Introduction to Hidden Surface Elimination

10

10.1. 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 algorithm 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 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 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 algorithm use image-space methods.

In this chapter, we are going to study various visible surface detection or hidden line removal and hidden surface removal algorithms.

The various hidden surface removal and detections algorithms are :

- Back-face removal algorithm
- Z-buffer Algorithm or Depth buffer algorithm

- A-buffer algorithm
- Scan-line algorithm
- Painters or Depth sorting algorithm
- Area subdivision algorithm

10.2. BACK FACE REMOVAL ALGORITHM

We know that a polygon has two surfaces, a front and a back just as a piece of paper does. We picture our polygons with one side painted light and the other painted dark. But the question is "find which surface is light or dark". When we are looking at the light surface the polygon will appe be drawn with counter clockwise pen motions and when we are looking at the dark surface the po will appear to be drawn with clockwise pen motions, as shown in the Fig. 10.1.

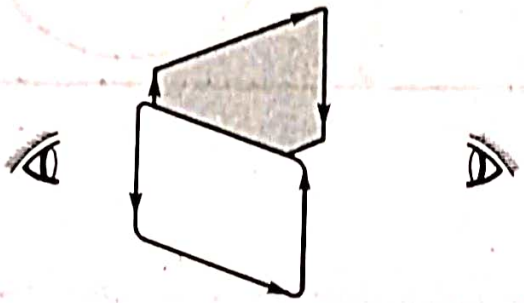


Fig. 10.1. Drawing directions.

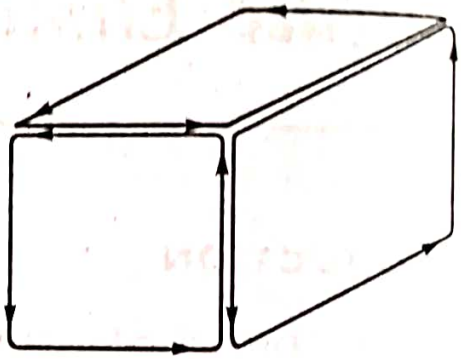


Fig. 10.2. Exterior surfaces are coloured light.

Let us assume that all solid objects are to be constructed out of polygons in such a way that only light surfaces are open to the air; the dark faces meet the material inside the object. This means that w we look at an object face from the outside, it will appear to be drawn counter clockwise, as shown in Fig. 10.2.

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

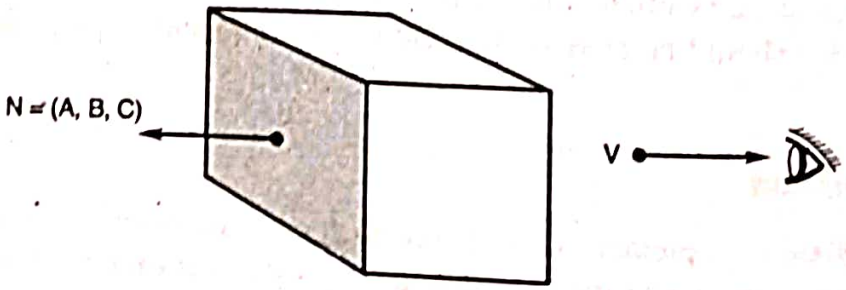


Fig. 10.3.

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

$$N_V > 0$$

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. (Refe Fig. 10.3).

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. Thus 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 direction are opposite ($\pi/2 < \theta \leq \pi$) then the cosine and the overall dot product is negative (Refer Fig. 10.4)

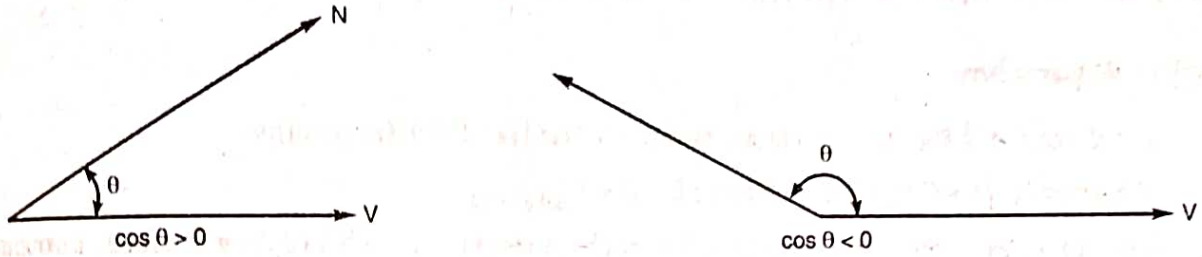


Fig. 10.4. Cosine angles between two vectors.

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 coordinates and our viewing direction is parallel to the viewing zv axis, then $V = (0, 0, V_z)$ and

$$VN = V_z C$$

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.

10.3. Z-BUFFER ALGORITHM (DEPTH 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 coordinates (x, y, z), each pixel position on the view plane is specified by x and y coordinate, and z-value gives the depth information. Thus object depths can be compared by comparing the z-values.

