9.1 Introduction

- In a given set of 3D objects and viewing specification, we wish to determine which lines or surfaces of the objects are visible, so that we can display only the visible lines or surfaces. This process is known as **hidden surfaces** or **hidden line elimination** or **visible surface determination.**
- The hidden line or hidden surface algorithm determines the lines, edges, surfaces or volumes that are visible or invisible to an observer located at a specific point in space.
- These 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.

Object-Space Method

- Object-space method is implemented in the **physical co-ordinate 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**.

Image-Space Method

- Image space method is implemented in the **screen co-ordinate 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.

Review Questions

1. *Why are hidden surface algorithms needed ?*
2. *Explain, how hidden lines and surfaces are removed ?*

9.2 Backface Removal Algorithm

AU : Dec.-15

- We know that a polygon has two surfaces, a front and a back, just as a piece of paper does. We might picture our polygons with one side painted light and the other painted dark. But the question is "how to find which surface is light or dark ?"

- When we are looking at the light surface, the polygon will appear to be drawn with counter clockwise pen motions and when we are looking at the dark surface the polygon will appear to be drawn with clockwise pen motions, as shown in the Fig. 9.2.1.

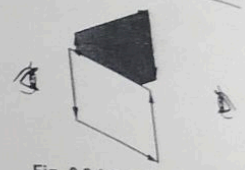


Fig. 9.2.1 Drawing directions

- Let us assume that all solid objects are to be constructed out of polygons in such a way that only the light surfaces are open to the air; the dark faces meet the material inside the object. This means that when we look at an object face from the outside, it will appear to be drawn counterclockwise, as shown in the Fig. 9.2.2.

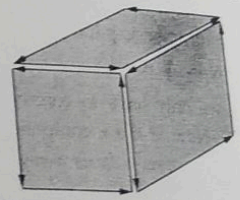


Fig. 9.2.2 Exterior surfaces are coloured light and drawn counter clockwise

- If a polygon is visible, the light surface should face towards us and the dark surface should face away from us. Therefore, if the direction of the light face is pointing towards the viewer, the face is visible (a front face), otherwise, the face is hidden (a back face) and should be removed.

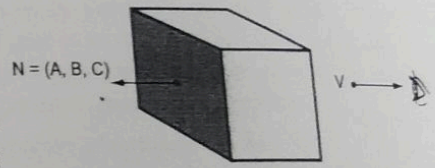


Fig. 9.2.3

- The direction of the light face can be identified by examining the result

$$N \cdot V > 0$$

where

N : Normal vector to the polygon surface with Cartesian components (A, B, C).

V : A vector in the viewing direction from the eye (or "camera") position (Refer Fig. 9.2.3)

- We know that, the dot product of two vectors, gives the product of the lengths of the two vectors times the cosine of the angle between them. This cosine factor is important to us because if the vectors are in the same direction ($0 \leq \theta < \pi/2$), then the cosine is positive and the overall dot product is positive. However, if the directions are opposite ($\pi / 2 < \theta \leq \pi$), then the cosine and the overall dot product is negative (Refer Fig. 9.2.4 on next page).

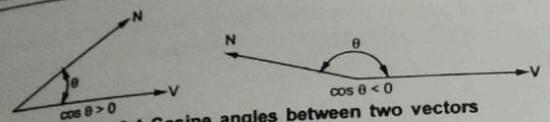


Fig. 9.2.4 Cosine angles between two vectors

- If the dot product is positive, we can say that the polygon faces towards the viewer; otherwise it faces away and should be removed.
- In case, if object description has been converted to projection co-ordinates and our viewing direction is parallel to the viewing z_v axis, then $V = (0, 0, V_z)$ and

$$V \cdot N = V_z C$$

- So that we only have to consider the sign of C, the z component of the normal vector N. Now, if the z component is positive, then the polygon faces towards the viewer, if negative, it faces away.

Review Questions

1. *Explain backface removal algorithm.*
2. *Write down the Backface detection algorithm.* **AU : Dec.-15, Marks 8**

9.3 Painter's Algorithm

- The basic idea of the painter's algorithm developed by Newell and Sancha, is to paint the polygons into the frame buffer in order of decreasing distance from the viewpoint. This process involves following basic functions.

