Computer Aided Design

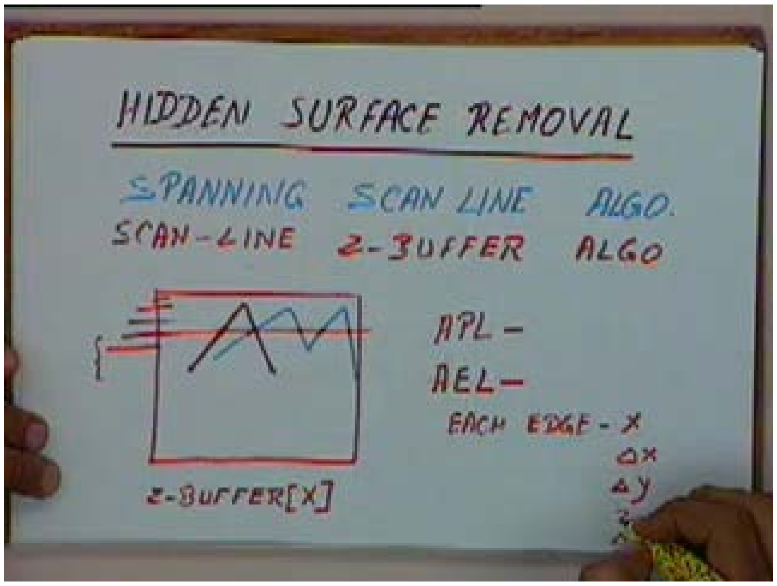
Dr. Anoop Chawla
Department of Mechanical Engineering
Indian Institute of Technology, Delhi

Lecture No. # 13

Hidden Surface Removal (Contd.)

In the last class we were talking of the spanning scan line algorithm sorry we were talking of the scan-line Z-buffer algorithm.

(Refer Slide Time: 00:01:13 min)

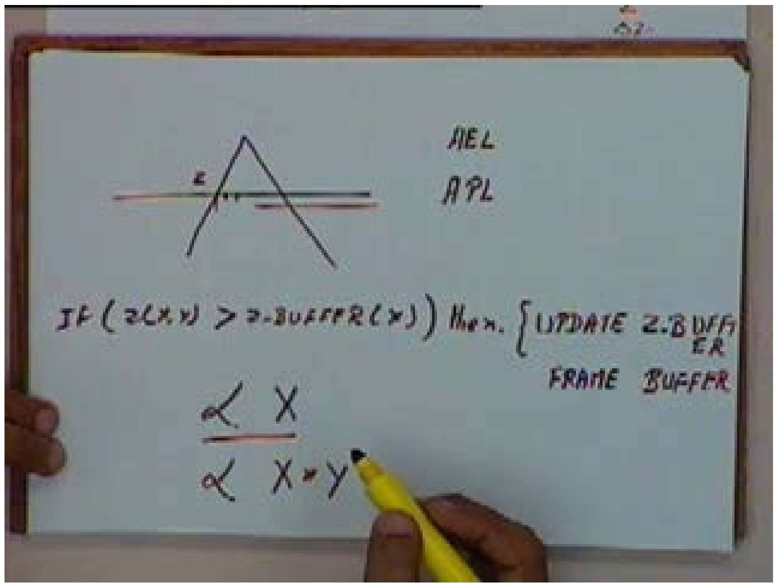


Today I will briefly repeat this scanning sorry I will briefly repeat the scan-line Z-buffer algorithm and then we will go onto the spanning scan-line algorithm. In a scan-line Z-buffer algorithm we had seen that if this is a screen, then we will cover the screen row by row. We first take the first row of pixels and the next row of pixels and so on and then at each row of pixels, we will consider all the polygons that are intersecting that row of pixels. So if we have a set of polygons like this then at each row let's say at this row, we will consider all the polygons that are active and within each row of pixels we will see at each pixel what is the closet polygon to the eye and for deciding the closet polygon to the eye at each pixel we are maintaining a Z-buffer.

And the size of the Z-buffer we said would be proportional to the x direction resolution of the screen. That means so each pixel will be maintaining the current closest polygon to the eye. So this way we will first scan convert one polygon then the next polygon and so on. And for this we said we need to maintain an active polygon list and for each polygon we will be maintaining an active edge list. This active edge list, this terminology is from the earlier algorithm that we had for polygon filling that is scan conversion of polygons. In this active edge list if you remember earlier we were keeping track of the x intersections, for each edge we are keeping track of the x intersection, we are keeping track of delta x that is the shift in x intersection as we move from one scan-line to the next scan-line.

