Reference/Resource:
J.D. Foley, A. Van Dan, Feiner, Hughes, "Computer Graphics Principles & Practice, 2nd edition, Publication Addison wesley, 1990.

FOLEY ch-15

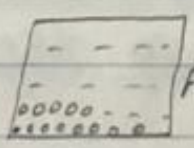
we do not want to draw surfaces that are hidden. In this chapter we will study three algorithms that compute which surfaces are hidden, we can then bypass them & draw only the surfaces that are visible.

three Algo.

- Z-buffer/ depth buffer Algo.
- warnock's Algo.
- Depth-Sort Algo.

(1). Z-buffer/Depth Buffer Algorithm

An example to understand the concept

Frame Buffer: It stores the information about each & every Pixel 

eg 10001011 1 → on, 0 → off] monochrome display

In case of colour display it will have multiple valu

Z-buffer: Size x * y

It will store z-value of each Pixel (x, y). we will maintain a z-buffer it will store the depth of the Pixel being displayed at each Pix

Put in frame buffer. The z-buffer will be updated after comparing the current depth (Polygon 2) with the Previous depth (stored value of polygon 1). Pixel with greater depth will be displayed.

① It is an image-space approach.
② Each surface (Polygon) is processed seperately one pixel position at a time.
③ The depth value for pixel are compared & the closest Polygon determines the color to be displayed.
④ The polygons can be scanned in any order.
⑤ It requires that we have available not only a frame buffer F in which color values are stored, but also a z-buffer Z, with the same number of entries, in which a z-value is stored for each pixel.
⑥ Algorithm's Steps
(i) Intialize frame buffer F to background color.
(ii) Intialize z-buffer Z to zero representing the z-value at the back clipping Plane. The largest value that can be stored in the z-buffer represents the z of the front Plane.
(iii) Scan convert each polygon in arbitrary order.

(iv). For each (x, y) pixel, calculate depth 'z' at that pixel.

(v). compare the calculated value from step (iv) to the value stored in z-buffer.

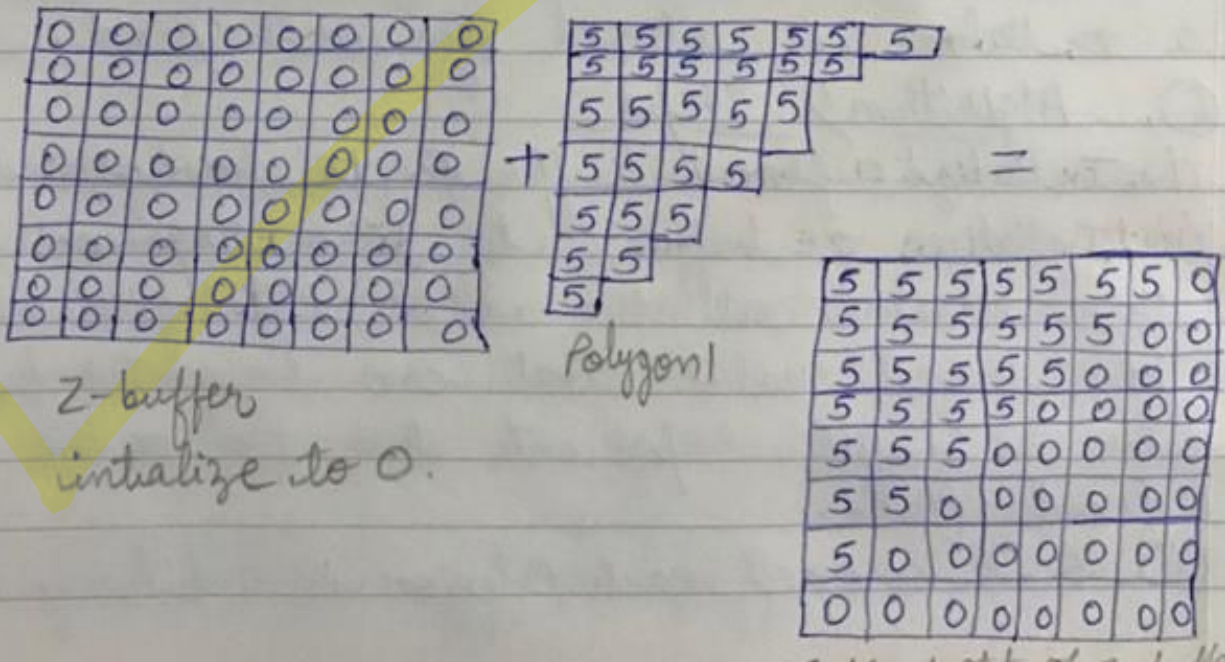
(vi). If new $z(x, y) >= z_{buff}(x, y)$, update z-buffer to the new value & update frame buffer.