1. Sorting of polygons in order of decreasing depth.
2. Resolving any ambiguities. This may cause when the polygon's z extents overlap, i.e., splitting polygons if necessary.
3. Scan conversion of polygons in order, starting with the polygon of greatest depth.

- The algorithm gets its name from the manner in which an oil painting is created. The artist begins with the background. He then adds the most distant object and then the nearer object and so forth. There is no need to erase portions of background; the artist simply paints on top of them. The new paint covers the old so that only the newest layer of paint is visible. This is illustrated in Fig. 9.3.1.

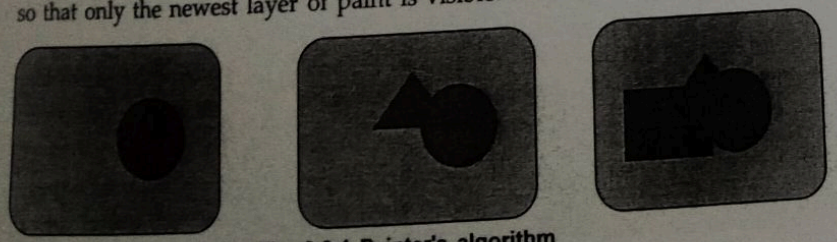


Fig. 9.3.1 Painter's algorithm

- Using the similar technique, we first sort the polygons according to their distance from the view point. The intensity values for the farthest polygon are then entered into the frame buffer. Taking each polygon is succeeding polygon in turn (in decreasing depth order), polygon intensities are painted on the frame buffer over the intensities of the previously processed polygons. This process is continued as long as no overlaps occur.
- If depth overlap is detected by any point in the sorted list, we have to make some additional comparisons to determine whether any of the polygon should be reordered.
- We can check whether any polygon Q does not obscure polygon P by performing following steps :