And we were keeping track of delta y that was the remaining number of scan-lines left for that edge. That means let's say if I am considering this edge and this edge ends over here then delta y correspond to the number of scan-lines in this portion. And every time we are going from one scan-line to the next scan-line, we were decreasing delta y by 1 and the moment delta y became 0 that edge would be deleted from the active edge list. Now in this case in addition to delta x and delta y, we will also maintain a z and a delta z. This z value would be the depth of this edge, the z value for this edge at this intersection point and this delta z will be a change in this depth value as we move in the x direction.

(Refer Slide Time: 00:04:51 min)

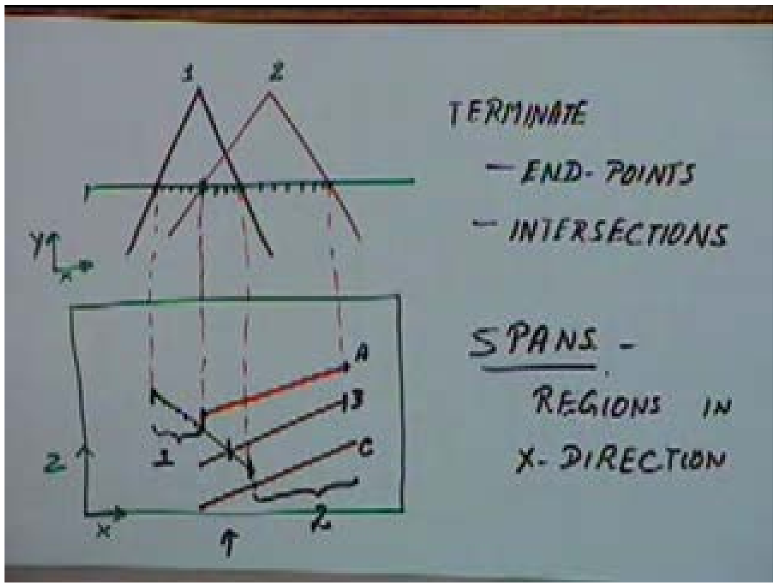


That means if you have a polygon like this and this is the scan-line you are considering, this is the value of z at this intersection, as I go onto the next pixel the depth of this pixel is going to change but since we were talking of a planar surface this change in depth is going to be the same as we move along the x direction. That means delta z in this portion would be the same as the delta z in this portion and so on. Therefore in the active edge list we will maintain a delta z value which will tell us the change in the depth of each pixel as we move in the x direction. And this way we will have an active edge list which will give us, that which will also give us at the z value that is the depth of each pixel in that polygon.

So when we are scan converting this polygon, at this scan-line we can easily compute that z value at each pixel. So z of x y can be very easily computed and if the z of xy is greater than the Z-buffer at x, in that case this particular pixel will be displayed according to this polygon. So, if this condition is satisfied and then we will have to update Z-buffer as well as the frame buffer. In this process of scan converting each polygon will be repeated for all the polygons in the active polygon list, so this is the scan-line Z-buffer algorithm. It is essentially a modification over Z-buffer algorithm, modification to extend that. Now we will be maintaining a Z-buffer for only one line and as we move from one scan-line to the next scan-line, we will use the same Z-buffer for our computations. The amount of space required in the Z-buffer will be proportional to the x direction resolution.

So this is an improvement upon the Z-buffer algorithm to this extent. In the Z-buffer algorithm we are using space which are proportional to x multiplied by y. Here we are using space which is proportional only to x. Student: Sir, in the delta z which we have it's the as you said, as we go along x or y. As we go along x, sir how we will account the (Refer Slide Time: 00:08:05 min) moment as we go along y. Good, you pointed that out. What we will have to do is we will have to maintain a delta z, a delta z in the x direction as well as a delta x in the y direction. The delta z in the y direction will be used as you go from one scan-line to the second scan-line, delta z in the x direction will be used as you go along the x direction, as you go from one pixel to the next pixel in the x direction. We will have to maintain both delta zx and delta zy. Thanks. Any other questions on this algorithm? Now within the same algorithm, let's see a small situation.

(Refer Slide Time: 00:09:03 min)



Let's say we have one polygon like this and we have a second polygon which looks let's say something like this and we are considering a scan-line at this location. In this algorithm what we are doing is we are starting from the first pixel on the screen, when we come to this intersection this is the first edge you have in the active edge list. From this intersection onwards up to this intersection, we are finding out the value of the z and then if this is my polygon number 1 and this is my polygon number 2, my Z-buffer in this region will contain details of polygon one, will contain depth according to the polygon number one. But for computing the depth, we are computing the depth at each of these pixels. Then I will start with this intersection and compute the depth at this pixel and so on.

