UNIT IV: Three-Dimensional viewing – viewing pipeline - Display Techniques – Parallel Projection – Perspective Projection – Hidden-Surface and Hidden-Line removal – Back face removal – Depth Buffer Method – Scan Line Method – BSP Tree Methods – Depth-Sorting Method – Area-subdivision Method – Octree Methods – Comparison of Hidden-Surface Methods.

TEXT BOOK

1. Donald Hearn and Pauline Baker, "Computer Graphics", Prentice Hall of India, 2001.

Prepared by **Dr.P.Sumathi**

Three-Dimensional Viewing

Viewing in 3D involves the following considerations:

- We can view an object from any spatial position, Eg., In front of an object, Behind the object, In the middle of a group of objects, Inside an object, etc.
- 3D descriptions of objects must be projected onto the flat viewing surface of the output device.
- The clipping boundaries enclose a volume of space

Viewing Pipeline

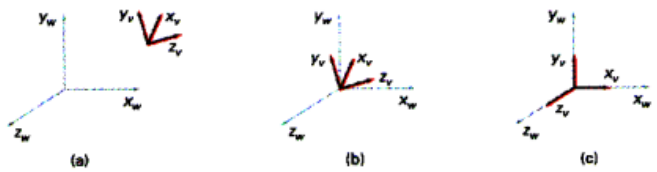
Once the scene has been modeled, world-coordinate positions are converted to viewing coordinates. The viewing-coordinate system is used in graphics packages as a reference for specifying the observer viewing position and the position of the projection plane, which we can think of in analogy with the camera film plane.

Next, 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. Objects outside the specified viewing limits are clipped h m further consideration, and the remaining objects are processed through visible-surface identification and surface-rendering procedures to produce the display within the device viewport.

Viewing Transformation

Conversion of objection descriptions from world to viewing coordinates is equivalent to a transformation that superimposes the viewing reference frame onto the world frame using the basic geometric translate-rotate operations:

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 (viewing coordinate system) with the world x_w, y_w, z_w axes, respectively.



Projections

Once world-coordinate descriptions of the objects in a scene are converted to viewing coordinates, we can project the three-dimensional objects onto the two-dimensional view plane. So **Projection in computer graphics** means the transformation of a three-dimensional (3D) area into a two-dimensional (2D) area.

(or)

It is the process of converting a 3D object into a 2D object. It is also defined as mapping or transformation of the object in projection plane or view plane. The view plane is displayed surface.

There are two basic projection methods, **Parallel projection and Perspective projection.**

In a **parallel projection**, coordinate positions are transformed to the view plane along parallel lines, as shown in the example of Fig.1.

A parallel projection preserves relative proportions of objects, and this is the method used in drafting to produce scale drawings of three-dimensional objects. Accurate views of the various sides of an object are obtained with a parallel projection, but this does not give us a realistic representation of the appearance of a three-dimensional object.

For a **perspective projection** (Fig.2), object positions are transformed to the view plane along lines that converge to a point called the **projection reference point** (or **Center of projection**). The projected view of an object is determined by calculating the intersection of the projection lines with the view plane.

A perspective projection, on the other hand, 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.

Important terms related to perspective

1. **View plane:** It is an area of world coordinate system which is projected into viewing plane.
2. **Center of Projection:** It is the location of the eye on which projected light rays converge.
3. **Projectors:** It is also called a projection vector. These are rays start from the object scene and are used to create an image of the object on viewing or view plane.

Orthographic projection:

In orthographic projection the direction of projection is normal to the projection of the plane.
There are three types of orthographic projections −

- Front Projection
- Top Projection
- Side Projection

Oblique projection

In this projection, the direction of projection is not normal to the projection of plane. In oblique projection, we can view the object better than orthographic projection.
There are two types of oblique projections − **Cavalier** and **Cabinet**.

The **Cavalier projection** makes 45° angle with the projection plane. The projection of a line perpendicular to the view plane has the same length as the line itself in Cavalier projection. In a cavalier projection, the foreshortening factors for all three principal directions are equal.

The **Cabinet projection** makes 63.4° angle with the projection plane. In Cabinet projection, lines perpendicular to the viewing surface are projected at ½ their actual length. Both the projections are shown in the following figure.

Isometric Projections

Orthographic projections that show more than one side of an object are called **axonometric orthographic projections**. The most common axonometric projection is an **isometric projection** where the projection plane intersects each coordinate axis in the model coordinate system at an equal distance. In this projection parallelism of lines are preserved but angles are not preserved. The following figure shows isometric projection −

Perspective Projection

In perspective projection, the distance from the center of projection to project plane is finite and the size of the object varies inversely with distance which looks more realistic.
The distance and angles are not preserved and parallel lines do not remain parallel. Instead, they all converge at a single point called **center of projection** or **projection reference point**. There are 3 types of perspective projections which are shown in the following chart.

- **One point** perspective projection is simple to draw.
- **Two point** perspective projection gives better impression of depth.
- **Three point** perspective projection is most difficult to draw.

