

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.

PROJECTIONS

After converting the description of objects from world co-ordinates to viewing co-ordinates, we can project the three dimensional objects onto the two dimensional view plane. There are two basic ways of projecting objects onto the view plane.

1. Parallel projection
2. Perspective projection

2.1. PARALLEL AND PERSPECTIVE PROJECTIONS

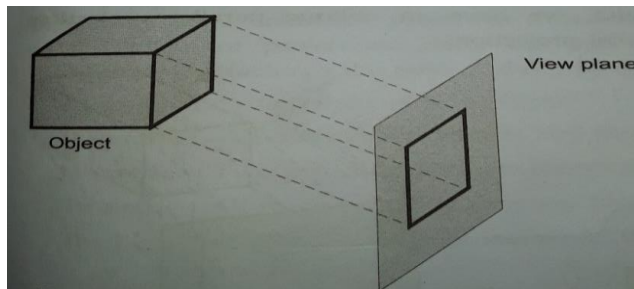
Parallel Projection:

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.



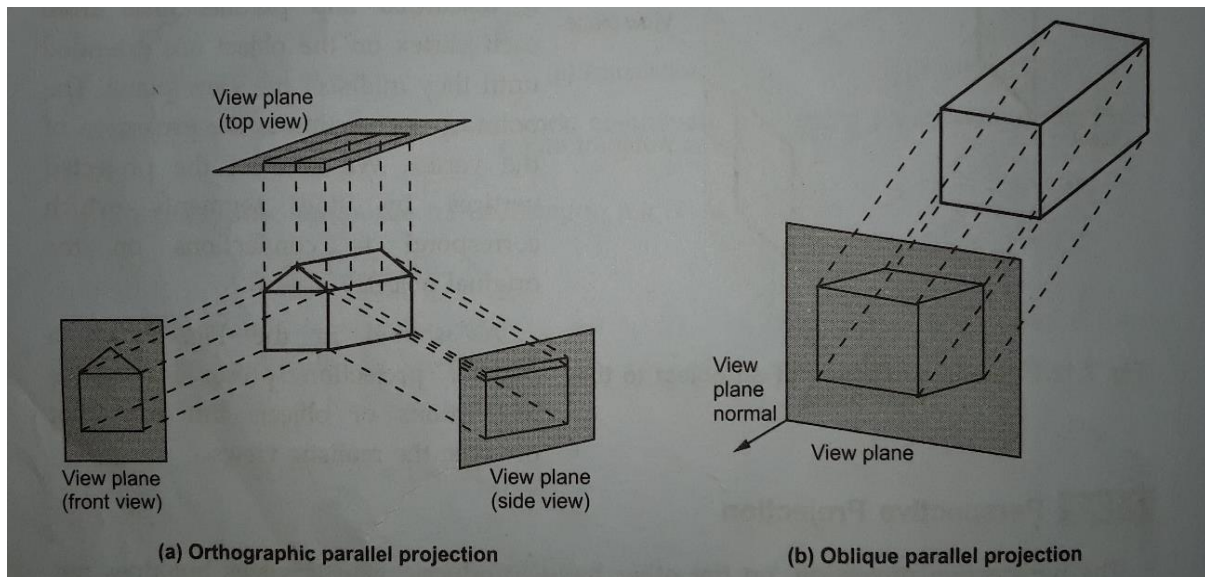
Types of parallel projections:

Parallel projections are basically categorized into two types.

Depending on the relation between the direction of projection and normal to the view plane.

- When the direction of projection is perpendicular to the view plane, we have an orthographic parallel projection.

Otherwise, we have an oblique parallel projection.



Orthographic projection:

- Orthographic projections are classified as **axonometric orthographic projection** and **multiview orthographic projection**.

Axonometric orthographic projection:

The most common types of orthographic projections are

- the front projection,
- top projection and
- side projection. In all these,

❖ the projection plane (view plane) is perpendicular to the principle axis. These projections are often used in engineering drawing to depict machine parts, assemblies, buildings and so on.

- ❖ The orthographic projections can display more than one fact of an object. Such an orthographic projection is called **axonometric orthographic projection**.

The most commonly used axonometric orthographic projection is the **isometric** projection.

Types of Axonometric Projection:

Axonometric projections of three types:

Isometric: All three principle axes are foreshortened equally.

