Computer Graphics
Prof. Sukhendu Das
Dept. of Computer Science and Engineering
Indian Institute of Technology, Madras
Lecture - 19
Scan Conversion of a Polygon (Contd…)

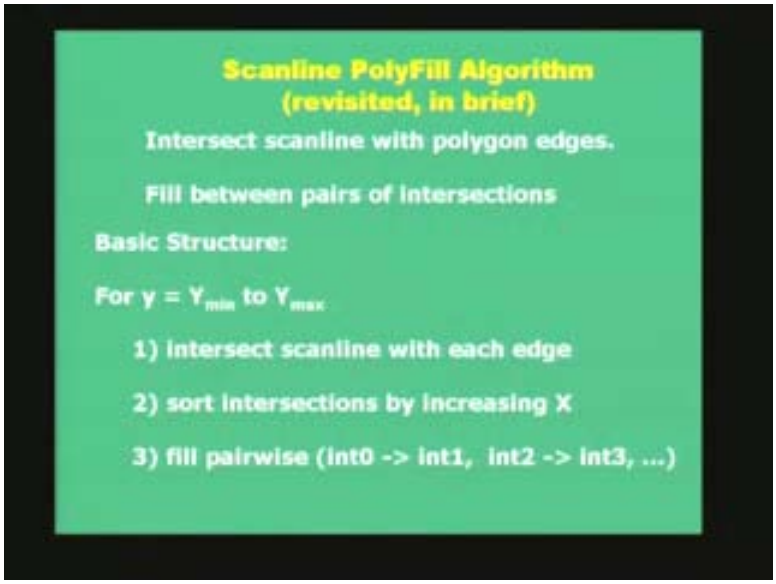
Hello and welcome everybody to the lecture series on computer graphics. In the last class we have discussed about scanline polygon filling algorithms or scanline based polyfill algorithm. And we have discussed the basic equations, the concept about the polygon filling algorithm to test whether a point is within a polygon. And then basically find out intersection of a scanline with the edges of the polygon and form pair wise intersections and fillings between. So that is the basic logic.

In addition to this we also discovered how to handle intersections at the vertices of the polygon. If you remember that we do expect that we have even number of intersections of a scanline with the edges of a polygon. So let there be a set of edges which enclose a polygonal or an area and any intersection line typically is expected to have even number of intersections with the polygon edges except when a polygon passes through a vertex we can have odd number of intersections. And in certain cases we increase the number of intersections by one more or reduce it by one.

We have seen those special cases where we basically look whether the adjacent edges of the polygon at the vertex are on the same side of the scanline or it is on either side or on both sides of the scanline. And so based on that we reduce the length of the corresponding edge of the polygon and hence that helps us to reduce the number of intersections and make from odd to even.

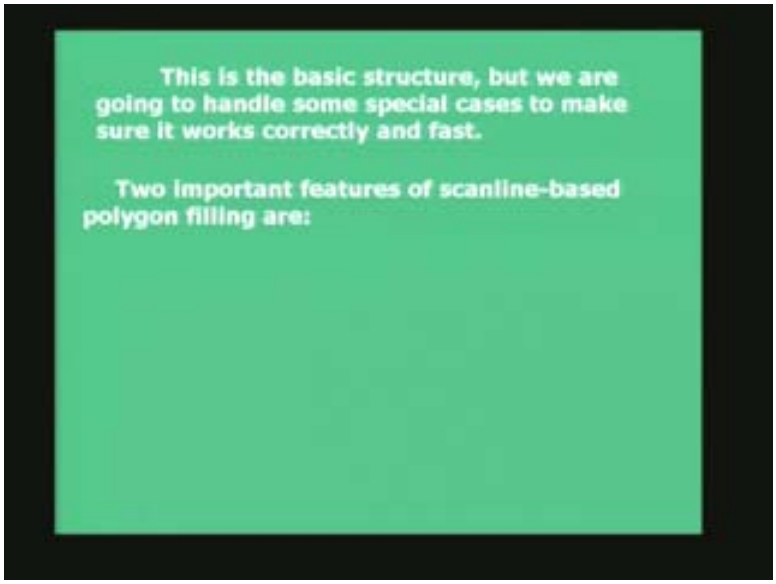
So we now know that the vertices need special cases of handling of the polygon. And if certain precaution is taken in the algorithm to handle such special cases then it means we are guaranteed to always have even number of intersections of a scanline with the edges of the polygon.

(Refer Slide Time: 00:03:22)