So effectively what we are doing is I am computing a depth at each of these pixels but in this region let's say from here up to this point computing the depth at each pixel is absolutely redundant because in this region, we have only one polygon. So even if I am computing the depth, the same polygon is going to be displayed. Even if I am comparing every time with the Z-buffer that's unnecessary because the same polygon is going to be displayed. Similarly in this region I need not do all the computation purely because I know that the same polygon is going to be displayed.

In this region where I have a two polygon, again here also I will be doing a lot of wasteful computation because since we were talking of planar surfaces, this polygon and this polygon can intersect in only one point in this plane. If I draw this plane like this, mind you on this screen I am drawing the x and the y planes and here I am drawing the plane corresponding to this scan-line that means if you are looking in the screen like this, I am drawing a plane perpendicular to the screen now. The plane perpendicular to the screen would be the xz plane.

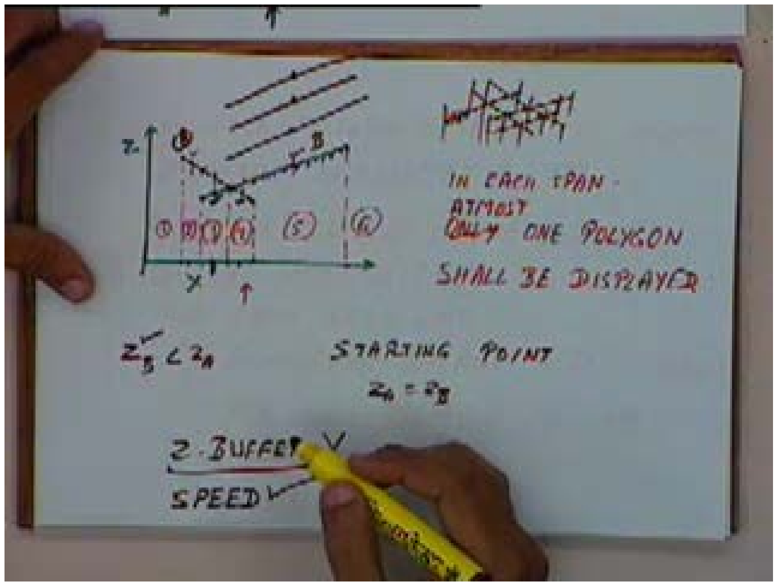
Now in this xz plane the intersection of a horizontal plane, this xz is a horizontal plane, the intersection of a horizontal xz plane with any of the polygons can only give me a straight line, two polygons can only intersect in a straight line. So if I consider this polygon number one, may be it will intersect something like this and polygon number two it can intersect either like this or maybe it can intersect like this or it can intersect like this. We will consider all the three cases. In the first region that is in this region, we have only one polygon as we can see here also we will have only one polygon. So we know that only this polygon has to be displayed.

In this region if this is the intersection that is case number A, this is case B and this is case C. In case A, this polygon is behind this polygon. So I am going to display this polygon because this polygon drawn in green is in front, polygon number one is in front, I am looking at it from this side right now. In case number B from this point to this point, polygon number two will be displayed and from this point to this point polygon number one will be displayed and in case C a polygon two is going to be displayed because that is closer to that eye that is in this region and then if I consider this region in all these three cases, I have only one polygon and that is polygon number two. So here I am going to have polygon one, here I am going to have polygon two and in this region I will have either polygon one or polygon two or a combination of the two depending on what type of line of intersection we have but nevertheless there is no need to compute and compare the Z-buffer at each of the pixels. I only need to decide that in this region only one polygon will be displayed and I will display that polygon, I need not compute the z value and compare that with the Z-buffer at each of these pixels that is unnecessary. So we will again modify the scan-line Z-buffer algorithm to take care of this.

What we will do is we will divide this screen into a number of spans where each span or the spans would be defined to go from one end point of a line. I am talking of these intersections, in these intersections the spans will be defined to go from one end point to another end point or to this is an end point, this is also an end point. So wherever we have end points, our spans will terminate there. I have an end point here, I have a end point here and I have an end point here, I also have an end point here plus wherever I have intersections, wherever I have any two lines intersecting, I will have another span ending over there. So in this case we will have a span ending here, we will have a span ending here, we will have a span ending here, here and here.

Now span is essentially dividing this x axis in to a number of regions. The spans are basically regions in x direction and spans they terminate either at end points, line end points or at intersections. So if we have spans terminating at end points or at intersections, we can have spans of this kind.