Dimetric: Two principle axes are foreshortened equally.

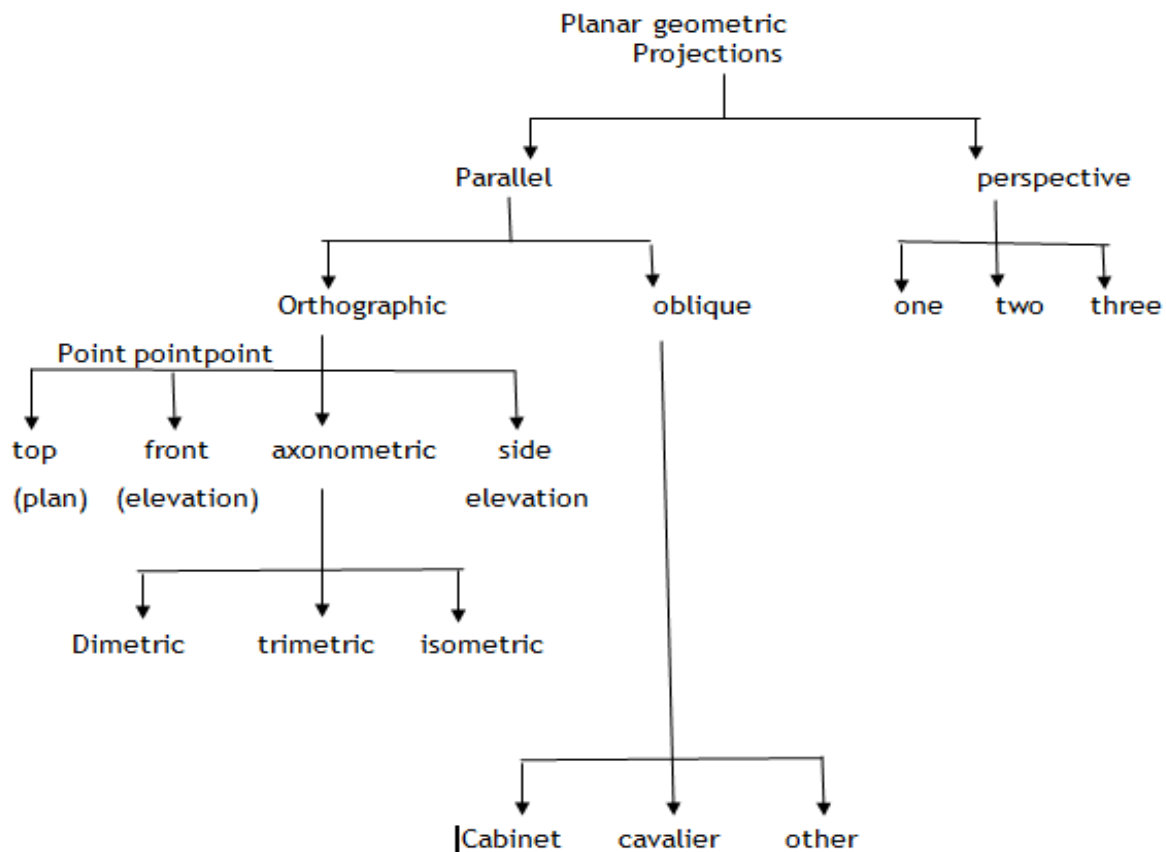
Trimetric: All three principle axes are foreshortened unequally.

Perspective Projection:

It is a method for generating a view of a three dimensional scene is to project points to the display plane along 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.



2.3.POLYGON SURFACES CONCEPT

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

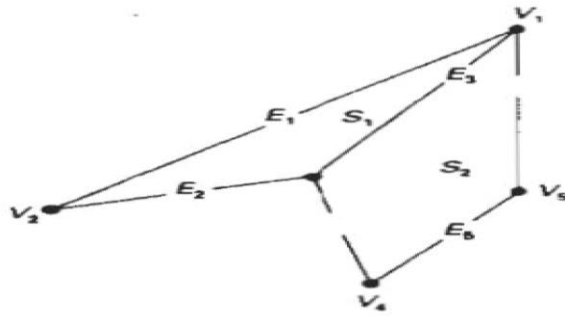
Polygon Tables

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.



