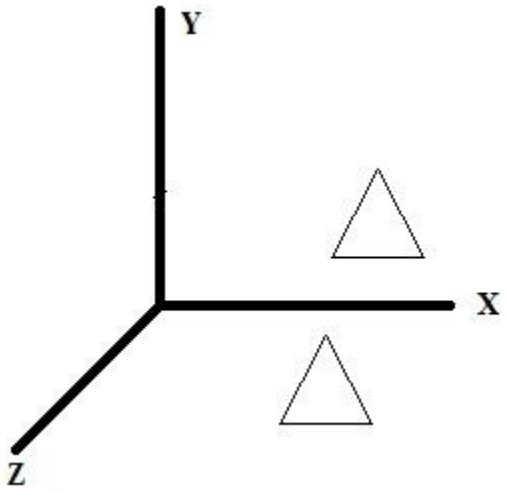
→ Translate the fixed point back to its original position.

$$\text{C.M.} = T^{-1}.S.T = \begin{bmatrix} 1 & 0 & 0 & x_f \\ 0 & 1 & 0 & y_f \\ 0 & 0 & 1 & z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_f \\ 0 & 1 & 0 & -y_f \\ 0 & 0 & 1 & -z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation

3D rotation is not same as 2D rotation. In 3D rotation, we have to specify the angle of rotation along with the axis of rotation. We can perform 3D rotation about X, Y, and Z axes.

i. Rotation about x-axis



Transformation equations are:

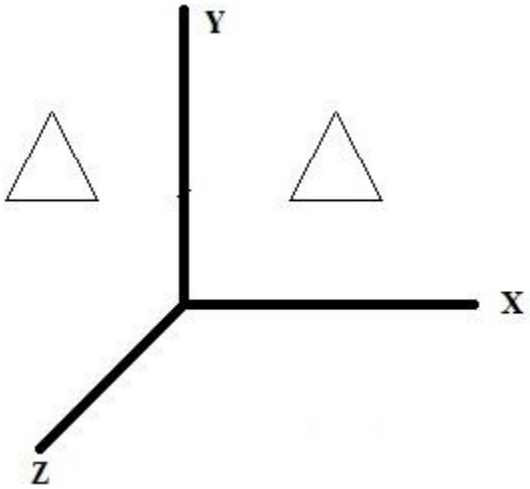
$$x' = x$$
$$y' = y\cos\theta - z\sin\theta$$
$$z' = y\sin\theta + z\cos\theta$$

i.e. $P' = R_x(\theta).P$

Transformation Matrix:

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

ii. Rotation about y-axis



Transformation equations are:

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

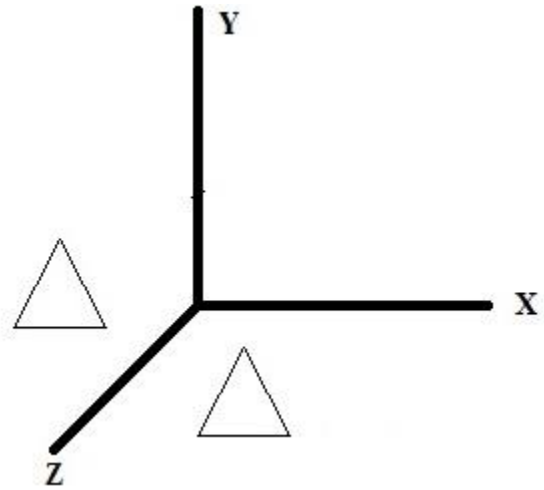
$$z' = z\cos\theta - x\sin\theta$$

i.e. $P' = R_y(\theta).P$

Transformation Matrix:

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

iii. Rotation about z-axis



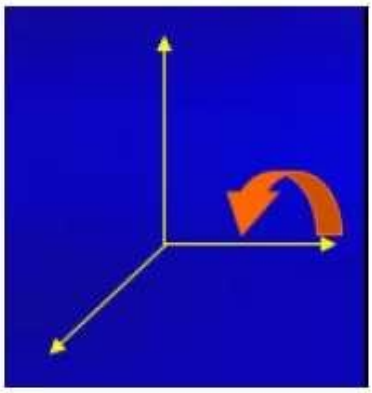
Transformation equations are:

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

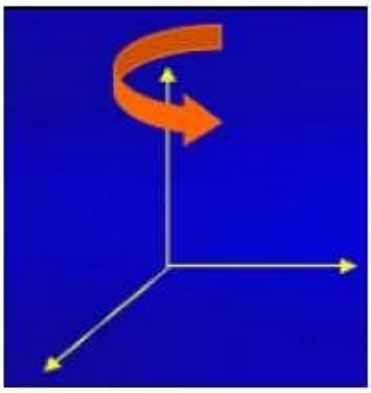
i.e. $P' = R_z(\theta).P$

Transformation Matrix:

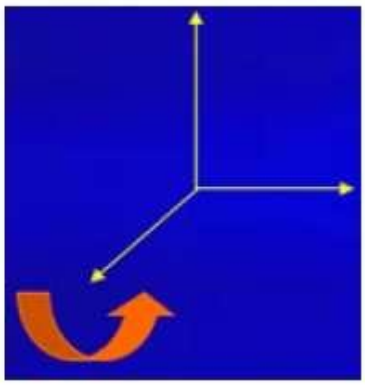
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Rotation about x-axis



Rotation about y-axis



Rotation about z-axis

iv. Rotation about an axis parallel to one of the co-ordinate axis

→ Translate the object so that rotation axis coincides with parallel co-ordinate axis.

→ Perform specified rotation about that axis.

→ Translate object back to its original position.

$P' = CM.P$ Where $CM = T^{-1}.R(\theta).T$

Reflection

i. Reflection about x-axis / Reflection about yz-plane

$$P' = Rf_x.P$$

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

ii. Reflection about x-axis / Reflection about yz-plane

$$P' = Rf_y.P$$

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

iii. Reflection about x-axis / Reflection about yz-plane

$$P' = Rf_z.P$$

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

Shear

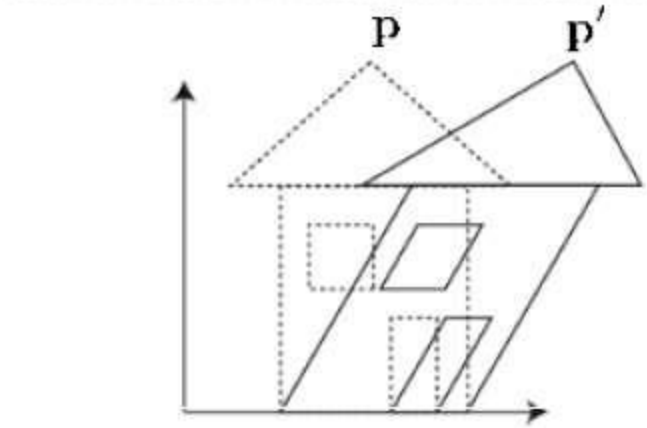
→ Shearing transformations are used to modify objects' shape.