(Refer Slide Time: 00:17:30 min)



If we have two lines two polygons intersecting the xz plane, at these two lines my spans would be defined as starting from x equal to 0 going up to this point, then from here going up to here then here, here and beyond that. So this is the our span number 1, this will be span number 2, this will be span number 3, this would be span number 4 and this would be span number 5, this way and this would be span number 6. This is how the whole screen or the complete scan-line along the x axis will be divided into a number of spans and the important property would be that in each span, only one polygon will be displayed and there will be more span in which more than one polygons can be displayed. In span number one nothing is being displayed and if you say only one I should say at most one, in span number two I will have this polygon being displayed. In span number 3 this polygon will be displayed, in span number 4 this one will be displayed, in span 5 this will be displayed and in span 6 nothing will be displayed.

Again I am looking at the, this is my direction of going but only thing that remains is how do I decide as to which polygon is to be displayed in a particular scan in a particular span. When I have only one polygon in the span that is very easy to decide which polygon will be displayed but when I have more than one polygon let's say in this case we have two polygons. I basically got to decide as to which polygon is closer to the eye, I can either look at the z value at the end of the spans, either at the starting end or at the last end. If I look at the z value at the starting end, at the starting point of the span, in polygon in span one there is no confusion, in span two there is no confusion. In span three I look at the z value here and the z value here and I will decide that this is the polygon to be displayed but in span four there will be a confusion because the z value of the starting point would be the same for this polygon as well as for this polygon.

Similarly if I look at the end point, there will be confusion in span number three because at the end point, the two polygons will have the same z value. So to avoid this confusion what we will do is we compute the z value of all the polygons at the midpoint of the span that means we will look at the midpoint of span number two, midpoint of span number three, midpoint of span number four and midpoint of span number 5. If I am looking at the midpoint of spans, I can easily decide which polygon will be visible and which will not be visible. If I am looking at span three at the midpoint I can easily decide this is the polygon to be displayed. In span four also if I look at the midpoint, I can decide that this is the span to be displayed. So to decide which polygon is closer to the eye in a particular span, we look at the midpoints of the spans and compute the z value of the polygons at the midpoint.

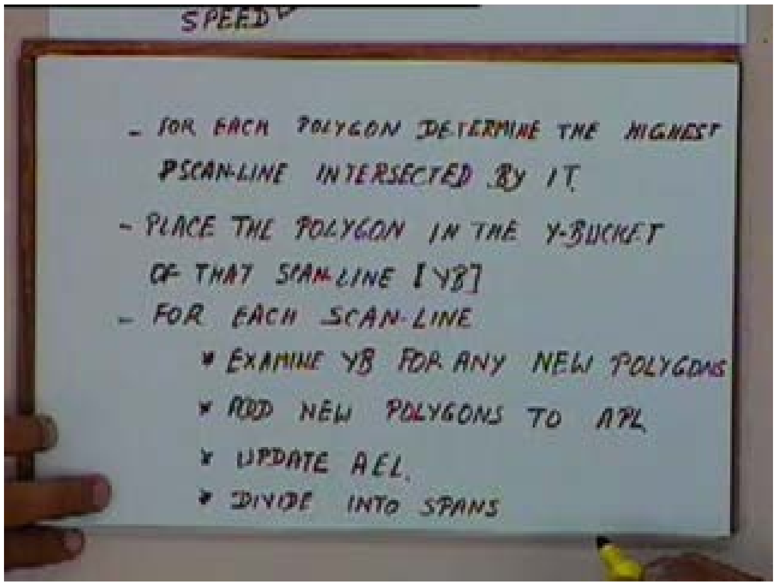
Now this is polygon one, let's say polygon A and this is polygon B. I compute the z value in span three for polygon A at the midpoint at this point. Similarly for polygon B I compute this z value at this midpoint and we will find that z_B is less than z_A. Therefore we will display z_B. Sir is there any necessity, it is in for computing at the midpoint. See the only thing is if you compute at any of the two end points there is a possibility of completion. Then you will have to also look at the other end point and then decide, so just to avoid confusion may be you can look at the midpoint. Let's say you are looking at the starting point. If you are looking at the starting point think of span number four, in span number four the z value for polygon A will be the same as the z value for the polygon B at the starting point because that is the point of intersection. Therefore what we will get here will be z_A will be equal to z_B. So how do you decide which is closer? You cannot. If you are looking at the end point then the same confusion will be there for span three. Just to cut down this confusion, you can uniformly decide that you look at the midpoint. Any other point on this algorithm?