Vertex table Edge Table Polygon surface table
V1 : X1, Y1, Z1 E1 : V1, V2 S1 : E1, E2, E3 V2 : X2, Y2, Z2
E2 : V2, V3 S2 : E3, E4, E5, E6 V3 : X3, Y3, Z3 E3 : V3, V1
V4 : X4, Y4, Z4 E4 : V3, V4 V5 : X5, Y5, Z5
E5 : V4, V5 E6 : V5, V1

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, edgetable can be expanded to include forward pointers into the polygon table so that common edges between polygons can be identified more rapidly.

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 $Ax + By + Cz + D = 0$ ----(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.

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.

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 is decorated to produce a polygon-mesh approximation to the surface.

Quadric Surfaces

The quadric surfaces are described with second degree equations (quadratics). They include spheres, ellipsoids, tori, paraboloids, 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.

$$x^2 + y^2 + z^2 = r^2$$

Ellipsoid

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

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.

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

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.

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 is described with a set of parametric coordinate functions or the form $x = x(u)$, $y = y(u)$, $z = z(u)$, $u_1 \leq u \leq u_2$

Zero order parametric continuity referred to as C0 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 C1 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.

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 to as G0 continuity means that the two curves sections must have the same coordinate position at the boundary point.

First order Geometric Continuity referred to as G1 continuity means that the parametric first derivatives are proportional at the intersection of two successive sections.

Second order Geometric continuity referred to as G2 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.

2.4.VISUALIZATION OF DATA SETS

CONCEPT:

The use of graphical methods as an aid in scientific and engineering analysis is commonly referred to as **scientific visualization**.

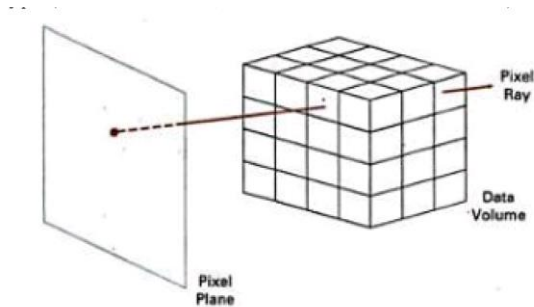
This involves the visualization of data sets and processes that may be difficult or impossible to analyze without graphical methods. Example medical scanners, satellite and spacecraft scanners. Visualization techniques are useful for analyzing process that occur over a long period of time or that cannot be 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).



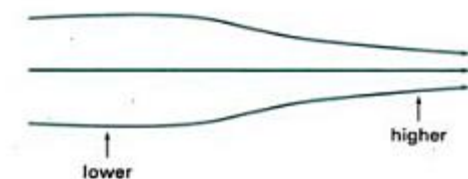
Visual representation for Vector fields

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 $|V|$ 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.



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.

SIGNIFICANCE:

The use of graphical methods as an aid in scientific and engineering analysis is commonly referred to as **scientific visualization**.

2.5 THREE DIMENSIONAL GEOMETRIC AND MODELING TRANSFORMATIONS:

CONCEPT:

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

Translation

In a three dimensional homogeneous coordinate representation, a point or an object is translated from position $P = (x, y, z)$ to position $P^1 = (x^1, y^1, z^1)$ with the matrix operation.

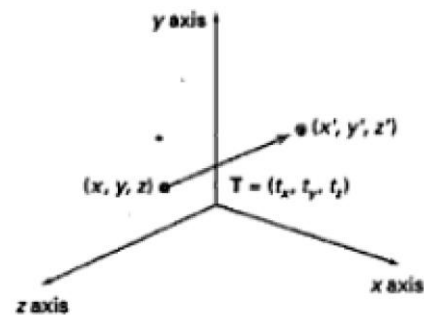
Rotation

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.