→ Shearing factors are sh_x, sh_y and sh_z in x, y and z direction respectively.

→ In space, one can push in two co-ordinate axes direction keeping the third axis fixed.

→ Like in 2D shear, we can shear an object along the X-axis, Y-axis, or Z-axis in 3D.

Shear



i. Shear in x and y direction keeping z co-ordinate same (along z-axis)

Transformation equations are:

$$x' = x + sh_x z$$
$$y' = y + sh_y . z$$
$$z' = z$$

i.e. $P' = sh_{xy} . P$

Transformation Matrix:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & sh_x & 0 \\ 0 & 1 & sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

It alters the x and y co-ordinates value by an amount that is proportional to the z-axis while leaving z-co-ordinate.

ii. Shear in y and z direction keeping x co-ordinate same (along x-axis)

Transformation equations are:

$$x' = x$$
$$y' = y + sh_y x$$
$$z' = z + sh_z . x$$

i.e. $P' = sh_{xy} . P$

Transformation Matrix:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ sh_y & 1 & 0 & 0 \\ sh_z & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

It alters the y and z co-ordinates value by an amount that is proportional to the x-axis while leaving x-co-ordinate.

iii. Shear in z and x direction keeping y co-ordinate same (along y-axis)

Transformation equations are:

$$x' = x + sh_x . y$$

$$y' = y$$
$$z' = z + sh_z.y$$

i.e. $P' = sh_{xy}.P$

Transformation Matrix:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & sh_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

It alters the z and x co-ordinates value by an amount that is proportional to the y-axis while leaving y-co-ordinate.

3D Viewing pipeline:

The steps for computer generation of a view of 3D scene are analogous to the process of taking photograph by a camera. For a snapshot, we need to position the camera at a particular point in space and then need to decide camera orientation. Finally, when we snap the shutter, the scene is cropped to the size of "window" (aperture) of the camera and the light from the visible surfaces is projected into the camera film.

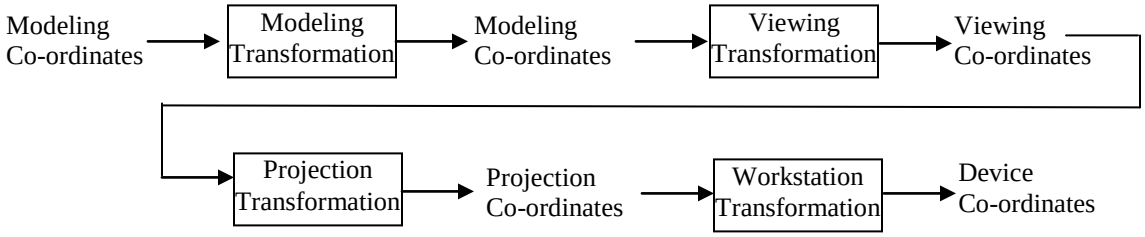


Fig: General three-dimensional transformation pipeline, from modeling coordinates to final device coordinates

→ Once the scene has been modeled, WC positions are converted to VC.
→ The VC system is used in graphics system as a reference for specifying the observer viewing position and the position of the projection plane.
→ Projection operations are performed to convert VC description of a scene to co-ordinate positions on the projection plane.
→ The projection plane is then mapped to output device.

Viewing Co-ordinates

→ Views of a scene can be generated, given the spatial position, orientation and aperture size of the camera.

Specifying the View Plane

→ Choosing a particular view for a scene by establishing VC system.
→ A *view plane* or *projection plane* is then set perpendicular to the z_v axis.
→ WC positions in the scene are transformed to VC, the VC are projected onto the view plane.

Establishing the VC reference frame

→ First, pick a WC position called ***view reference point***, the origin of our viewing co-ordinate system.
→ Next, the positive direction for the viewing z_v axis, and the orientation of the view plane is selected by specifying the **view plane normal plane, $\vec{N}$**.
→ We choose a WC position and this point establishes the direction for N relative either to the world origin or to the viewing co-ordinate origin.
→ Finally, we choose the up direction for the view by specifying a vector $\vec{V}$, called view-up vector.
→ This vector is used to establish the positive direction for Y-axis.

Fig: Orientations of the view plane for specified normal vector coordinates relative to the world origin, Position (1,0,0) orients the view plane as in (a), while (1,0,1) gives the orientation in (b).

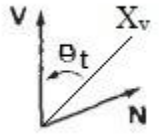


Fig: Specifying the view-up vector with a twist angle θ_t

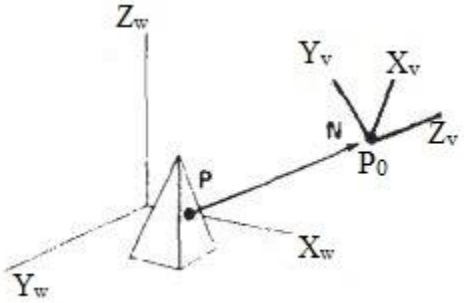


Fig: Orientation of the view plane for a specified look at point P, relative to the viewing coordinate origin P_0

Transformation from World co-ordinates to Viewing co-ordinates

→ Before object descriptions can be projected to the view plane, they must be transferred to viewing coordinates.

Steps:

i. Translate the view reference point to the origin of the world-coordinate system.
ii. Apply rotations to align the x_v, y_v and z_v axes with the world x_w, y_w and z_v axes respectively.

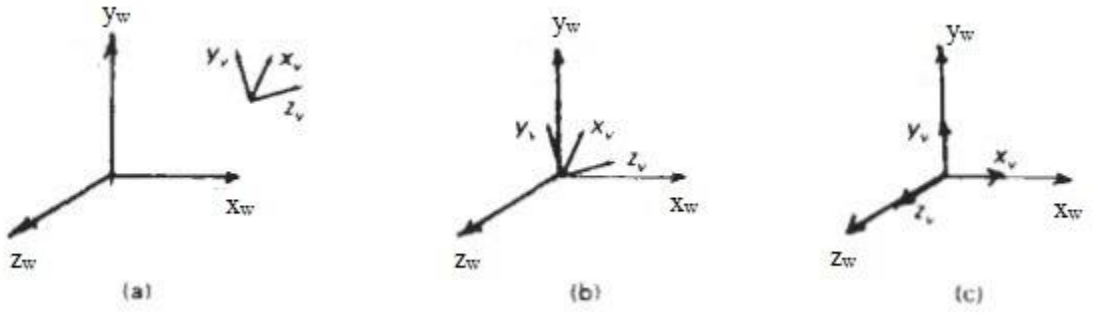
→ If the view reference point is specified at world position (x_0, y_0, z_0), this point is translated to the world origin with the matrix transformation.

$$T = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

→ The rotation sequence can require up to three coordinate-axis rotations, depending on the direction we choose for $\vec{N}$.

