

MODULE V

Projections – Parallel and perspective projections – vanishing points.

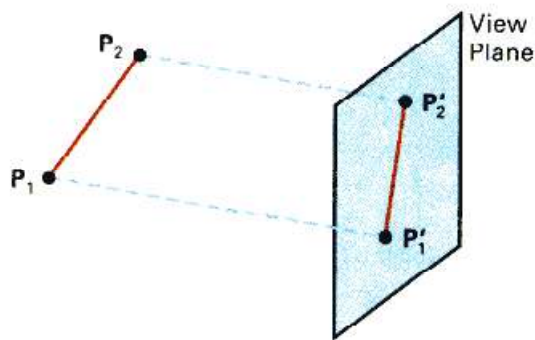
Visible surface detection methods– Back face removal- Z-Buffer algorithm, A-buffer algorithm, Depth-sorting method, Scan line algorithm.

Projections

There are 2 basic projection methods: Parallel Projection and Perspective Projection.

Parallel Projection

Parallel Projection transforms object positions to the view plane along parallel lines. A parallel projection preserves relative proportions of objects. Accurate views of the various sides of an object are obtained with a parallel projection. But this is not a realistic representation.



Parallel projection is classified into two.

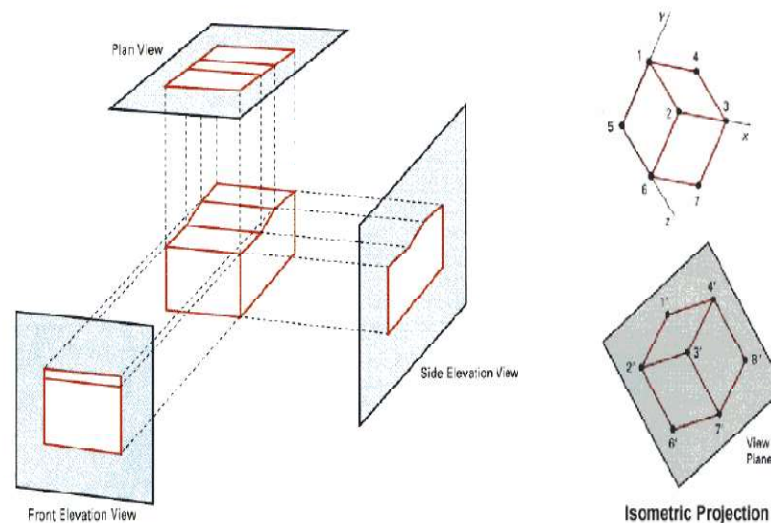
- Orthographic Parallel Projection
- Oblique Projection



Orthographic parallel projections are done by projecting points along parallel lines that are perpendicular to the projection plane.

Oblique projections are obtained by projecting along parallel lines that are NOT perpendicular to the projection plane.

Some special Orthographic Parallel Projections involve Plan View (Top projection), Side Elevations, and Isometric Projection as shown below in the figure:

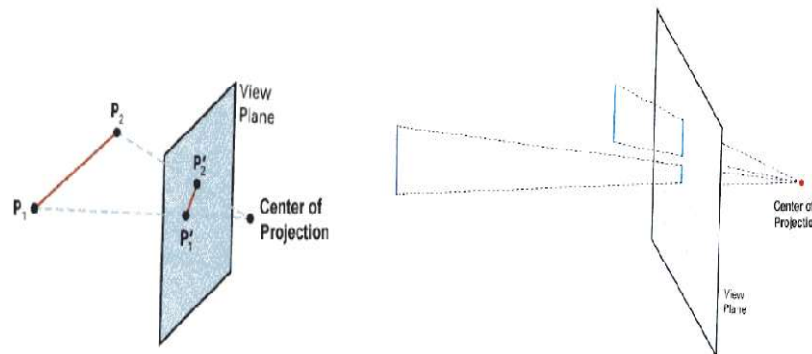


We can also form orthographic projections that display more than one phase of an object. Such views are called axonometric orthographic projections.

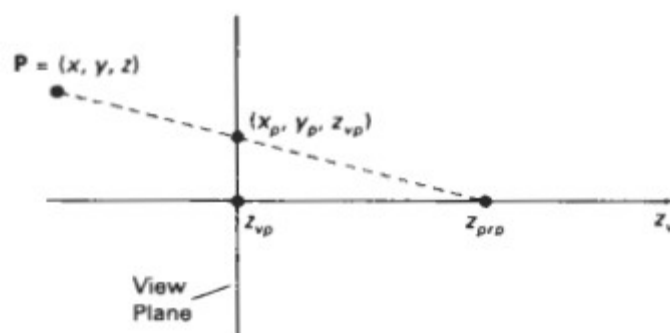
eg: isometric projections → The above figure shows the isometric projection of a cube. It is obtained by aligning the projection vector with the cube diagonal.

