UNIT - II THREE-DIMENSIONAL CONCEPTS

Parallel and Perspective projections-Three-Dimensional Object Representations – Polygons, Curved lines,Splines, Quadric Surfaces-Visualization of data sets- Three- Transformations – Three-Dimensional Viewing –Visible surface identification.

2.1 Three Dimensional Concepts

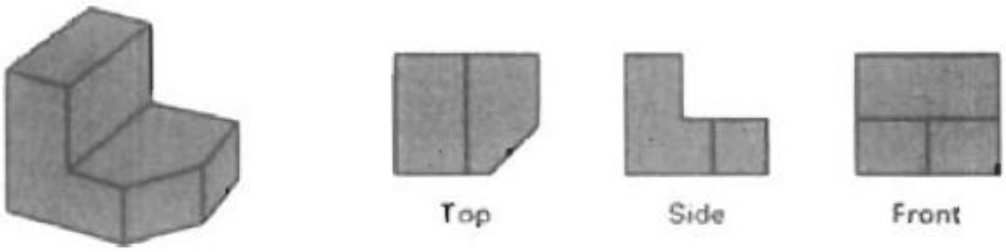
2.1.1 Three Dimensional Display Methods:

- To obtain a display of a three dimensional scene that has been modeled in world coordinates, we must setup a co-ordinate reference for the 'camera'.
- This coordinate reference defines the position and orientation for the plane of the camera film which is the plane we want to use to display a view of the objects in the scene.
- Object descriptions are then transferred to the camera reference coordinates and projected onto the selected display plane.
- The objects can be displayed in wire frame form, or we can apply lighting and surface rendering techniques to shade the visible surfaces.

Parallel Projection:

- Parallel projection is a method for generating a view of a solid object is to project points on the object surface along parallel lines onto the display plane.
- In parallel projection, parallel lines in the world coordinate scene project into parallel lines on the two dimensional display planes.
- This technique is used in engineering and architectural drawings to represent an object with a set of views that maintain relative proportions of the object.
- The appearance of the solid object can be reconstructed from the major views.

Fig. Three parallel projection views of an object, showing relative proportions from different viewing positions.



Perspective Projection:

- It is a method for generating a view of a three dimensional scene is to project points to the display plane alone converging paths.
- This makes objects further from the viewing position be displayed smaller than objects of the same size that are nearer to the viewing position.
- In a perspective projection, parallel lines in a scene that are not parallel to the display plane are projected into converging lines.
- Scenes displayed using perspective projections appear more realistic, since this is the way that our eyes and a camera lens form images.

Depth Cueing:

- Depth information is important to identify the viewing direction, which is the front and which is the back of displayed object.
- Depth cueing is a method for indicating depth with wire frame displays is to vary the intensity of objects according to their distance from the viewing position.
- Depth cueing is applied by choosing maximum and minimum intensity (or color) values and a range of distance over which the intensities are to vary.

Visible line and surface identification:

- A simplest way to identify the visible line is to highlight the visible lines or to display them in a different color.
- Another method is to display the non visible lines as dashed lines.

Surface Rendering:

- Surface rendering method is used to generate a degree of realism in a displayed scene.

- Realism is attained in displays by setting the surface intensity of objects according to the lighting conditions in the scene and surface characteristics.
- Lighting conditions include the intensity and positions of light sources and the background illumination.
- Surface characteristics include degree of transparency and how rough or smooth the surfaces are to be.

Exploded and Cutaway views:

- Exploded and cutaway views of objects can be to show the internal structure and relationship of the objects parts.
- An alternative to exploding an object into its component parts is the cut away view which removes part of the visible surfaces to show internal structure.

Three-dimensional and Stereoscopic Views:

- In Stereoscopic views, three dimensional views can be obtained by reflecting a raster image from a vibrating flexible mirror.
- The vibrations of the mirror are synchronized with the display of the scene on the CRT.
- As the mirror vibrates, the focal length varies so that each point in the scene is projected to a position corresponding to its depth.
- Stereoscopic devices present two views of a scene; one for the left eye and the other for the right eye.
- The two views are generated by selecting viewing positions that corresponds to the two eye positions of a single viewer.
- These two views can be displayed on alternate refresh cycles of a raster monitor, and viewed through glasses that alternately darken first one lens then the other in synchronization with the monitor refresh cycles.

2.1.2 Three Dimensional Graphics Packages

- The 3D package must include methods for mapping scene descriptions onto a flat viewing surface.
- There should be some consideration on how surfaces of solid objects are to be modeled, how visible surfaces can be identified, how transformations of objects are preformed in space, and how to describe the additional spatial properties.
- World coordinate descriptions are extended to 3D, and users are provided with output and input routines accessed with specifications such as
    - Polyline3(n, WcPoints)
    - Fillarea3(n, WcPoints)

- Text3(WcPoint, string)
- Getlocator3(WcPoint)
- Translate3(translateVector, matrix Translate)

Where points and vectors are specified with 3 components and transformation matrices have 4 rows and 4 columns.

2.2 Three Dimensional Object Representations

Representation schemes for solid objects are divided into two categories as follows:

1. Boundary Representation (B-reps)

   It describes a three dimensional object as a set of surfaces that separate the object interior from the environment. Examples are polygon facets and spline patches.

2. Space Partitioning representation

   It describes the interior properties, by partitioning the spatial region containing an object into a set of small, nonoverlapping, contiguous solids(usually cubes).

   Eg: Octree Representation

2.2.1 Polygon Surfaces

Polygon surfaces are boundary representations for a 3D graphics object is a set of polygons that enclose the object interior.

Polygon Tables

- The polygon surface is specified with a set of vertex coordinates and associated attribute parameters.
- For each polygon input, the data are placed into tables that are to be used in the subsequent processing.
- Polygon data tables can be organized into two groups: Geometric tables and attribute tables.

Geometric Tables

Contain vertex coordinates and parameters to identify the spatial orientation of the polygon surfaces.

Attribute tables

Contain attribute information for an object such as parameters specifying the degree of transparency of the object and its surface reflectivity and texture characteristics.

A convenient organization for storing geometric data is to create three lists:

1. The Vertex Table

Coordinate values for each vertex in the object are stored in this table.

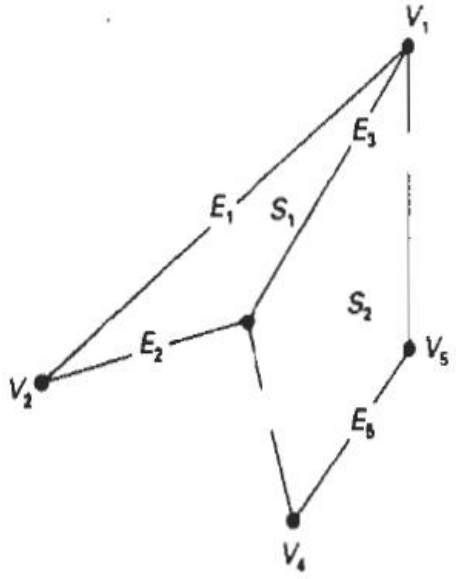
2. The Edge Table

It contains pointers back into the vertex table to identify the vertices for each polygon edge.

3. The Polygon Table

It contains pointers back into the edge table to identify the edges for each polygon.

This is shown in fig



Vertex table	**Edge Table**	**Polygon surface table**
$V_1 : X_1, Y_1, Z_1$	$E_1 : V_1, V_2$	$S_1 : E_1, E_2, E_3$
$V_2 : X_2, Y_2, Z_2$	$E_2 : V_2, V_3$	$S_2 : E_3, E_4, E_5, E_6$
$V_3 : X_3, Y_3, Z_3$	$E_3 : V_3, V_1$	
$V_4 : X_4, Y_4, Z_4$	$E_4 : V_3, V_4$	
$V_5 : X_5, Y_5, Z_5$	$E_5 : V_4, V_5$	
	$E6 : V_5, V_1$	

- Listing the geometric data in three tables provides a convenient reference to the individual components (vertices, edges and polygons) of each object.
- The object can be displayed efficiently by using data from the edge table to draw the component lines.
- Extra information can be added to the data tables for faster information extraction. For instance, edge table can be expanded

to include forward points into the polygon table so that common edges between polygons can be identified more rapidly.

$E_1 : V_1, V_2, S_1$

$E_2 : V_2, V_3, S_1$

$E_3 : V_3, V_1, S_1, S_2$

$E_4 : V_3, V_4, S_2$

$E_5 : V_4, V_5, S_2$

$E6 : V_5, V_1, S_2$

- This is useful for the rendering procedure that must vary surface shading smoothly across the edges from one polygon to the next. Similarly, the vertex table can be expanded so that vertices are cross-referenced to corresponding edges.
- Additional geometric information that is stored in the data tables includes the slope for each edge and the coordinate extends for each polygon. As vertices are input, we can calculate edge slopes and we can scan the coordinate values to identify the minimum and maximum x, y and z values for individual polygons.
- The more information included in the data tables will be easier to check for errors.
- Some of the tests that could be performed by a graphics package are:
    1. That every vertex is listed as an endpoint for at least two edges.
    2. That every edge is part of at least one polygon.
    3. That every polygon is closed.
    4. That each polygon has at least one shared edge.
    5. That if the edge table contains pointers to polygons, every edge referenced by a polygon pointer has a reciprocal pointer back to the polygon.

Plane Equations:

- To produce a display of a 3D object, we must process the input data representation for the object through several procedures such as,
    - Transformation of the modeling and world coordinate descriptions to viewing coordinates.
    - Then to device coordinates:
    - Identification of visible surfaces
    - The application of surface-rendering procedures.
- For these processes, we need information about the spatial orientation of the individual surface components of the object. This

information is obtained from the vertex coordinate value and the equations that describe the polygon planes.

The equation for a plane surface is

$$Ax + By + Cz + D = 0 \quad \text{----(1)}$$

Where (x, y, z) is any point on the plane, and the coefficients A,B,C and D are constants describing the spatial properties of the plane.

- We can obtain the values of A, B,C and D by solving a set of three plane equations using the coordinate values for three non collinear points in the plane.
- For that, we can select three successive polygon vertices (x_1, y_1, z_1), (x_2, y_2, z_2) and (x_3, y_3, z_3) and solve the following set of simultaneous linear plane equations for the ratios A/D, B/D and C/D.

$$(A/D)x_k + (B/D)y_k + (c/D)z_k = -1, \qquad k=1,2,3 \quad \text{-----(2)}$$

- The solution for this set of equations can be obtained in determinant form, using Cramer's rule as

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \qquad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix}$$

$$C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \qquad D = -\begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix} \quad \text{------(3)}$$

- Expanding the determinants , we can write the calculations for the plane coefficients in the form:

$$A = y_1 (z_2 - z_3) + y_2(z_3 - z_1) + y_3 (z_1 - z_2)$$

$$B = z_1 (x_2 - x_3) + z_2 (x_3 - x_1) + z_3 (x_1 - x_2)$$

$$C = x_1 (y_2 - y_3) + x_2 (y_3 - y_1) + x_3 (y_1 - y_2)$$

$$D = -x_1 (y_2 z_3 - y_3 z_2) - x_2 (y_3 z_1 - y_1 z_3) - x_3 (y_1 z_2 - y_2 z_1) \quad \text{------(4)}$$

- As vertex values and other information are entered into the polygon data structure, values for A, B, C and D are computed for each polygon and stored with the other polygon data.
- Plane equations are used also to identify the position of spatial points relative to the plane surfaces of an object. For any point (x, y, z) hot on a plane with parameters A,B,C,D, we have

$$Ax + By + Cz + D \neq 0$$

- We can identify the point as either inside or outside the plane surface according o the sigh (negative or positive) of Ax + By + Cz + D:

  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 inequality tests are valid in a right handed Cartesian system, provided the plane parmeters 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

- A single plane surface can be specified with a function such as **fillArea**. But when object surfaces are to be tiled, it is more convenient to specify the surface facets with a mesh function.
- One type of polygon mesh is the **triangle strip**.A triangle strip formed with 11 triangles connecting 13 vertices.

- This function produces n-2 connected triangles given the coordinates for n vertices.

- Another similar function in the **quadrilateral mesh**, which generates a mesh of (n-1) by (m-1) quadrilaterals, given the coordinates for an n by m array of vertices. Figure shows 20 vertices forming a mesh of 12 quadrilaterals.

2.2.2 Curved Lines and Surfaces

- Displays of three dimensional curved lines and surface can be generated from an input set of mathematical functions defining the objects or from a set of user specified data points.

- When functions are specified, a package can project the defining equations for a curve to the display plane and plot pixel positions along the path of the projected function.
- For surfaces, a functional description in decorated to produce a polygon-mesh approximation to the surface.

2.2.3 Quadric Surfaces

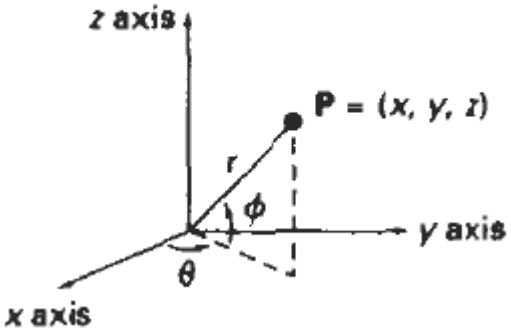
- The quadric surfaces are described with second degree equations (quadratics).
- They include spheres, ellipsoids, tori, parabolids, and hyperboloids.

Sphere

- In Cartesian coordinates, a spherical surface with radius r centered on the coordinates origin is defined as the set of points (x, y, z) that satisfy the equation.

$$x^2 + y^2 + z^2 = r^2 \qquad \text{----------(1)}$$

- The spherical surface can be represented in parametric form by using latitude and longitude angles



$$x = r\cos\varphi\cos\theta, \quad -\Lambda/2 <= \varphi <= \Lambda/2$$
$$y = r\cos\varphi\sin\theta, \quad -\Lambda <= \varphi <= \Lambda \qquad \text{-------(2)}$$
$$z = r\sin\varphi$$

- The parameter representation in eqn (2) provides a symmetric range for the angular parameter θ and φ.