→ If $\vec{N}$ is not aligned with any WC axis, we can superimpose the viewing and world systems with the transformation sequence $R_z.R_y.R_x$.

- i.e. First, rotate around the world x_w axis to bring z_v into the x_wz_w plane.
- Then, rotate around the world y_w axis to align the z_w and z_v axes.
- Finally, rotate about z_w axis to align y_w and y_v axes.



→ Another method for generating the rotation-transformation matrix is to calculate unit uvn vectors and form the composite rotation matrix directly.

→ Given vectors $\vec{N}$ and $\vec{V}$, these unit vectors are calculated as

$$\hat{n} = \frac{\vec{N}}{|\vec{N}|} = (n_1, n_2, n_3)$$

$$\hat{u} = \frac{\vec{V} \times \vec{N}}{|\vec{V} \times \vec{N}|} = (u_1, u_2, u_3)$$

$$\hat{v} = \hat{n} \times \hat{u} = (v_1, v_2, v_3)$$

→ This method also automatically adjusts the direction for $\vec{V}$ so that $\hat{v}$ is perpendicular to $\hat{n}$.

→ The composite rotation matrix for the viewing transformation is then

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

→ It transforms $\hat{u}$ onto the world x_w axis, $\hat{v}$ onto the y_w axis, and $\hat{n}$ onto the z_w axis.

→ The complete world-to-viewing coordinate transformation matrix is obtained as the matrix product

$$M_{WC,VC} = R.T$$

→ This transformation is then applied to coordinate descriptions of objects in the scene to transfer them to the viewing reference frame.

Projections

→ Once world co-ordinate description of the objects in a scene are converted to viewing co-ordinates, we can project the three dimensional objects onto the two dimensional view plane.

→ There are two basic projection methods:

Parallel Projection: In parallel projection, co-ordinate positions are transformed to the view plane along parallel lines.

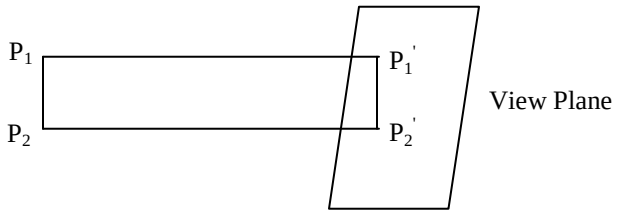


Fig: Parallel projection of an object to the view plane

Prospective projection: In prospective projection, object positions are transformed to the view plane along lines that converge to a point called ***projection reference point*** (centre of projection). The projected view of an object is determined by calculating the intersection of the projection lines with the view plane.

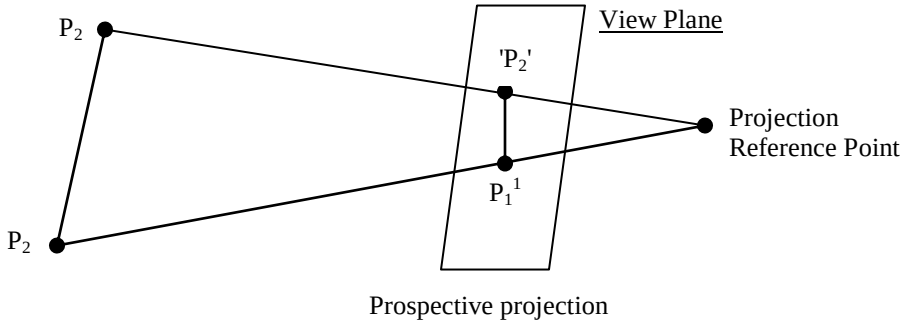
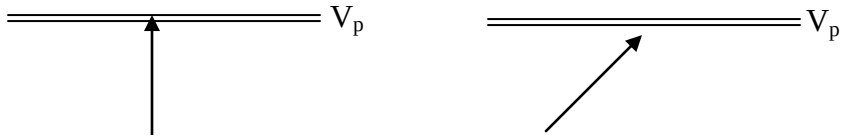


Fig: Perspective projection of an object to the view plane

A parallel projection preserve relative proportions of objects and this is the method used in drafting in drafting to produce scale drawing of three-dimensional objects. Accurate view of various sides of 3D object is obtained with parallel projection. But it does not given a realistic appearance of a 3D-object.
A prospective projection, on the other hand, produces realistic views but does not preserve relative proportions. Projections of distance objects from view plane are smaller than the projections of objects of the same size that are closer to the projection place.

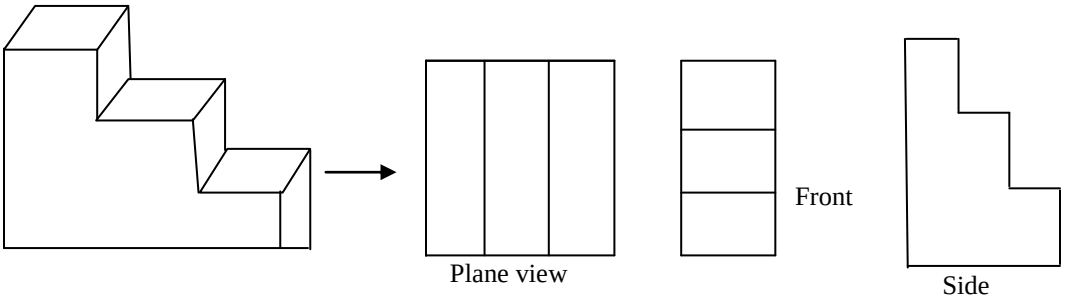
Parallel Projection: We can specify parallel projection with a projection vector that specifies the direction of projection line. When the projection lines are perpendicular to view plane, the projection is orthographic parallel projections.
It projection line are not parallel to view plane then it is oblique parallel projection.