Perspective Projection

Perspective Projection transforms object positions to the view plane while converging to a center point of projection. Perspective projection produces realistic views but does not preserve relative proportions. Projections of distant objects are smaller than the projections of objects of the same size that are closer to the projection plane.



To obtain a perspective projection of a three-dimensional object, we transform points along projection lines that meet at the projection reference point. Suppose we set the projection reference point at position z_{prp} along the z_v axis, and we place the view plane at z_{vp} as shown in the following figure.



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

We can write equations describing coordinate positions along this perspective projection line in parametric form as

$$\begin{aligned}x' &= x - xu \\y' &= y - yu \\z' &= z - (z - z_{prp})u\end{aligned}$$

Parameter u takes values from 0 to 1, and coordinate position (x', y', z') represents any point along the projection line. When $u = 0$, we are at position $P = (x, y, z)$. At the other end of the line, $u = 1$ and we have the projection reference point coordinates $(0, 0, z_{prp})$. On the view plane, $z' = z_{vp}$, and we can solve the z' equation for parameter u at this position along the projection line:

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

Substituting this value of u into the equations for x' and y' , we obtain the perspective transformation equations

$$\begin{aligned}x_p &= x \left(\frac{z_{prp} - z_{vp}}{z_{prp} - z} \right) = x \left(\frac{d_p}{z_{prp} - z} \right) \\y_p &= y \left(\frac{z_{prp} - z_{vp}}{z_{prp} - z} \right) = y \left(\frac{d_p}{z_{prp} - z} \right)\end{aligned}$$

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

Using a three-dimensional homogeneous-coordinate representation, we can write the perspective projection transformation shown above in matrix form as:

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

In this representation the homogeneous factor is,

$$h = \frac{z_{pp} - z}{d_p}$$

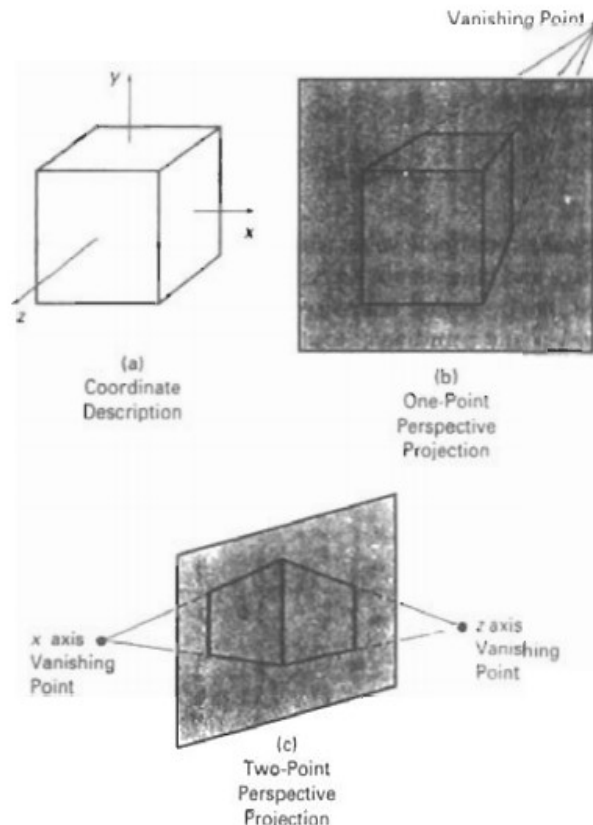
and the projection coordinates on the view plane are calculated from the homogeneous coordinates as

$$x_p = x_h / h$$

$$y_p = y_h / h$$

When a three-dimensional object is projected onto a view plane using perspective transformation equations, any set of parallel lines in the object that are not parallel to the plane are projected into converging lines. Parallel Lines that are parallel to the view plane will be projected as parallel lines. The point at which a set of projected parallel lines appears to converge is called a **vanishing point**. Each such set of projected parallel lines will have a separate vanishing point; and in general, a scene can have any number of vanishing points, depending on how many sets of parallel lines there are in the scene.

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



Visible Surface Detection Algorithms

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.

There are many approaches we can take to solve this problem, and numerous algorithms have been devised for efficient identification of visible objects for different types of applications.

The various algorithms are referred to as visible-surface detection methods. These are also referred as hidden surface elimination methods. Visible surface Detection algorithms are mainly classified into two:

Object Space methods

Image Space methods

An object-space method compares objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible.