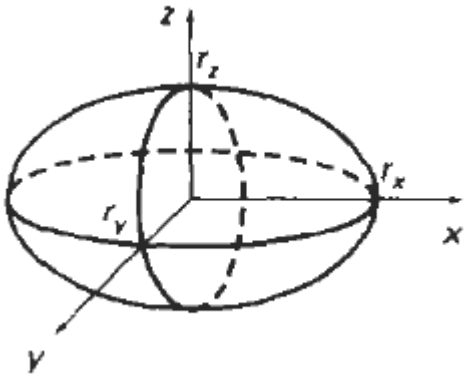
Ellipsoid

- Ellipsoid surface is an extension of a spherical surface where the radius in three mutually perpendicular directions can have different values

- The Cartesian representation for points over the surface of an ellipsoid centered on the origin is

$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1$$

- The parametric representation for the ellipsoid in terms of the latitude angle φ and the longitude angle θ is



$x = r_x \cos\varphi \cos\theta, \quad -\Lambda/2 <= \varphi <= \Lambda/2$

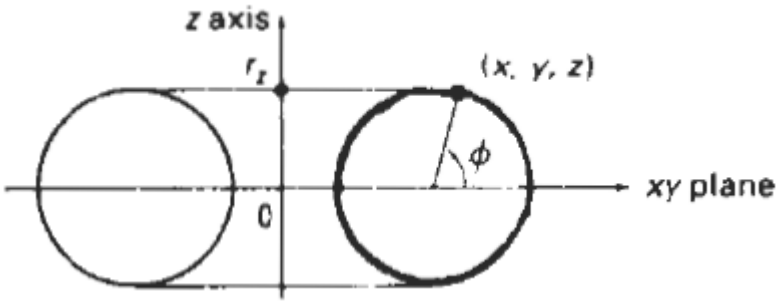
$y = r_y \cos\varphi \sin\theta, \quad -\Lambda <= \varphi <= \Lambda$

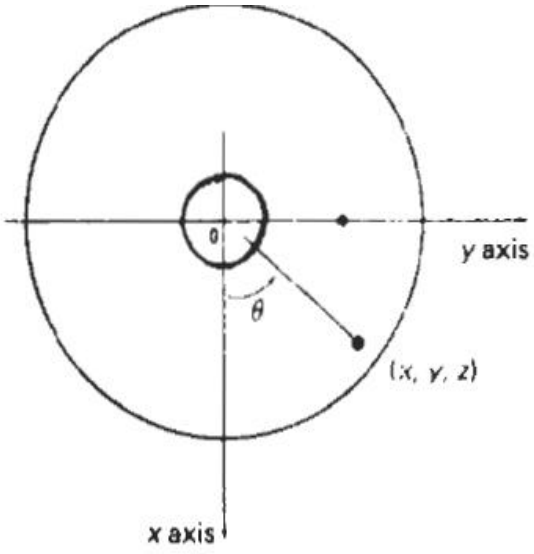
$z = r_z \sin\varphi$

Torus

- Torus is a doughnut shaped object.
- It can be generated by rotating a circle or other conic about a specified axis.

A torus with a circular cross section centered on the coordinate origin





- The Cartesian representation for points over the surface of a torus can be written in the form

$$\left[r - \sqrt{\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2}\right]^2 + \left(\frac{z}{r_z}\right)^2 = 1$$

  where r in any given offset value.

- Parametric representation for a torus is similar to those for an ellipse, except that angle φ extends over 360°.
- Using latitude and longitude angles φ and θ, we can describe the torus surface as the set of points that satisfy.

$$x = r_x (r + \cos\varphi) \cos\theta, \qquad -\Lambda <= \varphi <= \Lambda$$

$$y = r_y(r + \cos\varphi) \sin\theta, \qquad -\Lambda <= \varphi <= \Lambda$$

$$z = r_z \sin\varphi$$

2.2.4 Spline Representations

- A Spline is a flexible strip used to produce a smooth curve through a designated set of points.
- Several small weights are distributed along the length of the strip to hold it in position on the drafting table as the curve is drawn.
- The **Spline curve** refers to any sections curve formed with polynomial sections satisfying specified continuity conditions at the boundary of the pieces.
- A **Spline surface** can be described with two sets of orthogonal spline curves.
- Splines are used in graphics applications to design curve and surface shapes, to digitize drawings for computer storage, and to

specify animation paths for the objects or the camera in the scene. CAD applications for splines include the design of automobiles bodies, aircraft and spacecraft surfaces, and ship hulls.

Interpolation and Approximation Splines

- Spline curve can be specified by a set of coordinate positions called **control points** which indicates the general shape of the curve.
- These control points are fitted with piecewise continuous parametric polynomial functions in one of the two ways.

1. When polynomial sections are fitted so that the curve passes through each control point the resulting curve is said to **interpolate** the set of control points.

A set of six control points interpolated with piecewise continuous polynomial sections

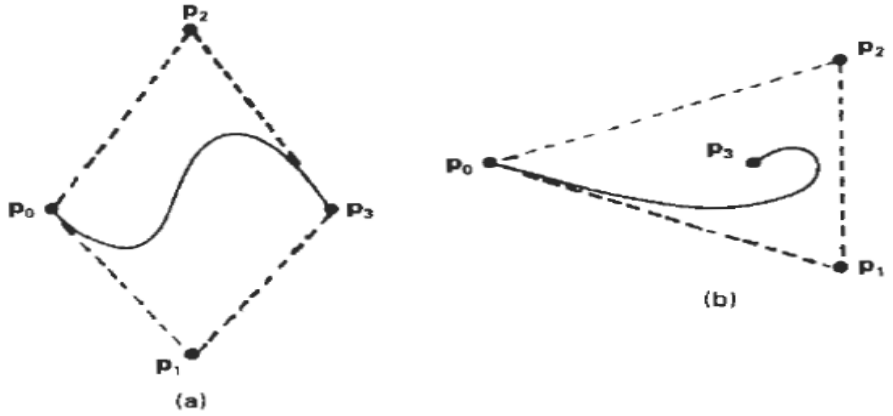
2. When the polynomials are fitted to the general control point path without necessarily passing through any control points, the resulting curve is said to **approximate** the set of control points.

A set of six control points approximated with piecewise continuous polynomial sections

- Interpolation curves are used to digitize drawings or to specify animation paths.
- Approximation curves are used as design tools to structure object surfaces.

- A spline curve is designed , modified and manipulated with operations on the control points.The curve can be translated, rotated or scaled with transformation applied to the control points.
- The convex polygon boundary that encloses a set of control points is called the **convex hull**.
- The shape of the convex hull is to imagine a rubber band stretched around the position of the control points so that each control point is either on the perimeter of the hull or inside it.

Convex hull shapes (dashed lines) for two sets of control points



Parametric Continuity Conditions

- For a smooth transition from one section of a piecewise parametric curve to the next various **continuity conditions** are needed at the connection points.
- If each section of a spline in described with a set of parametric coordinate functions or the form

$$x = x(u),\ y = y(u),\ z = z(u),\ u1 <= u <= u2 \qquad \text{-----(a)}$$

- We set **parametric continuity** by matching the parametric derivatives of adjoining curve sections at their common boundary.
- **Zero order parametric continuity** referred to as C^0 continuity, means that the curves meet. (i.e) the values of x,y, and z evaluated at u_2 for the first curve section are equal. Respectively, to the value of x,y, and z evaluated at u_1 for the next curve section.
- **First order parametric continuity** referred to as C^1 continuity means that the first parametric derivatives of the coordinate functions in equation (a) for two successive curve sections are equal at their joining point.
- **Second order parametric continuity**, or C2 continuity means that both the first and second parametric derivatives of the two curve sections are equal at their intersection.

- Higher order parametric continuity conditions are defined similarly.

Piecewise construction of a curve by joining two curve segments using different orders of continuity

a)Zero order continuity only

b)First order continuity only

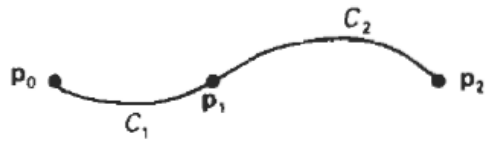
c) Second order continuity only

Geometric Continuity Conditions

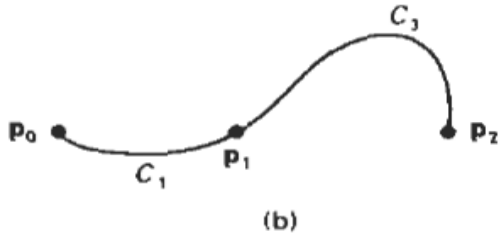
- To specify conditions for geometric continuity is an alternate method for joining two successive curve sections.
- The parametric derivatives of the two sections should be proportional to each other at their common boundary instead of equal to each other.
- Zero order Geometric continuity referred as G^0 continuity means that the two curves sections must have the same coordinate position at the boundary point.
- First order Geometric Continuity referred as G^1 continuity means that the parametric first derivatives are proportional at the interaction of two successive sections.
- Second order Geometric continuity referred as G^2 continuity means that both the first and second parametric derivatives of the two curve sections are proportional at their boundary. Here the curvatures of two sections will match at the joining position.

Three control points fitted with two curve sections joined with

a) parametric continuity



b)geometric continuity where the tangent vector of curve C_3 at point p1 has a greater magnitude than the tangent vector of curve C1 at p1.



(b)

Spline specifications

There are three methods to specify a spline representation:

1. We can state the set of boundary conditions that are imposed on the spline; (or)
2. We can state the matrix that characterizes the spline; (or)
3. We can state the set of **blending functions** that determine how specified geometric constraints on the curve are combined to calculate positions along the curve path.

- To illustrate these three equivalent specifications, suppose we have the following parametric cubic polynomial representation for the x coordinate along the path of a spline section.

$$x(u)=a_x u^3 + a_x u^2 + c_x u + d_x \qquad 0<= u <=1 \quad ----------(1)$$

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

From the boundary conditions we can obtain the matrix that characterizes this spline curve by first rewriting eq(1) as the matrix product

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

$$= U.C$$

where U is the row matrix of power of parameter u and C is the coefficient column matrix.

- Using equation (2) we can write the boundary conditions in matrix form and solve for the coefficient matrix C as

$$C = M_{spline} \cdot M_{geom} \quad \text{-----(3)}$$

Where M_{geom} in a four element column matrix containing the geometric constraint values on the spline and M_{spline} in the 4 * 4 matrix that transforms the geometric constraint values to the polynomial coefficients and provides a characterization for the spline curve.

- Matrix M_{geom} contains control point coordinate values and other geometric constraints.
- We can substitute the matrix representation for C into equation (2) to obtain.

$$x(u) = U \cdot M_{spline} \cdot M_{geom} \quad \text{------(4)}$$

- The matrix M_{spline}, characterizing a spline representation, called the **basis matriz** is useful for transforming from one spline representation to another.
- Finally we can expand equation (4) to obtain a polynomial representation for coordinate x in terms of the geometric constraint parameters.

$$x(u) = \sum \mathbf{g_k . BF_k(u)}$$

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.

2.3 Visualization of Data Sets

- The use of graphical methods as an aid in scientific and engineering analysis is commonly referred to as **scientific visualization**.
- This involves the visualization of data sets and processes that may be difficult or impossible to analyze without graphical methods. Example medical scanners, satellite and spacecraft scanners.

- Visualization techniques are useful for analyzing process that occur over a long period of time or that cannot observed directly. Example quantum mechanical phenomena and special relativity effects produced by objects traveling near the speed of light.
- Scientific visualization is used to visually display , enhance and manipulate information to allow better understanding of the data.
- Similar methods employed by commerce , industry and other nonscientific areas are sometimes referred to as business visualization.
- Data sets are classified according to their spatial distribution (2D or 3D) and according to data type (scalars , vectors , tensors and multivariate data).

Visual Representations for Scalar Fields

- A scalar quantity is one that has a single value. Scalar data sets contain values that may be distributed in time as well as over spatial positions also the values may be functions of other scalar parameters. Examples of physical scalar quantities are energy, density, mass , temperature and water content.
- A common method for visualizing a scalar data set is to use graphs or charts that show the distribution of data values as a function of other parameters such as position and time.
- **Pseudo-color methods** are also used to distinguish different values in a scalar data set, and color coding techniques can be combined with graph and chart models. To color code a scalar data set we choose a range of colors and map the range of data values to the color range. Color coding a data set can be tricky because some color combinations can lead to misinterpretations of the data.
- **Contour plots** are used to display **isolines** (lines of constant scalar value) for a data set distributed over a surface. The isolines are spaced at some convenient interval to show the range and variation of the data values over the region of space. Contouring methods are applied to a set of data values that is distributed over a regular grid.

  A 2D contouring algorithm traces the isolines from cell to cell within the grid by checking the four corners of grid cells to determine which cell edges are crossed by a particular isoline.

  The path of an isoline across five grid cells

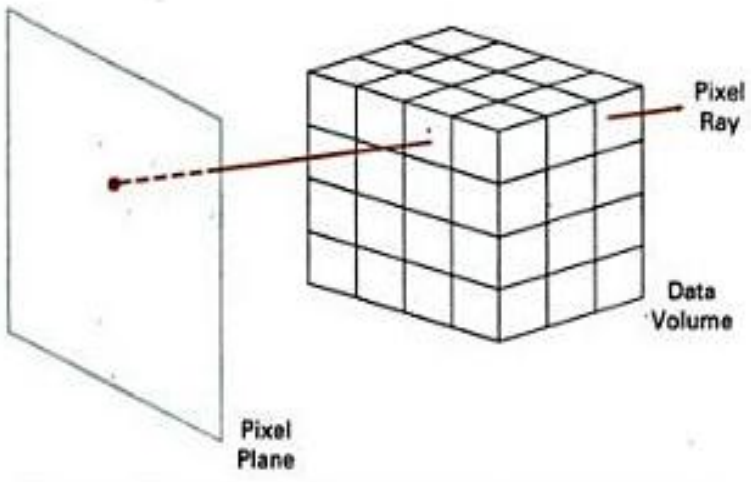
Sometimes isolines are plotted with spline curves but spline fitting can lead to misinterpretation of the data sets. Example two spline isolines could cross or curved isoline paths might not be a true indicator of data trends since data values are known only at the cell corners.