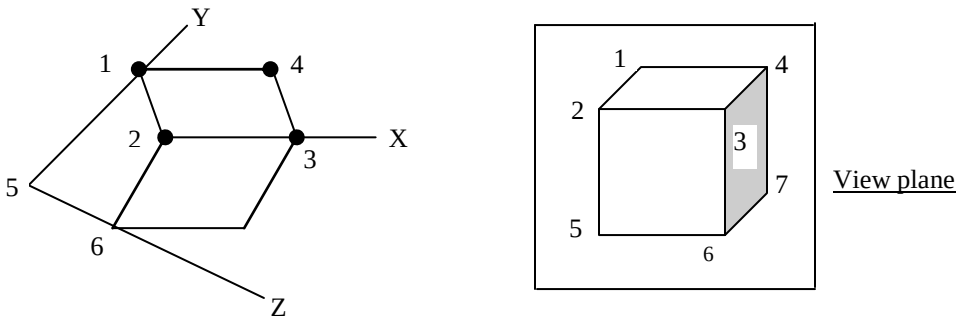
Orthographic parallel projection Oblique parallel projection

- Orthographic projections are most often used to produce the front, side, and top views of an object. Front, side and rear orthographic are called elevations and the top orthographic view

of object is known as plan view. Engineering and Architectural drawings commonly employ these orthographic projections.



We also form orthographic projections that display more than one face of an object. Such views are called ***axonometric orthographic projections***. The most commonly used axonometric projection is the ***isometric projection***.



Orthographic parallel projection

→ If the view plane is placed at position z_{vp} along the z_v axis, then any point (x,y,z) in viewing co-ordinate is transformed to projection co-ordinate as

$$x_p = x$$
$$y_p = y$$

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

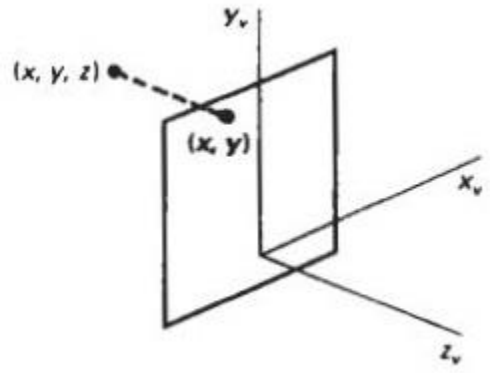


Fig: Orthographic projection of a point onto a viewing plane

Oblique parallel projection

→ Obtained by projecting points along parallel lines that are not perpendicular to the projection plane.
→ Often specified with two angles α and $\emptyset$.
→ Point (x, y, z) is projected to position (x_p, y_p) on the view plane.
→ Orthographic projection coordinates on the plane are (x, y).
→ The oblique projection line from (x, y, z) to (x_p, y_p) makes an angle α with the line on the projection plane that joins (x_p, y_p) and (x, y).
→ This line, of length L, is at an angle $\emptyset$ with the horizontal direction in the projection plane.

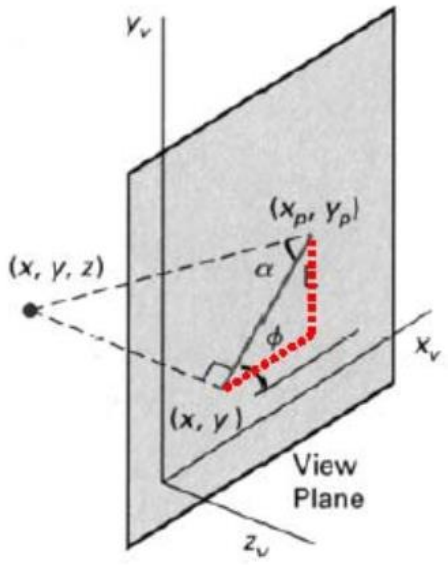


Fig: Oblique projection of co-ordinate position (x, y, z) to position (x_p, y_p) on the view plane.

→ Expressing projection co-ordinates in terms of x, y, L and $\emptyset$,

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

→ Also, $\boldsymbol{\tan\alpha = \frac{z}{L}}$. So, $\boldsymbol{L = \frac{z}{\tan\alpha} = zL_1}$, where $\boldsymbol{L_1 = \frac{1}{\tan\alpha}}$

→ So, oblique parallel projection equations are

$$x_p = x + z(L_1\cos\emptyset)$$
$$y_p = y + z(L_1\sin\emptyset)$$

→ Transformation matrix for producing oblique parallel projection onto the x_vy_v plane can be written as

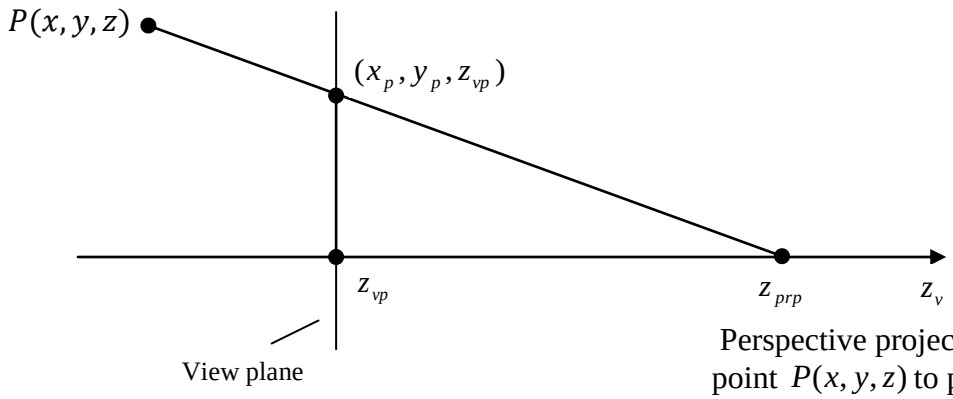
$$M_{parallel} = \begin{bmatrix} 1 & 0 & L_1\cos\emptyset & 0 \\ 0 & 1 & L_1\sin\emptyset & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

→ Orthographic projection is at $L_1 = 0$ and $\alpha = 90^0$.

Perspective projection:

→ To obtain a prospective projection of a three-dimensional object, we transform points along projection lines that meet at a point called projection reference point.

→ 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 is figure.



Perspective projection of a point $P(x, y, z)$ to position (x_p, y_p, z_{vp}) on the view plane.

→ We can write equations describing co-ordinates positions along this prospective projection line in parametric form as

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

→ Parameter u takes value from 0 to 1.

→ If $u = 0$, we are at position $P = (x, y, z)$.

→ If $u = 1$, we have projection reference point $(0,0, z_{prp})$.

→ On the view plane, $z' = z_{vp}$ then

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

→ Substituting these value in eq^n for x', y'.

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

→ Similarly,

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

where $dp = z_{prp} - z_{vp}$ is the distance of the view plane from projection reference point.

→ Using 3-D homogeneous Co-ordinate representation, we can write perspective projection transformation matrix as

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & z_{vp}/dp & \dfrac{-z_{vp}}{z_{prp}/dp} \\ 0 & 0 & 1/dp & z_{prp}/dp \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

In this representation, the homogeneous factor $h = \frac{z - z_{prp}}{dp}$

Projection co-ordinates,

$$xp = xh/h, yp = yh/h$$

There are special case for prospective transformation.