In an image-space algorithm, visibility is decided point by point at each pixel position on the projection plane.

Most visible-surface algorithms use image-space methods, although object space methods can be used effectively to locate visible surfaces in some cases.

Although there are major differences in the basic approach taken by the various visible-surface detection algorithms, most use sorting and coherence methods to improve performance.

Sorting is used to facilitate depth comparisons by ordering the individual surfaces in a scene according to their distance from the view plane.

Coherence methods are used to take advantage of regularities in a scene.

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

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, then this polygon is a back face if

$$V \cdot N > 0$$

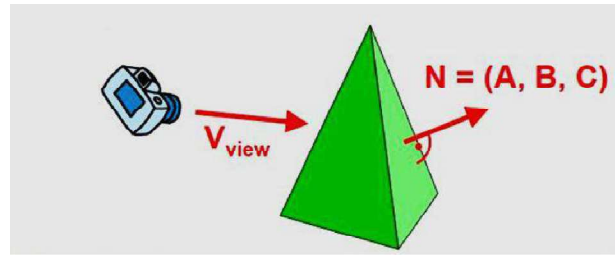
Furthermore, if object descriptions are converted to projection coordinates and your viewing direction is parallel to the viewing z-axis, then:

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

So that we only need to consider the sign of C the component of the normal vector N.

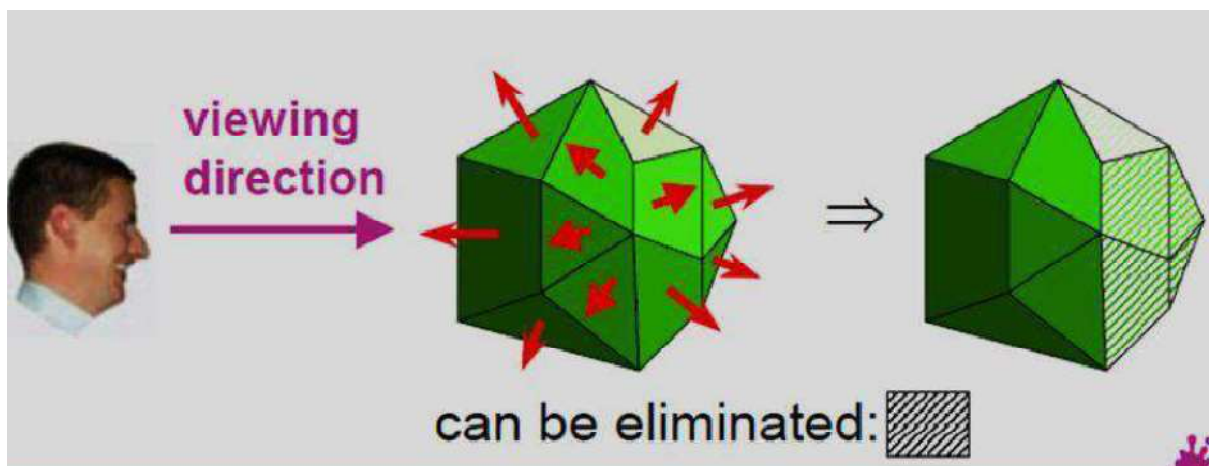
In a right-handed viewing system with viewing direction along the negative ZV axis, the polygon is a back face if $C < 0$. Also, we cannot see any face whose normal has z component $C = 0$, since your viewing direction is towards that polygon. Thus, in general, we can label any polygon as a back face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

$$C \leq 0$$



Similar methods can be used in packages that employ a left-handed viewing system. In these packages, plane parameters A, B, C and D can be calculated from polygon vertex coordinates specified in a clockwise direction (unlike the counter clockwise direction used in a right-handed system).

Also, back faces have normal vectors that point away from the viewing position and are identified by $C \geq 0$ when the viewing direction is along the positive z_v axis. By examining parameter C for the different planes defining an object, we can immediately identify all the back faces.



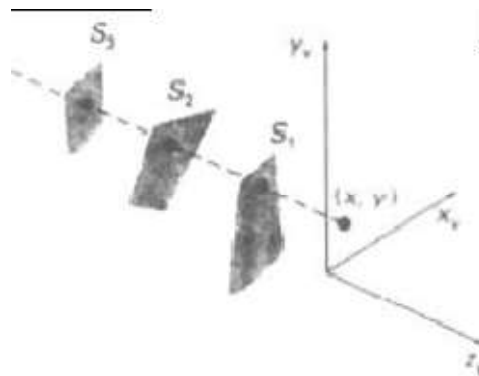
For a single convex polyhedron, this test identifies all the hidden surfaces on the object, since each surface is either completely visible or completely hidden. Also, if a scene contains only non overlapping convex polyhedra, then again all hidden surfaces are identified with the back-face method.