(vii). else do nothing.

⇒ The entire process is a search over each set of pairs $\{z_i(x, y), F_i(x, y)\}$ for fixed x & y, to find the largest z_i. The z-buffer & the frame buffer record the information associated with the largest z encountered thus far for each (x, y).

Eg. A pixel's shade is shown by its color, its z value is shown as a number.

(1).



current state of z-buffer

② .

5	5	5	5	5	5	5	0
5	5	5	5	5	5	0	0
5	5	5	5	5	0	0	0
5	5	5	5	0	0	0	0
5	5	5	0	0	0	0	0
5	5	0	0	0	0	0	0
5	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

current state of z-buffer

+

3					
4	3				
5	4	3			
6	5	4	3		
7	6	5	4	3	
8	7	6	5	4	3

Polygon 2

=

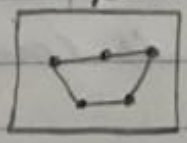
5	5	5	5	5	5	5	0
5	5	5	5	5	5	0	0
5	5	5	5	5	0	0	0
5	5	5	5	0	0	0	0
6	5	5	3	0	0	0	0
7	6	5	4	3	0	0	0
8	7	6	5	4	3	0	0
0	0	0	0	0	0	0	0

updated z-buffer

Calculating Depth-values for Polygons

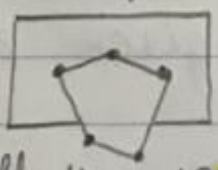
Polygons are of two types

Planar



All the vertices located at the same plane

Non-Palaner



All the vertices are not located in the same plane.

Depth value for ~~Pola~~ Planar Polygon

We know the depth values at the vertices. How can we calculate the depth at any other point on the surface of the polygon.

we have Plane equation

$$Ax + By + Cz + D = 0$$ Polygon surface equation

$$z = \frac{-Ax - By - D}{C}$$ At each step calculating z like this expensive

∴ we are using incremental algo.

For any scan line adjacent horizontal x positions or vertical y positions differ by 1 unit.

⇒ The depth value of the next position $(x+\Delta x, y)$ on the scan line can be obtained using

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

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

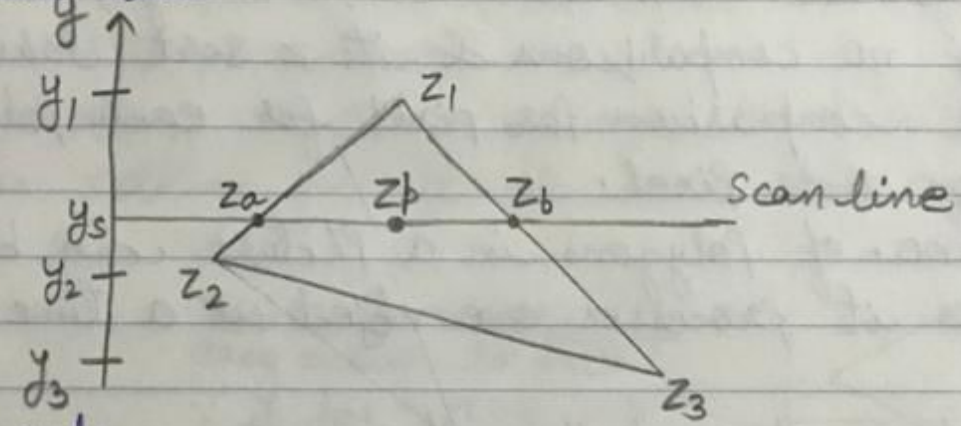
Only one subtraction is needed to calculate $z(x+1, y)$ given $z(x, y)$, since $\frac{A}{C}$ is constant & $\Delta x = 1$.

A similar incremental calculation can be performed to determine the first value of z on the next scan line, decrementing by $\frac{B}{C}$ for each Δy.

Depth Value for Non-Planar Polygons

Using interpolating method (It is a method of

constructing new data points within the range of discrete set of known data points.)
$z(x,y)$ can be determined by interpolating the z co-ordinates of the polygon's vertices along pairs of edges and then across each scan line.