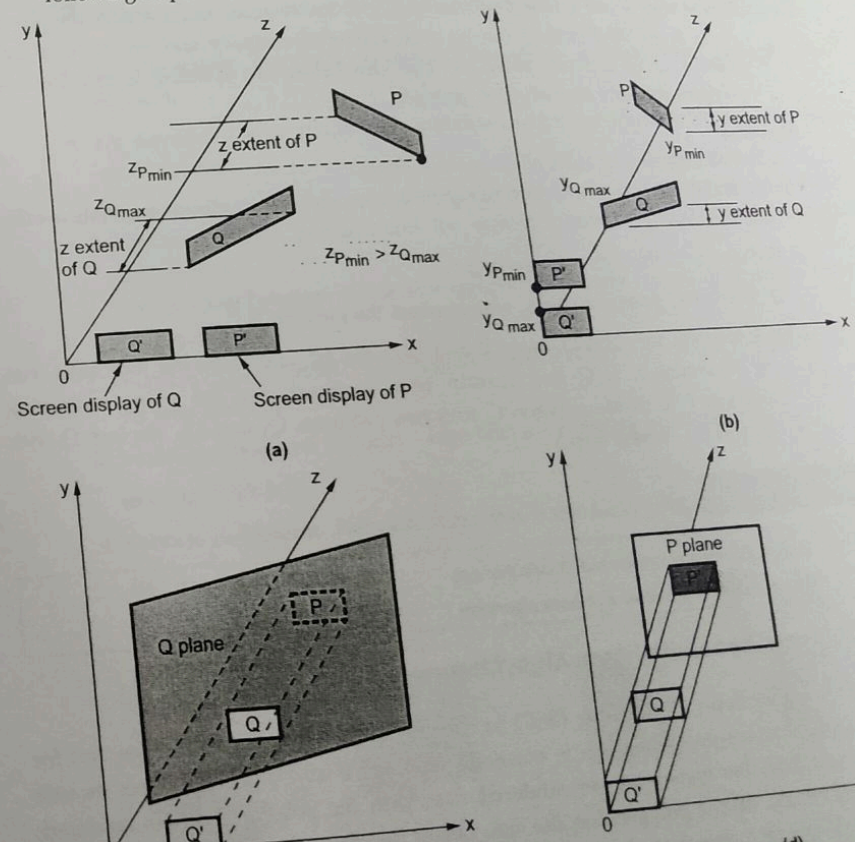


Fig. 9.3.2

1. The z-extents of P and Q do not overlap, i.e. $z_{Q\,max} < z_{P\,min}$. (See Fig. 9.3.2 (a) on previous page)
2. The y-extents of P and Q do not overlap. (See Fig. 9.3.2 (b) on previous page)
3. The x-extents of P and Q do not overlap.
4. Polygon P lying entirely on the opposite side of Q's plane from the view port. (See Fig. 9.3.2 (c) on previous page).
5. Polygon Q lying entirely on the same side of P's plane as the view port. (See Fig. 9.3.2 (d) on previous page).
6. The projections of the polygons P and Q onto the xy screen do not overlap.

- If all these five tests fail, we assume for the moment that P actually obscures Q, and therefore test whether Q can be scan-converted before P. Here, we have to repeat tests 4 and 5 for Q. If these tests also fail then we can say that there is no order in which P and Q can be scan converted correctly and we have to split either P or Q into two polygons. The idea behind the splitting is that the split polygons may not obscure other polygon.

Algorithm

1. Sort all polygons in order of decreasing depth.
2. Determine all polygons Q (preceding P) in the polygon list whose z-extents overlap that of P.
3. Perform test 2 through 6 for each Q
   a) If every Q passes the tests, scan convert the polygon P.
   b) If test fails for some Q, swap P and Q in the list, and make the indication that Q is swapped. If Q has already been swapped, use the plane containing polygon P to divide polygon Q into two polygons, Q_1 and Q_2. Replace Q with Q_1 and Q_2. Repeat step 3

Review Questions

1. *Explain Painter's algorithm and its applications.*
2. *Why is Painter's algorithm a priority algorithm ?*

9.4 Binary Space-Partition Algorithm

- The Binary Space Partitioning (BSP) tree visible surface algorithm assumes that for a given viewpoint a polygon is correctly rendered if all the polygons on its side away from the viewpoint are rendered first; then the polygon itself is rendered; and finally, all the polygons on the side nearer the viewpoint are rendered.
- It is a two-part algorithm, in which, a scene is subdivided into two sections at each step with a plane that can be at any position and orientation.

- The BSP tree algorithm uses one of the polygons in the scene as the separating or dividing plane. Other polygons in the scene that are entirely on one side of the separating plane are placed in the appropriate half space.
- Polygons that intersect the separating plane are split along the separating plane, and each portion is placed in the appropriate half space.
- Each half space is then recursively subdivided using one of the polygons in the half space as the separating plane. This subdivision continues until there is only a single polygon in each half space.
- The subdivided space is conveniently represented by a binary tree. This is illustrated in Fig. 9.4.1. Here, for simplicity, each of the polygons and the separating plane are assumed perpendicular to paper.