In this algorithm if you notice we are not using any Z-buffer, no Z-buffer will be used because we have divided the width of the screen into spans and at each span, we are deciding which is the closest polygon. So we don't need the Z-buffer plus, we are not scan converting every pixel of every polygon. So speed will also be better for this algorithm. In the previous algorithm we were taking each of the polygons, scan converting each and every pixel for all the polygons and then deciding which is closer to the eye and which is further off. In this case we are not doing that, so speed of the algorithm will be better and we are not choosing this Z-buffer, so the memory requirement will also be low. How do we get an intersection point? We know the equation of this line, we know the equation of this line, we we can find out the intersection.

Essentially if you have number of polygons, we will have a number of lines like this. You will have to find out all possible intersections and then divide the screen into spans like this. Why we have to come a same thing using a z buffer by seeing the, at the polygon list at each pixels. No, at each pixel. Then you are doing at each. Yes sir. Then you will, at each pixel will compute the z value for all the polygons. In this case I am not doing that in this case, let's say you have one polygon here another polygon behind it and another polygon you have set a polygons like this. In span number five you will compare the z value at the midpoints of all these polygons and decide that this is the polygon which is the closest and then you will compute the z value for only these pixels, only these points. You won't compute the z values for each of these polygons. For these polygons we will compute the z value only once. What you mean we have to divide it and span it anyway? If you are dividing it (Refer Slide Time: 00:27:11 min)) then you don't need that buffer at all. What you are saying is to compare the z value at the midpoints, you will maintain the Z-buffer. You will need that you only need to maintain one variable let's say which is the closest polygon. You don't need to maintain complete Z-buffer just because you want to decide which polygon is close at one particular pixel.

Any other point in this? That will happen in every algorithm, even in your scan-line Z-buffer as the number of polygons increase, you will have to scan them at each and every polygon. Sir and that's of order n, this is of a high level. Why. Sir therefore the. Calculating the number of point intersection points. Calculating the number of intersection points that can be speed up considerably by again dividing the whole thing in the span and deciding which are the, which polygons are active in a particular span, we compute the intersection only for that. Finding the intersection can be done much more efficiently. Intersection lines. Yes sir. What you are saying is this intersection point forms the line that will form a line that can be done. You can maintain that intersection line and as you move from one scan plane to a second scan plane, you can change that accordingly that can also be done. Intersection can be completed much more easily than from doing a complete scan conversion. That is why it is better to do more intersections than to spend all the time in doing the scan conversion. Any other question on this? In that case I will just write down this algorithm in a formal manner so that if there is any confusion we can take care of that.

(Refer Slide Time: 00:29:43 min)

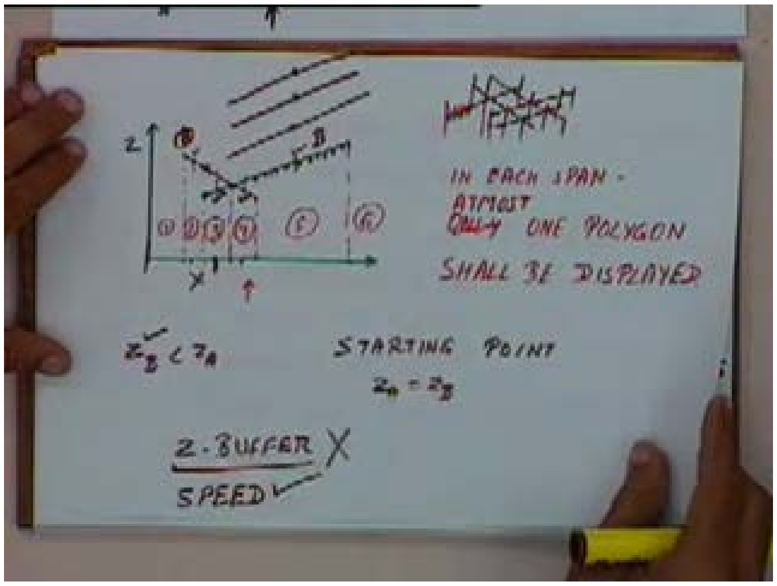


SPEED

- FOR EACH POLYGON DETERMINE THE HIGHEST SCAN-LINE INTERSECTED BY IT
- PLACE THE POLYGON IN THE Y-BUCKET OF THAT SCAN-LINE [YB]
- FOR EACH SCAN-LINE
    * EXAMINE YB FOR ANY NEW POLYGONS
    * ADD NEW POLYGONS TO APL
    * UPDATE AEL
    * DIVIDE INTO SPANS

The first step for each polygon we will determine the highest scan-line intersected by it and then we will place the polygon in the Y bucket of that scan-line. If you remember we have done a similar thing when we were doing polygon filling. There we had taken every edge and placed that in a Y bucket. Here we will take every polygon and place that in a Y bucket so that we will know when a polygon is starting and then we can compute when a polygon is going to end then for each scan-line let's say if I call this an array Y bucket YB, for each scan-line we will examine the Y bucket for any new polygon. We will add new polygons to the active polygon list then we will say update active edge list, we will have to update the active edge list and then divide into spans. Once I have updated the active edge list, I can easily find out the set of lines in my xz plane.