For other objects, such as the concave polyhedron more tests need to be carried out to determine whether there are additional faces that are totally or partly obscured by other faces.

Depth Buffer (Z-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 non planar 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 following figure shows three surfaces at varying distances along the orthographic projection line from position (x, y) in a view plane taken as the x - y plane. Surface S_1 , is closest at this position, so its surface intensity value at (x, y) is saved.



To implement depth buffer method, here we are using two buffer areas. One is **depth buffer** which is used to store depth values for each (x, y) position as surfaces are processed, and the second is **refresh buffer** which stores the intensity values for each position.

Depth Buffer algorithm can be summarized as follows:

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

$$\text{Depth}(x,y)=0, \quad \text{refresh}(x, y) = I_{\text{backgnd}}$$

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

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

where I_{backgnd} 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:

$$Z = (-Ax - By - D)/C$$

For any scan line as in the following figure, adjacent horizontal positions across the line differ by 1, and a vertical y value on an adjacent scan line differs by 1.

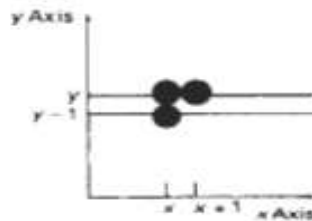


Figure 13-5
From position (x, y) on a scan line, the next position across the line has coordinates $(x + 1, y)$, and the position immediately below on the next line has coordinates $(x, y - 1)$.

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 the above equation as,

$$Z' = (-A(x+1) - By - D)/C$$

Or,

$$Z' = Z - A/C$$

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.

Advantages

- It is easy to implement.
- It reduces the speed problem if implemented in hardware.
- It processes one object at a time.

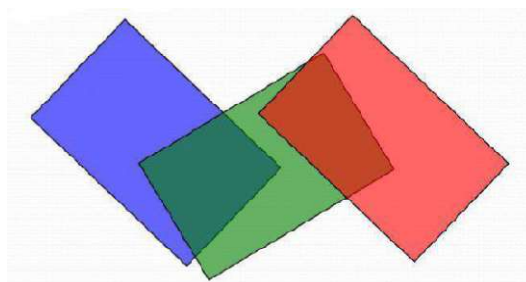
Disadvantages

- It requires large memory.
- It is time consuming process.

A-Buffer Method

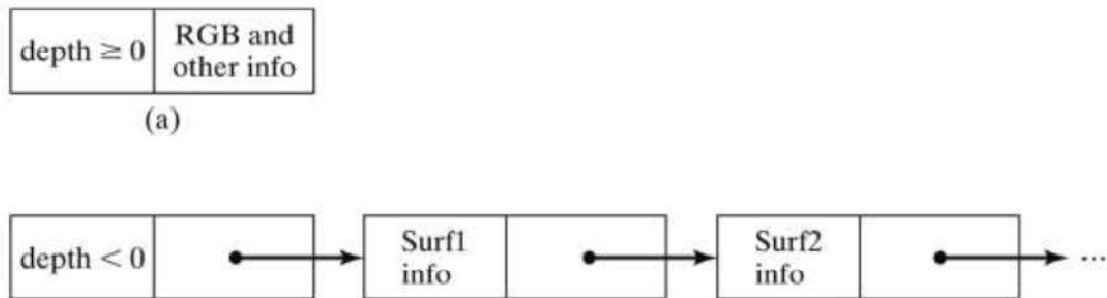
The A-buffer method is an extension of the depth-buffer method. The A-buffer method is a visibility detection method developed at Lucas film Studios for the rendering system Renders Everything You Ever Saw (REYES).

The A-buffer expands on the depth buffer method to allow transparencies. The key data structure in the A-buffer is the accumulation buffer.



Each position in the A-buffer has two fields:

- 1) **Depth field:** It stores a positive or negative real number
- 2) **Intensity field:** It stores surface-intensity information or a pointer value



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

If $\text{depth} < 0$, it indicates multiple-surface contributions to the pixel intensity. The intensity field then stores a pointer to a linked list of surface data. The surface buffer in the A-buffer includes:

- RGB intensity components
- Opacity Parameter
- Depth
- Percent of area coverage
- Surface identifier

The algorithm proceeds just like the depth buffer algorithm. The depth and opacity values are used to determine the final colour of a pixel.