When $z_{prp} = 0$;

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

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

Some graphics package, the projection point is always taken to be viewing co-ordinate origin. In this ease, $z_{prp} = 0$

$$\therefore x_p = x(\frac{zvp}{z}) = x(\frac{1}{z/zvp})$$

$$yp = y(\frac{zvp}{z}) = y\ (\frac{1}{z/zvp})$$

Hidden Line and Hidden Surface Removal Techniques

Visible Surface Detection or *Hidden Surface Removal* is major concern for realistic graphics for identifying those parts of a scene that are visible from a chosen viewing position. Numerous algorithms have been devised for efficient identification of visible objects for different types of applications. Some methods require more ***memory***, some involve more ***processing time***, and some apply only to ***special types of objects***. Deciding upon a method for a particular application can depend on such factors as the complexity of the scene, type of objects to be displayed, available equipment, and whether static or animated displays are to be generated.

Visible surface detection methods are broadly classified according to whether they deal with objects or with their projected images.

These two approaches are:

- **Object-Space methods:** An object-space method compares objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible. It deals with object definition directly. E.g. Back-Face Detection method.

- **Image-Space methods**: Visibility is decided point by point at each pixel position on the projection plane. It deals with projected image of an object. E.g. Z-Buffer (Depth-Buffer) method, A-Buffer method, Scan-line method.

Most visible surface detection algorithm use image-space-method but in some cases object space methods can also be effectively used.

Back-Face Detection method

→ Used for removing hidden surface in 3D object.

→ A fast and simple object-space method for identifying the back faces of a polyhedron.

→ Based on "inside-outside" tests.

→ A point (x, y, z) is "inside" a polygon surface with plane parameters A, B, C, and D if When an inside point is along the line of sight to the surface, the polygon must be a back face (we are inside that face and cannot see the front of it from our viewing position).

→ Consider normal vector **N** to a polygon surface, which has Cartesian components (A, B, C).

→ If V is a vector in the viewing direction from the eye (or "camera") position, then this polygon is a back face if

$$\mathbf{V.N > 0}$$

→ If V.N < 0, the surface is visible.

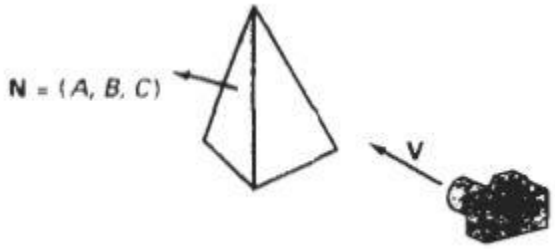


Fig: Vector V in the viewing direction and a back-face normal vector N of a polyhedron

→ If object descriptions are converted to projection coordinates and your viewing direction is parallel to the viewing z_v-axis, then,

$$V = (0, 0, V_z) \text{ and } V.N = V_Z C$$

→ So, we only need to consider the sign of C, the z-component of the normal vector **N**.

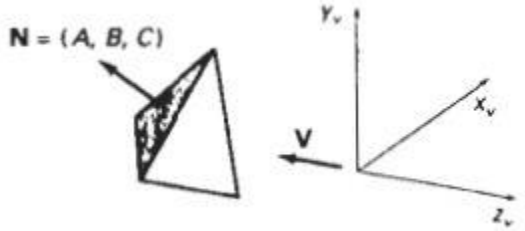


Fig: A polygon surface with plane parameter C < 0 in a right-handed viewing coordinate system is identified as a back face when the viewing direction is along the negative z_v axis.

→ If the viewing direction is along the negative Z_v-axis, the polygon is a back face if C < 0. Also, we cannot see any face whose normal has z component, C = 0, since your viewing

direction is towards that polygon. Thus, in general, we can label any polygon as a back face if its normal vector has a z component value

C <= 0

→ Alternatively,

- Make the center of object at origin.
- Calculate value of D of the plane by taking three points in anti-clockwise direction.
- If D < 0, surface is visible.
- If D > 0, surface is invisible.

→ Does not work well for

- Overlapping from faces due to
  - — Multiple objects
  - — Concave objects
- Non-closed objects
- Non-polygonal models

Q. Consider a rectangular pyramid defined by A(1,0,0), B(0,0,-1), C(-1,0,0), D(0,0,1) and E(0,1,0). If the observer is at

i) (5,1,5)

ii) (0,0.5,0)

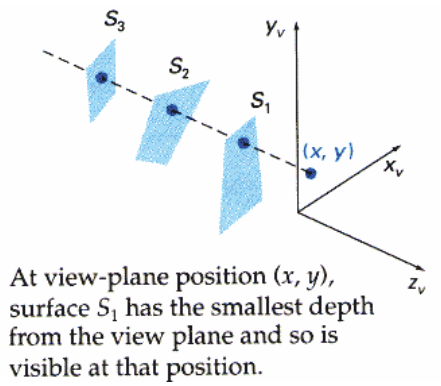
Is the face AED visible?

Depth-Buffer (Z-Buffer) Method

Depth Buffer Method is the commonly used *image-space method* for detecting visible surface. It compares surface depths at each pixel position on the projection plane. It is called z-buffer method since object depth is usually measured from the view plane along the z-axis of a viewing system.

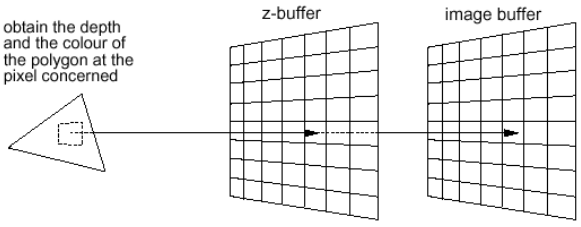
Each surface of scene is processed separately, one point at a time across the surface. The method is usually applied to scenes containing only polygon surfaces, because depth values can be computed very quickly and method is easy to implement. This method can be applied also to non planer surfaces.

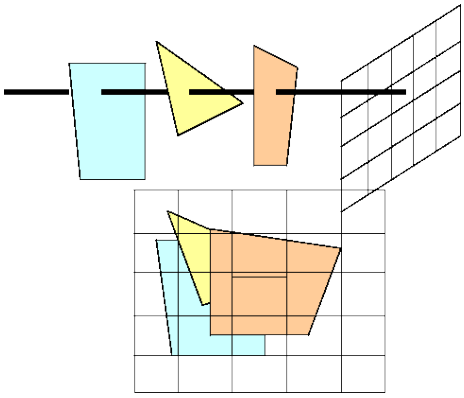
With object description converted to projection co-ordinates, each (x, y, z) position on polygon surface corresponds to the orthographic projection point (x, y) on the view plane. Therefore for each pixel position (x, y) on the view plane, object depth is compared by z-values.