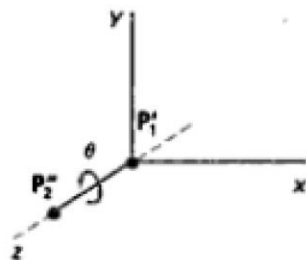
Co-ordinate Axes Rotations

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

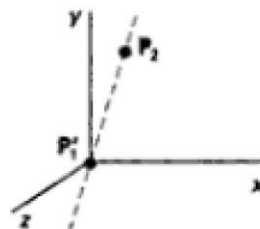


$$X^1 = x \cos \theta - y \sin \theta$$

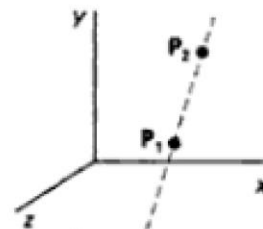
$$Y^1 = x \sin \theta + y \cos \theta$$



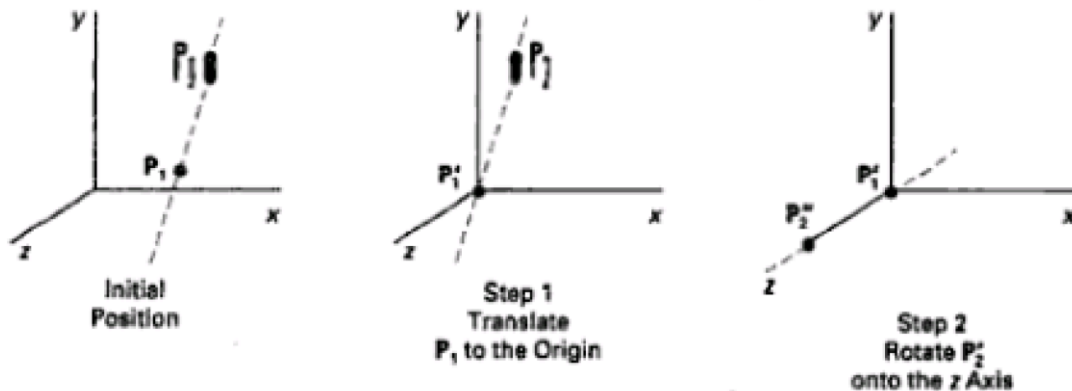
Step 3
Rotate the
Object Around the
z Axis



Step 4
Rotate the Axis
to the Original
Orientation



Step 5
Translate the
Rotation Axis
to the Original
Position



Scaling

The matrix expression for the scaling transformation of a position $P = (x, y, z)$

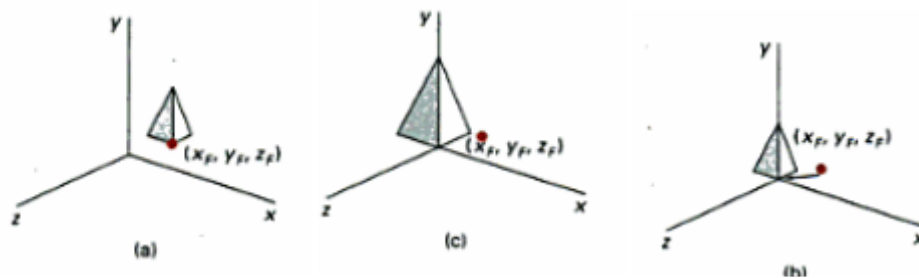
Scaling an object changes the size of the object and repositions the object relative to the coordinate origin.

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



Other Transformations

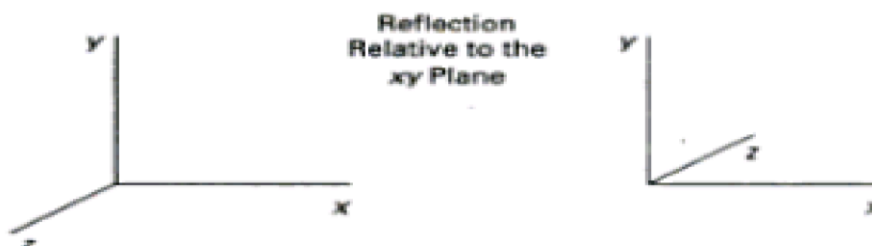
Reflections

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° rotations about the axis.

Reflection relative to a plane are equivalent to 180° rotations in 4D space.

When the reflection plane is a coordinate plane (either xy, xz or yz) then the transformation can be a conversion between left-handed and right-handed systems.