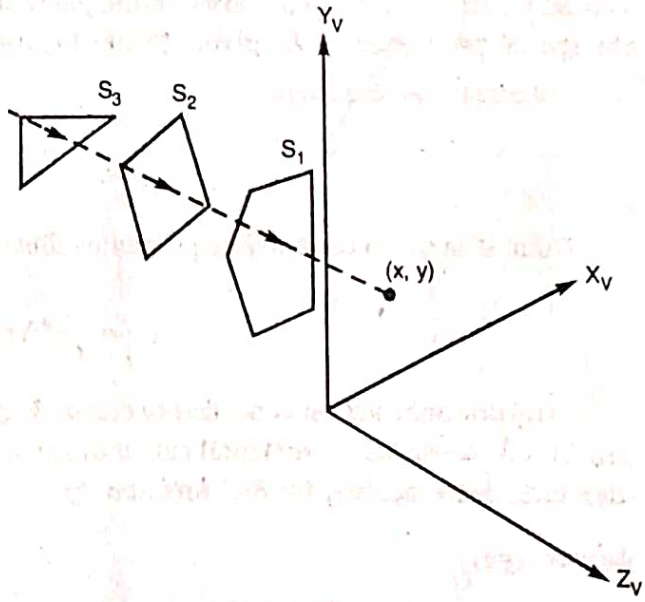


Fig. 10.5.

The Z-buffer algorithm is usually implemented in the normalized coordinates, 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 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 ten 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 previous stored in the Z-buffer at that position. If the calculated depth values is greater than the value stored in t Z-buffer, the new depth value is stored, and the surface intensity at that position is determined and plac in the same xy-location in the frame buffer.

For example in Fig. 10.5 among three surfaces, surface S_1 has the smallest depth at view position (x, and hence highest z-value. So it is visible at that position.

Z-buffer Algorithm

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

   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 dep 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 intensi value for the surface at pixel position (x, y). After processing of all surfaces, the Z-buffer contains dep values for the visible surfaces and the frame buffer contains the corresponding intensity values for the 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 describi the spatial properties of the plane. (Refer to Appendix A for detials).

Therefore, we can write

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

Note, if at (x, y) the above equation evaluates to z_p then at $(x + \Delta x, y)$ 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 const and $\Delta x = 1$. A similar incremental calculation can be performed to determine the first value of z on the n 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 c 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.

10.4. A-BUFFER ALGORITHM

In the year 1984. Loren Carpenter introduced this method in the 11th annual conference of Computer Graphics and Interactive Techniques. The A-buffer, the anti-aliased, area-averaged, accumulation buffer, is developed by Lucas Films and used demo sequence in Star Treck-II for the first time.

This is an algorithm, which takes care of not only the opaque surfaces but also consider transparent surfaces. Thus, this algorithm shows the true distance of each and every pixel. This is an algorithm which falls under the image-space method category.

A-buffer algorithm is an expansion of Z-buffer algorithm. In A-buffer algorithm, a link list is attached with each pixel position and this link list carries the intensity information of each surface associated with that position as shown in Fig. 10.6.

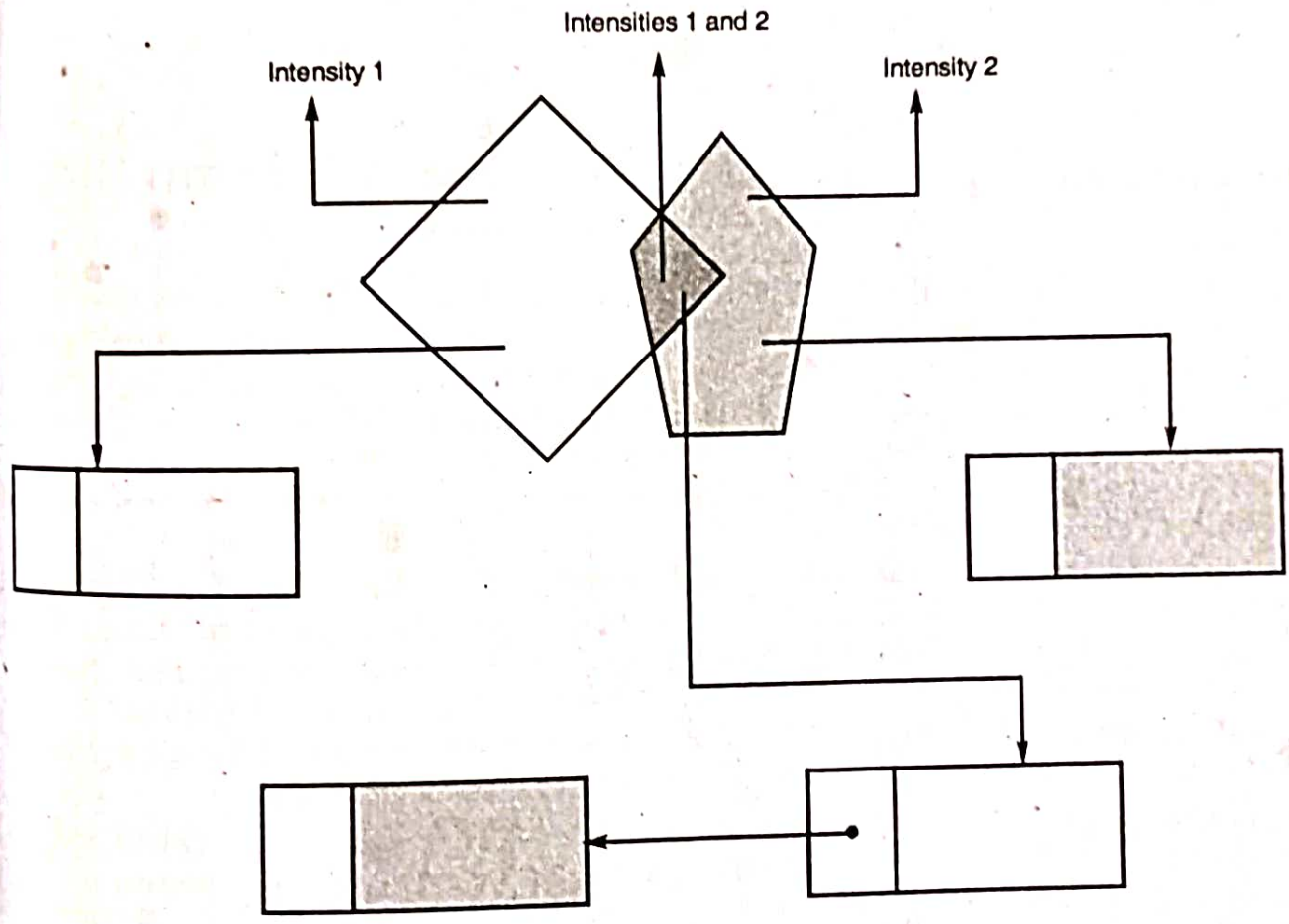


Fig. 10.6. Link list associated with different parts of a scene.