At view-plane position (x, y), surface S_1 has the smallest depth from the view plane and so is visible at that position.

Three surfaces at varying distance from view plane x_vy_v, the projection along (x, y) surface S_1 is closest to the view-plane, so surface intensity value of S_1 at (x, y) is saved.





In Z-buffer method, two buffers area are required.

- **Depth buffer (z-buffer)**: stores the depth value for each (x, y) position as surfaces are processed.
- **Refresh buffer (Image buffer or Frame Buffer)**: stores the intensity value for each position.

Initially, all the positions in depth buffer are set to 0, and refresh buffer is initialized to background color. Each surfaces listed in polygon table are processed one scan line at a time, calculating the depth (z-val) for each position (x, y). The calculated depth is compared to the value previously stored in depth buffer at that position. If calculated depth is greater than stored depth value in depth buffer, new depth value is stored and the surface intensity at that position is determined and placed in refresh buffer.

Algorithm: Z-buffer

1. Initialize depth buffer and refresh buffer so that for all buffer position (x, y)

   depth (x, y) = 0, **refresh (x, y)** = $I_{background.}$
2. For each position on each polygon surface, compare depth values to previously stored value in depth buffer to determine visibility.
   - Calculate the depth ***z*** for each (x, y) position on polygon
   - If $z >$ depth (x, y) then

     depth (x, y) = z, **refresh (x, y)** = $I_{surface}$(x, y)

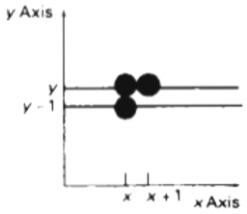
where, $I_{background}$ = Intensity value for background

$I_{surface}$ (x, y) = Intensity value for surface at pixel position (x, y) on projected plane.

3. After all pixels and surfaces are processed (compared), the depth buffer contains the depth value of the visible surface and refresh buffer contains the corresponding intensity values for those surfaces. Draw the object.

The depth values of the surface position (x, y) are calculated by plane equation of surface.

$$Z = \frac{-Ax - By - D}{C}$$



Let Depth Z' at position (x+1, y) along the scan line is,

$$Z' = \frac{-A(x+1) - By - D}{C}$$

$$\Rightarrow Z' = Z - A/C \qquad (1)$$

$-A/C$ is constant for each surface so succeeding depth value across a scan line are obtained from preceding values by single addition.

Drawback: It deals only with opaque surface, but not with transparent surface. It can only find visible surface of each pixel position. A-buffer described below solves this problem.

A-Buffer Method

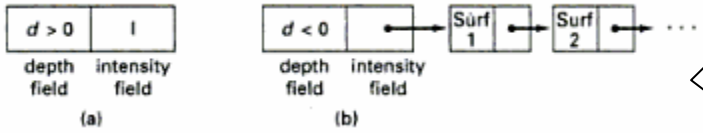
An extension of the ideas in the depth-buffer method is the A-buffer method. A-buffer expands z-buffer so that each position in the buffer can reference a linked list of surfaces. So, more than one surface intensity can be taken into consideration at each pixel.

Each pixel position in the A-buffer has two fields:

→ Depth field: stores a +ve or –ve real number.
- If +ve, single surface contributes to pixel intensity.
- If -ve, multiple surfaces contribute to pixel intensity.

→ Intensity field: stores surface intensity value or a pointer value
- Surface intensity if single surface (stores the RGB components of the surface color at that point and percent of pixel coverage)
- Pointer value if multiple surfaces.

Viewing an opaque surface through a transparent surface requires multiple surface intensity contributions for pixel positions



Organization of an A-buffer pixel position:
a) Single-surface overlap of the corresponding pixel area (d >= 0)
b) Multiple surface overlap (d < 0)

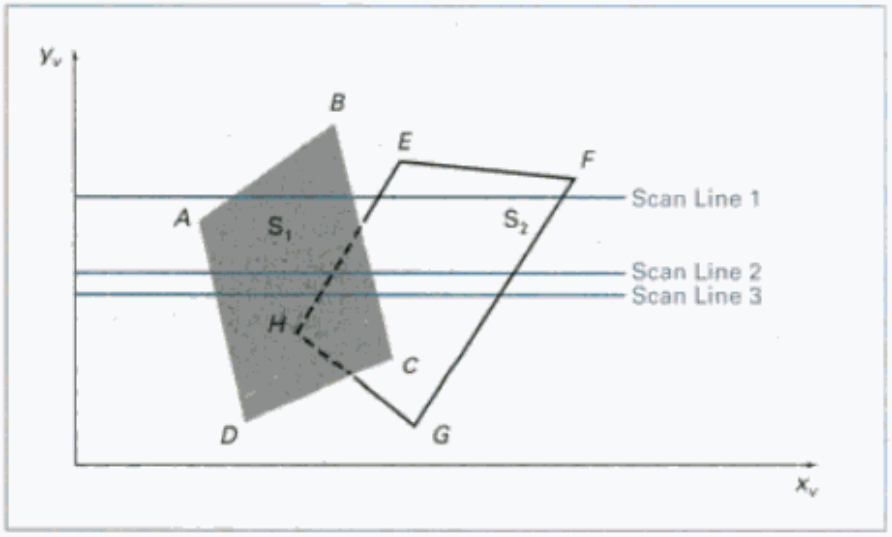
If depth field >= 0, the number stored at that position is the depth of a single surface overlapping the corresponding pixel area. The intensity field then stores the RGB components of the surface color at that point and the percent of pixel coverage.

If depth field < 0, this indicates multiple surface contributions to the pixel intensity. The intensity field then stores a pointer to a linked list of surface data. Data for each surface includes:

- RGB intensity components
- Opacity parameter (% of transparency)
- Depth
- Percent of area coverage
- Surface identifier
- Other surface-rendering parameters
- Pointer to next surface

Scan-Line Method

- This is image-space method for removing hidden surface which is extension of the scan line polygon filling for polygon interiors. Instead of filling one surface we deal with multiple surfaces here.
- In this method, as each scan line is processed, all polygon surfaces intersecting that line are examined to determine which are visible. Across each scan line, depth calculations are made for each overlapping surface to determine which is nearest to the view plane. When the visible surface has been determined, the intensity value for that position is entered into the image buffer.



Scan line crossing the projection of two surfaces s1 and s2, in the view plane. Dashed lines indicate the boundaries of hidden surfaces.