For 3D scalar data fields we can take cross sectional slices and display the 2D data distributions over the slices. Visualization packages provide a slicer routine that allows cross sections to be taken at any angle.

Instead of looking at 2D cross sections we plot one or more **isosurfaces** which are simply 3D contour plots. When two overlapping isosurfaces are displayed the outer surface is made transparent so that we can view the shape of both isosurfaces.

- **Volume rendering** which is like an X-ray picture is another method for visualizing a 3D data set. The interior information about a data set is projected to a display screen using the ray-casting method. Along the ray path from each screen pixel.

Volume visualization of a regular, Cartesian data grid using ray casting to examine interior data values



. Data values at the grid positions. are averaged so that one value is stored for each voxel of the data space. How the data are encoded for display depends on the application.

For this volume visualization, a color-coded plot of the distance to the maximum voxel value along each pixel ray was displayed.

Visual representation for Vector fields

- A vector quantity V in three-dimensional space has three scalar values

  (V_x , V_y,V_z,) one for each coordinate direction, and a two-dimensional vector has two components **(V_x, V_y,).** Another way to describe a vector quantity is by giving its magnitude I**V** I and its direction as a unit vector **u**.

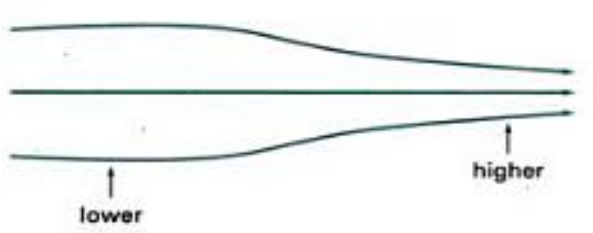
  As with scalars, vector quantities may be functions of position, time, and other parameters. Some examples of physical vector quantities are velocity, acceleration, force, electric fields, magnetic fields, gravitational fields, and electric current.

  One way to visualize a vector field is to plot each data point as a small arrow that shows the magnitude and direction of the vector. This method is most often used with cross-sectional slices, since it can be difficult to see the trends in a three-dimensional region cluttered with overlapping arrows. Magnitudes for the vector values can be shown by varying the lengths of the arrows.

  Vector values are also represented by plotting **field lines** or **streamlines .**

  Field lines are commonly used for electric , magnetic and gravitational fields. The magnitude of the vector values is indicated by spacing between field lines, and the direction is the tangent to the field.

Field line representation for a vector data set



Visual Representations for Tensor Fields

A tensor quantity in three-dimensional space has nine components and can be represented with a 3 by 3 matrix. This representation is used for a **second-order tensor,** and higher-order tensors do occur in some applications.

Some examples of physical, second-order tensors are stress and strain in a material subjected to external forces, conductivity of an electrical conductor, and the metric tensor, which gives the properties of a particular coordinate space.

The stress tensor in Cartesian coordinates,for example, can be represented as

$$\begin{bmatrix} \sigma_x & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_y & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_z \end{bmatrix}$$

Tensor quantities are frequently encountered in anisotropic materials, which have different properties in different directions. The x, xy, and xz elements of the conductivity tensor, for example, describe the contributions of electric field components in the x, y, and *z* diretions to the current in the *x* direction.

Usually, physical tensor quantities are symmetric, so that the tensor has only six distinct values. Visualization schemes for representing all six components of a second-order tensor quantity are based on devising shapes that have six parameters.

Instead of trying to visualize all six components of a tensor quantity, we can reduce the tensor to a vector or a scalar. And by applying **tensor-contraction** operations, we can obtain a scalar representation.

Visual Representations for Multivariate Data Fields

In some applications, at each grid position over some region of space, we may have multiple data values, which can be a mixture of scalar, vector, and even tensor values.

A method for displaying multivariate data fields is to construct graphical objects, sometimes referred to as **glyphs**, with multiple parts. Each part of a glyph represents a physical quantity. The size and color of each part can be used to display information about scalar magnitudes. To give directional information for a vector field, we can use a wedge, a cone, or some other pointing shape for the glyph part representing the vector.

2.4 Three Dimensional Geometric and Modeling Transformations

Geometric transformations and object modeling in three dimensions are extended from two-dimensional methods by including considerations for the z-coordinate.

2.4.1 Translation

In a three dimensional homogeneous coordinate representation, a point or an object is translated from position P = (x,y,z) to position P' = (x',y',z') with the matrix operation.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & y_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{--------(1)}$$

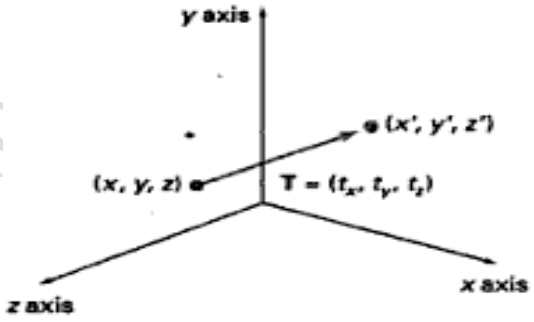
(or) **P' = T.P** ----------(2)

Parameters t_x, t_y and t_z specifying translation distances for the coordinate directions x,y and z are assigned any real values.

The matrix representation in equation (1) is equivalent to the three equations

$$x' = x + t_x$$
$$y' = y + t_y$$
$$z' = z + t_z \quad \text{----------(3)}$$

Translating a point with translation vector T = (t_x, t_y, t_z)



Inverse of the translation matrix in equation (1) can be obtained by negating the translation distance t_x, t_y and t_z.

This produces a translation in the opposite direction and the product of a translation matrix and its inverse produces the identity matrix.

2.4.2 Rotation

- To generate a rotation transformation for an object an axis of rotation must be designed to rotate the object and the amount of angular rotation is also be specified.
- Positive rotation angles produce counter clockwise rotations about a coordinate axis.

Co-ordinate Axes Rotations

The 2D z axis rotation equations are easily extended to 3D.

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

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

$z' = z$ ----------(2)

Parameters θ specifies the rotation angle. In homogeneous coordinate form, the 3D z axis rotation equations are expressed as

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{-------(3)}$$

which we can write more compactly as

$$\mathbf{P' = R_z(\theta) \cdot P} \quad \text{----------(4)}$$

The below figure illustrates rotation of an object about the z axis.

Transformation equations for rotation about the other two coordinate axes can be obtained with a cyclic permutation of the

coordinate parameters x, y and z in equation (2) i.e., we use the replacements

$$x \rightarrow y \rightarrow z \rightarrow x \qquad \text{---------(5)}$$

Substituting permutations (5) in equation (2), we get the equations for an x-axis rotation

$$\begin{aligned} y' &= y\cos\theta - z\sin\theta \\ z' &= y\sin\theta + z\cos\theta \qquad \text{----------(6)} \\ x' &= x \end{aligned}$$

which can be written in the homogeneous coordinate form

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad \text{-------(7)}$$

$$\text{(or)} \quad \mathbf{P' = R_x(\theta).\ P} \qquad \text{----------(8)}$$

Rotation of an object around the x-axis is demonstrated in the below fig

Cyclically permuting coordinates in equation (6) give the transformation equation for a y axis rotation.

$$\begin{aligned} z' &= z\cos\theta - x\sin\theta \\ x' &= z\sin\theta + x\cos\theta \qquad \text{----------(9)} \\ y' &= y \end{aligned}$$

The matrix representation for y-axis rotation is

$$\begin{bmatrix} x' \\ \end{bmatrix} \quad \begin{matrix} \cos\theta & 0 & \sin\theta & 0 \end{matrix} \quad \begin{bmatrix} x \\ \end{bmatrix}$$

$$\begin{bmatrix} y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y \\ z \\ 1 \end{bmatrix} \quad \text{--------(10)}$$

(or) $\mathbf{P' = R_y(\theta).\ P}$ **----------(** 11)

An example of y axis rotation is shown in below figure

- An inverse rotation matrix is formed by replacing the rotation angle θ by –θ.
- Negative values for rotation angles generate rotations in a clockwise direction, so the identity matrix is produces when any rotation matrix is multiplied by its inverse.
- Since only the sine function is affected by the change in sign of the rotation angle, the inverse matrix can also be obtained by interchanging rows and columns. (i.e.,) we can calculate the inverse of any rotation matrix R by evaluating its transpose ($R^{-1} = R^T$).

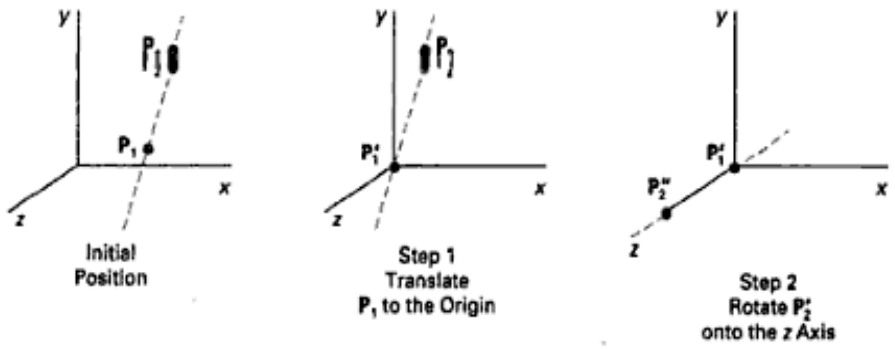
General Three Dimensional Rotations

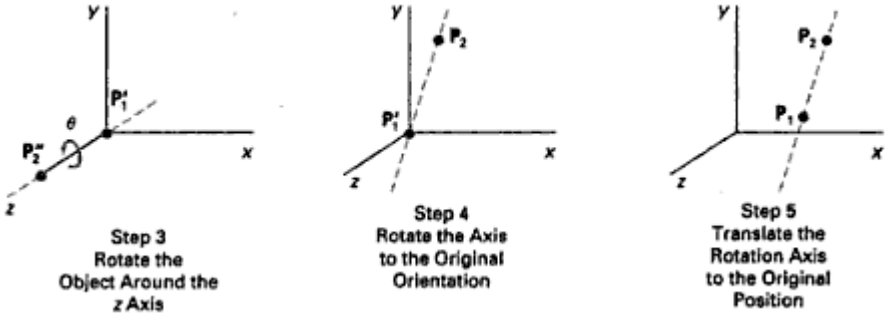
- A rotation matrix for any axis that does not coincide with a coordinate axis can be set up as a composite transformation involving combinations of translations and the coordinate axes rotations.
- We obtain the required composite matrix by
    1. Setting up the transformation sequence that moves the selected rotation axis onto one of the coordinate axis.
    2. Then set up the rotation matrix about that coordinate axis for the specified rotation angle.

3. Obtaining the inverse transformation sequence that returns the rotation axis to its original position.

- In the special case where an object is to be rotated about an axis that is parallel to one of the coordinate axes, we can attain the desired rotation with the following transformation sequence:
    1. Translate the object so that the rotation axis coincides with the parallel coordinate axis.
    2. Perform the specified rotation about the axis.
    3. Translate the object so that the rotation axis is moved back to its original position.
- When an object is to be rotated about an axis that is not parallel to one of the coordinate axes, we need to perform some additional transformations.
- In such case, we need rotations to align the axis with a selected coordinate axis and to bring the axis back to its original orientation.
- Given the specifications for the rotation axis and the rotation angle, we can accomplish the required rotation in five steps:
    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.

Five transformation steps





2.4.3 Scaling

- The matrix expression for the scaling transformation of a position P = (x,y,.z) relative to the coordinate origin can be written as

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{--------(11)}$$

(or) $\mathbf{P' = S.P}$ ---------(12)

where scaling parameters s_x, s_y, and s_z are assigned any position values.

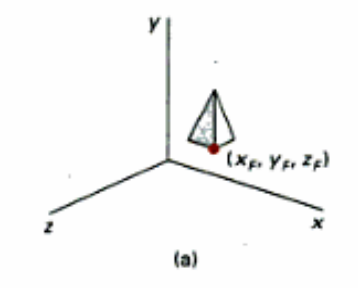
- Explicit expressions for the coordinate transformations for scaling relative to the origin are

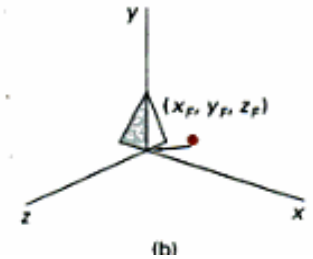
$$x' = x.s_x$$
$$y' = y.s_y \quad \text{----------(13)}$$
$$z' = z.s_z$$

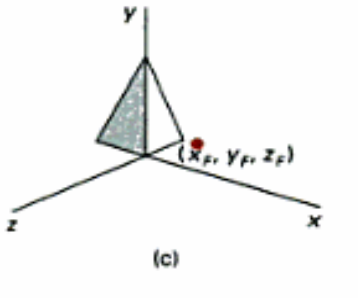
- Scaling an object changes the size of the object and repositions the object relatives to the coordinate origin.
- If the transformation parameters are not equal, relative dimensions in the object are changed.
- The original shape of the object is preserved with a uniform scaling ($s_x = s_y = s_z$) .
- Scaling with respect to a selected fixed position (x_f, y_f, z_f) can be represented with the following transformation sequence:

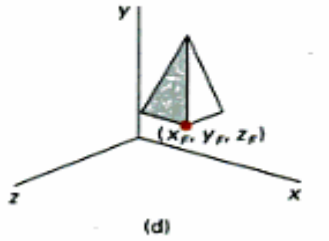
  1. Translate the fixed point to the origin.
  2. Scale the object relative to the coordinate origin using Eq.11.

3. Translate the fixed point back to its original position. This sequence of transformation is shown in the below figure .









- The matrix representation for an arbitrary fixed point scaling can be expressed as the concatenation of the **translate-scale-translate** transformation are

$$T(x_f, y_f, z_f) \cdot S(s_x, s_y, s_z) \cdot T(-x_f, -y_f, -z_f) =$$

$$\begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{----------(14)}$$

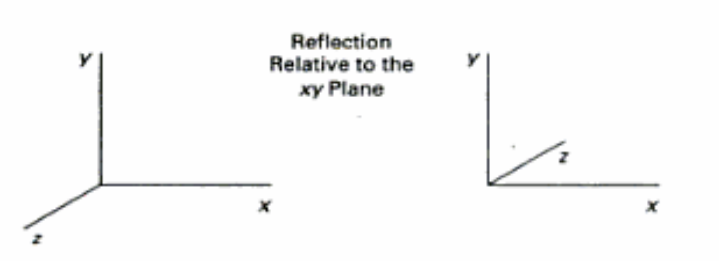
- Inverse scaling matrix m formed by replacing the scaling parameters s_x, s_y and s_z with their reciprocals.
- The inverse matrix generates an opposite scaling transformation, so the concatenation of any scaling matrix and its inverse produces the **identity matrix**.