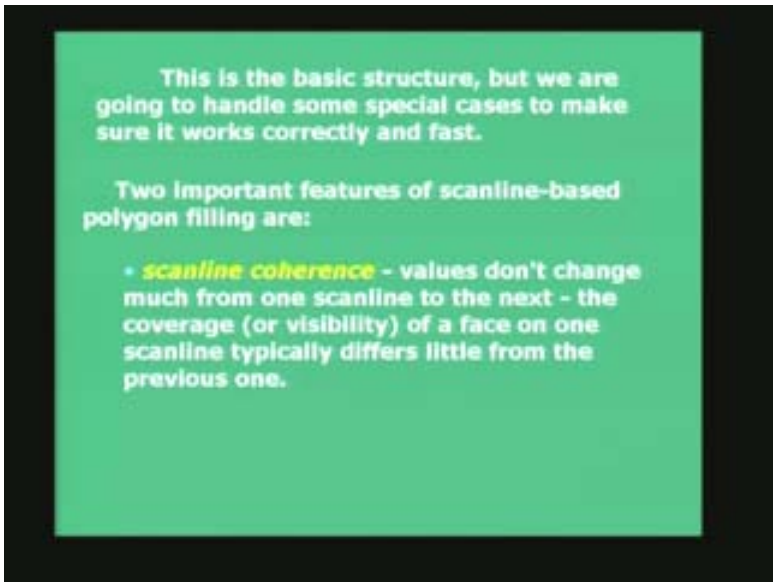
So, if we look back into the last slide which we talked about in the last class, this is the scanline polyfill algorithm revisited again in brief where we intersect a scanline with the polygon edges and assuming that the fact that we have taken care of the vertices of the polygon that is intersection points of the scanline with vertices. Then you find out the intercepts, fill pair of intersections in between and the basic structure is a loop where we run from scanline Y_{min} to Y_{max}, the first step within that loop is intersect scanline with each edge, sort intersections with increasing X and since you are guaranteed to have even number of intersections, you fill pair wise in between from intersection number 0 to intersection number 1, do not fill between intersection 1 to 2, fill again from intersection 2 to 3. There are arrows that indicate you are filling pixels in between intersection 0 on to the intersection 1 in the particular scanline, then intersection 2 to intersection number 3 and so on. This is how you fill pair wise intersections in between and leave gaps and even number of intersections you typically have the scanline filled. So assuming this we look into certain other aspects of about scanline filling algorithm.

(Refer Slide Time: 00:04:44)



Well this is the basic structure but we are going to handle some special cases to make sure it works correctly and fast. And to ensure that this works fast and correctly, in fact incorporate certain concepts which are extensions of the Bresenham's scanline drawing algorithm which is integer based to find out how it works and helps us to obtain the algorithm and make it work fast in the case of scanline polygon filling.

(Refer Slide Time: 00:05:20)



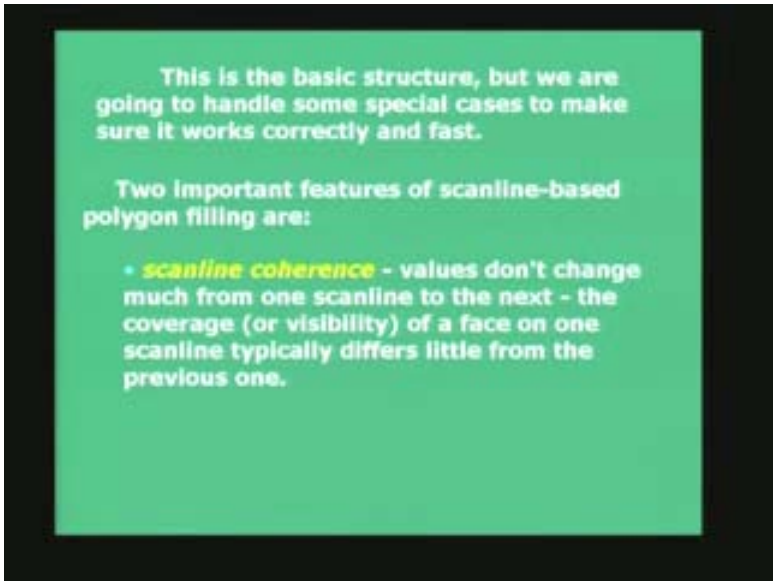
So two important features of scanline polygon filling are; first one you can see on the screen again, one is called scanline coherence. Scanline coherence, we know that or assume that the values typically do not change much from one scanline to the next. What

do you mean by values? It is the value of the intersections of the scanline with the corresponding edges of the polygon. So, if you move from one scanline to the other, next when you are moving scanline from top to bottom or from bottom to top whatever the case may be when you are given or you have already obtained the even number of intersections of a particular scanline with all the edges of the polygon whichever it intercepts and when you are moving to the next scanline on the top or on the bottom the XY coordinates of the values of intersections do not change much. And that is an example of the concept of scanline coherence.

So we read the sentence once again on the screen where it says that the values do not change much from one scanline to the next. And the coverage or visibility of a face on one scanline typically differs very little from the previous one. This is the extension with which we say, that the visibility means basically you fill intersection in pairs and those intersections in pairs is what I will call as the visibility of a polygon for a particular scanline.

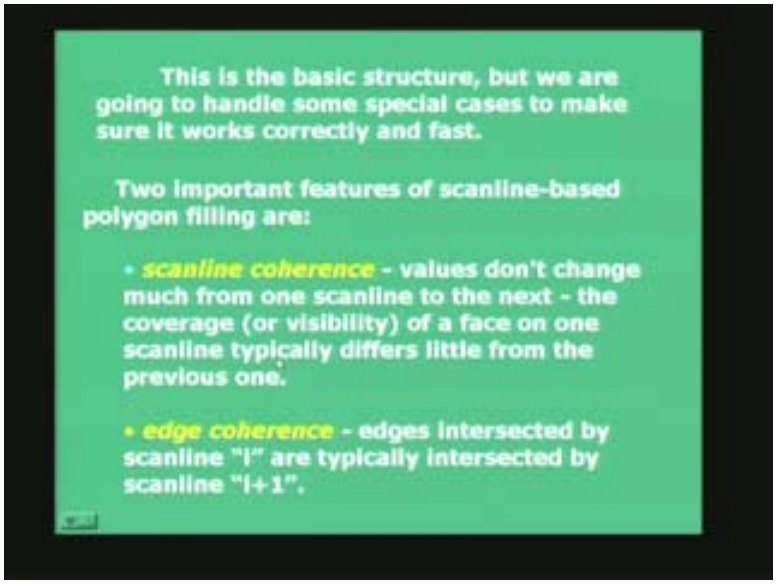
When we move from one scanline to another there is a coherence, the intersection points do not change much in terms of values of coordinates of X and Y. Of course Y increments or decrements by 1 depending upon whether you are going up or going down. But the X value also does not change much; it is typically a concept similar to the Bresenham's line algorithm in terms of pixel drawing from one scanline to the other, from one point to the other. So the visibility or part of the scanline which will be visible inside the polygon will not change much from one scanline to the other.