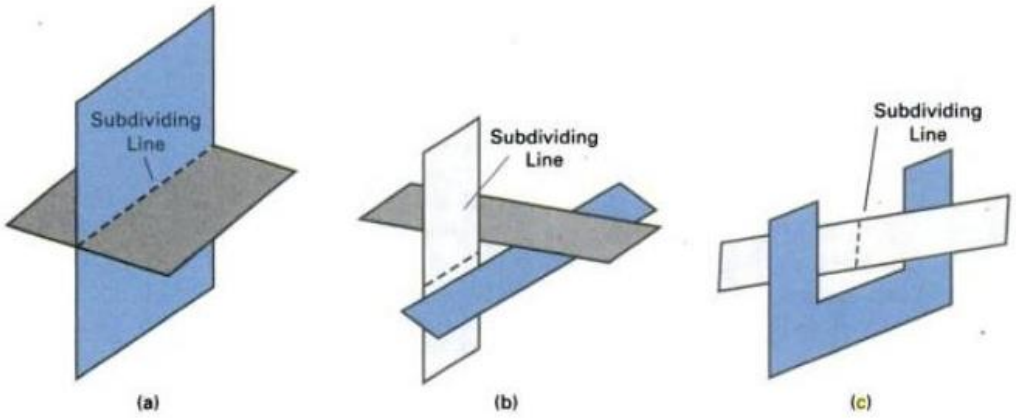
Figure above illustrates the scan-line method for locating visible portions of surfaces for pixel positions along the line. The active list for scan line 1 contains information from the edge table for edges AB, BC, EH, and FG. For positions along this scan line between edges AB and BC, only the flag for surface S_1 is on.

Therefore, no depth calculations are necessary, and intensity information for surface S_1 is entered from the polygon table into the refresh buffer. Similarly, between edges EH and FG, only the flag for surface S_2 is on. No other positions along scan line 1 intersect surfaces, so the intensity values in the other areas are set to the background intensity. The background intensity can be loaded throughout the buffer in an initialization routine.

For scan lines 2 and 3, the active edge list contains edges *AD, EH, BC,* and *FG.* Along scan line 2 from edge *AD* to edge *EH,* only the flag for surface S_1 is on. **But between edges EH and *BC,***

the flags for both surfaces are on. In this interval, depth calculations must be made using the plane coefficients for the two surfaces. For this example, the depth of surface S_1 is assumed to be less than that of S_2, so intensities for surface S_1 are loaded into the refresh buffer until boundary BC is encountered. Then the flag for surface S_1 goes off, and intensities for surface S_2 are stored until edge FG is passed.
We can take advantage of coherence along the scan lines as we pass from one scan line to the next. In Fig., scan line 3 has the same active list of edges as scan line 2. Since no changes have occurred in line intersections, it is unnecessary again to make depth calculations between edges *EH* and *BC***.** The two surfaces must be in the same orientation as determined on scan line 2, so the intensities for surface S_1 can be entered without further calculations.

Any number of overlapping polygon surfaces can be processed with this scan-line method. Flags for the surfaces are set to indicate whether a position is inside or outside, and depth calculations are performed when surfaces overlap. When these coherence methods are used, we need to be careful to keep track of which surface section is visible on each scan line. This works only if surfaces do not cut through or otherwise cyclically overlap each other.



If any kind of cyclic overlap is present in a scene, we can divide the surfaces to eliminate the overlaps. The dashed lines in this figure indicate where planes could be subdivided to form two distinct surfaces, so that the cyclic overlaps are eliminated.

Introduction to Non-planar Surfaces

1. Spline Representation

A Spline is a flexible strips used to produce smooth curve through a designated set of points. A curve drawn with these set of points is spline curve. Spline curves are used to model 3D object surface shape smoothly.
Mathematically, spline are described as piece-wise cubic polynomial functions. In computer graphics, a spline surface can be described with two set of orthogonal spline curves. Spline is used in graphics application to design and digitalize drawings for storage in computer and to specify animation path. Typical CAD application for spline includes the design of automobile bodies, aircraft and spacecraft surface etc.

Interpolation and approximation spline

- Given the set of control points, the curve is said to **interpolate** the control point if it passes through each points.

- If the curve is fitted from the given control points such that it follows the path of control point without necessarily passing through the set of point, then it is said to **approximate** the set of control point.

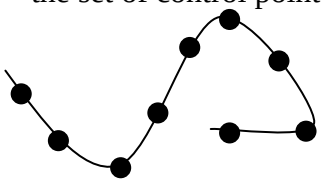


Fig: *A* set of nine control point **interpolated** with piecewise continuous polynomial sections.

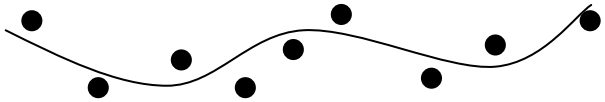


Fig: *A* set of nine control points **approximated** with the piecewise continuous polynomial sections

Spline Specifications

There are three equivalent methods for specifying a particular spline representation:

A. ***Boundary conditions*:** We can state the set of boundary conditions that are imposed on the spline.

For illustration, suppose we have parametric cubic polynomial representation for the x-coordinate along the path of a spline section:

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x \qquad 0 \le u \le 1$$

Boundary conditions for this curve might be set, for example, on the endpoint coordinates *x(0)* and x(1) and on the parametric first derivatives at the endpoints *x'(0)* and x'(1). These four boundary conditions are sufficient to determine the values of the four coefficients a_x, b_x, c_x, and d_x.

B. ***Characterizing matrix:*** We can state the matrix that characterizes the spline.

From the boundary condition, we can obtain the characterizing matrix for spline. Then the parametric equation can be written as:

$$x(u) = [u^3\ u^2\ u\ 1] \begin{bmatrix} a_x \\ b_x \\ c_x \\ d_x \end{bmatrix}$$

$$= \mathbf{U} \cdot \mathbf{C}$$

Where U is the row matrix of powers of parameter u**,** and C is the coefficient column matrix.

C. Blending Functions: We can state the set of blending functions (or basis functions) that determine how specified geometric constraints on the curve are combined to calculate positions along the curve path.

From matrix representation in B, we can obtain polynomial representation for coordinate x in terms of the geometric constraint parameters:

$$x(u) = \sum_{k=0}^{3} g_k . BF_k(u)$$

where g_k are the constraint parameters, such as the control-point coordinates and slope of the curve at the control points, and $BF_k(u)$ are the polynomial blending functions.

Cubic spline

- It is most often used to set up path for object motions o to provide a representation for an existing object or drawing. To design surface of 3D object, any spline curve can be represented by piece-wise cubic spline.
- Cubic polynomial offers a reasonable compromise between flexibility and speed of computation. Cubic spline requires less calculation with comparison to higher order polynomials and requires less memory. Compared to lower order polynomial cubic spline are more flexible for modeling arbitrary curve shape.