(Refer Slide Time: 00:33:06 min)



IN EACH SPAN - ATMOST ~~ONLY~~ ONE POLYGON SHALL BE DISPLAYED

$Z_B < Z_A$

STARTING POINT

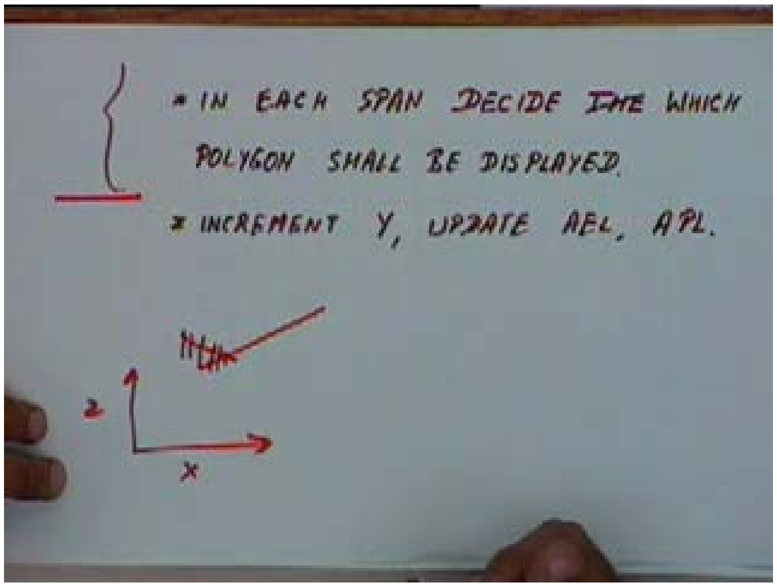
$Z_A = Z_B$

Z-BUFFER X

SPEED ✓

Each of these lines like this can be obtained from the active edge list. This point is going to be on the active edge list, this point is also going to be on the active edge list. Similarly this point will be on the active edge list and this point will also be on the active edge list. So from the active edge list I can divide my region into spans and once I have divided the region into spans then in each span we will decide the closest polygon or decide which is the closest polygon and then we will say increment Y and go onto next scan line and again update active edge list and the active polygon list.

(Refer Slide Time: 00:33:44 min)

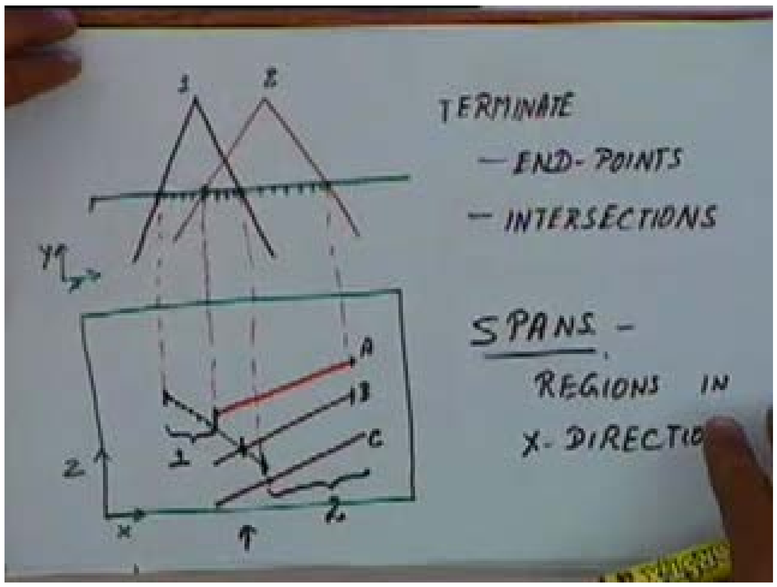


* IN EACH SPAN DECIDE ~~THE~~ WHICH POLYGON SHALL BE DISPLAYED.
* INCREMENT Y, UPDATE AEL, APL.

Some edges would have come to an end, we will have to delete those edges, some polygons would have come to an end, we will have to delete those polygons. This will be a sequence of depth involved in this. If you notice between this algorithm and the previous algorithm that is the scan-line Z-buffer, the portion that is actually a different is this portion. In the previous algorithm instead of dividing in two spans, we were scan converting every polygon. The rest of it was model of the same, even there we have to maintain an active edge list and an active polygon list, only this portion was different in that algorithm. Here we are dividing it into spans and thereby getting it exactly.