2.4.4 Other Transformations

Reflections

- A 3D reflection can be performed relative to a selected reflection axis or with respect to a selected reflection plane.
- Reflection relative to a given axis are equivalent to 180^0 rotations about the axis.

- Reflection relative to a plane are equivalent to 180^0 rotations in 4D space.
- When the reflection plane in a coordinate plane (either x_y, x_z or y_z) then the transformation can be a conversion between left-handed and right-handed systems.
- An example of a reflection that converts coordinate specifications from a right handed system to a left-handed system is shown in the figure



- This transformation changes the sign of z coordinates, leaves the x and y coordinate values unchanged.
- The matrix representation for this reflection of points relative to the xy plane is

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

- Reflections about other planes can be obtained as a combination of rotations and coordinate plane reflections.

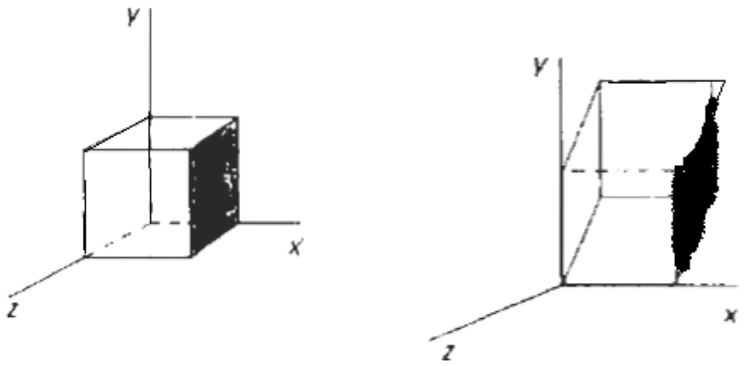
Shears

- Shearing transformations are used to modify object shapes.
- They are also used in three dimensional viewing for obtaining general projections transformations.
- The following transformation produces a z-axis shear.

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

- Parameters a and b can be assigned any real values.

- This transformation matrix is used to alter x and y coordinate values by an amount that is proportional to the z value, and the z coordinate will be unchanged.
- Boundaries of planes that are perpendicular to the z axis are shifted by an amount proportional to z the figure shows the effect of shearing matrix on a unit cube for the values a = b = 1.



2.4.5 Composite Transformation

- Composite three dimensional transformations can be formed by multiplying the matrix representation for the individual operations in the transformation sequence.
- This concatenation is carried out from right to left, where the right most matrixes is the first transformation to be applied to an object and the left most matrix is the last transformation.
- A sequence of basic, three-dimensional geometric transformations is combined to produce a single composite transformation which can be applied to the coordinate definition of an object.

2.4.6Three Dimensional Transformation Functions

Some of the basic 3D transformation functions are:

translate (translateVector, matrixTranslate)

rotateX(thetaX, xMatrixRotate)

rotateY(thetaY, yMatrixRotate)

rotateZ(thetaZ, zMatrixRotate)

scale3 (scaleVector, matrixScale)

- Each of these functions produces a 4 by 4 transformation matrix that can be used to transform coordinate positions expressed as homogeneous column vectors.
- Parameter translate Vector is a pointer to list of translation distances tx, ty, and tz.

- Parameter scale vector specifies the three scaling parameters sx, sy and sz.
- Rotate and scale matrices transform objects with respect to the coordinate origin.
- Composite transformation can be constructed with the following functions:

  composeMatrix3

  buildTransformationMatrix3

  composeTransformationMatrix3

- The order of the transformation sequence for the **buildTransformationMarix3** and **composeTransfomationMarix3** functions, is the same as in 2 dimensions:

  1. scale
  2. rotate
  3. translate

- Once a transformation matrix is specified, the matrix can be applied to specified points with

  transformPoint3 (inPoint, matrix, outpoint)

- The transformations for hierarchical construction can be set using structures with the function

  setLocalTransformation3 (matrix, type)

where parameter matrix specifies the elements of a 4 by 4 transformation matrix and parameter type can be assigned one of the values of:

Preconcatenate,

Postconcatenate, or replace.

2.4.7 Modeling and Coordinate Transformations

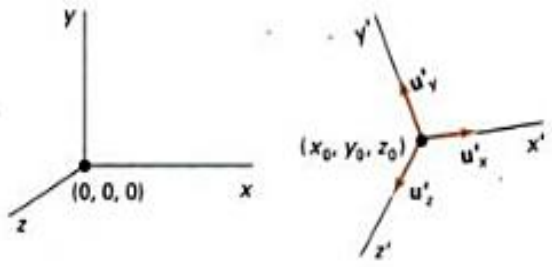
- In modeling, objects are described in a local (modeling) coordinate reference frame, then the objects are repositioned into a world coordinate scene.
- For instance, tables, chairs and other furniture, each defined in a local coordinate system, can be placed into the description of a room defined in another reference frame, by transforming the furniture coordinates to room coordinates. Then the room might be transformed into a larger scene constructed in world coordinate.
- Three dimensional objects and scenes are constructed using structure operations.
- Object description is transformed from modeling coordinate to world coordinate or to another system in the hierarchy.

- Coordinate descriptions of objects are transferred from one system to another system with the same procedures used to obtain two dimensional coordinate transformations.
- Transformation matrix has to be set up to bring the two coordinate systems into alignment:
  - First, a translation is set up to bring the new coordinate origin to the position of the other coordinate origin.
  - Then a sequence of rotations are made to the corresponding coordinate axes.
  - If different scales are used in the two coordinate systems, a scaling transformation may also be necessary to compensate for the differences in coordinate intervals.
- If a second coordinate system is defined with origin (x_0, y_0,z_0) and axis vectors as shown in the figure relative to an existing Cartesian reference frame, then first construct the translation matrix T($-x_0$, $-y_0$, $-z_0$), then we can use the unit axis vectors to form the coordinate rotation matrix

$$R = \begin{bmatrix} u'_{x1} & u'_{x2} & u'_{x3} & 0 \\ u'_{y1} & u'_{y2} & u'_{y3} & 0 \\ u'_{z1} & u'_{z2} & u'_{z3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

which transforms unit vectors u'_x, u'_y and u'_z onto the x, y and z axes respectively.

Transformation of an object description from one coordinate system to another.



- The complete coordinate-transformation sequence is given by the composite matrix **R .T.**
- This matrix correctly transforms coordinate descriptions from one Cartesian system to another even if one system is left-handed and the other is right handed.

2.5Three-Dimensional Viewing

In three dimensional graphics applications,

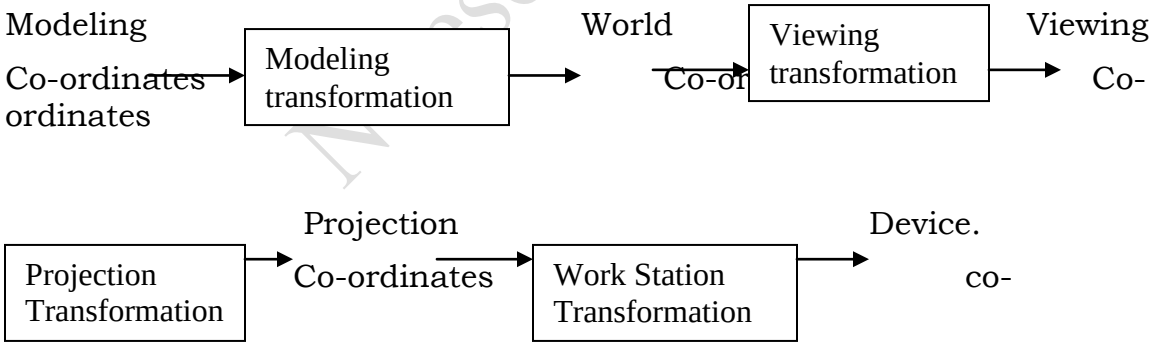
- we can view an object from any spatial position, from the front, from above or from the back.
- We could generate a view of what we could see if we were standing in the middle of a group of objects or inside object, such as a building.

2.5.1Viewing Pipeline:

In the view of a three dimensional scene, to take a snapshot we need to do the following steps.

1. Positioning the camera at a particular point in space.
2. Deciding the camera orientation (i.e.,) pointing the camera and rotating it around the line of right to set up the direction for the picture.
3. When snap the shutter, the scene is cropped to the size of the 'window' of the camera and light from the visible surfaces is projected into the camera film.

In such a way the below figure shows the three dimensional transformation pipeline, from modeling coordinates to final device coordinate.



Processing Steps

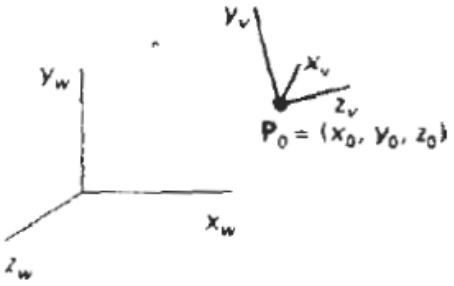
1. Once the scene has been modeled, world coordinates position is converted to viewing coordinates.
2. The viewing coordinates system is used in graphics packages as a reference for specifying the observer viewing position and the position of the projection plane.
3. Projection operations are performed to convert the viewing coordinate description of the scene to coordinate positions on the projection plane, which will then be mapped to the output device.

4. Objects outside the viewing limits are clipped from further consideration, and the remaining objects are processed through visible surface identification and surface rendering procedures to produce the display within the device viewport.

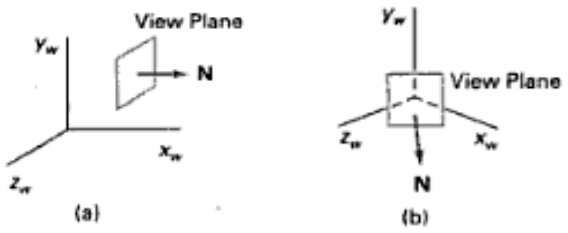
2.5.2Viewing Coordinates

Specifying the view plane

- The view for a scene is chosen by establishing the viewing coordinate system, also called the **view reference coordinate system.**

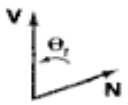


- A **viewplane** or **projection plane** is set-up perpendicular to the viewing Z_v axis.
- World coordinate positions in the scene are transformed to viewing coordinates, then viewing coordinates are projected to the view plane.
- The **view reference point** is a world coordinate position, which is the origin of the viewing coordinate system. It is chosen to be close to or on the surface of some object in a scene.
- Then we select the positive direction for the viewing Z_v axis, and the orientation of the view plane by specifying the **view plane normal vector, N**. Here the world coordinate position establishes the direction for N relative either to the world origin or to the viewing coordinate origin.



- Then we select the up direction for the view by specifying a vector V called the **view-up vector**. This vector is used to establish the positive direction for the y_v axis.

Specifying the view –up vector with a twist angle θ_t

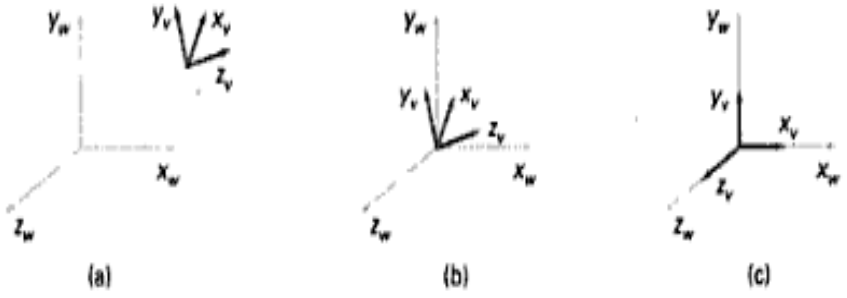


Transformation from world to viewing coordinates

- Before object descriptions can be projected to the view plane, they must be transferred to viewing coordinate. This transformation sequence is,
    1. Translate the view reference point to the origin of the world coordinate system.
    2. Apply rotations to align the x_v, y_v and z_v axes with the world x_w,y_w and z_w 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 3 coordinate axis rotations depending on the direction chosen for N. Aligning a viewing system with the world coordinate axes using a sequence of translate – rotate transformations



- Another method for generation the rotation transformation matrix is to calculate unit uvn vectors and form the composite rotation matrix directly.
- Given vectors N and V, these unit vectors are calculated as

$$n = N / (|N|) = (n_1, n_2, n_3)$$

$$u = (V*N) / (|V*N|) = (u_1, u_2, u_3)$$

$$v = n*u = (v_1, v_2, v_3)$$

- This method automatically adjusts the direction for v, so that v is perpendicular to n.
- The composite rotation matrix for the viewing transformation is

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

which transforms u into the world x_w axis, v onto the y_w axis and n onto the z_w axis.

- The complete world-to-viewing transformation matrix is obtained as the matrix product. $\mathbf{M_{wc,\ vc} = R.T}$

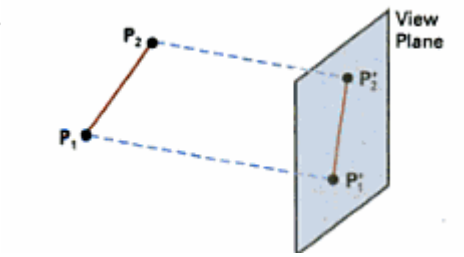
This transformation is applied to coordinate descriptions of objects in the scene transfer them to the viewing reference frame.

2.5 Projections

- Once world coordinate descriptions of the objects are converted to viewing coordinates, we can project the 3 dimensional objects onto the two dimensional view planes.
- There are two basic types of projection.

