MODULE 5

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.

Vanishing points

- 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 vanishmg point for any set of lines that are parallel to one of the princi-pal axes of an object is referred to as a principal vanishing point.
- The number of principal vanish-ing points in a projection is determined by the number of principal axes intersect-ing the view plane.

Visible surface detection methods

- Visible-surface detection algorithms are broadly classified according to whether they deal with object definitions directly or with their projected images. These two approaches are called object-space methods and image-space methods, respectively.
- An object-space method compare 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

Back face removal

- 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

$$Ax+By+Cz<0$$

- When an inside point is along the line of sight to the surface, the polygon must be a back face. if V is a vector in the viewing direction from the eye (or "camera") position, then this polygon is a back face if

$$V.N>0$$

- In a right-handed viewing system with viewing direction along the negative z axis the polygon is a back face if **C < 0.**
- in general, we can label any polygon as a back face if its normal vector has a z Component value:

$$C<=0$$

Also, back faces have normal vectors that point away from the viewing position and are identified by C >= 0 when the viewing direction is along the positive z, axis.

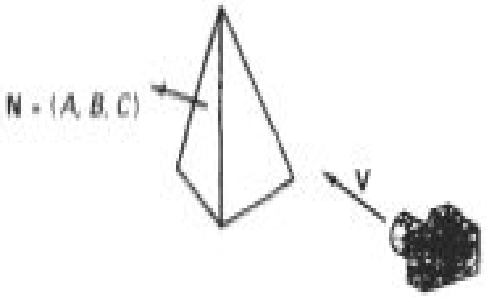

Figure 13-1
Vector V in the viewing direction and a back-face normal vector N of a polyhedron

Z-Buffer algorithm OR 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 mcthod can be applied to nonplanar surfaces.
- 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),

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

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

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

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.

-

- Here 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.
- Each surface listed in the polygon tables is then processed, one scan line at a time, calculating the depth (z value) at each (x, y) pixel position.
- The calculated depth is compared to the value previously stored in the depth buffer at that position.
- If the calculated depth is greater than the value stored in the depth buffer, the new depth value is stored, and the surface intensity at that position is determined and in the same xy location in the refresh buffer.
  Depth values for a surface position (x, y) are calculated from the plane equation for each surface:

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

For any scan line 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:

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

$$z' = z - \frac{A}{C}$$

The ratio -A/C is constant for each surface, so succeeding depth values across a scan line are obtained from precrd~ng values with a single addition.
Depth values down the edge are then obtained recursively as :

$$z' = z + \frac{A/m + B}{C}$$

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

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

A-BUFFER METHOD

- An extension of the ideas in the depth-buffer method is the A-buffer method (at the other end of the alphabet from "z-buffer", where z represents depth).
- The A buffer method represents an antialiased, area-averaged, accumulation buffer method .

- A drawback of the depth-buffer method is that it can only find one visible surface at each pixel position. In other words, it deals only with opaque surfaces and cannot accumulate intensity values for more than one surface, as is necessary if transparent surfaces are to be displayed.
-
- 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:
    - depth field - stores a positive or negative real number
    - intensity field - stores surface-intensity information or a pointer value.

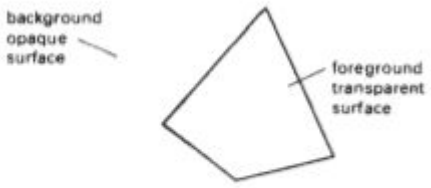

Figure 13-8
Viewing an opaque surface through a transparent surface requires multiple surface-intensity contributions for pixel positions.

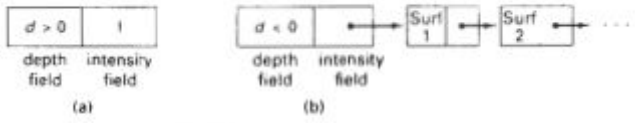

Figure 13-9
Organization of an A-buffer pixel position: (a) single-surface overlap of the corresponding pixel area, and (b) multiple-surface overlap.