In A-buffer, each node contains two fields, a depth value and intensity value of a given pixel. If the pixel is having only one surface, then the depth and intensity information will be stored in one node. Whereas, if there is more than one surface, one node each will be there in a link list containing the surface information. In this case, each node also contains the link information of the next node. The intensity field carries all the required data like RGB values, transparency value, surface identification information, etc. It is important to note that this algorithm works for the anti-aliasing too.

10.5. SCAN-LINE ALGORITHM

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.

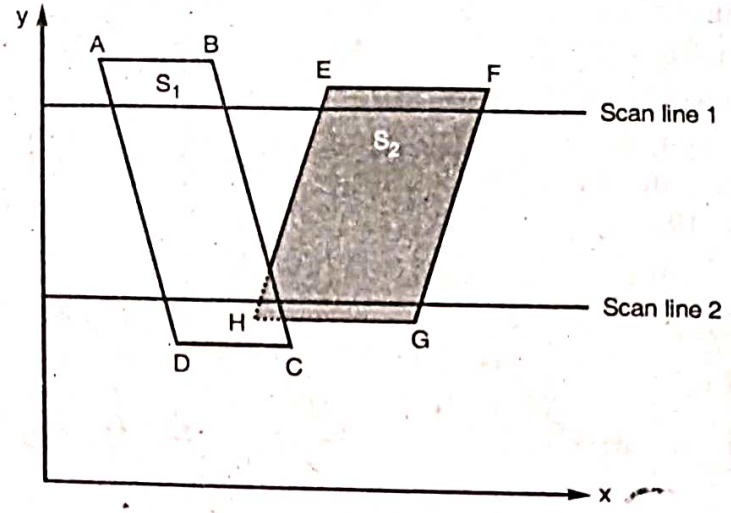


Fig. 10.7. (a) Illustration of scan line method of hidden surface removal.

We know that scan line algorithm maintains the edge list in the Edge Table (ET). The Active Edge Table (AET) 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. 10.7(a) illustrated the scan line method for hidden surface removal. As shown in the Fig. 10.7(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.

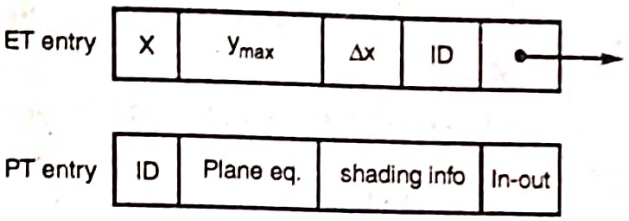
For scan line 2 in the Fig. 10.7 (a), the active edge list contains edges AD, EH, BC and FG. Along the scan line 2 from edge AD to edge EH, only the flag for surface S_1 is ON. However, between edges EH and BC, the flags for both surfaces are ON. In this portion of scan line 2, the depth calculations 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. 10.7(*b*) shows the ET, PT and AET for the scan line algorithm.



AET contents

Scanline	Entries
Scanline 1	AD BC EH FG
Scanline 2	AD EH BC FG

Fig. 10.7(b). ET, PT, AET, for the scan line algorithm.

10.6. THE PAINTER'S ALGORITHM OR DEPTH SORTING ALGORITHM

As the name suggests, the algorithm follows the standard practice of a painter. In creating an oil painting, an artist first paints the background colour next, the most distant objects are added then the nearer objects and so forth. The new paint layer covers the old so that only the newest layer of paint is visible. Frame buffer has the same property, we enter a filled polygon into the frame buffer by changing the proper pixels to values corresponding to the polygon's interior style. If we then enter a second polygon "on top of" the first, some of those same pixels will be changed corresponding to the second polygon's interior style. Wherever the second polygon lies first. The Painter's algorithm tells, that those polygons which are farthest from the viewer enter first, *i.e.*, the Background, the objects closest to the viewer enter, last, *i.e.* foreground. Hidden surface can be covered up by choosing the correct order to draw them and taking advantage of the properties of frame buffers.

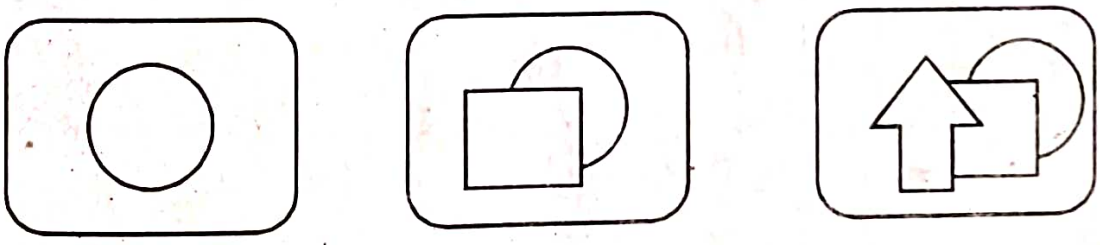


Fig. 10.8.