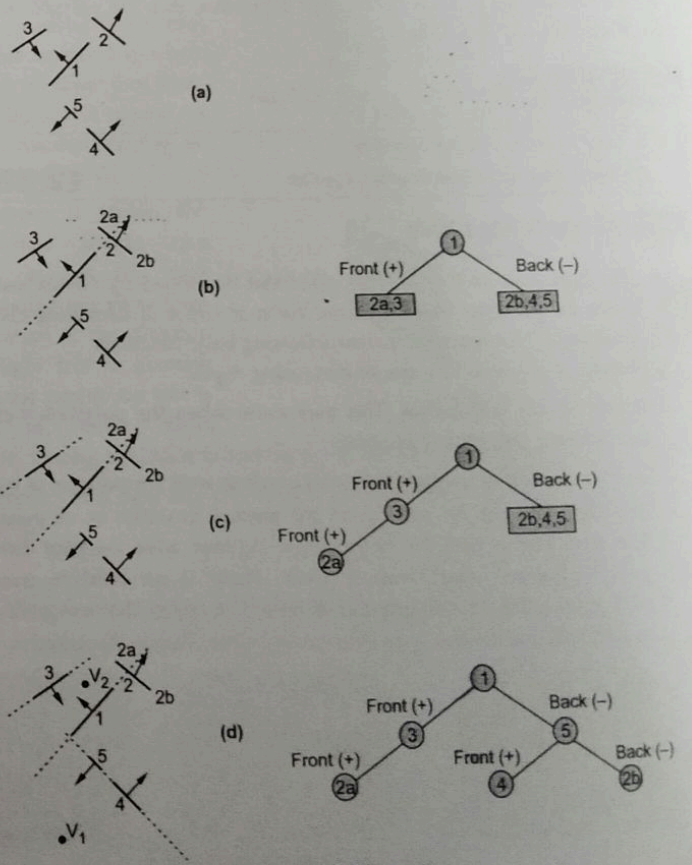


Fig. 9.4.1 Binary tree

Let us see the pseudocode for building a BSP tree.

```
Struct
{       polygon root ;
        BSP_tree * backChild, * frontChild ;
        BSP_tree ;
}
BSP_tree * BSP_makeTree (polygon * polyList)
{       Polygon root ;
        Polygon * backList, * frontList ;
        Polygon P, backPart, FrontPart ;
        if (PolyList = = NULL) Then
                BSP_tree = NULL ;
        else
    {   root = selectpolygon (&polyList) ;
        backList = NULL ;
        frontList = NULL ;
        for (each remaining polygon P in polyList)
    {   if (Polygon P in front of root) then
                AddtoBSPList (P, &frontList) ;
        elseif (polygon P in back of root) then
                AddtoBSPList (P, &backList) ;
        else
                {       Sliptpolygon ( P, root, &frontPart, &backPart) ;
                        AddtoBSPList (frontPart, &frontList) ;
                        AddtoBSPList (backPart, &backList) ;
                }
        endif
    }
    return combineBSP_tree (BSP_makeTree (frontList),
                                        root,
                                        BSP_makeTree (backList) ;
}
}   / * BSP_makeTree */
```

Let us see the pseudocode for displaying a BSPtree

```
Void DisplayBSP_Tree (BSP_tree * tree)
{
        if (tree ! = NULL) then
        {       if (viewer is in front of tree → root) then
                {       /* Display back child, root, and front child. */
                        DisplayBSP_Tree (tree → backChild) ;
                        DisplayPolygon (tree → root) ;
                        DisplayBSP_Tree (tree → frontChild) ;
                }
                else
                {       /* Display front child, root, and back child. * /
                        DisplayBSP_Tree (tree → frontChild) ;
```



```
                        DisplayPolygon (tree → root) ;
                        DisplayBSP_Tree (tree → backChild) ;
                }
                endif
        }
        endif
}       /* DisplayBSP_Tree */
```

Review Questions

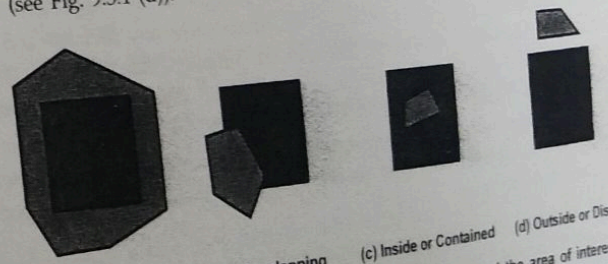
1. *Explain binary space partition algorithm for hidden surfaces.*
2. *Explain any two algorithms used for removing hidden surfaces.*

9.5 Warnock's (Area Subdivision) Algorithm

AU : Dec.-13

- An interesting approach to the hidden-surface problem was developed by Warnock.
- He developed area subdivision algorithm which subdivides each area into four equal squares.
- At each stage in the recursive-subdivision process, the relationship between projection of each polygon and the area of interest is checked for four possible relationships :

1. Surrounding Polygon - One that completely encloses the (shaded) area of interest (see Fig. 9.5.1 (a))
2. Overlapping or Intersecting Polygon - One that is partly inside and partly outside the area (see Fig. 9.5.1 (b))
3. Inside or Contained Polygon - One that is completely inside the area (see Fig. 9.5.1 (c)).
4. Outside or Disjoint Polygon - One that is completely outside the area (see Fig. 9.5.1 (d)).