(Refer Slide Time: 00:07:24)



So this is the meaning of the word scanline coherence where I repeat again, values do not change much from one scanline to the next. The coverage or visibility of a face on one scanline typically differs little from the previous one. Let us look at the next point which we call as the edge coherence.

(Refer Slide Time: 00:07:37)

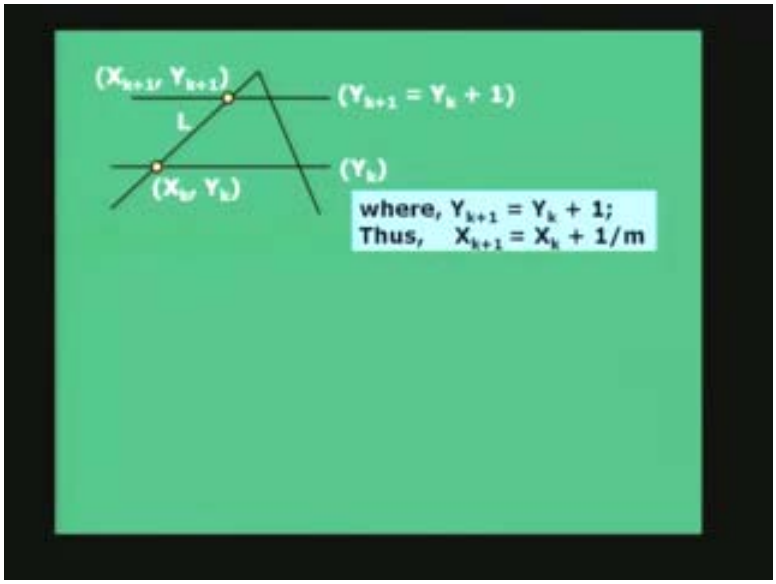


The edge coherence says that the edges intersected by this scanline I, any particular arbitrary scanline I are typically again intersected by scanline $I_{plus\ 1}$. This is also true, if the word edge coherence means that when you move from scanline I to scanline $I_{plus\ 1}$ going on the top or going to the bottom, $I_{plus\ 1}$ or minus 1 does not matter, the number of edges which intersects scanline $I_{plus\ 1}$ with respect to I in general will not change. It will only change if you reach a vertex or getting out of a vertex.

If you get a new vertex or leaving a vertex then, basically when you get a new vertex the number of scanlines could increment or decrement. But those vertices are few and far between in general for a polygon and hence the numbers of edges which intersect adjacent scanlines are almost the same number. That is the edge coherence which we talked about, you know what is scanline coherence now. And we try to see how we can incorporate these concepts to increase the speed of the polygon filling algorithm and make it work very fast. This means the falling.

At any given scanline I, if you know how many number of intersections exist and which are the edges through which the scanline is intersecting and the corresponding coordinates of intersection and if you have known that or computed that by some set of steps within the algorithm then when you move to scanline $I_{plus\ 1}$ or I minus 1, that means in the next adjacent scanline the number of its intersections will almost remain the same, it might change a little bit, again I say when you are probably at the vertex but otherwise the number of intersections in general almost remain the same and sometimes it is exactly the same. That is number one, number of intersections remaining the same and the intersection coordinates also change a very little. These are the concepts of scanline coherence and edge or edge line coherence or edge coherence to be very specific which or will be exploited to make the algorithm very fast in addition to the concepts of Bresenham's line algorithm which would be borrowed and extrapolated and put here.
So let us look into the next slide.

(Refer Slide Time: 00:09:51)



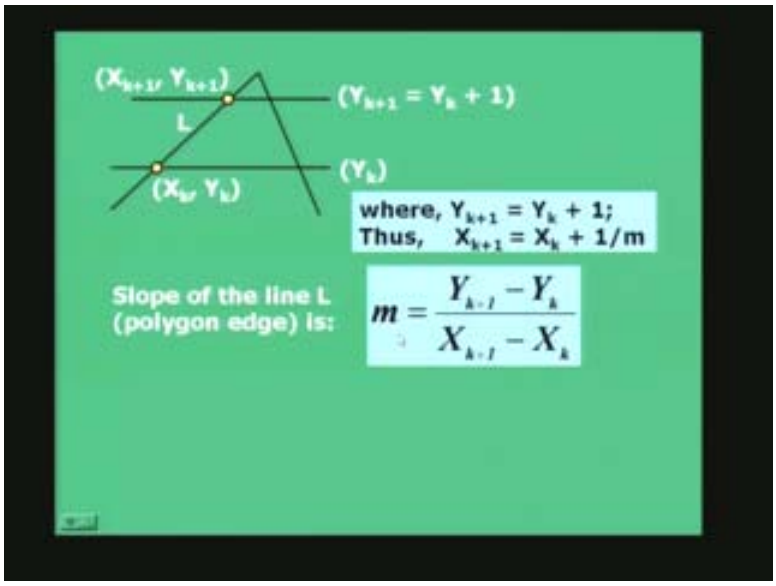
This exploits the concept of the edge coherence and the scanline coherence as well. We are going to discuss this. This basically means that if for a particular scanline at the bottom which is at Y is equal to Y_k, at the kth iteration let us say that this is the bottom line is the scanline with integer coordinates Y is equal to Y_k. And let us say that this particular scanline intersects the left edge of a polygon. I have just shown a part of the polygon here with one vertex and two adjacent edges and out of the two adjacent edges to the left and the right I am looking into the left edge of this polygon which is marked by the line L. So line L is something like a draw line command using Bresenham's is what you can visualize that this is the line L and what were you interested in? If the scanline Y_k intersects this edge of the polygon or the line L on the left hand side at the integer coordinates X_k, Y_k that is the intersection at the line L.