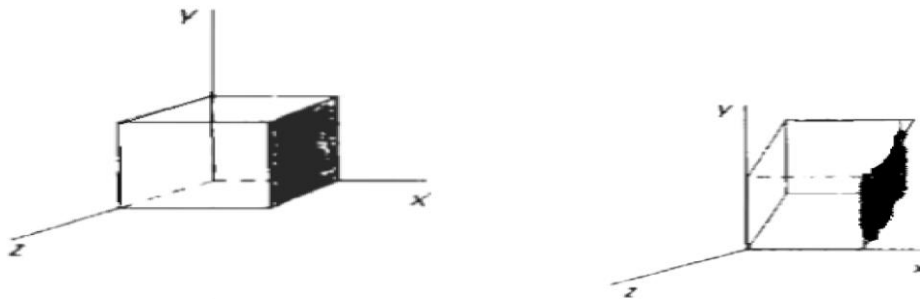


Shears

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.



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 matrix 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.

Three 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 **`buildTransformationMatrix3`** and **`composeTransformationMatrix3`** 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.

2.6.THREE-DIMENSIONAL VIEWING

CONCEPT:

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.

Viewing 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 sight 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

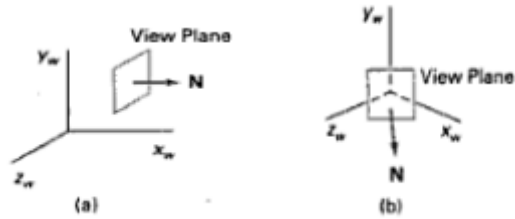
1. Once the scene has been modeled, world coordinates position is converted to viewing coordinates.
2. The viewing coordinates system is used in graphics packages as a reference for specifying the observer viewing position and the position of the projection plane.
3. Projection operations are performed 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.

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.

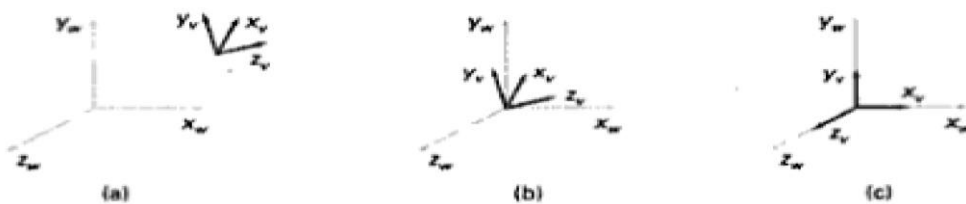


Transformation from world to viewing coordinates

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



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

$$n = N / (|N|) = (n_1, n_2, n_3) \quad u = (V * N) / (|V * N|) = (u_1, u_2, u_3) \quad v = n * u = (v_1, v_2, v_3)$$

This method automatically adjusts the direction for v , so that v is perpendicular to n .

2.7.VISIBLE SURFACE IDENTIFICATION

CONCEPT

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.

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.

BACK-FACE DETECTION

A fast and simple object-space method for identifying the back faces of a polyhedron is based on the "inside-outside" tests discussed in Chapter 10. A point (x, y, z) is "inside" a polygon surface with plane parameters A, B, C , and D if

$$Ax + By + Cz + D < 0$$

When an inside point is along the line of sight to the surface, the polygon must be a back face (we are inside that face and cannot see the front of it from our viewing position).

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 (or "camera") position, as shown in Fig. 13-1, then this polygon is a back face if

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

Furthermore, if object descriptions have been converted to projection coordinates and our viewing direction is parallel to the viewing z_v -axis, then $\mathbf{V} = (0, 0, \mathbf{v}_z)$ and

$$\mathbf{V} \cdot \mathbf{N} = \mathbf{v}_z C$$

so that we only need to consider the sign of **C**, the z component of the normal vector **N**

In a right-handed viewing system with viewing direction along the negative z_v axis (Fig. 13-21, the polygon is a back face if $C < 0$. Also, we cannot see any face whose normal has z component $C=0$, since our viewing direction is grazing that polygon. Thus, in general, we can label any polygon as a back face if its normal vector has a z component value:

$$C \leq 0$$

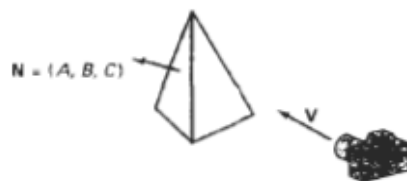


Figure 13-1
Vector **V** in the viewing direction and a back-face normal vector **N** of a polyhedron



Figure 13-2
A polygon surface with plane parameter $C < 0$ in a right-handed viewing coordinate system is identified as a back face when the viewing direction is along the negative z_v axis.

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, since object depth is usually measured from the view plane along the z axis of a viewing system. 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 method 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 viewplane, object depths can be compared by comparing z values.