The pointer's algorithm is also called as **depth sorting algorithm** or **priority algorithm.**

The basic idea of the depth sort algorithm developed by **Newell** and **Sancha,** is to paint the polygons into the frame buffer in order of decreasing distance from the view point. This process involves following basic functions :

(*a*) Sorting of polygons in order of decreasing depth.

(*b*) Resolving any ambiguities this may cause when the polygon's z extents overlap *i.e.*, splitting polygon if neccessary.

(*c*) Scan conversion of polygon in order, starting with the polygon of greatest depth.

To determine the specific nature of the relative position of polygons, it is necessary to define the *x*-, *y*- and *z*-extents, of a polygon in 3D space as the respective coordinate ranges of the bounding (*i.e.*, enclosing) box, as shown in Fig. 10.9.

For the quadrilateral *ABCD*, the *x*-extent is $(x_D - x_B)$, *y*-extent is $(y_A - y_C)$, and *z*-extent is $(z_B - z_D)$. The *x*- and *y*-extents of two polygons will enable decisions to be made on their overlap, and the *z*-extent will define their relative position with respect to the viewer.

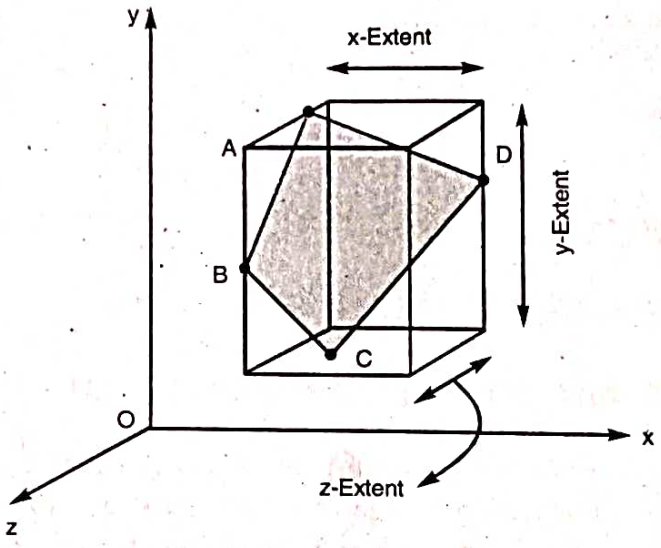


Fig. 10.9. Extents for a polygon.

Based on the extents, the farthest polygon is stored first, and over it the next farther polygon, and so on, until all the polygons are stored. The second and subsequent polygons automatically superpose themselves over the earlier ones, and reveal only the visible portions. Figure 10.10 shows the sequence of operations :

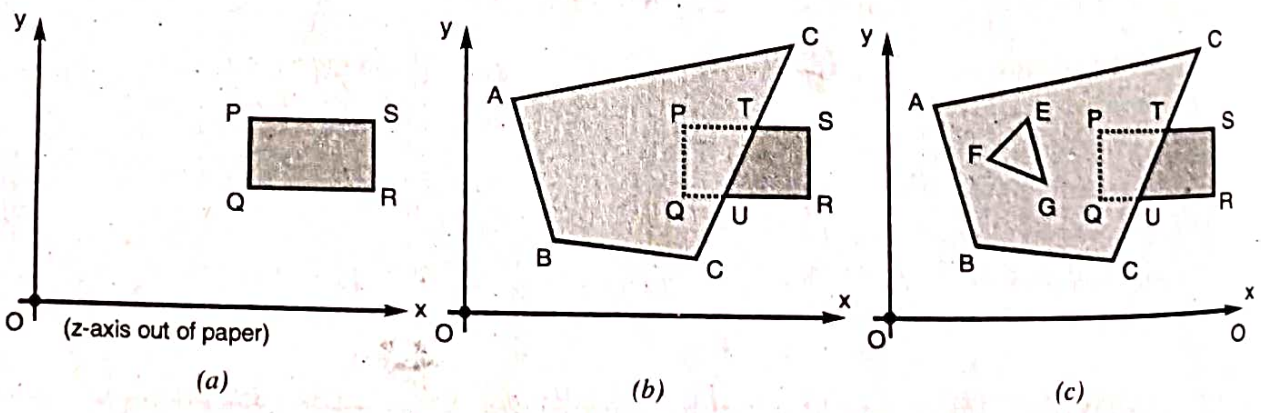


Fig. 10.10.

We can check whether any polygon Q does not obscure polygon P by performing following steps :

1. The *z*-extents of P and Q do not overlap, *i.e.*, $zQ_{max} < zP_{min}$. [see Fig. 10.11(*a*)]
2. The *y*-extents of P and Q do not overlop. [see Fig.10.11(*b*)]
3. The *x*-extents of P and Q do not overlop.
4. Polygon P lying entirely on the opposite side of 'Q's plane from the view port. [see fig. 10.11(*c*)]

5. Polygon Q lying entirely on the same side of P's plane as the viewport. [see Fig. 1011(d)]
6. The projections of the polygons P and Q onto the xy screen do not overlap.