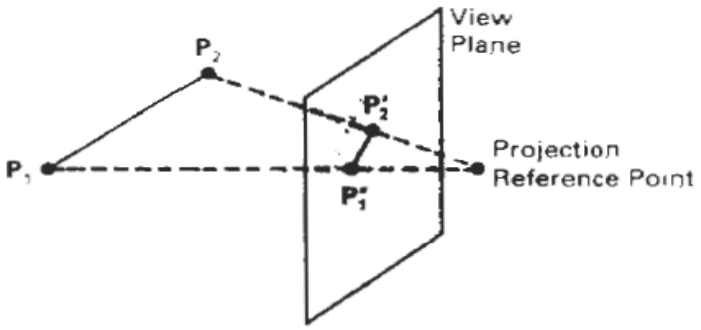
1. **Parallel Projection** - Here the coordinate positions are transformed to the view plane along parallel lines.

Parallel projection of an object to the view plane



2. **Perspective Projection** – Here, object positions are transformed to the view plane along lines that converge to a point called the **projection reference point**.

Perspective projection of an object to the view plane



Parallel Projections

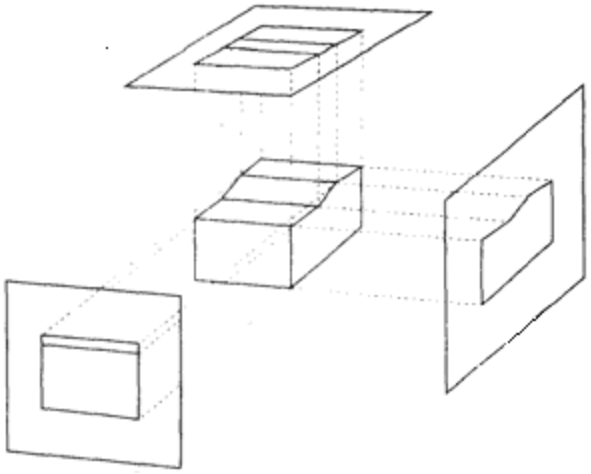
- Parallel projections are specified with a **projection vector** that defines the direction for the projection lines.
- When the projection in perpendicular to the view plane, it is said to be an **Orthographic parallel projection**, otherwise it said to be an **Oblique parallel projection**.

Orientation of the projection vector V_p to produce an orthographic projection (a) and an oblique projection (b)

Orthographic Projection

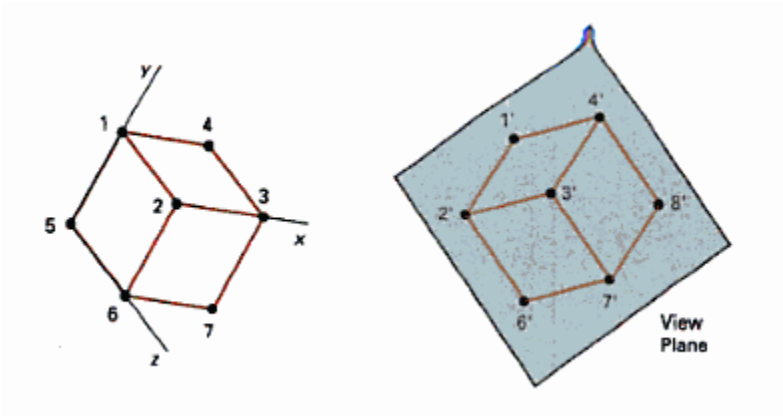
- Orthographic projections are used to produce the front, side and top views of an object.
- Front, side and rear orthographic projections of an object are called **elevations**.
- A top orthographic projection is called a **plan view**.
- This projection gives the measurement of lengths and angles accurately.

Orthographic projections of an object, displaying plan and elevation views



- The orthographic projection that displays more than one face of an object is called **axonometric orthographic projections**.
- The most commonly used axonometric projection is the **isometric projection**.
- It can be generated by aligning the projection plane so that it intersects each coordinate axis in which the object is defined as the same distance from the origin.

Isometric projection for a cube



- Transformation equations for an orthographic parallel projection are straight forward.

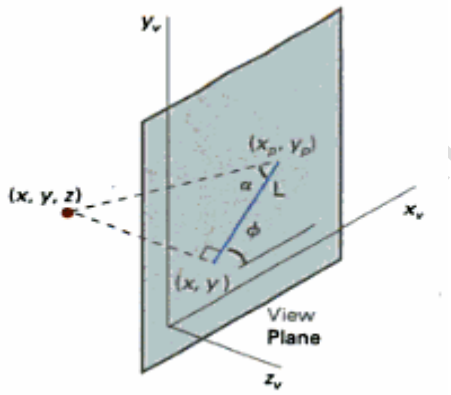
- If the view plane is placed at position z_{vp} along the z_v axis then any point (x,y,z) in viewing coordinates is transformed to projection coordinates as

$$x_p = x, \quad y_p = y$$

where the original z coordinates value is kept for the depth information needed in depth cueing and visible surface determination procedures.

Oblique Projection

- An oblique projection in obtained by projecting points along parallel lines that are not perpendicular to the projection plane.
- The below figure α and φ are two angles.



- Point (x,y,z) is projected to position (x_p,y_p) on the view plane.
- The oblique projection line form (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 in at an angle φ with the horizontal direction in the projection plane.
- The projection coordinates are expressed in terms of x,y, L and φ as

$$x_p = x + L\cos\varphi \qquad \text{- - - -(1)}$$

$$y_p = y + L\sin\varphi$$

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

$$\tan\alpha = z / L$$

thus,

$$L = z / \tan\alpha$$

$$= z\, L_1$$

where L_1 is the inverse of tanα, which is also the value of L when z = 1.

- The oblique projection equation (1) can be written as

$$x_p = x + z(L_1 \cos\varphi)$$

$$y_p = y + z(L_1 \sin\varphi)$$

- The transformation matrix for producing any parallel projection onto the $x_v y_v$ plane is

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

- An orthographic projection is obtained when $L_1 = 0$ (which occurs at a projection angle α of 90^0)
- Oblique projections are generated with non zero values for L_1.

Perspective Projections

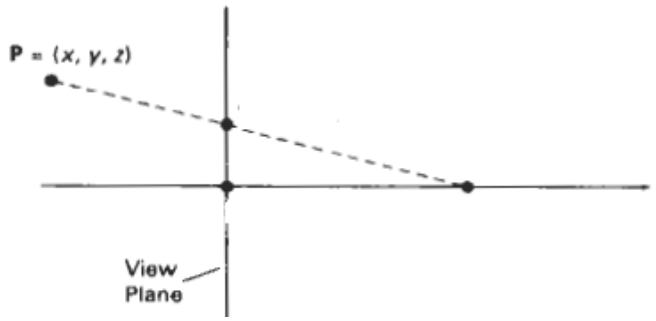
- To obtain perspective projection of a 3D object, we transform points along projection lines that meet at the projection reference point.
- If the projection reference point is set at position z_{prp} along the zv axis and the view plane is placed at z_{vp} as in fig , we can write equations describing coordinate positions along this perspective projection line in parametric form as

$$x' = x - x_u$$

$$y' = y - y_u$$

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

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



- Parameter u takes values from 0 to 1 and coordinate position (x', y',z') represents any point along the projection line.

- When u = 0, the point is at P = (x,y,z).
- At the other end of the line, u = 1 and the projection reference point coordinates $(0,0,z_{prp})$
- On the view plane $z` = z_{vp}$ and z` can be solved for parameter u at this position along the projection line:
- Substituting this value of u into the equations for x` and y`, we obtain the perspective transformation equations.

$$x_p = x((z_{prp} - z_{vp}) / (z_{prp} - z)) = x(d_p/(z_{prp} - z))$$

$$y_p = y((z_{prp} - z_{vp}) / (z_{prp} - z)) = y(d_p / (z_{prp} - z)) \text{ ----------(2)}$$

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

- Using a 3D homogeneous coordinate representation we can write the perspective projection transformation (2) in matrix form as

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -(z_{vp}/d_p) & z_{vp}(z_{prp}/d_p) \\ 0 & 0 & -1/d_p & z_{prp}/d_p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \text{ --------(3)}$$

- In this representation, the homogeneous factor is

$$h = (z_{prp}-z)/d_p \text{ ----------(4)}$$

and the projection coordinates on the view plane are calculated from eq (2)the homogeneous coordinates as

$$x_p = x_h / h$$

$$y_p = y_h / h \text{ ----------(5)}$$

where the original z coordinate value retains in projection coordinates for depth processing.

2.6 CLIPPING

- An algorithm for three-dimensional clipping identifies and saves all surface segments within the view volume for display on the output device. All parts of objects that are outside the view volume are discarded.
- Instead of clipping against straight-line window boundaries, we now clip objects against the boundary planes of the view volume.
- To clip a line segment against the view volume, we would need to test the relative position of the line using the view volume's boundary plane equations. By substituting the line endpoint coordinates into the plane equation of each boundary in turn, we

could determine whether the endpoint is inside or outside that boundary.

- An endpoint **(x, y, *z)*** of a line segment is outside a boundary plane if **Ax** + **By** + **Cz** + **D** > **0,** where ***A, B* , C**, and ***D*** are the plane parameters for that boundary.
- Similarly, the point is inside the boundary if **Ax** + **By** + **Cz** +**D** < **0**. Lines with both endpoints outside a boundary plane are discarded, and those with both endpoints inside all boundary planes are saved.
- The intersection of a line with a boundary is found using the line equations along with the plane equation.
- Intersection coordinates **(x_1, y_1, z_1)** are values that are on the line and that satisfy the plane equation **Ax_1,** + **By_1** + **Cz_1** + **D** = **0.**
- To clip a polygon surface, we can clip the individual polygon edges. First, we could test the coordinate extents against each boundary of the view volume to determine whether the object is completely inside or completely outside that boundary. If the coordinate extents of the object are inside all boundaries, we save it. If the coordinate extents are outside all boundaries, we discard it. Other-wise**,** we need to apply the intersection calculations.

Viewport Clipping

- Lines and polygon surfaces in a scene can be clipped against the viewport boundaries with procedures similar to those used for two dimensions, except that objects are now processed against clipping planes instead of clipping edges.
- The two-dimensional concept of region codes can be extended to three dimensions by considering positions in front and in back of the three-dimensional viewport, as well as positions that are left, right, below, or above the volume. For three dimensionalpoints, we need to expand the region code to six bits. Each point in the description of a scene is then assigned a six-bit region code that identifies the relative position of the point with respect to the viewport.
- For a line endpoint at position **(x, y, z)**, we assign the bit positions in the region code from right to left as

  bit 1 = 1, if $x < xv_{min}$(left)

  bit 2 = 1, if $x > xv_{max}$(right)

  bit 3 = 1, if $y < yv_{min}$(below)

  bit 4 = 1, if $y > yv_{max}$(above)

  bit 5 = 1, if $z < zv_{min}$(front)

  bit 6 = 1, if $z > zv_{max}$(back)

- For example, a region code of 101000 identifies a point as above and behind the viewport, and the region code 000000 indicates a point within the volume.
- A line segment can immediately identified as completely within the viewport if both endpoints have a region code of 000000. If either endpoint of a line segment does not have a region code of 000000, we perform the logical and operation on the two endpoint codes. The result of this and operation will be nonzero for any line segment that has both endpoints in one of the six outside regions.
- As in two-dimensional line clipping, we use the calculated intersection of a line with a viewport plane to determine how much of the line can be thrown away.

- The two-dimensional parametric clipping methods of **Cyrus-Beck** or **Liang-Barsky** can be extended to three-dimensional scenes. For a line segment with endpoints $\mathbf{P_1} = (\mathbf{x_1, y_1, z_1,})$ and $\mathbf{P_2} = (\mathbf{x_2, y_2, z_2),}$ we can write the parametric line equations as

$$x = x_1 + (x_2 - x_1)u \qquad 0<=u<=1$$

$$y = y_1 + (y_2 - y_1)u$$

$$z = z_1 + (z_2 - z_1)u \qquad \text{----------(1)}$$

- Coordinates **(x, y, z)** represent any point on the line between the two endpoints.
- At **u** = 0, we have the point **PI,** and **u** = 1 puts us at **P2.**
- To find the intersection of a line with a plane of the viewport, we substitute the coordinate value for that plane into the appropriate parametric expression of Eq.1 and solve for u. For instance, suppose we are testing a line against the $\mathbf{zv_{min},}$ plane of the viewport. Then

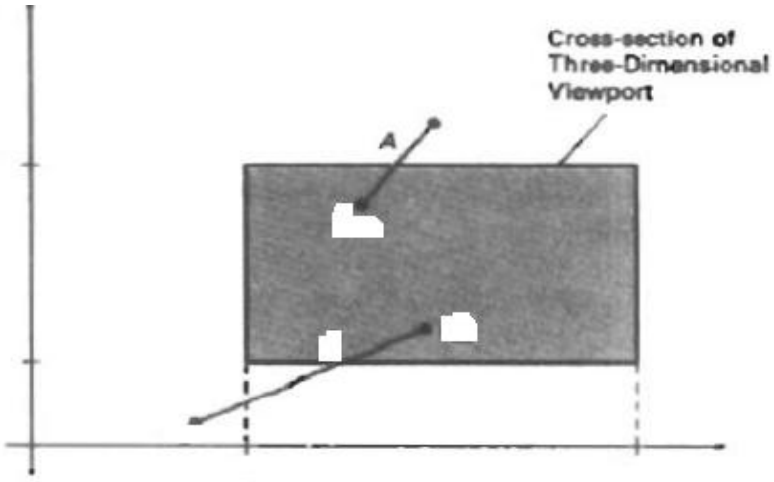
$$u = \frac{zv_{min} - z_1}{z_2 - z_1} \qquad \text{---------- (2)}$$

- When the calculated value for u is **not in the range from 0 to 1**, the line segment does not intersect the plane under consideration at any point between endpoints P1 and P2 (line A in fig).
- If the calculated value for *u* in Eq.2 **is in the interval from 0 to 1,** we calculate the intersection's x and y coordinates as

$$x_1 = x_1 + (x_2 - x_1) \left[\frac{zv_{min} - z_1}{z_2 - z_1} \right]$$

$$y_1 = y_1 + (y_2 - y_1) \left[\frac{zv_{min} - z_1}{z_2 - z_1} \right]$$

- If either x_1 or y_1 is not in the range of the boundaries of the viewport, then this line intersects the front plane beyond the boundaries of the volume (line ***B*** in Fig.)