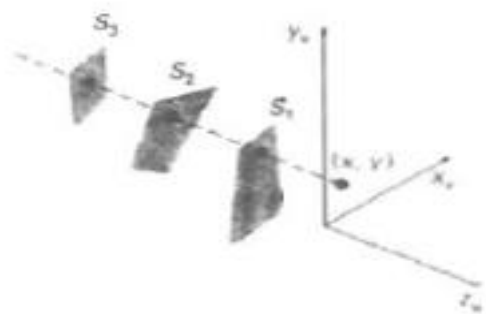


Figure 13-4
At view-plane position (x, y) , surface S_1 has the smallest depth from the view plane and so is visible at that position.

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) ,

$$\text{depth}(x, y) = 0, \quad \text{refresh}(x, y) = I_{\text{background}}$$

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 $z > \text{depth}(x, y)$, then set

$$\text{depth}(x, y) = z, \quad \text{refresh}(x, y) = I_{\text{surf}}(x, y)$$

where $I_{\text{background}}$ is the value for the background intensity, and $I_{\text{surf}}(x, y)$ 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:

$$z = \frac{-Ax - By - D}{C} \quad (13-4)$$

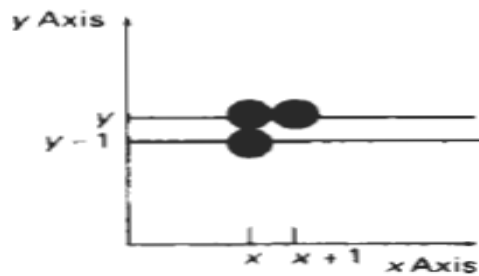


Figure 13-5
From position (x, y) on a scan line, the next position across the line has coordinates $(x + 1, y)$, and the position immediately below on the next line has coordinates $(x, y - 1)$.

For any scan line (Fig. 13-5), adjacent horizontal positions across the line differ by 1, 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. 13-4 as

$$z' = \frac{-A(x + 1) - By - D}{C} \quad (13-5)$$

$$z' = z - \frac{A}{C} \quad (13-6)$$

The ratio $-A/C$ is constant for each surface, so succeeding depth values across a scan line are obtained from preceding values with a single addition. On each scan line, we start by calculating the depth on a left edge of the polygon that intersects that scan line (Fig. 13-6). Depth values at each successive position across the scan line are then calculated by Eq. 13-6. We first determine the y-coordinate extents of each polygon, and process the surface from the topmost scan line to the bottom scan line, as shown in Fig.

13-6. 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 (Fig. 13-7). Depth values down the edge are then obtained recursively as

$$z' = z + \frac{A/m + B}{C} \quad (13-7)$$

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

$$z' = z + \frac{B}{C}$$

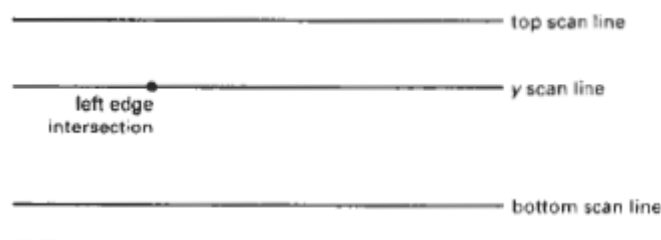


Figure 13-6
Scan lines intersecting a polygon surface.

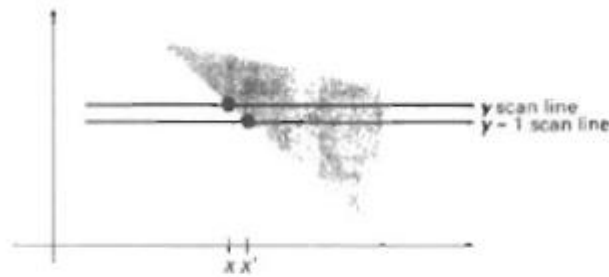


Figure 13-7
Intersection positions on successive scan lines along a left polygon edge.

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.

SCAN-LINE METHOD

This imagespace method for removing hidden surface is an extension of the scan-line algorithm for tiling polygon interiors. Instead of filling just one surface, we now deal with multiple surfaces. As each scan line is processed, all polygons 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, as discussed in Chapter 10, 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.