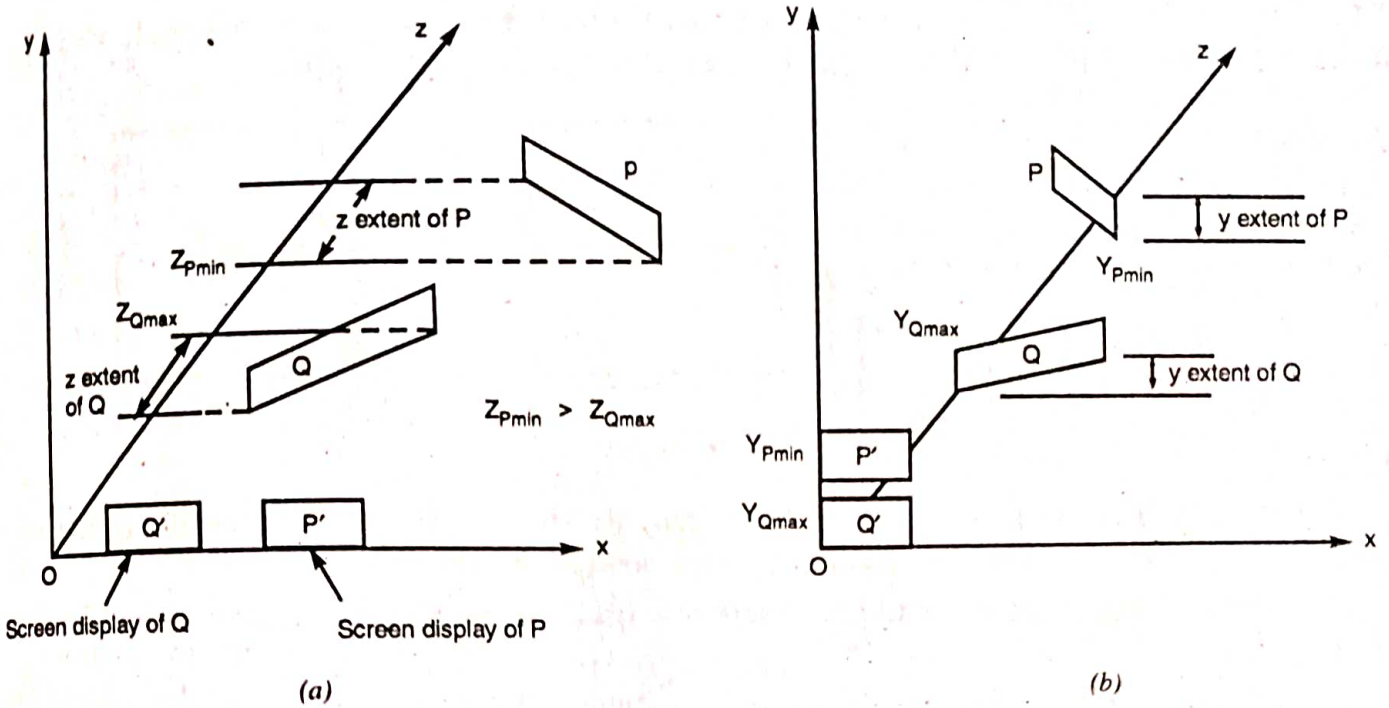


Fig. 10.11

If all these five tests fail, we assume for the moment that P actually obscure 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 spliting is that the splitted polygons may not obscure other polygon.

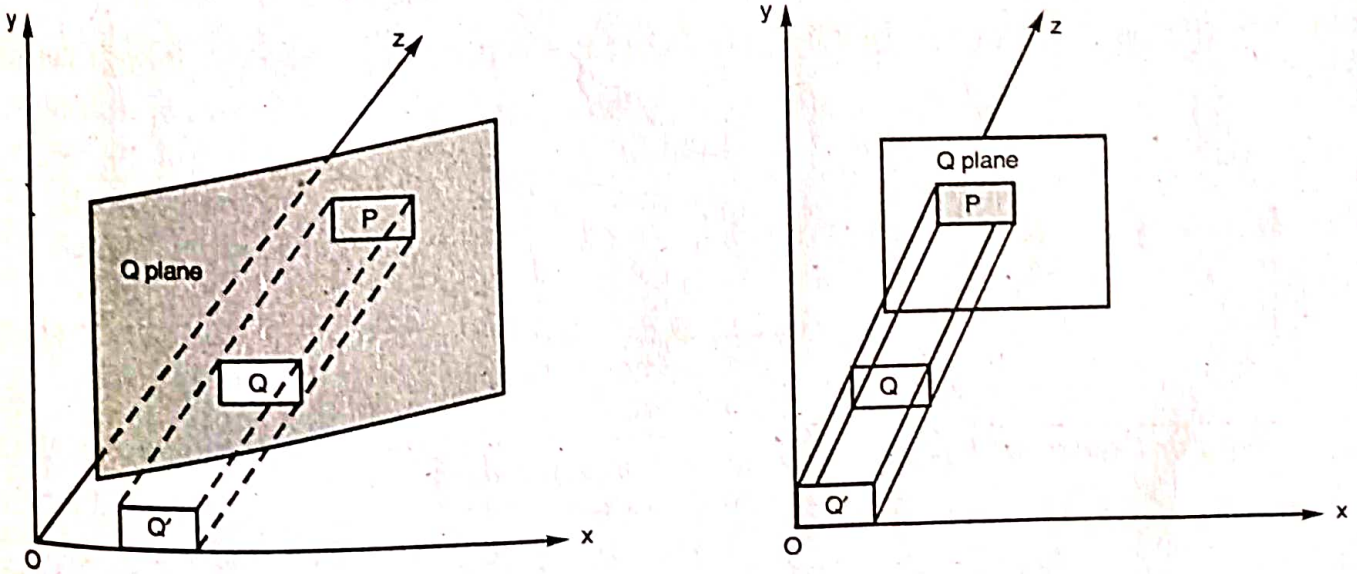


Fig. 10.12.

10.7. WARNOCK'S ALGORITHM (AREA SUBDIVISION METHOD)

An intersecting approach to the hidden-surface problem was developed by Warnock. He developed area subdivision algorithm which subdivides each area into four equal squares.