Formula

$$z_a = z_1 - (z_1 - z_2)\frac{y_1 - y_s}{y_1 - y_2}$$ z_a is interpolated b/w z_1 & z_2

$$z_b = z_1 - (z_1 - z_3)\frac{y_1 - y_s}{y_1 - y_3}$$ z_b is interpolated b/w z_1 & z_3

$$z_p = z_b - (z_b - z_a)\frac{x_b - x_p}{x_b - x_a}$$ z_p is interpolated b/w z_a & z_b

Advantages of Z-buffer Algorithm

1. The z-buffer algo does not require that objects be polygons. It can be used to render any object if a shade and a z-value can

be determined for each point in its projection.
(2). It is easy to implement.
(3). It can be saved along with the image and used later to merge in other objects whoose z can be computed.
(4). It performs radix sort in x and y, requiring no comparisons & its z sort takes only one comparison per pixel for each polygon containing that pixel.
(5). Total no. of polygons in a picture can be large, as it processes one object at a time.

Disadvantages of Z-buffer Algo

(1). It requires a large amount of space for z-buffer.
(2). It has an aliasing problem, those pixels shared by the rendered edges may assigned slightly different z values because of numerical inaccuracies in performing the z interpolation.
(3). It is a time-consuming process.

(2) Warnock's Algorithm

Area-subdivision algorithm follow the divide and conquer strategy which states, if it is
- easy to decide which polygons are visible in the area, they are displayed. Otherwise the area is subdivided into smaller areas to which the decision logic is applied recursively.

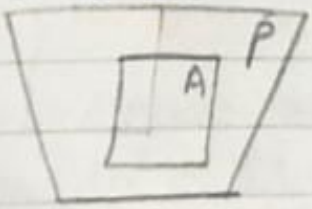
(1). Subdivides each area into four equal squares.
(2). It is a recursive subdivision process.
(3).

initial area first division second division

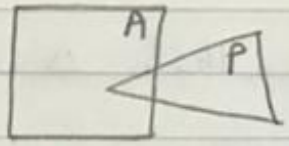
If the visibility decision cannot be made then this area is further subdivided either until a visibility decision can be made or until the screen area is a single pixel. This method is also known as quadtree method.

(4). At each stage in recursion, each polygon has one of four possible relationship with the area of interest (window)-:

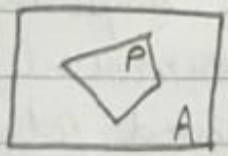
(i). Surrounding Polygons: completely contain the area of interest.



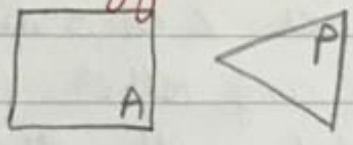
(ii). ~~Interes~~ Intersecting polygons: Intersects the area.



(iii). Contained Polygons : are completely inside the area.



(iv). Disjoint Polygons : are completely outside the area.



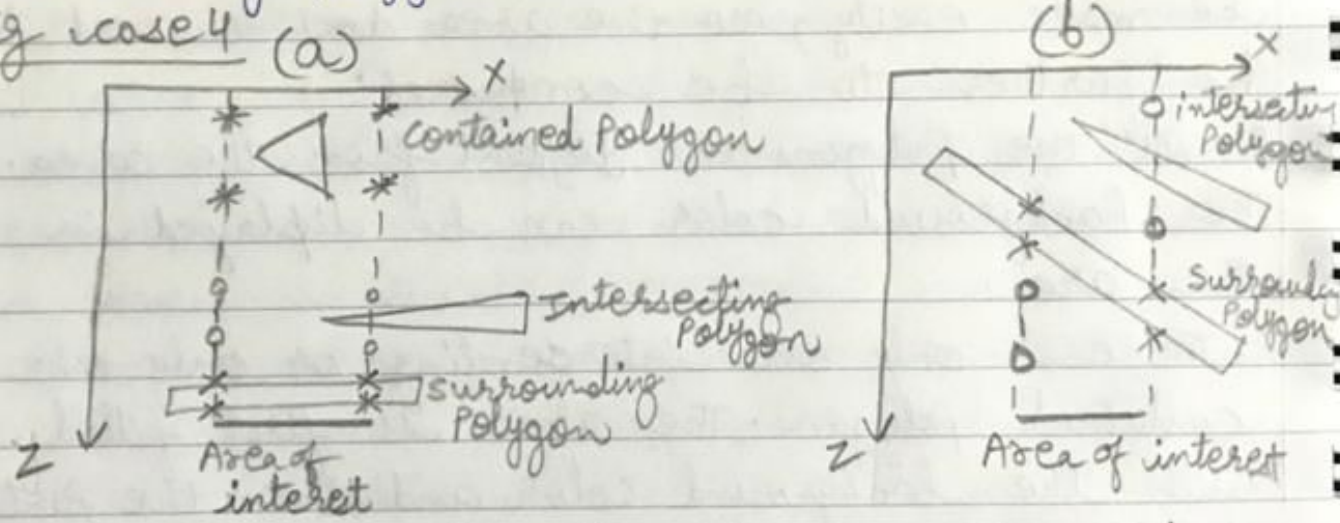
In four cases, a decision about an area can be made easily, so the area does not need to be further to be conquered:-