Figure 13-10 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 **S1** is on.

Therefore, no depth calculations are necessary, and intensity information for surface **S1** 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. The background intensity can be loaded throughout the buffer in an initialization routine.

For scan lines 2 and 3 in Fig. 13-10, 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 **S1** 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 **S1** is assumed to be less

than that of **S2**, so intensities for surface **S1** are loaded into the refresh buffer until

boundary **BC** is encountered. Then the flag for surface **S1** goes off, and intensities

for surface **S2** are stored until edge **FG** is passed.

We can take advantage of coherence along the scan lines as we pass from one scan line to the next. In Fig. 13-10, scan line 3 has the same active list of edges

as scan line 2. Since no changes have occurred in line intersections, it is unnecessary

again to make depth calculations between edges **EH** and **BC**. The two surfaces must be in the same orientation as determined on scan line 2, so the intensities

for surface **S1** can be entered without further calculations.

Any number of overlapping polygon surfaces can be processed with this scan-line method. Flags for the surfaces are set to indicate whether a position is

inside or outside, and depth calculations are performed when surfaces overlap.

When these coherence methods are used, we need to be careful to keep track of which surface section is visible on each scan line. This works only if surfaces do not cut through or otherwise cyclically overlap each other (Fig. 13-11). If any kind of cyclic overlap is present in a scene, we can divide the

surfaces to eliminate the overlaps. The dashed lines in this figure indicate where planes could be subdivided to form two distinct surfaces, so that the cyclic overlaps are eliminated

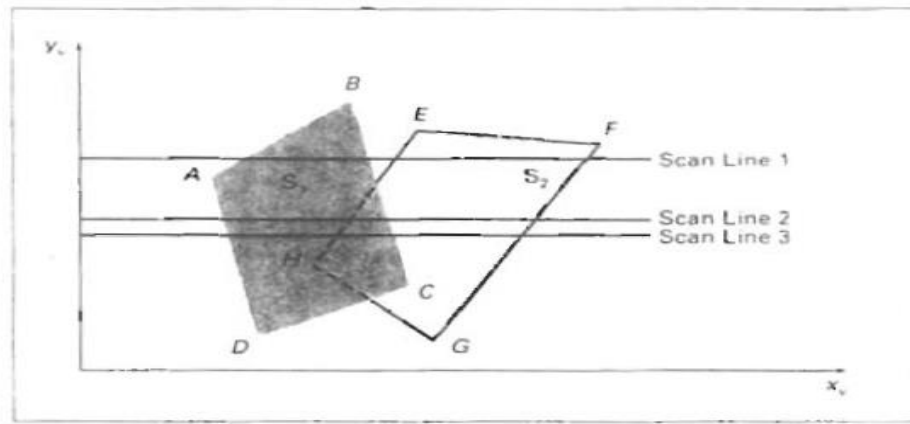


Figure 13-10
Scan lines crossing the projection of two surfaces, S_1 and S_2 , in the view plane. Dashed lines indicate the boundaries of hidden surfaces.

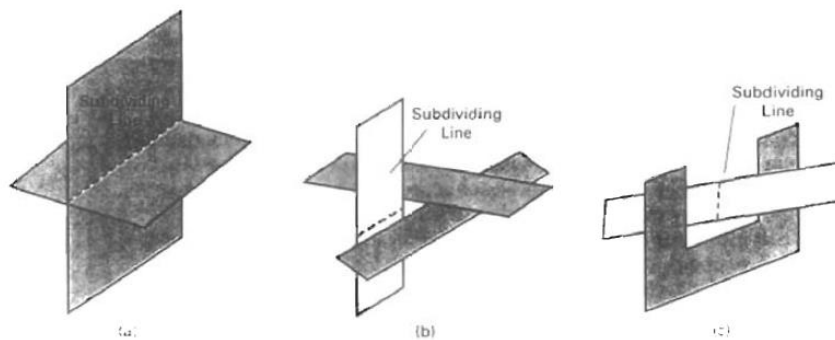


Figure 13-11
Intersecting and cyclically overlapping surfaces that alternately obscure one another.

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. Figure 13-19 illustrates the basic concept in this algorithm. 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. 13-19(b). In this tree, the objects are represented as terminal nodes, with front objects as left branches and back objects as right branches.