2.7 Three Dimensional Viewing Functions

1. With parameters specified in world coordinates, elements of the matrix for transforming world coordinate descriptions to the viewing reference frame are calculated using the function.

EvaluateViewOrientationMatrix3(x_0,y_0,z_0,xN,yN,zN,xV,yV,zV,error,viewMatrix)

- This function creates the viewMatrix from input coordinates defining the viewing system.
- Parameters x_0,y_0,z_0 specify the sign of the viewing system.
- World coordinate vector (xN, yN, zN) defines the normal to the view plane and the direction of the positive z_v viewing axis.
- The world coordinates (xV, yV, zV) gives the elements of the view up vector.
- An integer error code is generated in parameter error if input values are not specified correctly.

2. The matrix proj matrix for transforming viewing coordinates to normalized projection coordinates is created with the function.

EvaluateViewMappingMatrix3
(xwmin,xwmax,ywmin,ywmax,xvmin,xvmax,yvmin,yvmax,zvmin,zvmax,
projType,xprojRef,yprojRef,zprojRef,zview,zback,zfront,error,projMatrix)

- Window limits on the view plane are given in viewing coordinates with parameters xwmin, xwmax, ywmin and ywmax.
- Limits of the 3D view port within the unit cube are set with normalized coordinates xvmin, xvmax, yvmin, yvmax, zvmin and zvmax.
- Parameter projType is used to choose the projection type either parallel or perspective.
- Coordinate position (xprojRef, yprojRdf, zprojRef) sets the projection reference point. This point is used as the center of projection if projType is set to perspective; otherwise, this point and the center of the viewplane window define the parallel projection vector.
- The position of the viewplane along the viewing z_v axis is set with parameter z view.
- Positions along the viewing z_v axis for the front and back planes of the view volume are given with parameters z front and z back.
- The error parameter returns an integer error code indicating erroneous input data.

2.8VISIBLE SURFACE IDENTIFICATION

A major consideration in the generation of realistic graphics displays is identifying those parts of a scene that are visible from a chosen viewing position.

2.8.1 Classification of Visible Surface Detection Algorithms

These are classified into two types based on whether they deal with object definitions directly or with their projected images

1. **Object space methods:** 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.
2. **Image space methods:** visibility is decided point by point at each pixel position on the projection plane. Most Visible Surface Detection Algorithms use image space methods.

2.8.2 Back Face Detection

A point (x, y,z) is "inside" a polygon surface with plane parameters A, B, C, and D if

$$A_x + B_y + C_z + D < 0 \qquad \text{----------(1)}$$

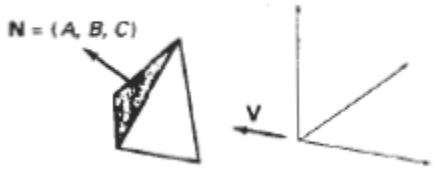
When an inside point is along the line of sight to the surface, the polygon must be a back face .

We can simplify this test by considering the normal vector **N** to a polygon surface, which has Cartesian components ***(A, B, C).*** In general, if V is a vector in the viewing direction from the eye position, as shown in Fig.,

then this polygon is a back face if **V . N > 0**

Furthermore, if object descriptions have been converted to projection coordinates and our viewing direction is parallel to the viewing z_v. axis, then **V = (0, 0, V_z)** and **V . N = V_zC** so that we only need to consider the sign of C, the ; component of the normal vector N.

In a right-handed viewing system with viewing direction along the negative z_v axis in the below Fig. the polygon is a back face if C < 0.



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

C<= 0

By examining parameter C for the different planes defining an object, we can immediately identify all the back faces.

2.8.3 Depth Buffer Method

A commonly used image-space approach to detecting visible surfaces is the depth-buffer method, which compares surface depths at each pixel position on the projection plane. This procedure is also referred to as the z-buffer method.

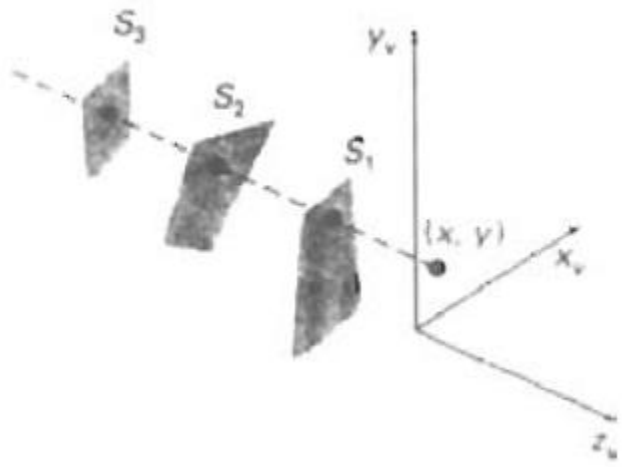
Each surface of a scene is processed separately, one point at a

time across the surface. The method is usually applied to scenes containing only polygon surfaces, because depth values can be computed very quickly and the method is easy to implement. But the mcthod can be applied to nonplanar surfaces.

With object descriptions converted to projection coordinates, each (x, y, z) position on a 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 depths can be compared by comparing z values. The figure shows

three surfaces at varying distances along the orthographic projection line from position (x,y) in a view plane taken as the (x_v,y_v) plane. Surface S_1, is closest at this position, so its surface intensity value at (x, y) is saved.



We can implement the depth-buffer algorithm in normalized coordinates, so that ***z*** values range from **0** at the back clipping plane to $\mathbf{Z_{max}}$ at the front clipping plane.

Two buffer areas are required.A **depth buffer** is used to store depth values for each (x, y) position as surfaces are processed, and the **refresh buffer** stores the intensity values for each position.

Initially,all positions in the depth buffer are set to 0 (minimum depth), and the refresh buffer is initialized to the background intensity.

We summarize the **steps of a depth-buffer algorithm as follows**:

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

depth (x, y)=0, refresh(x , y)=$I_{backgnd}$

2. 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 , refresh(x,y)= I_{surf}(x, y)

where $\mathbf{I_{backgnd}}$ is the value for the background intensity, and $\mathbf{I_{surf}(x, y)}$ 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.

Depth values for a surface position (x, y) are calculated from the plane equation for each surface:

$$z = \frac{-Ax - By - D}{C} \qquad \text{----------(1)}$$

For any scan line adjacent horizontal positions across the line differ by1, and a vertical y value on an adjacent scan line differs by 1. If the depth of position (x, y) has been determined to be z, then the depth z' of the next position (x +1, y) along the scan line is obtained from Eq. (1) as

$$z' = \frac{-A(x+1) - By - D}{C} \qquad \text{----------(2)}$$

Or $$z' = z - \frac{A}{C} \qquad \text{----------(3)}$$

On each scan line, we start by calculating the depth on a left edge of the polygon that intersects that scan line in the below fig. Depth values at each successive position across the scan line are then calculated by Eq. (3).

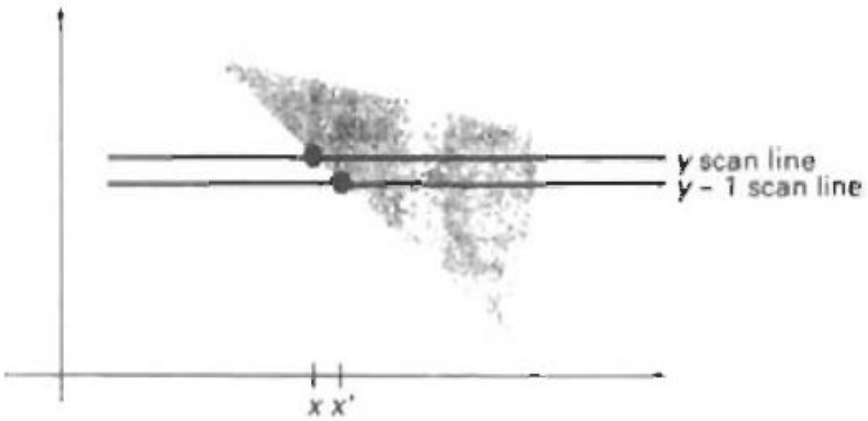
Scan lines intersecting a polygon surface

We first determine the y-coordinate extents of each polygon, and process the surface from the topmost scan line to the bottom scan line. Starting at a top vertex, we can recursively calculate x positions down a left edge of the polygon as x' = x - 1/m, where m is the slope of the edge.

Depth values down the edge are then obtained recursively as

$$z' = z + \frac{A/m + B}{C} \qquad \text{----------(4)}$$

Intersection positions on successive scan lines along a left polygon edge



If we are processing down a vertical edge, the slope is infinite and the recursive calculations reduce to

$$z' = z + \frac{B}{C} \qquad \text{----------(5)}$$

An alternate approach is to use a midpoint method or Bresenham-type algorithm for determining x values on left edges for each scan line. Also the method can be applied to curved surfaces by determining depth and intensity values at each surface projection point.

For polygon surfaces, the depth-buffer method is very easy to implement, and it requires no sorting of the surfaces in a scene. But it does require the availability of a second buffer in addition to the refresh buffer.

2.8.4 A- BUFFER METHOD

An extension of the ideas in the depth-buffer method is the A-buffer method. The A buffer method represents an **antialiased, area-averaged, accumulation-buffer method** developed by Lucasfilm for implementation in the surface-rendering system called **REYES** (an acronym for "Renders Everything You Ever Saw").

A drawback of the depth-buffer method is that it can only find one visible surface at each pixel position. The A-buffer method expands the depth buffer so that each position in the buffer can reference a linked list of surfaces.

Thus, more than one surface intensity can be taken into consideration at each pixel position, and object edges can be antialiased. Each position in the A-buffer has two fields:

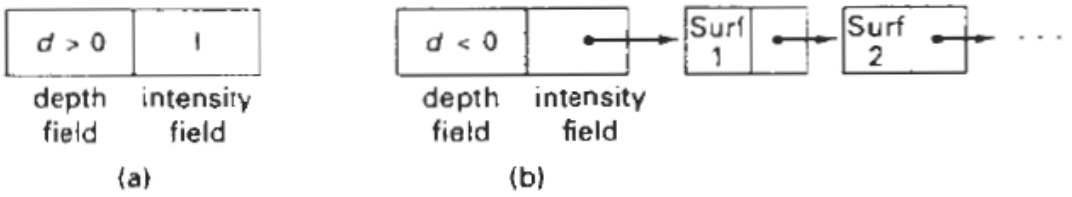
1)depth field - stores a positive or negative real number

2)intensity field - stores surface-intensity information or a pointer value.

If the depth field is positive, the number stored at that position is the depth of a single surface overlapping the corresponding pixel area. The intensity field then stores the RCB components of the surface color at that point and the percent of pixel coverage, as illustrated in Fig.A

If the depth field is negative, this indicates multiple-surface contributions to the pixel intensity. The intensity field then stores a pointer to a linked Iist of surface data, as in Fig. B.

Organization of an A-buffer pixel position (A) single surface overlap of the corresponding pixel area (B) multiple surface overlap



Data for each surface in the linked list includes

- RGB intensity components
- opacity parameter (percent of transparency)
- depth
- percent of area coverage
- surface identifier
- other surface-rendering parameters
- pointer to next surface

.

2.8.5 SCAN-LINE METHOD

This image-space method for removing hidden surfaces is an extension of the scan-line algorithm for filling polygon interiors. 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 refresh buffer.

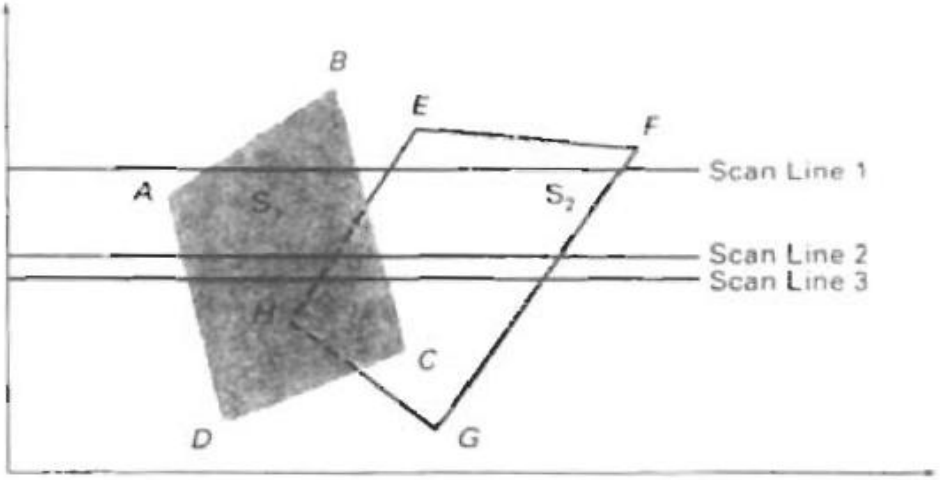
We assume that tables are set up for the various surfaces, which include both an edge table and a polygon table. The **edge table** contains coordinate endpoints for each line in-the scene, the inverse slope of each line, and pointers into the polygon table to identify the surfaces bounded by each line.

The **polygon table** contains coefficients of the plane equation for each surface, intensity information for the surfaces, and possibly pointers into the edge table.

To facilitate the search for surfaces crossing a given scan line, we can set up an active list of edges from information in the edge table. This active list will contain only edges that cross the current scan line, sorted in order of increasing x**.**

In addition, we define a flag for each surface that is set on or off to indicate whether a position along a scan line is inside or outside of the surface. Scan lines are processed from left to right. At the leftmost boundary of a surface, the surface flag is turned on; and at the rightmost boundary, it is turned off.

Scan lines crossing the projection of two surfaces S_1 and S_2 in the view plane. Dashed lines indicate the boundaries of hidden surfaces



The figure 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 S2 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.

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.

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.

2.8.6 Depth Sorting Method

Using both image-space and object-space operations, the depth-sorting method performs the following basic functions:

1. Surfaces are sorted in order of decreasing depth.

2. Surfaces are scan converted in order, starting with the surface of greatest depth.

Sorting operations are carried out in both image and object space, and the scan conversion of the polygon surfaces is performed in image space.