1. All the Polygons are disjoint from the area. The background color can be displayed in the area.
2. There is only one intersecting or only one contained Polygon. The area is first filled with the background color and then the part of the Polygon contained in the area is scan converted.
3. There is a single surrounding Polygon, the area is filled with the color of the surrounding Polygon.
4. More than one Polygon is intersecting, contained in or surrounding the area, but one

is a surrounding polygon that is in front of all other polygons.

Determining whether a surrounding polygon is in front is done by computing the z-co-ordinate of the planes of all surrounding, ~~polygon~~ intersecting and contained polygons at the four corners of the area, if there is a surrounding polygon whose four corner z co-ordinates are larger than those of any other polygons, then the entire area is filled with the color of surrounding polygon.

Eg case 4



X ; marks the intersection of surrounding polygon
O ; " " " " intersecting "
* ; " " " " contained "

In (a) the four intersections of the surrounding polygon are all closer to the viewport than are any of the other intersections. ∴ the entire area is filled by the surrounding polygon's color.

In (b) no decision can be made, ∵ on the left the plane of intersecting Polygon seems to be in front of the ~~intersecting Polygon~~ surrounding Polygon.
Warnock's algo. subdivides the area to simply the problem.

There is no point in examining case 1 (surrounding Polygon) & case 4 (Disjoint Polygon) only intersecting and contained Polygons need to be re-examined.

Advantage
Since it follows the divide and conquer strategy, so parallel computer can ~~be~~ be used to speed up the process.

Disadvatage
Test are complex & slow.

A worst case scenario occurs when the case is not one of the four.
Please refer Pg 688 example.

(3). Painter's Algorithm / Depth-Sort Algorithm / Priority Algorithm

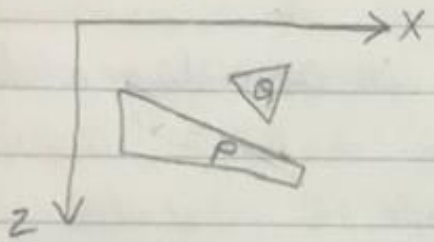
The Basic idea of the depth-sort algorithm, is to paint the polygons into the frame buffer in order of decreasing distance from the viewpoint.

Three conceptual steps are performed :-
1. Sort all polygons according to the smallest (farthest) z coordinate of each.
2. Resolve any ambiguities this may cause when the polygon's z extents overlap, splitting polygons if necessary.
3. Scan convert each polygon in ascending order of smallest z-coordinate.

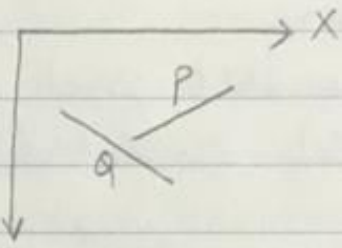
TO resolve the ambiguities in step 2, the algorithm performs 5 tests.
Let the polygon currently at the far end of the sorted list of polygons be called P. Before this polygon is scan-converted into the frame buffer, it must be tested each polygon Q whose z-extent overlaps the z-extent of P, to prove P cannot hide Q and that P can ∴ be written before Q upto 5 tests are performed in order of ↑ing complexity.
(The 5 test are in order. If test 1 fails enter in test 2 & so on. As soon as one succeeds, stop performing the test. let's say one of test pass, ∴ P has been shown not to obscure Q, & the next polygon Q overlapping P in z is tested. If all such polygons pass, then P is scan converted & the next polygon on the list becomes the new P)

Five test (order matters)

1. Do the Polygons' x-extents not overlap?
2. Do the Polygons' y-extents not overlap? (fail when overlap)
3. Is P entirely on the opposite side of Q's plane from the viewpoint?