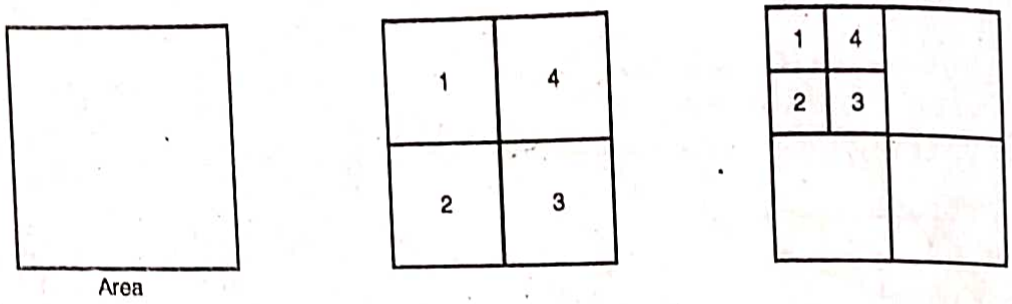


Fig. 10.13.

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 area of interest.

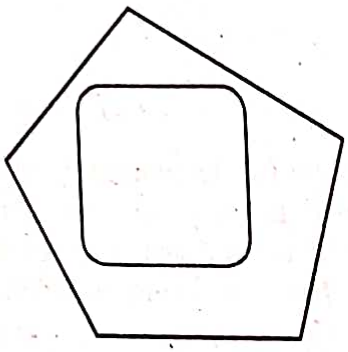


Fig. 10.14.

2. **Overlapping or Intersecting Polygon :** One that is partly inside and partly outside the area.

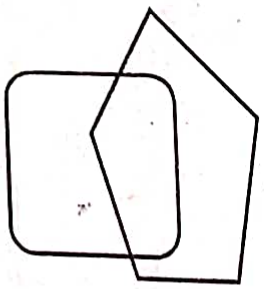


Fig. 10.15.

3. **Inside or Contained Polygon :** One that is completely inside the area.

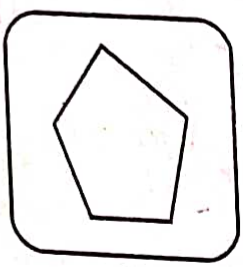


Fig. 10.16.

4. **Outside or Disjoint Polygon :** One that is completely outside the area.

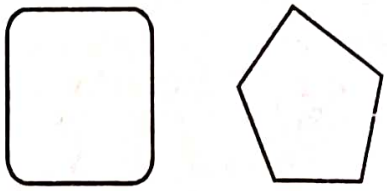


Fig. 10.17.

After checking 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 color of polygon.
3. If there is a single surrounding polygon but no intersecting or contained polygons, then the area is filled with the color 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.

The tests for determining surface visibility within an area can be stated in terms of these four classifications. No further subdivisions of a specified area are needed if one of the following conditions is true :

1. All surfaces are outside surfaces with respect to the area.
2. Only one side, overlapping, or surrounding surface is in the area.
3. A surrounding surface obscures all other surfaces within the area boundaries.

Test 1 can be carried out by checking the bounding rectangles of all surfaces against the area boundaries. Test 2 can also use the bounding rectangles in the *xy*-plane to identify an inside surface. For other types of surfaces, the bounding rectangles can be used as an initial check. If a single bounding rectangle intersects the area in some way, additional checks are used to determine whether the surface is surrounding, overlapping or outside. Once a single inside, overlapping, or surrounding surface has been identified, its pixel intensities are transferred to the appropriate area within the frame buffer.

One method for implementing test 3 is to order surfaces according to their minimum depth from the view plane. For each surrounding surface, we then compute the maximum depth within the area under consideration.

Thus, the polygons which are coming under category four *i.e.* disjoint polygon, are totally invisible. So we are not considering them at all for visibility. Now we have to concentrate on only first three categories. We will keep these polygons in a separate polygon list, which is called as **Potentially Visible Polygon List (PVPL).**

If the polygon is of category one *i.e.* surrounding polygon, then whole area of the screen will get the colour of surrounding polygon. For category-two *i.e.* intersecting polygon, we will apply clipping algorithm to convert category two into category three and four.

For category three *i.e.*, contained polygon, we are going to subdivide the area into four subareas and again apply the same logic. There is no point in applying subdivision for case one and case four *i.e.* surrounding polygon and disjoint polygon case. Because as in surrounding case whole area *A* is visible so

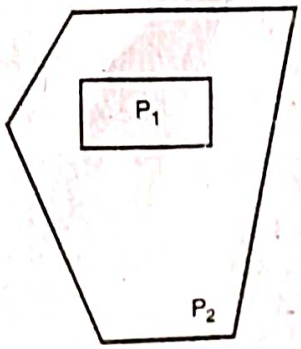


Fig. 10.18

even if we further subdivide it, that area will be visible only. The same thing is for disjoint category. It means the subdivision procedure is to be applied to contained and intersection category only.

Now, coming back to our warnock algorithm, if there are two polygons then the polygon is not visible, if it is in back of surrounding polygon. If there are two polygons p_1 and p_2 and p_2 surrounds p_1 then it then it means p_1 should not be displayed *i.e.*, p_1 should be removed from PVRL.

Actually, PVRL, could be the list of smallist z-coordinates of the polygons which are within the area *i.e.*, z_{min} and let z_{max} be the maximum z value of surrounding polygon.

For example : Figure 10.18 is shown with different look as Fig. 10.19.