(a) Surrounding (b) Overlapping (c) Inside or Contained (d) Outside or Disjoint

Fig. 9.5.1 Possible relationships with polygon surfaces and the area of interest

After checking four relationships we can handle each relationship as follows :

1. If all the polygons are disjoint from the area, then the background colour is displayed in the area.
2. If there is only one intersecting or only one contained polygon, then the area is first filled with the background colour and then the part of the polygon contained in the area is filled with colour of polygon.
3. If there is a single surrounding polygon, but no intersecting or contained polygons, then the area is filled with the colour of the surrounding polygon.
4. If there are more than one polygon intersecting, contained in, or surrounding the area then we have to do some more processing.

- See Fig. 9.5.2. In Fig. 9.5.2 (a), the four intersections of surrounding polygon are all closer to the viewpoint than any of the other intersections. Therefore, the entire area is filled with the colour of the surrounding polygon.

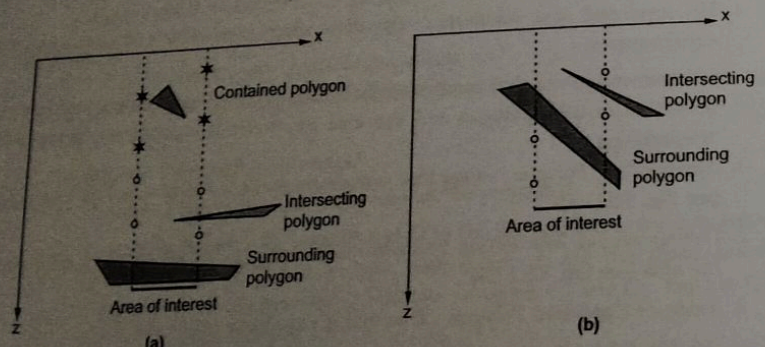


Fig. 9.5.2

- However, Fig. 9.5.2 (b) shows that surrounding polygon is not completely in front of the intersecting polygon. In such case, we cannot make any decision and hence Warnock's algorithm subdivides the area to simplify the problem. This is illustrated in Fig. 9.5.3. As shown in the Fig. 9.5.3 (a) we cannot make any decision about which polygon is in front of the other. But after dividing area of interest polygon 1 is ahead of the polygon 2 in left area and polygon 2 is ahead of polygon 1 in the right area. Now we can fill these two areas with corresponding colours of the polygons.
- The Warnock's algorithm stops subdivision of area only when the problem is simplified or when area is only a single pixel.

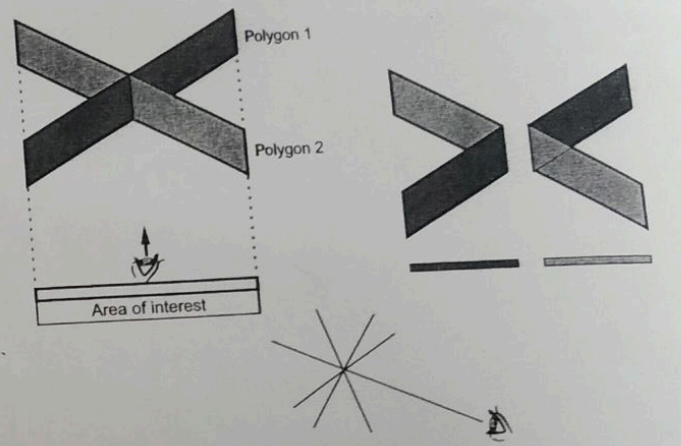


Fig. 9.5.3

Algorithm

1. Initialize the area to be the whole screen.
2. Create the list of polygons by sorting them with their z-values of vertices. Do not include disjoint polygons in the list because they are not visible.
3. Find the relationship of each polygon.
4. Perform the visibility decision test.
   a) If all the polygons are disjoint from the area, then fill area with background colour.
   b) If there is only one intersecting or only one contained polygon then first fill entire area with background colour and then fill the part of the polygon contained in the area with the colour of polygon.
   c) If there is a single surrounding polygon, but no intersecting or contained polygons, then fill the area with the colour of the surrounding polygon.
   d) If surrounding polygon is closer to the viewpoint than all other polygons, so that all other polygons are hidden by it, fill the area with the colour of the surrounding polygon.