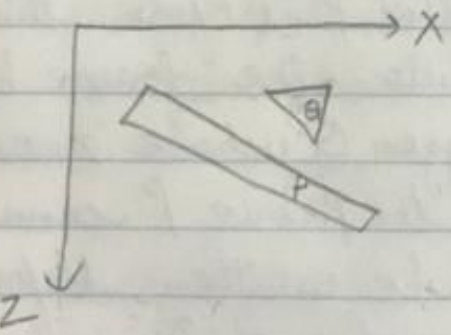


Test 3 fails

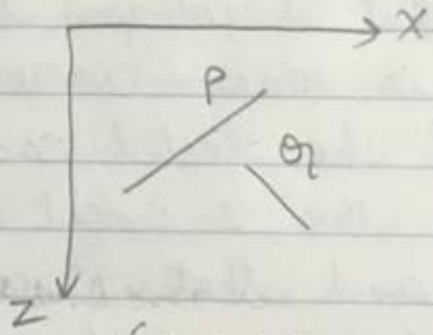


Test 3 is true

4. Is Q entirely on the same side of P's plane as the viewpoint?



(Viewpoint ↑↑↑↑
test 4 fails)



(↑↑↑↑ viewpoint)
test 3 is false
(∵ P is not behind Q)
but test 4 is true

5. Do the projections of the polygons onto the (x, y) plane not overlap? (This can be determined by comparing the edges.)

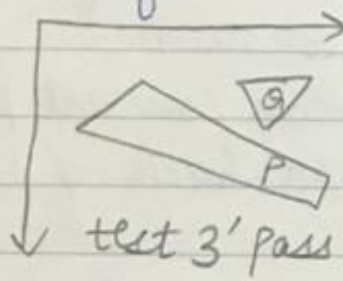
If all of the 5 test fails, then we have to do the ~~re ording~~ re-ordering of polygons (P, Q) to (Q, P), i.e P hides Q.

X ✓

∴ we need to perform test whether Q can be scan-converted before P.

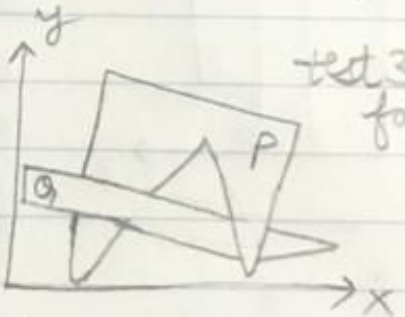
Test 1, 2 and 5 do not need to be repeated, but new version of test 3 & 4 are used, with order (Q, P)

3'. Is Q entirely on the opposite side of P's plane from the viewpoint?



test 3' pass

∴ we move Q to the end of the list & it becomes the new P.



test 3' & 4' fails

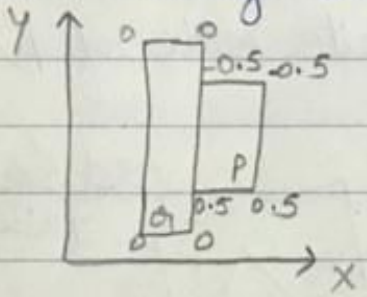
there is no order in which P and Q can be scan converted correctly. Either P or Q must be split. The original unsplit polygon is discarded, its pieces are inserted in proper z-order & the algorithm proceeds from the start.

4'. Is P entirely on the same side of Q's plane as the viewpt?

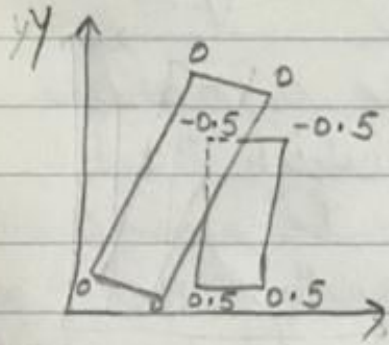
TO avoid looping, we modify our approach by marking each polygon that is moved to the end of the list. Then, whenever the first five test fails & the current polygon Q is marked, we do not tests 3' and 4'. Instead, we split either P or Q & reinsert the pieces.

Worst-case scenario

can two polygons fails all the tests even when they are already ordered correctly?



only the z-coordinates of each vertex is shown.
The algorithm scan convert P first. ∵ it is on back of Q.



Rotate Q clockwise until it begin to obscure P, do not intersect P & Q. Since, P & Q have overlapping z-extents, so they must be compared.
All the test 1, 2, 3, 4, 5, 3' & 4' fails.
∴ the polygon will be split, even though P can be scan converted before Q.

Disadvantage of Depth-Sort Algo

1. Polygons have to be sort first.
2. we need to split polygons to solve cyclic & intersecting objects.