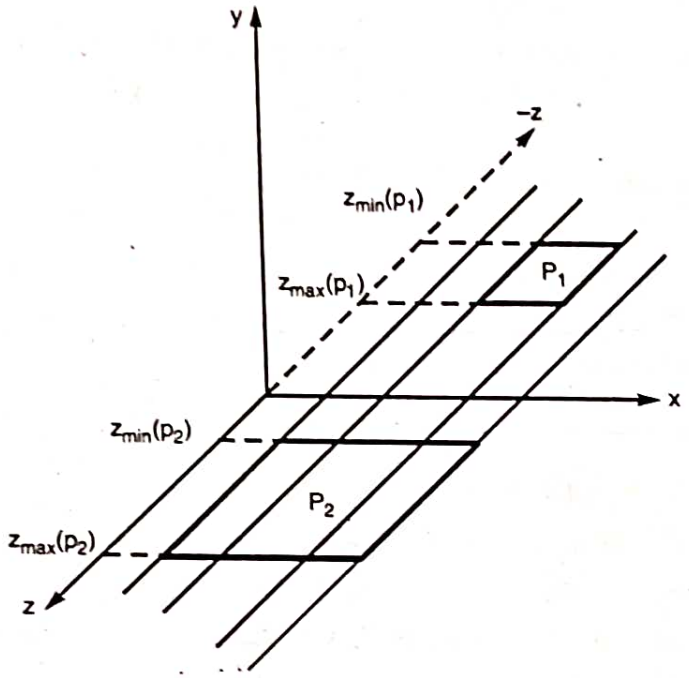


Fig. 10.19

From the Fig. 10.19, we can easily say that if $z_{max}(p_1) < z_{min}(p_2)$, then polygon p_1 is hidden. Here polygon p_2 is surrounding polygon. So we can say $z_{max}(p_1) < z_{min}$ then p_1 is hidden. In fact, all other polygons after p_1, *i.e.*, whose z is smaller than $z_{max}(p_1)$, on the list will be hidden by p_2 so remove all them from PVRL.

But if polygon p_1 is completely inside the polygon p_2 *i.e.*, contained category then it will be exactly opposite.

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. Don't 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 (x, y), and neither a, b, c nor d applied, 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. Extra memory buffer is not required.
2. It follows the divide-and conquer startegy, therefore, parallel computer can be used to speed up the process.

SOLVED PROBLEMS

1. What is Back-face detection?

Ans. A simple object space algorithm is a back-face removal (or Back face cull) where no faces on the back of the object are displayed. Since in general about half of the faces of the objects are back faces,Here we give the algorithm that will remove about half of the total polygons in the image look at a left handed viewing system :

If $C < 0$ then a back face is in right handed system. So, an algorithm (for left handed system) is –

1. Compute N for every face of the object
2. If C(Z–component) > 0 then a back face and don't draw it.

Some advantages for this back face detection algorithm, are :

1. The back face cull is a good preprocessing step, once it removes about half of the polygons in the science.
2. For colour shading, we must complete the normal for all of the polygons anyway.

2. How does the z-buffer algorithm determine which surfaces are hidden ?

Ans. From the plane equation of the face currently being scan converted, the depth at the point (x, y) is easily evaluated where y is the scan line and x is the position in the scan line. If the eqn of the plane is $ax + by + cz + d = 0$

then
$$z = \frac{-(ax + by + d)}{c}$$

along a scan line y stays constant while x increases by one, hence z changes by an incremental abount dz, given by

$$dz = \frac{a}{c}$$

for any scan line adjacent horizontal positions across the line differ by 1 and a vertical y value on an adjacent scan line diffewrs by 1. If depth of position (x, y) has been determined to be 2, then depth of z of the next position $(x + 1, y)$ along the scan line is obtained from eqn as

$$z = \frac{-a(x+1-by-d}{c}$$

$$z' = z - \frac{a}{c}$$

the ratio –a/c is constant for each surface Now the z-buffer algorithm can be stated as

Step 1: Initialize frame buffer to background colour.

Step 2: Initialize z-buffer to minimum z value.

Step 3: Scan convert each polygon in arbitrary order.

Step 4: For each (x, y) pixel, calculate depth 'z' at that pixel $z(x, y)$.

Step 5: Compare calculated now depth z (x, y) with value previously stored in z buffer at the location z (x, y)

Step 6: If $Z_{buff}(x, y) > z(x, y)$ then write the now depth value to z buffer and update frame buffer.

Step 7: Otherwise, no action is taken.

3. Given points $P_1(1, 1, 1)$ $P_2(5, 10, 9)$ and $P_3(1, 2, 1)$ and a view point $M(0, 0, 1)$ determine which points obscure the other when viewed from M.

Ans. We first need to find a line joining point P_1 and M and then finding which all points lie on this line. Then we need to find the distance of these points from M, the nearest point will be obscure the other colinear points. Parametric line equation will useful for this.

$\therefore$ Line joining view point M (0, 0, 1) and P_1 (1, 2, 1) is

$$x = t$$

$$y = 2t$$

$$z = -1 + 2t$$

$\therefore$ We can see that point P_2 (5, 10, 9) lies on this line at $x = 5$, when $t = 5$, the y coordinate = 10 and $z = 9$. Hence B lies on a line through M and P_1.

Similarly we can check point P_3 does not lies on this line, so point P_1 obscure P_2 and doesn't obscure point P_3.

4. Describe the scan line method.

Ans. A scan line algorithm consists of two nested loop an x scan loop nested with in a y-scan loop.

y-scan – for each y value, say $y = \alpha$ intersect the polygons to be rendered with the scan plane $y = \alpha$. This scan plane is parallel to the xz plane and the resulting intersection are line segments in this plane.

x-scan – (1) for each value of x, say $x = \beta$, intersect the line segments found above with the x-scan line $x = \beta$ lying on the y-scan plane. This intersection results in a set of points that lies on the x-scan line.

(2) Sort these points with respect to their z-coordinates.