Hidden-Surface and Hidden-Line Removal

When we view a picture containing non-transparent objects and surfaces, then we cannot see those objects from view which are behind from objects closer to eye. We must remove these hidden surfaces to get a realistic screen image. The identification and removal of these surfaces is called **hidden-surface elimination methods**. These methods are also called as **visible surface determination**.
There are two approaches for removing hidden surface,

1. Object space methods
2. Image space methods

Object space methods:

Object-space method is implemented in the physical coordinate system in which objects are described. It 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. Object-space methods are generally used in line-display algorithms. So, these algorithms are line based instead of surface based.

Object space methods: In this method, various parts of objects are compared. After comparison, visible, invisible or hardly visible surfaces are determined. These methods generally decide visible surface. In the wireframe model, these are used to determine a visible line. So, these algorithms are line based instead of surface based. Method proceeds by determination of parts of an object whose view is obstructed by other object and draws these parts in the same color.

Image space methods: Image space method is implemented in the screen coordinate system in which the objects are viewed. In an image-space algorithm, visibility is decided point by point at each pixel Position on the view plane. Most hidden line/surface algorithms use image-space methods.
Here positions of various pixels are determined. It is used to locate the visible surface instead of a visible line. Each point is detected for its visibility. If a point is visible, then the pixel is on, otherwise off.

1) 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

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

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, then this polygon is a back-face if

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,

$$C <= 0$$

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 counterclockwise 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 >= 0 when the viewing direction is along the positive *Zv* axis. By examining parameter C for the different planes defining an object, we can immediately identify all the back faces.

However, this is still useful as a pre-processing step as almost 50% of the surfaces are eliminated.

---

2) 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 nonplanar surfaces.

We can implement the depth-buffer algorithm in normalized coordinates, so that ***z*** values range from 0 at the back clipping plane to z_{max} at the front clipping plane. The value of z, can be set either to 1 (for a unit cube) or to the largest value that can be stored on the system.

As implied by the name of this method, 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 or Frame buffer** stores the intensity values for each position.

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

Algorithm

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

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

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

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

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

---

3) Scan-Line Method

It is an image-space method to identify visible surface. This method has a depth information for only single scan-line. In order to require one scan-line of depth values, we must group and process all polygons intersecting a given scan-line at the same time before processing the next scan-line. Two important tables, **Edge table** and **Polygon table,** are maintained for this.

The Edge Table - It contains coordinate endpoints of each line in the scene, the inverse slope of each line, and pointers into the polygon table to connect edges to surfaces.

The Polygon Table - It contains the plane coefficients, surface material properties, other surface data, and may be pointers to the edge table.

Therefore, no depth calculations are necessary, and intensity information for surface S_1 is entered from the polygon table into the refresh buffer. Similarly, between edges EH and FG, only the flag for surface S_2 is on. No other positions along scan line 1 intersect surfaces, so the intensity

values in the other areas are set to the background intensity. The background intensity can be loaded throughout the buffer in an initialization routine.

For scan lines 2 and 3 in figure, 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 S_1 is assumed to be less than that of S_2 so intensities for surface S_1, are loaded into the refresh buffer until boundary BC is encountered. Then the flag for surface S_1 goes off, and intensities for surface S_2 are stored until edge FG is passed.

We can take advantage of-coherence along the scan lines as we pass from one scan line to the next. In this figure, 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 sur faces must be in the same orientation as determined on scan line 2, so the intensities for surface S, 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.

4) Binary Space Partition (BSP) Trees

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 sub diversion, relative to the viewing direction. Figure 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.

For objects described with polygon facets, we chose the partitioning planes to coincide with tne 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 DSP trees are used in some systems.

5) Depth-Sorting Method (Painter's algorithm)

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.

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 (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 viewing 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.

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 completely 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 atleast one of these tests, none of them is behind S. 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. An example of two surfaces that overlap in the z direction but not in the x direction is shown in figure 2.

6) Area-Subdivision Method (Warnock's Algorithm)

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, as shown in above figure.

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 and is given in the following figure.

- 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.

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.

7) Octree Methods

Octree is a tree data structure in which each internal node can have at most 8 children. Like Binary tree which divides the space two segments, Octree divides the space into at most eight-part which is called as octanes. It is used to store the 3-D point which takes a large amount of space. if all the internal node of the Octree contains exactly 8 children is called full Octree. It is also useful for high-resolution graphics like 3D computer graphics.

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 figure, 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 rear of the front octants or in the back octants (4,5,6, and 7) may be hidden by the front surfaces.

Back surfaces are eliminated, for the viewing direction given in the above figure, by 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 sub octants of octant 0 are visited before the nodes for the four back sub octants. 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. We assume that the octree representation is always set up so that octants 0,1,2, and 3 of a region form the front face.

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. Above figure depicts the octants in a region of space and the corresponding quadrants on the view plane. Contributions to quadrant 0 come from octants 0 and 4. Color values in quadrant 1 are obtained from surfaces in octants 1 and 5, and values in each of the other two quadrants are generated from the pair of octants aligned with each of these quadrants.

Recursive processing of octree nodes is applied, which accepts an octree description and creates the quadtree representation for visible surfaces in the region. 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.

For heterogeneous regions, the procedure is recursively called, passing as new arguments the child of the heterogeneous octant and a newly created quadtree node. If the front is empty, the rear octant is processed. Otherwise, two recursive calls are made, one for the rear octant and one for the front octant.