Given a set of control points, cubic interpolation spines are obtained by fitting the input points with a piecewise cubic polynomial curve that passes through every control points.

Suppose, we have n+1 control points specified with co-ordinates.

$$p_k = (x_k, y_k, z_k), \quad k = 0,1,2,3\ldots\ldots\ldots\ldots\ldots\ldots n$$

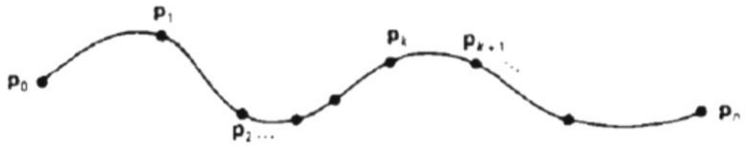


Fig: A piecewise continuous cubic-spline interpolation of $n + 1$ control points

We can describe the parametric cubic polynomial that is to be fitted between each pair of control points with the following set of parametric equations.

$$\begin{aligned} x(u) &= a_x u^3 + b_x u^2 + c_x u + d_x \\ y(u) &= a_y u^3 + b_y u^2 + c_y u + d_y \qquad (0 \le u \le 1) \\ z(u) &= a_z u^3 + b_z u^2 + c_z u + d_z \end{aligned}$$

For each of these three equations, we need to determine the values of the four coefficients a, b, c, and d in the polynomial representation for each of the n curve sections between the $n + 1$ control points. We do this by setting enough boundary conditions at the "joints" between curve sections so that we can obtain numerical values for all the coefficients.

2. Bezier curve and surface

This is spline approximation method, developed by the French Engineer Pierre Bezier for use in the design of automobile body. Beziers spline has a number of properties that make them highly useful and convenient for curve and surface design. They are easy to implement. For this reason, Bezier spline is widely available in various CAD systems.

Beziers curves

In general, Bezier curve can be fitted to any number of control points. The number of control points to be approximated and their relative position determine the degree of Bezier polynomial.

The Bezier curve can be specified with boundary condition, with characterizing matrix or blending functions. But for general Bezier curves, blending function specification is most convenient.

Suppose we have n+1 control points: $p_k(x_k, y_k, z_k)$, $0 \le k \le n$. These co-ordinate points can be blended to produce the following position vector P(u) which describes path of an approximating Bezier polynomial function p_0 and p_n.

$$P(u) = \sum_{k=0}^{n} p_k . BEZ_{k,n}(u), \qquad 0 \le u \le 1 \text{ ---------- } 1$$

The Bezier belending function $BEZ_{k,n}(u)$ are the Bernstein polynomial:

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

Where C(n, k) are the binomial coefficients:

$$C(n, k) = n! / k!(n-k)!$$

The vector equation (1) represents a set of three parametric equations for individual curve coordinates.

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

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

$$z(u) = \sum_{k=0}^{n} z_k . BEZ_{k,n}(u)$$

As a rule, a Bezier curve is a polynomial of degree one less than the number of control points used: Three points generate a parabola, four points a cubic curve, and so forth.

Fig below demonstrates the appearance of some Bezier curves for various selections of control points in the xy-plane *(z* = 0), with certain control-point placements:

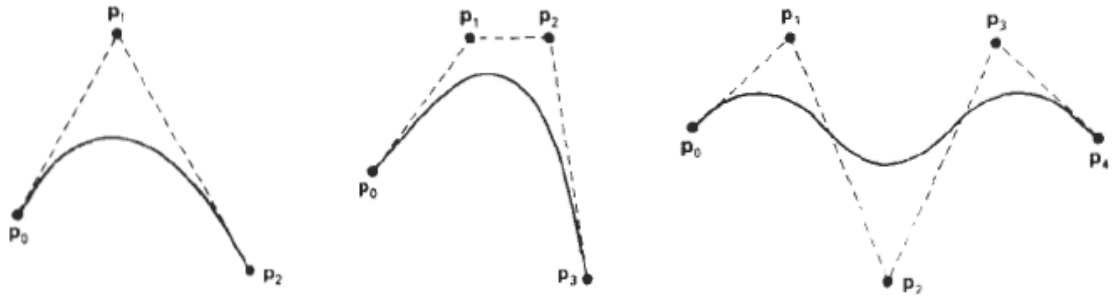


Fig: Examples of two-dimensional Bezier curves generated from three, four, and five control points

Properties of Bezier Curves

1. It always passes through initial and final control points. i.e $P(0) = p_0$ and $P(1) = p_n$.
2. Values of the parametric first derivatives of a Bezier curve at the end points can be calculated from control points as:

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

3. The slope at the beginning of the curve is along the line joining the first two points and slope at the end of curve is along the line joining last two points.
4. Parametric second derivative of a Bezier curve at end points are calculated as:
$$P''(0) = n(n-1)[(p_2-p_1)-(p_1-p_0)]$$
$$P''(1) = n(n-1)[(p_{n-2}-p_{n-1}) - (p_{n-1}-p_n)]$$
5. It lies within the convex hull of the control points. This follows from the properties of Bezier blending functions: they are all positive and their sum is always 1.
$$\sum_{k=0}^{n} BEZ_{k,n}(u) = 1$$

Bezier surfaces

Two sets of orthogonal Bezier curves can be used to design an object surface by specifying by an input mesh of control points. The parametric vector function for the Bezier surface is formed as the Cartesian product of Bezier blending functions:

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

With $p_{j,k}$ specifying the location of the (m + 1) by(n + 1) control points.

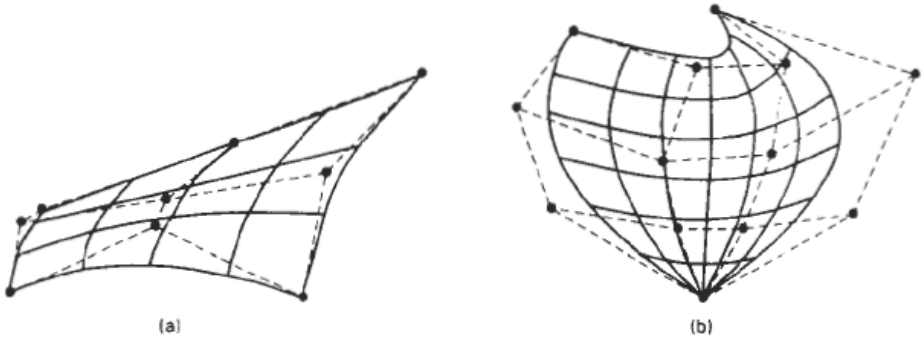


Fig: Bezier surfaces constructed tor (a) m = 3, n = 3, and (b) m = 4, n = 4. Dashed lines connect the control points