Then at the next scanline $Y_{k \text{ plus } 1}$ which is equal to Y_k incremented by 1 because we are moving from bottom to top integer coordinates of Y. We are going up from bottom to top. If that is the case, it means for the next scanline $Y_{k \text{ plus } 1}$ the next intersection can be easily computed which are given as coordinates $X_{k \text{ plus } 1}$ $Y_{k \text{ plus } 1}$ which can be computed easily. First of all we know that $Y_{k \text{ plus } 1}$ is just Y_k with an increment 1. So that is very straight forward, which is the next Y intersection. What is the next X intersection? Well this is what we exploit similar to the Bresenham's line algorithm. We can substitute this increment and easily obtain that the next X intersection will be increased by 1 by m where m is the slope of the line. Y equal to mX plus c is the equation of the line. You can rewrite it as X equal to m by Y plus c plus m. So if Y increments by 1 you can re substitute in the equation and find out that the difference in the X coordinate will be the slope 1 by m.

So I leave that as a small exercise to you because we have manipulated this equation a lot in not only Bresenham's line algorithm but also a little bit in the previous lecture.

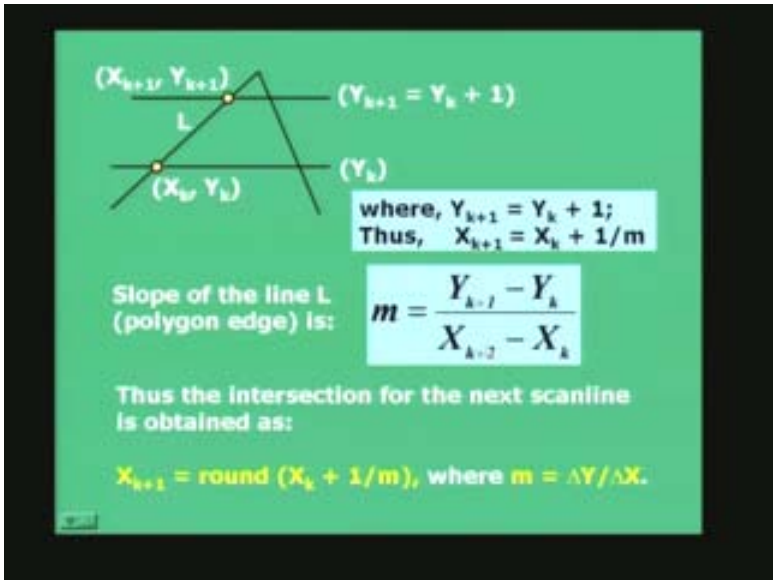
So please do that and you can substitute back because you know your X_k Y_k is the point through which the line passes. The next point will be for the next scanline $Y_{k\ plus\ 1}$ will be the X coordinates which will increase by a fraction 1 by m. That is a constant factor by which the intersection goes up. This is the concept of the scanline coherence where we know that the intersection increments always by 1 by m from one scanline to the other.

(Refer Slide Time: 00:12:52)



Slope of the line L we know for a polygon edge is Y. This is the expression of the slope of the line. We know that given two intersection points $Y_{k\ plus\ 1}$ Y_k and $X_{k\ plus\ 1}$ and X_k this is the slope of the line L. And that can be substituted and obtained as $X_{k\ plus\ 1}$. And the numerator is basically 1 because you know that $Y_{k\ plus\ 1}$ minus Y_k will be 1.

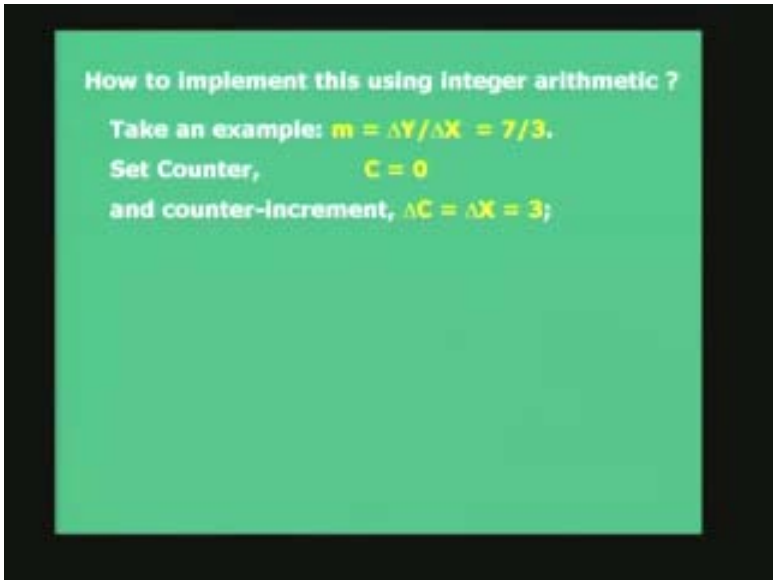
(Refer Slide Time: 00:13:05)



So the numerator is 1 in this equation of m and this helps you to derive that $X_{k \text{ plus } 1}$ is $X_{k \text{ plus } 1}$ by m. So, using the bottom equation you can easily obtain $X_{k \text{ plus } 1}$ that is very simple and straight forward. Thus the intersection for the next scanline can be easily obtained by a round function of $X_{k \text{ plus } 1}$ by m where m is delta Y by delta. This is dY by dX of your Bresenham's. So we know that.