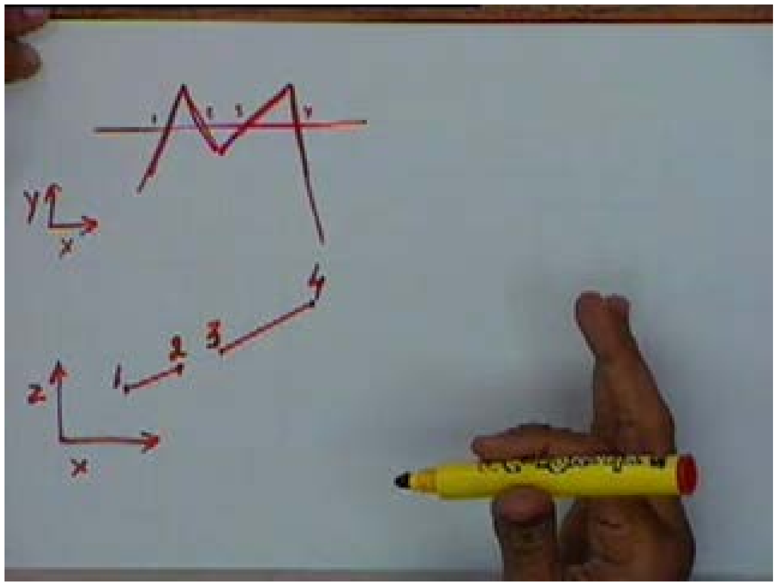
Intersection we found mathematically not by comparing z values at each point. Why? This is your x direction, this is your z direction, I have a line here and I have a line here. I know the equation of this line, I know the equation of this line, I can find out the intersection. I know how to find the, you know how to find intersection between two lines. I don't have to compare the z value here, here, here, here and keep comparing and then decide. That is not the way I will find the intersection. Intersection we found just by mathematically writing down the equation of the line. Any other question on this? What is the active edge list? In active edge list, see this is my xz plane. My active edge list will tell me that for every polygon, if I am looking at it from front that is in the xy plane, so you remember this figure.

(Refer Slide Time: 00:37:16 min)



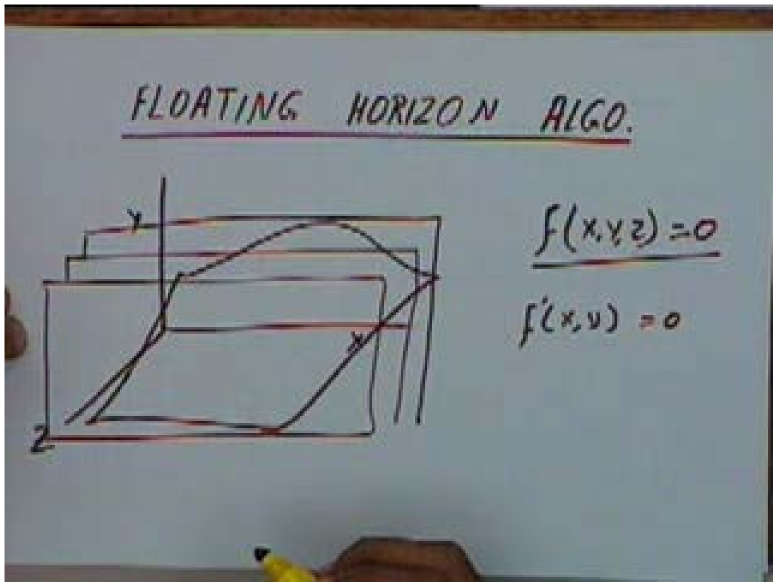
This edge is active, this edge is active on this scan-line. Similarly this edge is active and this edge is active. So the active edge list will give me the intersection of this edge, the intersection of this edge, intersection of this edge and the intersection of this edge with the xz plane. So if I am looking at the xz plane now the active edge list will give me a point here and a point here plus a point here and a point here. In the xz plane, the active edge list will only give me points. Once I know these points, I know the equation of the line between them.

(Refer Slide Time: 00:38:12 min)



If I have a let’s say, if I have polygon like this and a scan-line like this, now this I am talking in the xy plane. My active edge list will contain this edge, this edge, this edge and this edge. If I look at the same thing in the xz plane maybe I will get lines like this 1 2 3 4, this point is point 1, this point is 2, this is 3 and this is 4. So the active edge list will give me these end points from that I will get the equation of this line and then use that to divide it into spans. Any other question on this? In that case we will go into another algorithm and that is called the floating horizon algorithm.