e) If the area is the pixel [illegible] z co-ordinate at pixel [illegible] colour of the polygon [illegible]

5. If none of the above tests are [illegible]

Advantages

1. It follows the divide-and- [illegible] used to speed up the proc
2. Extra memory buffer is [illegible]

Review Questions

1. Explain Warnock's alg
2. Why this algorithm [illegible]
3. Discuss on area sub

9.6 Z-buffer Alg

- One of the hidden surfa Catmull.
- This algori plane.
- The surfa system.
- When o positio the de z-valu
- The that plan
- Th th
- A

e) If the **area is the pixel** (x, y), and neither a, b, c, nor d applies, compute the z co-ordinate at pixel (x, y) of all polygons in the list. The pixel is then set to colour of the polygon which is closer to the viewpoint.

5. If none of the above tests are true then subdivide the area and go to step 2.

Advantages

1. It follows the divide-and-conquer strategy, therefore, parallel computers can be used to speed up the process.
2. Extra memory buffer is not required.

Review Questions

1. *Explain Warnock's algorithm.*
2. *Why this algorithm is also called as area subdivision algorithm ?*
3. *Discuss on area subdivision method of hidden surface identification algorithm.*

AU : Dec.-13, Marks 8

9.6 Z-buffer Algorithm

- One of the simplest and commonly used image space approach to eliminate hidden surfaces is the **Z-buffer** or **depth buffer** algorithm. It is developed by Catmull.
- This algorithm **compares surface depths** at each pixel position on the projection plane.
- The surface depth is measured from the view plane along the z axis of a viewing system.
- When object description is converted to projection co-ordinates (x, y, z), each pixel position on the view plane is specified by x and y co-ordinate, and z value gives the depth information. Thus object depths can be compared by comparing the z-values.
- The Z-buffer algorithm is usually implemented in the normalized co-ordinates, so that z values range from 0 at the back clipping plane to 1 at the front clipping plane.
- The implementation requires another buffer memory called **Z-buffer** along with the frame buffer memory required for raster display devices.
- A Z-buffer is used to store depth values for each (x, y) position as surfaces are processed and the frame buffer stores the intensity values for each position.

- At the beginning Z-buffer is initialized to zero, representing the z-value at the back clipping plane and the frame buffer is initialized to the background colour. Each surface listed in the display file is then processed, one scan line at a time, calculating the depth (z-value) at each (x, y) pixel position. The calculated depth value is compared to the value previously stored in the Z-buffer at that position.
- If the calculated depth values is greater than the value stored in the Z-buffer, the new depth value is stored and the surface intensity at that position is determined and placed in the same xy location in the frame buffer.
- For example, in Fig. 9.6.1 among three surfaces, surface S_1 has the smallest depth at view position (x, y) and hence highest z value. So it is visible at that position.

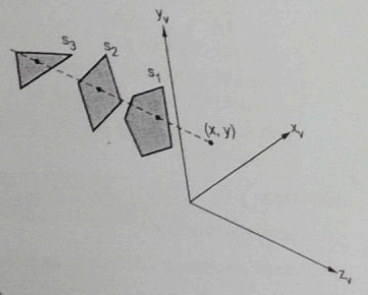


Fig. 9.6.1

Z-buffer Algorithm

1. Initialize the Z-buffer and frame buffer so that for all buffer positions

   Z-buffer (x, y) = 0 and frame-buffer (x, y) = $I_{background}$

2. During scan conversion process, for each position on each polygon surface, compare depth values to previously stored values in the depth buffer to determine visibility.