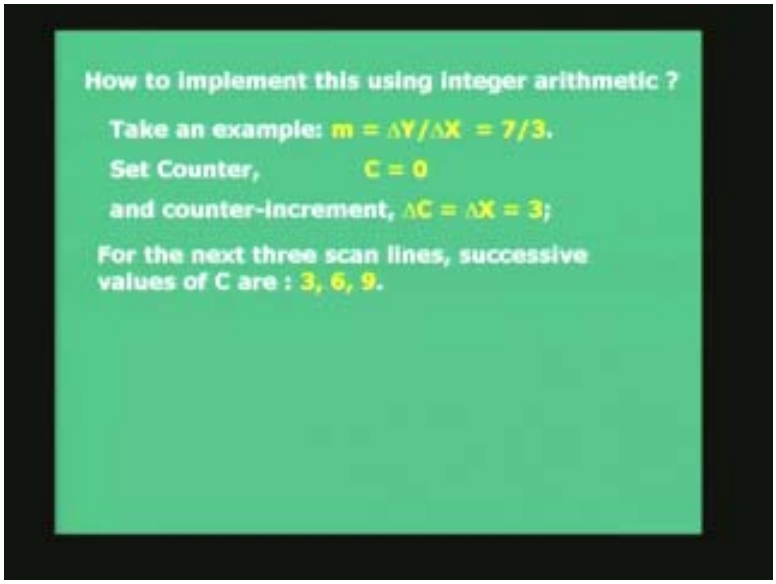
But now again as like Bresenham's scanline algorithm we avoided all floating point calculations, remember that, we also avoided the round function because we wanted to use integer arithmetic to make the algorithm work as fast as possible. So we will see how to avoid floating point calculations and make this calculation easier by avoiding the floating point addition and the long function. Let us take an example, how to implement this using integer arithmetic.

(Refer Slide Time: 00:14:08)



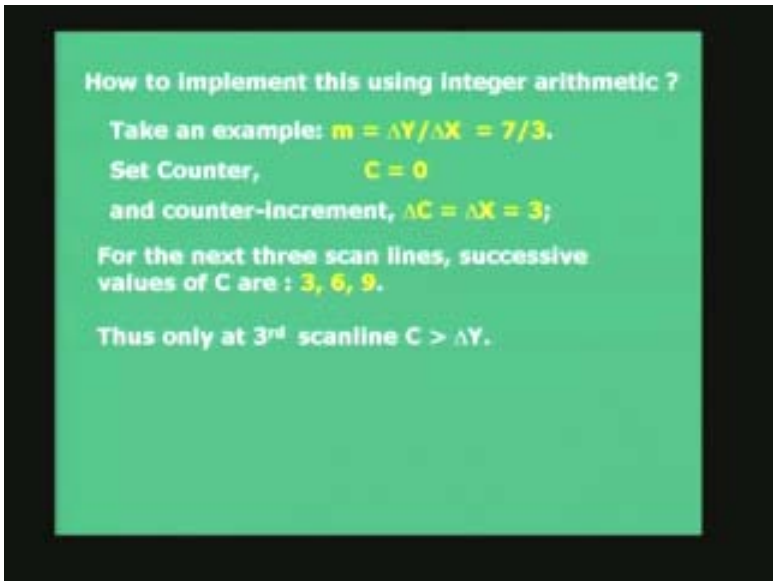
Take an example, the slope of the line del Y by del X which is the difference in the Y coordinates divided by the difference in X coordinate. This equation should not be new to you if you have understood the Bresenham's line drawing algorithm. The m was calculated as dY by dX which is 7 by 3. You initialize a counter variable C is equal to 0. And you can also initialize a counter increment which I will call as delta C and take that or consider that equal to delta X or delta X is 3. So these are the three initialization commands m C and delta C. In fact we will not use m because it is a floating point number. We use the counter which is an integer quantity and it will be also incremented by integers because this del Y and del X is the difference in Y coordinates and the difference in the X coordinates are all integer quantities but you will not divide them. So what do you do?

(Refer Slide Time: 00:15:07)



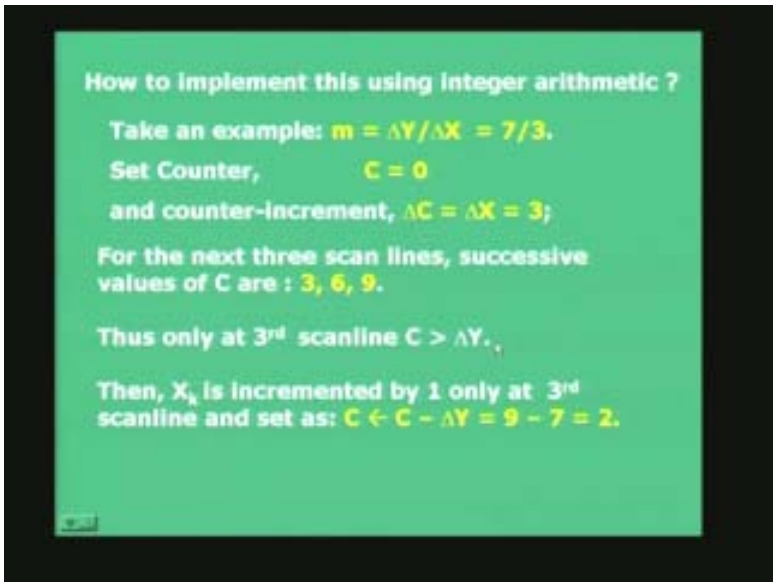
What you do is starting from a scanline say scanline 0 or 1 whatever the case may be, for the next three scanlines successive values of C are incremented by delta C. So, if the counter value is set C is equal to 0 for a particular scanline, say the starting scanline for an edge, then you increment the values of C by delta C. So C is equal to 0, then you have delta C, delta C increments by 3. So from 0 it will reach 3, 6, 9, 12 and so on. This is the next set of three scanlines, the successive values of C are given as 3, 6 and 9 respectively. Where do you stop? You will stop somewhere.