(Refer Slide Time: 00:39:35 min)



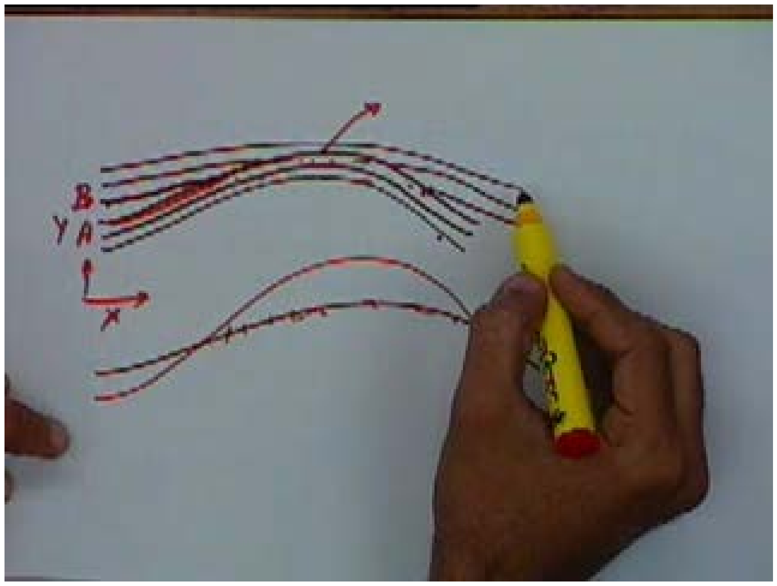
FLOATING HORIZON ALGO.

$f(x, y, z) = 0$

$f'(x, y) = 0$

Now this floating horizon algorithm will be used for displaying surfaces. If I have a surface something like this where the surface can be given mathematically by equation of the type F of xyz is equal to 0. If my surface is given in this form, in that case I can use this floating horizon algorithm. Essentially what we will do is we will take vertical planes parallel to the xy plane. I will take one vertical plane like this then I will take another vertical plane which will be just behind it and then another one just behind that. We will take a set of vertical planes and compute it, compute the intersections of these planes with the surface that we have. So if I take the intersection with the z equal to constant plane, I will get a curve. This is one surface and I have a vertical plane. The intersection of this surface with this plane is going to give me a curve that curve will be of the form let's say F of xy or a prime of xy will be 0. This way we will get a set of successive curves one behind another.

(Refer Slide Time: 00:41:52 min)



Let's say with the first plane if I get a curve like this with the plane just behind it, I get another curve which looks something like this, with the plane behind that I get another curve which looks something like this. I keep displaying each of these curves but if I get a curve which looks something like this in the next plane then I will display only this portion of the curve. So the next curve that will look something like this. Again if I look at the next curve, it might look like this and so on. If I start with the back planes then I will have lots of problems, look at this. If I start from the back plane let's say I have drawn this curve.

Now I am taking a plane which is ahead of it, so some part of it is going to be obscured. Let's say the next curve looks something like this and this curve in red is in a plane closer to the eye. So this portion of the curve is not going to be displayed now, I will have to delete that. So a portion, a figure which has already been drawn I have to go and rub this out, I don't want to do that. I am first looking at the plane which is closest to the eye. Once I have drawn that, I know for sure that is not going to be rubbed ever because all the other planes are behind that. That is why I first consider the plane which is closest to the eye that is this plane.

Then I will consider a plane which is just behind it and then I will consider the next plane which will be behind it like this. So one by one we will consider all the planes and we will keep drawing these curves and along the length of the curve, we keep track of the fact that portion of the curve might be behind portions that have already been displayed and when would this portion, behind a curve which has already been displayed that would be when the y value for this curve, mind you this was in the xy plane now. When the y value for this curve becomes less than the y value for a curve which has already been displayed. This curve, this one have already been displayed.

Now I am trying to display this curve. At this point the y value for this curve, this is let's say curve number two or curve number B and this is curve number A. The moment the y value for the B curve becomes less than the y value for the A curve that means this B curve will be hidden. So we will make use of that fact and then we will display each of these curves one by one keeping track of the y values. I have to store the y values of all the curves. Yes, not all the curves, the current highest. The maximum y. The maximum y value of at the current location. Suppose it is like a wave. Doesn't matter. Then different curves will have different maximum y values, so we store the maximum values of. No, at each x pixel you have to store the maximum y value of that point. Of any curve. Means whatever the curve with the maximum y value at that point, we will see this algorithm more detail in the next class because the detail will take some time, we will see the details in the class. How do you decide the increment of the plane? That is by a decision which has to be taken on the basis of the nature of the curve that would be random, so that's all for today. We will complete this algorithm in the next class.