The point (x, y, z) with the smallest z value is visible, and the color of the polygon containing this point is the color set at the pixel corresponding to this point. In order to reduce the amount of

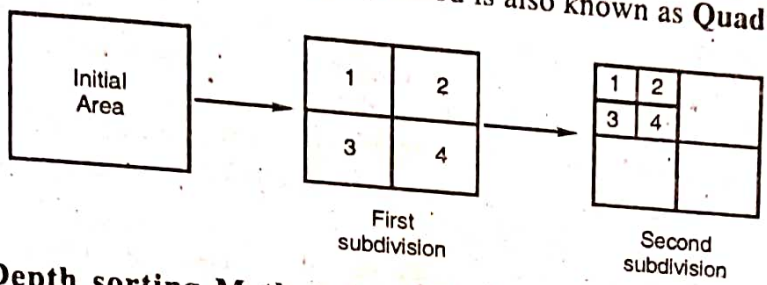
calculation in each scan line loop, we try to take advantage of relationships and dependencies called coherences, between different elements that comprise a scene.

5. Discuss the area subdivision method ?

Ans. It was developed by warnock and it uses divide and congues strategy. This algorithm has two steps :

Step 1: First of all, decide which all polygons are visible in area, they are displayed.

Step 2: Else the area is divided into four equal areas, on each area those polygons are further tested to determine which ones should therefore be displayed. If a visibility decision cannot be mad either this second area is further subdivided either until a visibility decision can be made or until the screen area is a single pixel. The method is also known as **Quadtree method.**



6. Discuss the Depth sorting Method for hidden surface removal. Why is the polygon approximations required.

Ans. Z-buffer/depth Buffer is a simplest algorithms of the hidden surface removal. This method was originally proposed by catmull. It is an image space method. Here a frame buffer is used to store the attributes (intensity) of each pixel in image space. The z-buffer is a separate depth buffer used to store the z-coordinate or depth of every visible pixel in image space. The depth or 2 value of a new pixel to be written to a frame buffer is compared to the depth or z-value of the pixel already stored in the z-buffer.

If the 2 value of the new pixel > Z-value already stored in Z-buffer then Z-buffer is updated with the new Z-value and intensity of new pixel is written in frame buffer else no action is taken.

As each polygon is scan converted, the depth at each pixel is calculated and compared with the corresponding in the depth buffer. If the depth is than that stored in the depth buffer (*i.e.* nearer the viewer) then that pixel is set in the frame buffer with the polygon color at that point and the depth buffer is set to polygon depth. If the polygon depth is greater (*i.e.* further away from the viewer) than the depth buffer at that point then the pixel is not written to the frame buffer.

7. Discuss A-buffer algorithm?

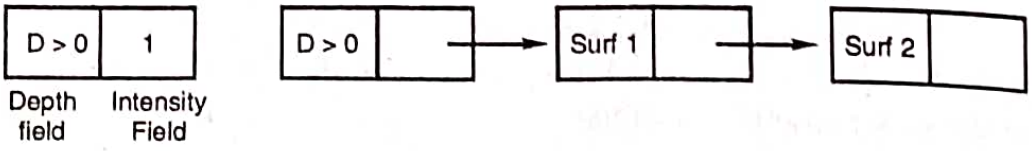
Ans. A buffer method is an extension depth buffer method. This method represented an antialised, area averaged, accumulation buffer method is that it can only find one visible surface at each pixel position. That means, it deals only with opaque surfaces and cannot accumulate intensity, values for more than one surface, as is necessary if transparent surfaces are to be displayed.

Each pixel position in the A buffer has following fields :

(*i*) **Depth field** – Stores a positive or negative real numbers.

(*ii*) **Intensity field** – Stores surface – intensity information or a pointer value.

If depth field is positive, the number stored at that position is the dpth 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 as shown below.



8. How do we detect the back face in any polygon ?

Ans. Step to detect the back face in any polygon is as follow:

1. Compute N (Normal) for every face of the object/polygon.
2. If C (Z-component) > 0 (positive) then its a back face and don't draw.

9. Discuss the Depth Sorting Method for hidden surface removal. Why is the Polyg Approximations required ? Justify.

Ans. Z-buffer/depth Buffer is a simplest algorithms of the hidden surface removal. This method w originally proposed by catmull. It is an image space method. Here a frame buffer is used store the attributes (intensity) of each pixel in image space. The z-buffer is a separate dep buffer used to store the z-coordinate or depth of every visible pixel in image space. The dep or 2 value of a new pixel to be written to a frame buffer is compared to the depth or z-value the pixel already stored in the z-buffer.

If the 2 value of the new pixel > Z-value already stored in Z-buffer then Z-buffer is updat with the new Z-value and intensity of new pixel is written in frame buffer else no action taken.

As each polygon is scan converted, the depth at each pixel is calculated and compared w the corresponding in the depth buffer. If the depth is than that stored in the depth buffer (i nearer the viewer) then that pixel is set in the frame buffer with the polygon color at that po and the depth buffer is set to polygon depth. If the polygon depth is greater (*i.e.* further aw from the viewer) than the depth buffer at that point then the pixel is not written to the fra buffer.

EXERCISES

1. Explain Z-buffer method. Which is better and why Z-buffer OR A-buffer.
2. Explain Depth Sorting Method.
3. What happens when two polygons have the same z value and the z-buffer algorithm is used ?
4. How can the amount of computation required by the scan line method be reduced.
5. Explain and write expressions for Z-bugger algorithm. How A-buffer method removes the drawbs of Z-buffer algorithm.
6. Explain Depth Sorting Method.
7. What is the difference between A buffer amd Z buffer method.
8. Define area subdivision method and its various variants available.