(Refer Slide Time: 00:15:49)



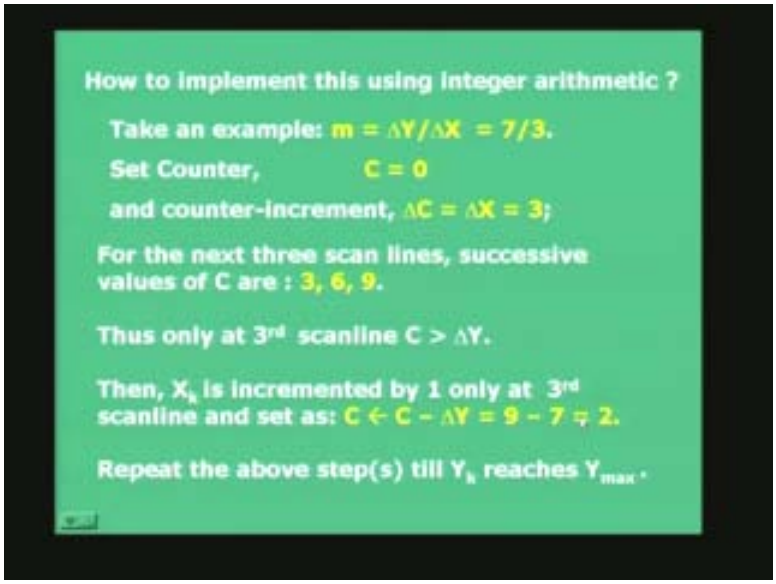
You find that you keep on incrementing the counter value by the counter increment. That means increment C from 0, 3, 6, 9 and keep observing when the value of C crosses del Y that is the value 7. You will find that the value of counter is 0 which is less than del Y the first scanline then it reaches 3 then it is 6 which is also less than delta Y. So, only in the third scanline when the value of C reaches 9 the counter value will be more than del Y. And at that particular point you increment X_k or the X coordinate of the intersection by one only when at the third scanline this counter value crosses del Y and at that point you reset the counter value or decrement it by the value delta Y.

(Refer Slide Time: 00:16:20)



You are actually handling floating point numbers by integer manipulations only. And I actually divide all of these by a denominator and see that you are basically rounding off at some point. So I leave it as an exercise for you to verify that. So you minimize C by del Y so when it reaches 9 at that point you increment X and reduce C by the counter value by del Y so 9 minus 7 will be 2. This is how you keep proceeding, repeat the above steps till Y_k reaches Y_{max} because you are moving from scanline Y to Y_{min} to Y_{max} or for a particular edge you are starting from the vertex of a particular edge Y_{min} to the maximum Y value for the particular edge.

(Refer Slide Time: 00:17:07)



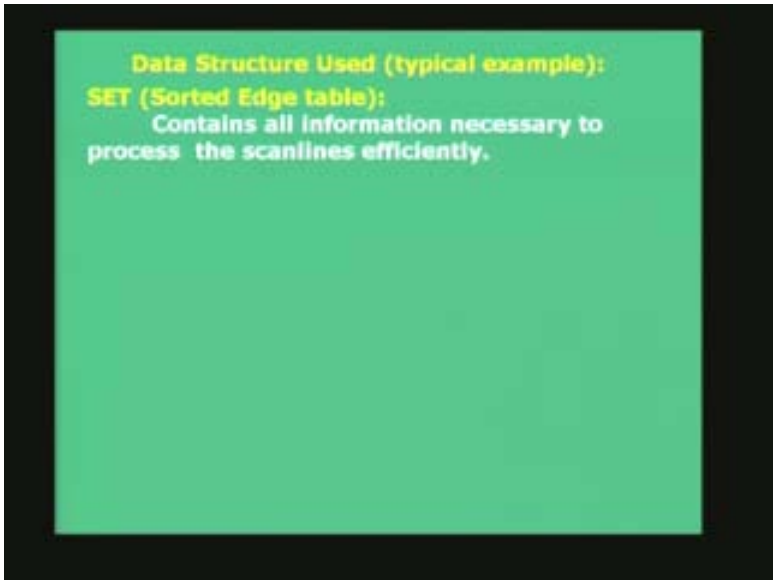
So Y_{min} to Y_{max} means one vertex of an edge to another vertex of the edge and you start with C equal to 0 keep watching when the C reaches del Y or crosses that, reduce it by del Y again keep on incrementing again you keep on doing that and each point when it crosses when it decrements you increase the X value by 1, that is how you do and you stop when you have reached Y_{min} to Y_{max}. This is how you use integer based algorithm to obtain or to exploit and disexploit the scanline coherence as well. To compute the next intersection from the previous one and it does it very fast because it is based on integer algorithm.