- 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.
- If the depth field is negative, this indicates multiple-surface contributions to the pixel intensity. The intensity field then stores a pinter to a linked Iist of surface data,.
- Data for each surface in the linked list includes
    1. RGB intensity components
    2. opacity parameter (percent of transparency)
    3. depth percent of area coverage
    4. surface identifier
    5. other surface-rendering parameters
    6. pointer to next surface

SCAN-LINE METHOD

- This image space method for removing hidden surface5 is an extension of the scan-line algorithm for tilling 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, d~pth calculations are made for each overlapping surface to determine which is nearest to the view plane. When the visible surface has been determined, the mtensity value for that position is entered into the refresh buffer.
- Tables are-set up for the various surfaces, 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.
- 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.

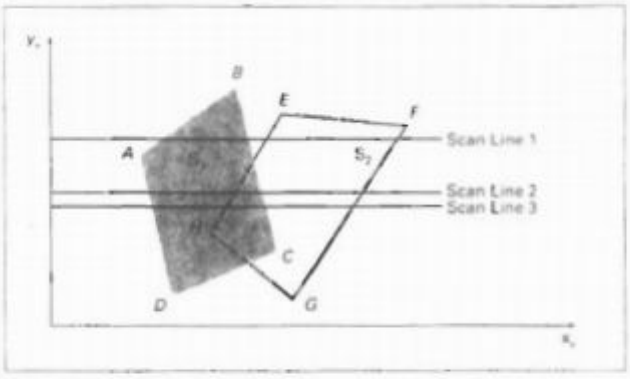

Figure 13-10
Scan lines crossing the projection of two surfaces, S_1 and S_2, in the view plane. Dashed lines indicate the boundaries of hidden surfaces.

- The 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 Sl is on. Therefo~, no depth calculations are necessary, and intensity information for surface S, 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 S, 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 SI is assumed to be less than that of S,, so intensities for surface S, are loaded into the refresh buffer until boundary BC is encountered.

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

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.
- 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 hxrn (in decreasing depth order), 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 wewing along the-z direction, surfaces are ordered on the first pass according to the smallest z value on each surface. Surface 5 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.
- 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. 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.
- We make the following tests for each surface that overlaps with 5. 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 5 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.

- We perform these tests in the order listed and proceed to the next overlapping surface as soon as we find one of the tests is true. If all the overlapping surfaces pass at least one of these tests, none of them is behind 5. No reordering is then necessary and S is scan converted.
- 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.

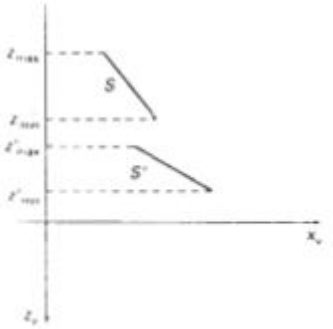

Figure 13-12
Two surfaces with no depth overlap.

- 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 set up 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' .
- . Similarly, S' is completely in front of S if all vertices of S are "outside" of 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.
- Should all four tests fail with a particular overlapping surface S', we interchange surfaces S and S' in the sorted list.

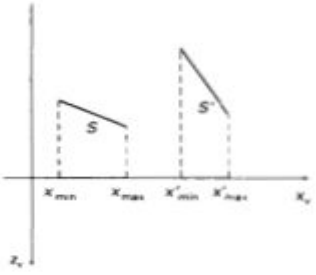

Figure 13-13
Two surfaces with depth overlap but no overlap in the *x* direction.

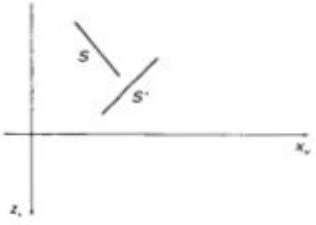

Figure 13-14
Surface S is completely behind ("inside") the overlapping surface S'.

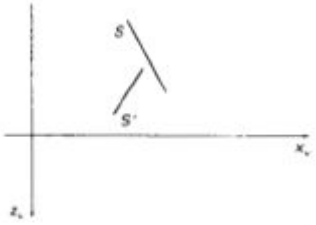

Figure 13-15
Overlapping surface S' is completely in front ("outside") of surface S, but S is not completely behind S'.