This method for solving the hidden-surface problem is often referred to as the **painter's algorithm**. In creating an oil painting, an artist first paints the background colors. Next, the most distant objects are added, then the nearer objects, and so forth. At the final step, the foreground objects are painted on the canvas over the background and other objects that have been painted on the canvas. Each layer of paint covers up the previous layer.

Using a similar technique, we first sort surfaces according to their distance from the view plane. The intensity values for the farthest surface are then entered into the refresh buffer. Taking each succeeding surface

in turn we "paint" the surface intensities onto the frame buffer over the intensities of the previously processed surfaces.

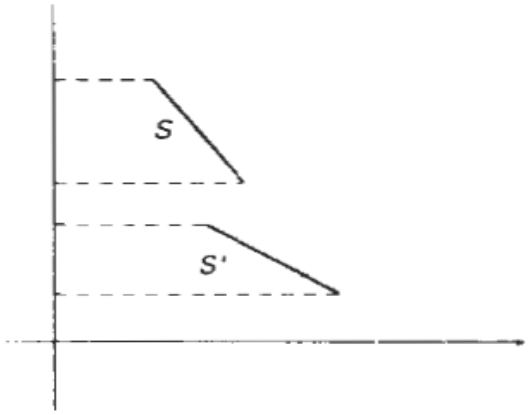
Painting polygon surfaces onto the frame buffer according to depth is carried out in several steps. Assuming we are viewing along the-z direction,

1.surfaces are ordered on the first pass according to the smallest ***z*** value on each surface.

2.Surfaces with the greatest depth is then compared to the other surfaces in the list to determine whether there are any overlaps in depth. If no depth overlaps occur, S is scan converted. Figure shows two surfaces that overlap in the xy plane but have no depth overlap.

3.This process is then repeated for the next surface in the list. As long as no overlaps occur, each surface is processed in depth order until all have been scan converted.

4. If a depth overlap is detected at any point in the list, we need to make some additional comparisons to determine whether any of the surfaces should be reordered.**Two surfaces with no depth overlap**



We make the following tests for each surface that overlaps with S. If any one of these tests is true, no reordering is necessary for that surface. The tests are listed in order of increasing difficulty.

1. The bounding rectangles in the xy plane for the two surfaces do not overlap

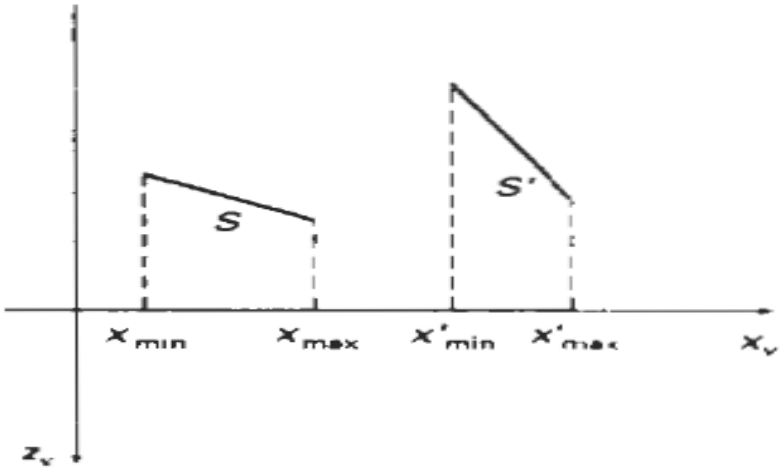
2. Surface S is completely behind the overlapping surface relative to the viewing position.

3**.** The overlapping surface is completelv in front of S relative to the viewing position.

4. The projections of the two surfaces onto the view plane do not overlap.

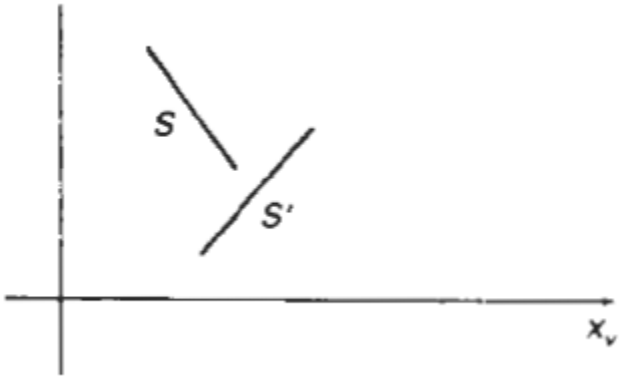
Test 1 is performed in two parts. We first check for overlap in the x direction,then we check for overlap in the y direction. If either of these directions show no overlap, the two planes cannot obscure one other. An

example of two surfaces that overlap in the z direction but not in the ***x*** direction is shown in Fig.



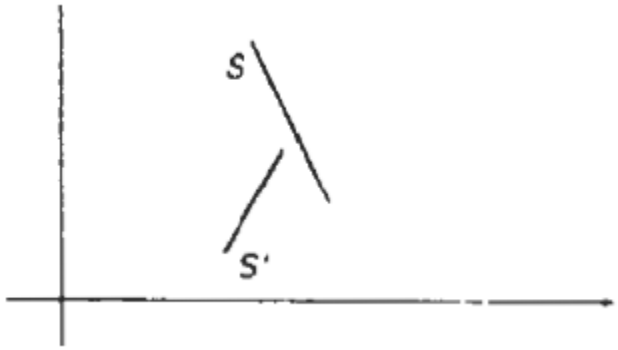
We can perform tests 2 and 3 with an "inside-outside" polygon test. That is,we substitute the coordinates for all vertices of S into the plane equation for the overlapping surface and check the sign of the result. If the plane equations are setup so that the outside of the surface is toward the viewing position, then *S* is behind S' if all vertices of *S* are "inside" S'

Surface S is completely behind (inside) the overlapping surface S'



Similarly, S' is completely in front of S if all vertices of S are "outside" of ***S'.*** Figure shows an overlapping surface S' that is completely in front of S, but surface S is not completely inside S'.

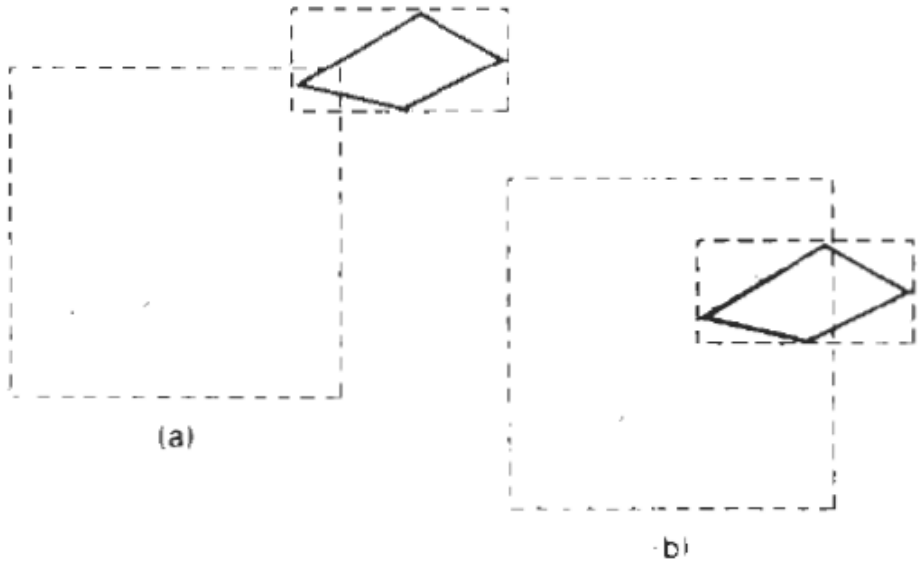
Overlapping surface S' is completely in front(outside) of surface S but s is not completely behind S'



If tests 1 through 3 have all failed, we try test 4 by checking for intersections between the bounding edges of the two surfaces using line equations in the xy plane. As demonstrated in Fig., two surfaces may or may not intersect even though their coordinate extents overlap in the x, y, and z directions.

Should all four tests fail with a particular overlapping surface S', we interchange surfaces S and S' in the sorted list.

Two surfaces with overlapping bounding rectangles in the xy plane



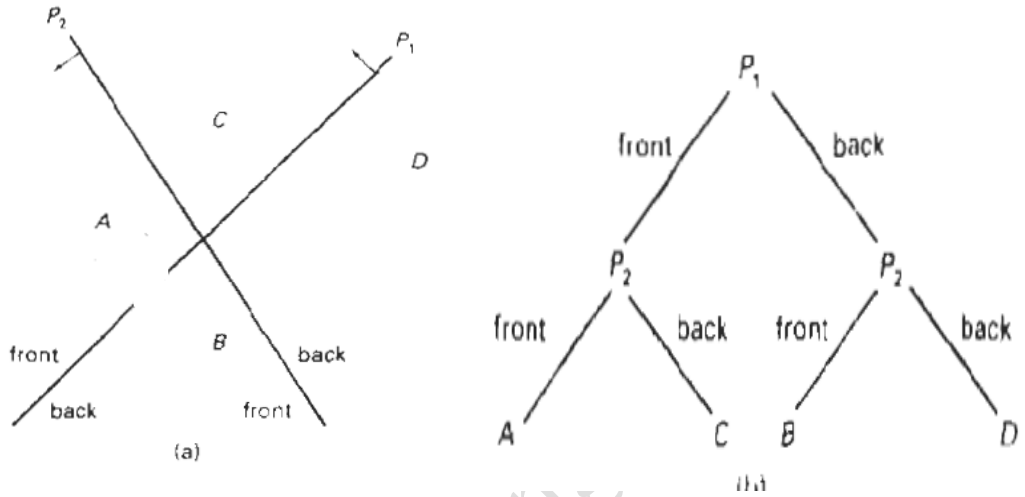
2.8.7 BSP-Tree Method

A binary space-partitioning (BSP) tree is an efficient method for determining object visibility by painting surfaces onto the screen from back to front, as in the painter's algorithm. The BSP tree is particularly

useful when the view reference point changes, but the objects in a scene are at fixed positions.

Applying a BSP tree to visibility testing involves identifying surfaces that are "inside" and "outside" the partitioning plane at each step of the space subdivision, relative to the viewing direction. The figure(a) illustrates the basic concept in this algorithm.

A region of space (a) is partitioned with two planes P_1 and P_2 to form the BSP tree representation in (b)



With plane P_1**,**we first partition the space into two sets of objects. One set of objects is behind, or in back of, plane P_1, relative to the viewing direction, and the other set is in front of P_1. Since one object is intersected by plane P_1, we divide that object into two separate objects, labeled A and B.

Objects A and C are in front of P_1 and objects B and D are behind P_1. We next partition the space again with plane P_2 and construct the binary tree representation shown in Fig.(b).

In this tree, the objects are represented as terminal nodes, with front objects as left branches and back objects as right branches.

2.8.8 Area – Subdivision Method

This technique for hidden-surface removal is essentially an image-space method ,but object-space operations can be used to accomplish depth ordering of surfaces.

The area-subdivision method takes advantage of area coherence in a scene by locating those view areas that represent part of a single surface. We apply this method by successively dividing the total viewing

area into smaller and smaller rectangles until each small area is the projection of part of a single visible surface or no surface at all.

To implement this method, we need to establish tests that can quickly identify the area as part of a single surface or tell us that the area is too complex to analyze easily. Starting with the total view, we apply the tests to determine whether we should subdivide the total area into smaller rectangles. If the tests indicate that the view is sufficiently complex, we subdivide it. Next. we apply the tests to each of the smaller areas, subdividing these if the tests indicate that visibility of a single surface is still uncertain. We continue this process until the subdivisions are easily analyzed as belonging to a single surface or until they are reduced to the size of a single pixel. An easy way to do this is to successively divide the area into four equal parts at each step.

Tests to determine the visibility of a single surface within a specified area are made by comparing surfaces to the boundary of the area. There are four possible relationships that a surface can have with a specified area boundary. We can describe these relative surface characteristics in the following way (Fig.):

- Surrounding surface-One that completely encloses the area.
- Overlapping surface-One that is partly inside and partly outside the area.
- Inside surface-One that is completely inside the area.
- Outside surface-One that is completely outside the area.

Possible relationships between polygon surfaces and a rectangular area

The tests for determining surface visibility within an area can be stated in terms of these four classifications. No further subdivisions of a specified area are needed if one of the following conditions is true:

1. All surfaces are outside surfaces with respect to the area.

2. Only one inside, overlapping, or surrounding surface is in the area.

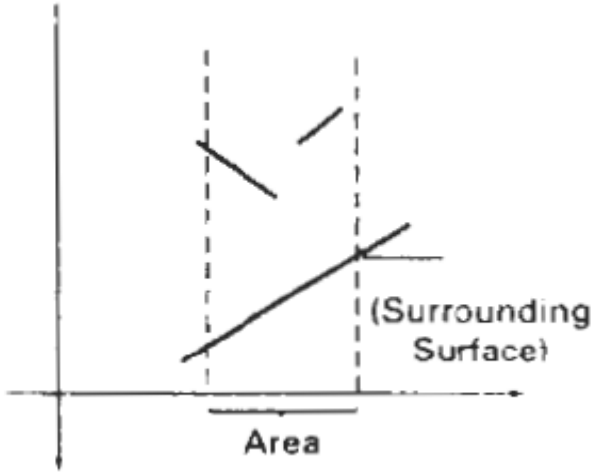
3. A surrounding surface obscures all other surfaces within the area boundaries.

Test 1 can be carrieded out by checking the bounding rectangles of all surfaces against the area boundaries.

Test 2 can also use the bounding rectangles in the xy plane to identify an inside surface

One method for implementing **test 3** is to order surfaces according to their minimum depth from the view plane. For each surrounding surface, we then compute the maximum depth within the area under consideration. If the maximum depth of one of these surrounding surfaces is closer to the view plane than the minimum depth of all other surfaces within the area, test 3 is satisfied.

Within a specified area a surrounding surface with a maximum depth of Z_{max} obscures all surfaces that have a minimum depth beyond Z_{max}



Another method for carrying out test 3 that does not require depth sorting is to use plane equations to calculate depth values at the four vertices of the area for all surrounding, overlapping, and inside surfaces, If the calculated depths for one of the surrounding surfaces is less than the calculated depths for all other surfaces, test 3 is true. Then the area can be filled with the intensity values of thesurrounding surface.