You just have to add couple of integers, one integer addition and one integer division as there is no multiplication, subtraction as well, as was the case for Bresenham's. Of course that involve multiplication by 2 which also can be done could have been done faster by bit wise manipulation in the standard C program. And this is how you exploit, coming back to the scanline polyfill algorithm this how you implement this scanline coherence to compute the next intersection from the previous one as you go from one scanline to the other. So I hope you will re-look at these steps again.

I repeat, set counter C is equal to 0 set counter increment equal to del X keep on incrementing from one scanline by delta C whenever you find here in this particular example you find the C value at the third scanline is 9 which crossed the del Y which is 7, what you do is increment X by 1 you reduce the counter value by delta Y and then make it 2 and keep on increasing. So what will happen in the next scanline after the third one? From two you increase that is from 2 plus 3 is 5 and then 5 plus 3 is 8. So when it crosses 8 at the fourth, fifth and sixth one. So 2, 5 and then 8, third, fourth, fifth itself will be 8 again, it will be crossing del Y reduce it by 2. So 8 minus 7 you will reach 1 so keep on doing this cycle till you reach Y max. This is the example. You can easily work it out extensively for certain set of values.

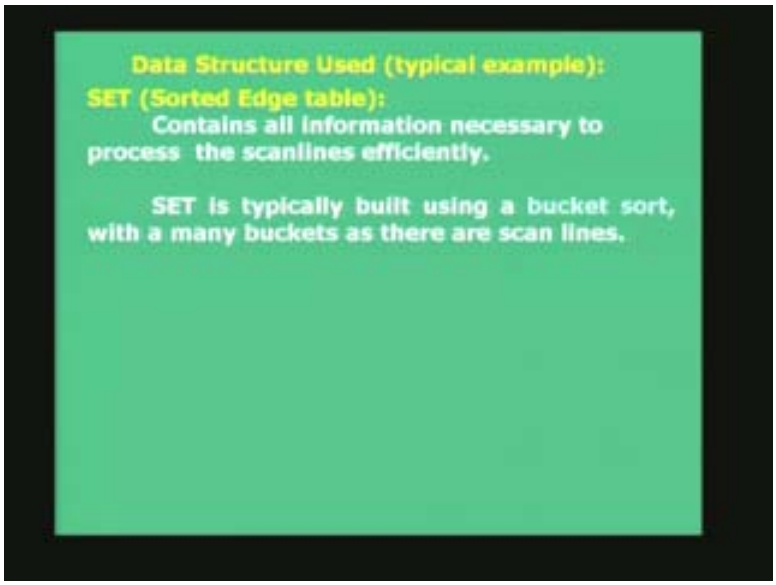
Now coming deep into the algorithm we look at the data structures which may be used. We will follow the textbooks mainly by Hearn and Baker here and see what type of data structures we have been recommended in solving this particular problem. A typical example of a data structure used to solve the problem.

(Refer Slide Time: 00:19:57)



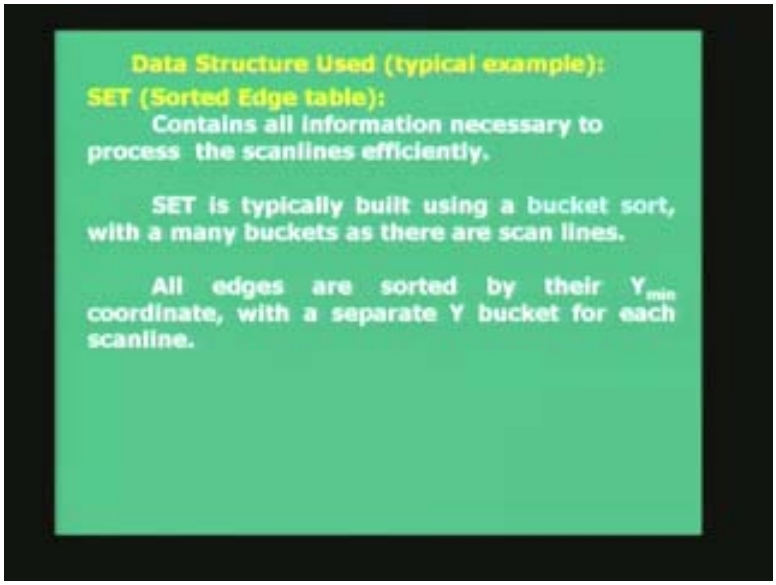
We use what is called as sorted edge table. It contains all information necessary to process the scanline efficiently.

(Refer Slide Time: 00:20:09)



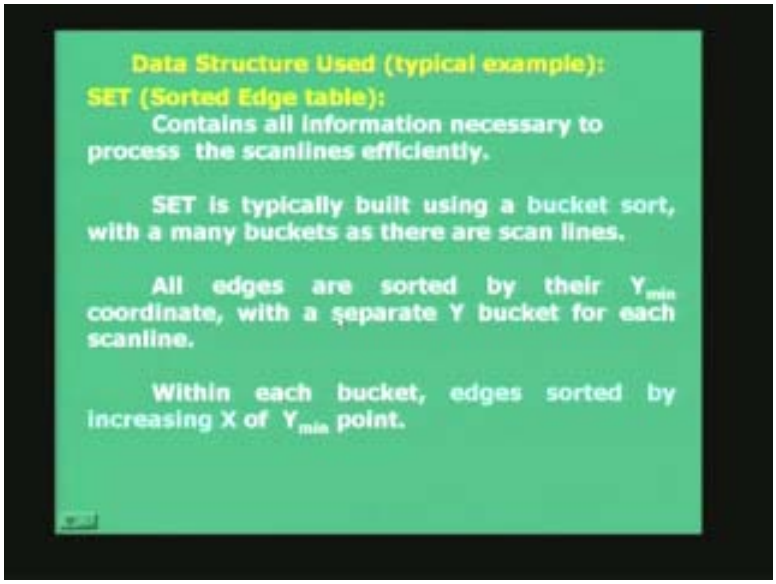
What is a sorted edge table? A SET, we will call this as SET from now on. It is not a typical set, it is a sorted edge table, codings, or ACTs probably we are pronouncing it right, you can use SET. But SET is used. The SET is typically built using a bucket sort algorithm and with as many buckets as there are scanlines. You now the bucket sort is an efficient sorting algorithm when you have integers. We are sorting integers with that too within a certain range. That is, typically the scanline range between Y_{min} to Y_{max} is a very good example where you can use bucket sort.