   Calculate z-value for each (x, y) position on the polygon.

   If z > Z-buffer (x, y), then set

   Z-buffer (x, y) = z, frame-buffer (x, y) = $I_{surface}$ (x, y)

3. Stop

- Note that, $I_{background}$ is the value for the background intensity and $I_{surface}$ is the projected intensity value for the surface at pixel position (x, y). After processing of all surfaces, the Z-buffer contains depth values for the visible surfaces and the frame buffer contains the corresponding intensity values for those surfaces.
- To calculate z-values, the plane equation

  $Ax + By + Cz + D = 0$

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

- Therefore, we can write

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

- Note, if at (x, y) the above equation evaluates to z_1, then at (x + Δx, y) the value of z, is

$$z_1 = \frac{A}{C}(\Delta x)$$

- Only one subtraction is needed to calculate z(x + 1, y), given z(x, y), since the quotient A/C is constant and Δx = 1. A similar incremental calculation can be performed to determine the first value of z on the next scan line, decrementing by B/C for each Δy.

Advantages

1. It is easy to implement.
2. It can be implemented in hardware to overcome the speed problem.
3. Since the algorithm processes objects one at a time, the total number of polygons in a picture can be arbitrarily large.

Disadvantages

1. It requires an additional buffer and hence the large memory.
2. It is a time consuming process as it requires comparison for each pixel instead of for the entire polygon.

Review Questions

1. *How does the Z-buffer algorithm determine which surfaces are hidden ?*
2. *Explain Z-buffer algorithm and its applications.*

9.7 A Buffer Algorithm

- 'A' buffer method is an extension of the ideas in the 'Z'-buffer method. It represents an antialiased, area-averaged, accumulation-buffer method developed by Lucasfilm for implementation in the surface-rendering system called REYES (an acronym for "Renders Everything You EverSaw").
- We know that Z-buffer method has a drawback that it can only define one visible surface at each pixel position. That is it deals only with opaque surfaces and cannot accumulate intensity values for more than one surfaces. Since transparent surfaces require to accumulate intensity values for more than one surfaces it is not possible to display transparent surfaces using Z-buffer method. As an extension to

Z-buffer method, in A-buffer method each position in the buffer can reference a linked list of surfaces. Due to this more than one surface intensity can be taken into consideration at each pixel position and object edges can be antialiased. To accommodate depth information and link information A-buffer has two fields :

- **Depth field :** Holds a positive or negative real number
- **Intensity field :** Holds surface intensity information or a pointer value

- The depth field in the A-buffer indicates whether a single surface or multiple surfaces contribute the intensity of the corresponding pixel. When the depth field is positive, a single surface contributes the intensity of the corresponding pixel. When the depth field is negative, multiple surfaces contribute the intensity of the corresponding pixel. The information of multiple surface intensity is stored in the linked list. This is illustrated in Fig. 9.7.1.

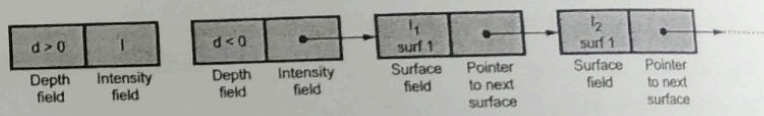


Fig. 9.7.1 Organization of an A-buffer

- The intensity field in the A-buffer stores the RGB components of the surface colour at that point and the percent of pixel coverage. The surface field stores the information of that particular surface. It includes :
  - Surface identifier
  - Depth
  - RGB intensity components
  - Opacity parameter (percent of transparency)
  - Percentage of area coverage and
  - Other surface rendering parameters
- The A-buffer algorithm is implemented similar to Z-buffer algorithm. However, in A-buffer algorithm the intensity of pixel is determined by considering opacity parameter and percentage of overlaps of the overlapping surfaces.