For some situations, both methods of implementing test 3 will fail to identify correctly a surrounding surface that obscures all the other surfaces. It is faster to subdivide the area than to continue with more complex testing.

Once outside and surrounding surfaces have been identified for an area, they will remain outside and surrounding surfaces for all subdivisions of the area. Furthermore, some inside and overlapping surfaces can be expected to be eliminated as the subdivision process continues, so that the areas become easier to analyze.

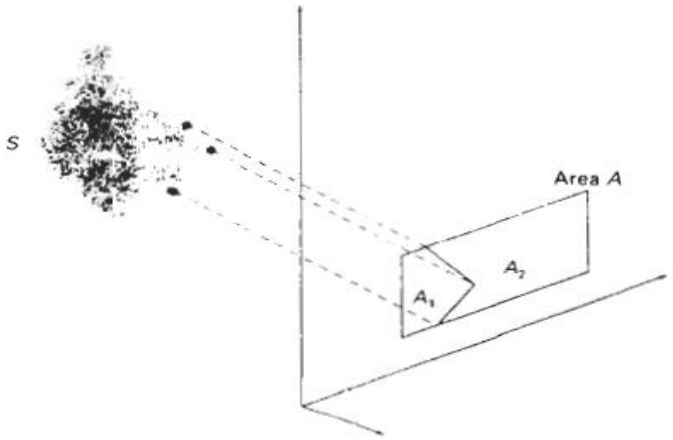
In the limiting case, when a subdivision the size of a pixel is produced, we simply calculate the depth of each relevant surface at that

point and transfer the intensity of the nearest surface to the frame buffer.

As a variation on the basic subdivision process, we could subdivide areas along surface boundaries instead of dividing them in half. The below Figure illustrates this method for subdividing areas. The projection of the boundary of surface S is used to partition the original area into the subdivisions A_1 and A2**.** Surface S is then a surrounding surface for A_1**,** and visibility tests 2 and 3 can be applied to determine whether further subdividing is necessary.

In general, fewer subdivisions are required using this approach, but more processing is needed to subdivide areas and to analyze the relation of surfaces to the subdivision boundaries.

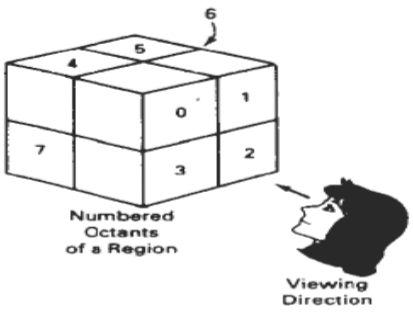
Area A is subdivided into A_1 and A_2, using the boundary of surface *S* on the view plane.



2.8.9 OCTREE METHODS

When an **octree** representation is used for the viewing volume, hidden-surface elimination is accomplished by projecting octree nodes onto the viewing surface in a front-to-back order.

In the below Fig. the front face of a region of space (the side toward the viewer) is formed with octants 0, 1**,** 2, and 3. Surfaces in the front of these octants are visible to the viewer. Any surfaces toward the re in the back octants (4,5,6, and 7) may be hidden by the front surfaces.



Back surfaces are eliminated, for the viewing directionby processing data elements in the octree nodes in the order 0, 1, 2,3,4, 5, 6, 7.

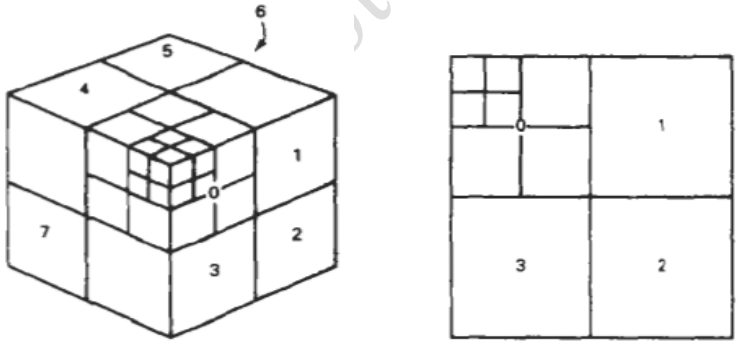
This results in a depth-first traversal of the octree**,** so that nodes representing octants 0, 1.2, and 3 for the entire region are visited before the nodes representing octants 4,5,6, and 7.

Similarly, the nodes for the front four suboctants of octant 0 are visited before the nodes for the four back suboctants. The traversal of the octree continues in this order for each octant subdivision.

When a color value is encountered in an octree node, the pixel area in the frame buffer corresponding to this node is assigned that color value only if no values have previously been stored in this area. In this way, only the front colors are loaded into the buffer. Nothing is loaded if an area is void. Any node that is found to be completely obscured is eliminated from further processing, so that its subtrees are not accessed.

Different views of objects represented as octrees can be obtained by applying transformations to the octree representation that reorient the object according to the view selected.

A method for displaying an octree is first to map the octree onto a quadtree of visible areas by traversing octree nodes from front to back in a recursive procedure. Then the quadtree representation for the visible surfaces is loaded into the frame buffer. The below Figure depicts the **octants in a region of space and the corresponding quadrants on the view plane.**



View Plane

Contributions to quadrant 0 come from octants 0 and 4. Color values in quadrant 1 are obtained from surfaces in octants1 and 5, and values in each of the other two quadrants are generated from the pair of octants aligned with each of these quadrants.

In most cases, both a front and a back octant must be considered in determining the correct color values for a quadrant. But if the front octant is homogeneously filled with some color, we do not process the

back octant. If the front is empty the, the rear octant is processed. Otherwise, two ,.recursive calls are made, one for the rear octant and one for the front octant.

```
typedef enum ( SOLID, MIXED } Status;
bdefine EMPTY -1
typedef struct tOctree (
int id;
Status status;
union (
int color;
struct tOctree  *children[8]:
) data;
}Octree:
typedef struct tQuadtree i
int id:
Status status;
union [
int color;
struct tQuadtree *children[4];
) data;
) Quadtree;
int nQuadtree = 0.
void octreeToQuadtree (Octree  *oTree. Quadtree *qTree)
(
Octree  *front. *back:
Quadtree  *newQuadtree;
int i, j;
if (oTree->status == SOLID) (
qTree->status = SOLID:
qTree->data.color = oTree->data color:
return:
)
qTree->status = MIXED:
/*Fill in each quad of the quadtree *I
for ( i = O ; i<4; i++)
 {
front = oTree->data.children[il;
back = oTree->data..children[i+4];
newQuadtree = (Quadtree *) malloc (sizeof (Quadtree)):
newQuadtree->id = nQuadtree++;
newQuadtree->status = SOLID;
qTree->data.childrenIil = newQuadtree;
if (front->status == SOLID)
if (front->data.color != EMPTY)
qTree->data.children[i]->data.color = front->data.color;
else
if (back->status == SOLID)
if (back->data.color != EMPTY)
qTree->data.children[i]->data.color = back->data.color;
```

```
else
qTree->data.children[il->data.color = EMPTY;
else ( / * back node is mixed * /
newQuadtree->status = MIXED;
octreeToQuadtree (back, newguadtree);
octreeToQuadtree (front, newQuadtree):
}
}
}
```

2.8.10 RAY CASTING METHOD

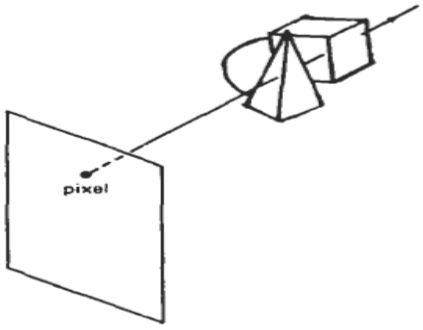
If we consider the line of sight from a pixel position on the view plane through a scene, as in the Fig. below, we can determine which objects in the scene intersect this line.

After calculating all ray-surface intersections, we identify the visible surface as the one whose intersection point is closest to the pixel. This visibility detection scheme uses ray-casting procedures.

Ray casting, as a visibility detection tool, is based on geometric optics methods, which trace the paths of light rays. Since there are an infinite number of light rays in a scene and we are interested only in those rays that pass through pixel positions, we can trace the light-ray paths backward from the pixels through the scene.

The ray-casting approach is an effective visibility-detection method for scenes with curved surfaces, particularly spheres.

A ray along a line of sight from a pixel position through a scene



We can think of ray casting as a variation on the depth-buffer method . In the depth-buffer algorithm, we process surfaces one at a time and calculate depth values for all projection points over the surface. The calculated surface depths are then compared to previously stored depths to determine visible surfaces at each pixel.

In ray casting, we process pixels one at a time and calculate depths for all surfaces along the projection path to that pixel. Ray casting is a special case of **ray-tracing algorithms** that trace multiple ray paths to pick up global reflection and refraction contributions from multiple objects in a scene. With ray casting, we only follow a ray out from each pixel to the nearest object.

2.8.11 Curved Surfaces

Effective methods for determining visibility for objects with curved surfaces include ray-casting and octree methods.

With ray casting, we calculate ray-surface intersections and locate the smallest intersection distance along the pixel ray.

With octree, once the representation has been established from the input definition of the objects, all visible surfaces are identified with the same processing procedures.

No special considerations need be given to different kinds of curved surfaces. We can also approximate a curved surface as a set of plane, polygon surfaces and use one of the other hidden-surface methods.With some objects, such as spheres, it can be more efficient as well as more accurate to use ray casting and the curved-surface equation.

Curved-Surface Representations

We can represent a surface with an implicit equation of the form **f(x, y, *z)* = 0** or with a parametric representation .

Spline surfaces, for instance, are normally described with parametric equations.

In some cases, it is useful to obtain an explicit surface equation, as, for example, a height function over an **xy** ground plane:

z=f(x,y)

Many objects of interest, such as spheres, ellipsoids, cylinders, and cones, have quadratic representations.

Scan-line and ray-casting algorithms often involve numerical approximation techniques to solve the surface equation at the intersection point with a scan line or with a pixel ray. Various techniques, including parallel calculations and fast hardware implementations, have been developed for solving the curved-surface equations for commonly used objects.

Surface Contour Plots

For many applications in mathematics, physical sciences, engineering and other fields, it is useful to display a surface function with a set of contour lines that shows the surface shape. The surface may be described with an equation or with data tables.

With an explicit functional representation, we can plot the visible surface contour lines and eliminate those contour sections that are hidden by the visible parts of the surface.

To obtain an **xy** plot of a functional surface, we write the surface representation in the form

y=f(x,z) ----------(1)

A curve in the **xy** plane can then be plotted for values of **z** within some selected range, using a specified interval Δ**z.** Starting with the largest value of ***z,*** we plot the curves from "front" to "back" and eliminate hidden sections.

We draw the curve sections on the screen by mapping an xy range for the function into an xy pixel screen range. Then, unit steps are taken in x and the corresponding y value for each x value is determined from Eq. (1) for a given value of z.

One way to identify the visible curve sections on the surface is to maintain a list of y_{min}, and y_{max}, values previously calculated for the pixel x coordinates on the screen.

As we step from one pixel **x** position to the next, we check the calculated y value against the stored range, y_{min}, and y_{max}, for the next pixel.

If $y_{min} <= y <= y_{max}$ that point on the surface is not visible and we do not plot it. But if the calculated y value is outside the stored y bounds for that pixel, the point is visible. We then plot the point and reset the bounds for that pixel.

2.8.12 WireFrame Methods

When only the outline of an object is to be displayed, visibility tests are applied to surface edges. Visible edge sections are displayed, and hidden edge sections can either be eliminated or displayed differently from the visible edges. For example,hidden edges could be drawn as dashed lines.

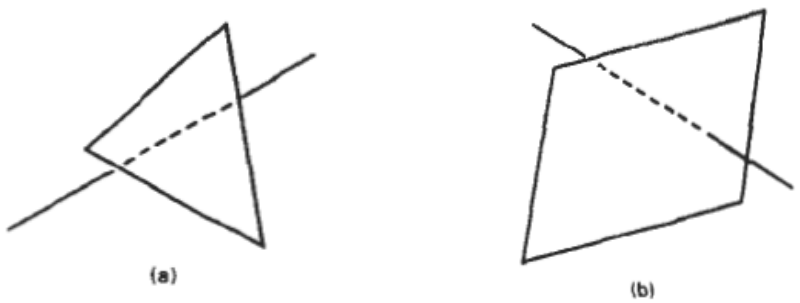
Procedures for determining visibility of object edges are referred to as **wireframe-visibility methods**. They are also called **visible line detection methods** or **hidden-line detection methods**.

A direct approach to identifying the visible lines in a scene is to compare each line to each surface, we now want to determine which sections of the lines are hidden by surfaces.

For each line, depth values are compared to the surfaces to determine which line sections are not visible. We can use coherence methods to identify hidden line segments without actually testing each coordinate position. If both line intersections with the projection of a surface boundary have greater depth than the surface at those points, the line segment between the intersections is completely hidden, as in Fig. (a).

This is the usual situation in a scene, but it is also possible to have lines and surfaces intersecting each other. When a line has greater depth at one boundary intersection and less depth than the surface at the other boundary intersection, the line must penetrate the surface interior, as in Fig. (b). In this case, we calculate the intersection point of the line with the surface using the plane equation and display only the visible sections.

Hidden line sections (dashed) for a line that (a) passes behind a surface and (b) penetrates a surface



Some visible-surface methods are readily adapted to wireframe visibility testing. Using a back-face method, we could identify all the back surfaces of an object and display only the boundaries for the visible surfaces. With depth sorting, surfaces can be painted into the refresh buffer so that surface interiors are in the background color, while boundaries are in the foreground color.

By processing the surfaces from back to front, hidden lines are erased by the nearer surfaces.An area-subdivision method can be adapted to hidden-line removal by displaying only the boundaries of visible surfaces. Scan-line methods can be used to display visible lines by setting points along the scan line that coincide with boundaries of visible surfaces.

2.8.13 VISIBILITY-DETECTION FUNCTIONS

Often, three-dimensional graphics packages accommodate several visible-surface detection procedures, particularly the back-face and depth-buffer methods.

A particular function can then be invoked with the procedure name, such as **back-Face** or **depthBuffer**.