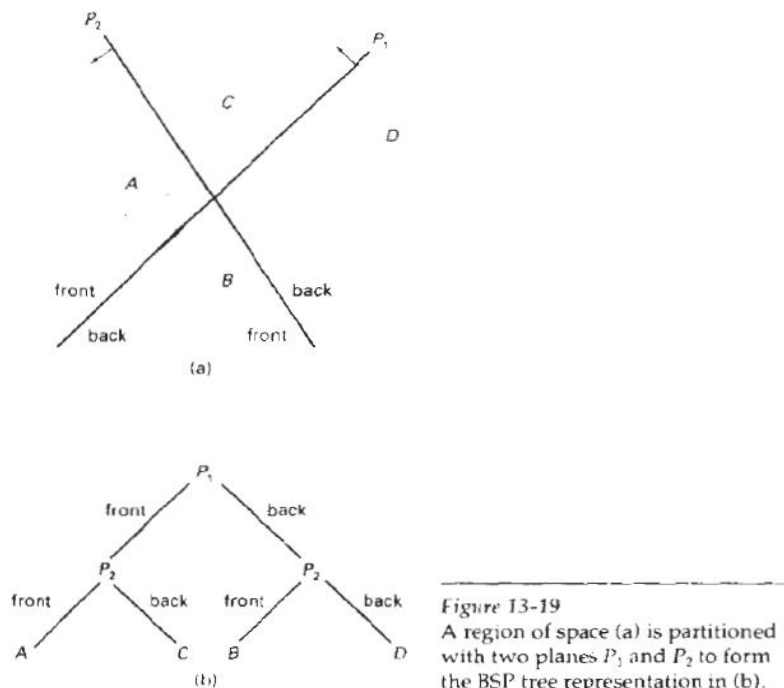


Figure 13-19
A region of space (a) is partitioned with two planes P_1 and P_2 to form the BSP tree representation in (b).

For objects described with polygon facets, we chose the partitioning planes to coincide with the polygon planes. The polygon equations are then **used** to identify "inside" and "outside" polygons, and the tree is constructed with one partitioning plane for each polygon face. Any polygon intersected by a partitioning plane is split into two parts. When the BSP tree is complete, we process the tree by selecting the surfaces for display in the order back to front, so that foreground objects are painted over the background objects. Fast hardware implementations for constructing and processing BSP trees are used in some systems.

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 *n* 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, as shown in Fig. 13-20. This approach is similar to that used in constructing a quadtree.

A viewing area with a resolution of 1024 by 1024 could be subdivided ten times in this way before a subarea is reduced to a point.

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. 13-21):

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.

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 carried 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. For other **types** of surfaces, the bounding rectangles can be **used** as an initial check. If a single bounding rectangle intersects the area in some way, additional checks are used to determine whether the surface is surrounding, overlapping, or outside. Once a single inside, overlapping, or surrounding surface has been identified, its pixel intensities are transferred to the appropriate area within the frame buffer.

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.

Figure 13-22 shows an example of the conditions for this method. 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 the surrounding surface.

For some situations, both methods of implementing test 3 will fail to identify correctly a surrounding surface that obscures all the other surfaces. Further testing could be carried out to identify the single surface that covers the area, but 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.

-

Section 13-8

Area-Subdivision Method

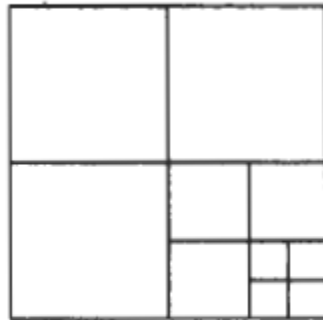


Figure 13-20

Dividing a square area into equal-sized quadrants at each step.

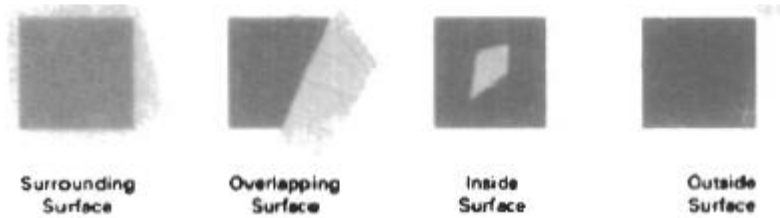


Figure 13-21

Possible relationships between polygon surfaces and a rectangular area.