Review Question

1. *Explain A-buffer visible surface algorithm.*

9.8 Scan Line Algorithm

AU : Dec.-13

A scan line method of hidden surface removal is an another approach of image space method. It is an extension of the scan line algorithm for filling polygon interiors. Here, the algorithm deals with more than one surfaces. As each scan line is processed, it examines all polygon surfaces intersecting that line to determine which are visible. It then does the depth calculation and finds which polygon is nearest to the view plane. Finally, it enters the intensity value of the nearest polygon at that position into the frame buffer.

We know that scan line algorithm maintains the edge list in the edge table (ET). The AET (Active Edge Table) contains only edges that cross the current scan line, sorted in order of increasing x. The scan line method of hidden surface removal also stores 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.

The Fig. 9.8.1 (a) illustrates the scan line method for hidden surface removal. As shown in the Fig. 9.8.1 (a), the active edge list for scan line 1 contains the information for edges AD, BC, EH and FG. For the positions along this scan line between edges AD and BC, only the flag for surface S_1 is ON. Therefore, no depth calculations are necessary, and intensity information for surface S_1 is entered into the frame buffer. Similarly, between edges EH and FG, only the flag for surface S_2 is ON and during that portion of scan line the intensity information for surface S_2 is entered into the frame buffer.

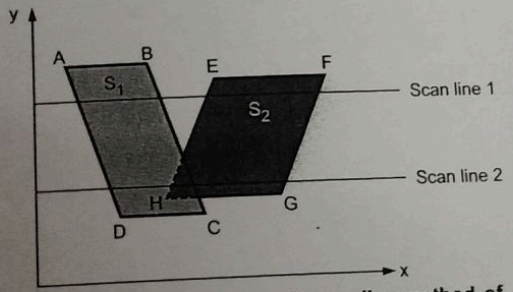


Fig. 9.8.1 (a) Illustration of scan line method of hidden surface removal

For scan line 2 in the Fig. 9.8.1 (a), the active edge list contains edges AD, EH, BC and FG. Along the scan line 2 from edge AD to edge EH, only the flag for surface S_1 is ON. However, between edges EH and BC, the flags for both surfaces are ON. In this portion of scan line 2, the depth calculations are necessary. Here we have assumed that the depth of S_1 is less than the depth of S_2 and hence the intensities of surface S_1 are loaded into the frame buffer. Then, for edge BC to edge FG portion of scan line 2 intensities of surface S_2 are entered into the frame buffer because during that portion only flag for S_2 is ON.

To implement this algorithm along with AET we are required to maintain a Polygon Table (PT) that contains at least the following information for each polygon, in addition to ID.

1. The coefficient of the plane equation.
2. Shading or colour information for the polygon.
3. A in-out Boolean flag, initialized to false and used during scan line processing.

- The Fig. 9.8.1 (b) shows the ET, PT and AET for the scan line algorithm.

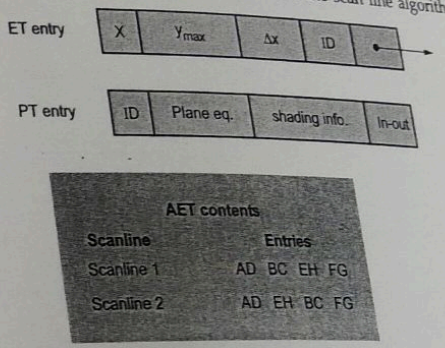


Fig. 9.8.1 (b) ET, PT, AET, for the scan line algorithm

Review Questions

1. *Explain the scanline algorithm for hidden surface removal.*
2. *Discuss on the various visualization techniques in detail.* AU : Dec.-13, Marks 8

9.9 Two Marks Questions with Answers

Q.1 What do you mean by visible surface determination ? (Refer section 9.1)

Q.2 Give the two basic types of hidden line algorithm. (Refer section 9.1)

Q.3 What is object-space method ? (Refer section 9.1)

Q.4 What is image-space method ? (Refer section 9.1)

Q.5 State the advantages of Warnock's algorithm. (Refer section 9.5)

Q.6 State the advantages of Z-buffer algorithm. (Refer section 9.6)

Q.7 State the disadvantages of Z-buffer algorithm. (Refer section 9.6)

Q.8 How A-buffer algorithm differ from Z-buffer ? (Refer section 9.7)