Depth Sorting Method

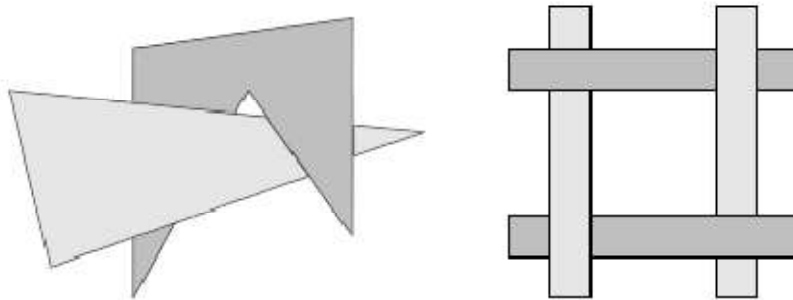
Depth sorting method uses both image space and object-space operations. The depth sorting method performs two basic functions:

- First, the surfaces are sorted in order of decreasing depth.

- Second, the surfaces are scan-converted in order, starting with the surface of greatest depth.

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.

The following figure shows the effect of depth sorting:



The algorithm begins by sorting by depth. For example, the initial “depth” estimate of a polygon may be taken to be the closest z value of any vertex of the polygon.

Let us take the polygon P at the end of the list. Consider all polygons Q whose z -extents overlap P 's. Before drawing P , we make the following tests. If any of the following tests is positive, then we can assume P can be drawn before Q .

1. Do the x -extents not overlap?
2. Do the y -extents not overlap?
3. Is P entirely on the opposite side of Q 's plane from the viewpoint?
4. Is Q entirely on the same side of P 's plane as the viewpoint?
5. Do the projections of the polygons not overlap?

If all the tests fail, then we split either P or Q using the plane of the other. The new cut polygons are inserted into the depth order and the process continues.

Theoretically, this partitioning could generate $O(n^2)$ individual polygons, but in practice, the number of polygons is much smaller.

Scan Line Algorithm

This image space 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 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.

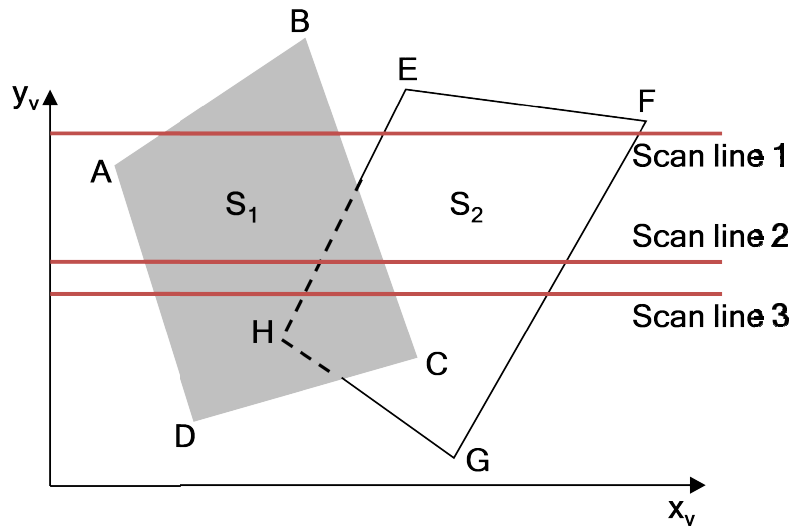
Here two tables are maintained for various surfaces. They are:

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.

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.

Example:



When processing scan line 1, the active list contains the edges AB, BC, EH and FG. Between AB and BC only the flag for surface S_1 is on. So no depth calculation is necessary. Intensity for surface S_1 is entered into the refresh buffer. Similarly between EH and FG only the flag for surface S_2 is on, intensity for surface S_2 will be entered into refresh buffer.

For scan line 2 and 3, the active list contains the edges AD, EH, BC and FG. Between AD and EH, only the flag for S_1 is on. But between EH and BC, the flags for both surfaces are on. In this interval, depth calculations must be made using the plane coefficients for the two surfaces. For this example, the depth of surface S_1 is assumed to be less than that of S_2 , so intensities for surface S_1 are loaded into the refresh buffer until boundary BC is encountered. Then the flag for surface S_1 goes off, and intensities for surface S_2 are stored until edge FG is passed.

We can take advantage of coherence as we pass from one scan line to the next. Here scan line 3 cuts the same edges as scan line 2. So it is not needed to do all the calculations for scan line 3. So without any further calculations we can enter intensity of surface S_1 into the refresh buffer when processing 3rd scan line.