(Refer Slide Time: 00:20:41)



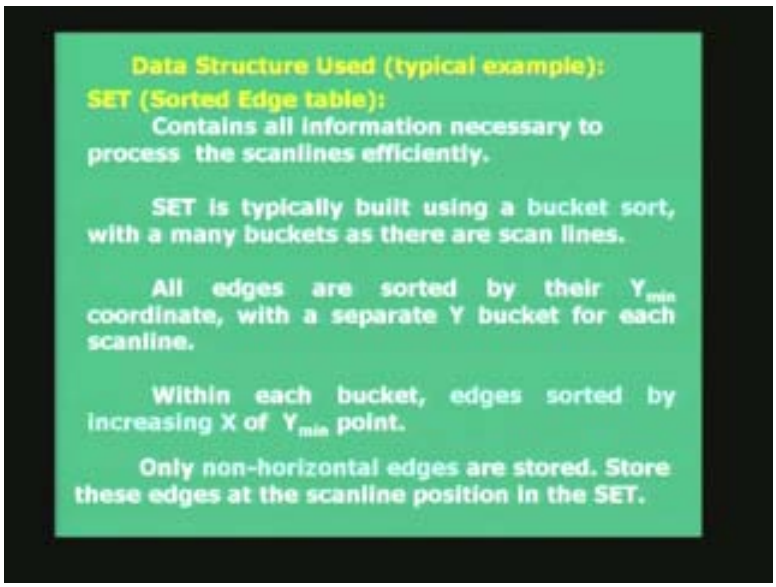
All edges are sorted by their Y_{min} coordinates. What is this Y_{min} coordinates? With a separate Y bucket for each scanline. So you have a bucket for each scanline and all edges are sorted by their minimum Y coordinate. That means each edge you know, a polygon is defined or given as an input to this algorithm by a set of vertices. You have a set of vertices X_1 Y_1 to X_2 Y_2 X_3 Y_3 and so on up to X_n Y_n the last vertex being the first one.
So actually you can form pair wise these set of vertices and each is an edge starting from first to the second and second to the third and so on. So, if this each is an edge, so you take each of these edges and look into their minimum values of Y_1 and Y_2 those pair of Y values and the minimum of them is the Y min. So, for each edge you look into their Y_{min} coordinate. Minimum Y value for each edge is what you look and that is how the edges are sorted based on their Y_{min} coordinate with a separate Y bucket for each scanline.

(Refer Slide Time: 00:21:40)



We will have a look at this SET or the sorted edge table now. Within each bucket edges are sorted by increasing X of Y_{min} point. We will also see what is done here. We talked of each a separate bucket for each scanline, so for each scanline since we have a separate Y bucket within that the edges are sorted by increasing X of the Y_{min} point.

(Refer Slide Time: 00:22:01)

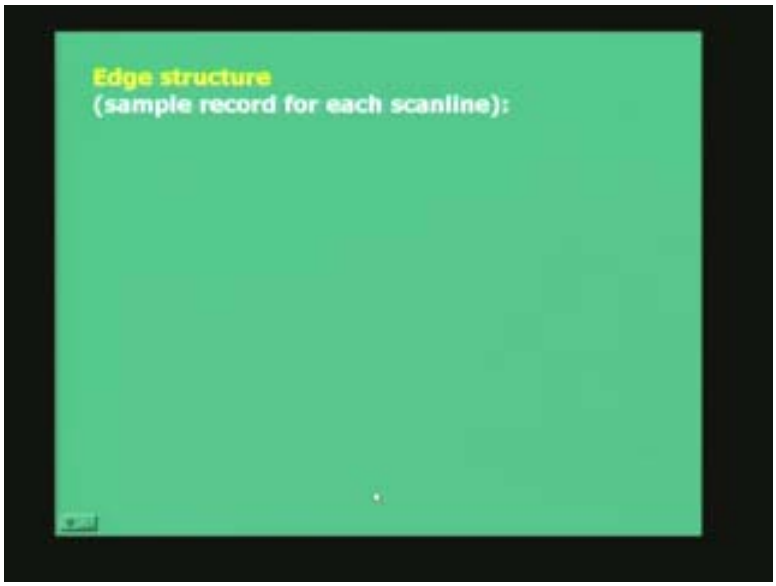


Only non-horizontal edges are stored and stored these edges at the scanline position in the SET or the sorted edge table. You do not need to process, at some point if you remember in the last class in the algorithm we said we only process non-horizontal edges. If you have horizontal edges what will typically happen is, those points will form an

intersection, a suitable pair and you will fill in between. So in fact you need not want to bother about horizontal lines. The algorithm itself is well suited were the horizontal edges will be automatically taken care of. We are talking about non-horizontal edges. So this is the set of lines which we describe as the shortest edge SET or sorted edge table where we talk of a bucket sort as many buckets as there are in a scanline, all edges are sorted with their Y_{min} coordinates with the separate Y bucket for each scanline.

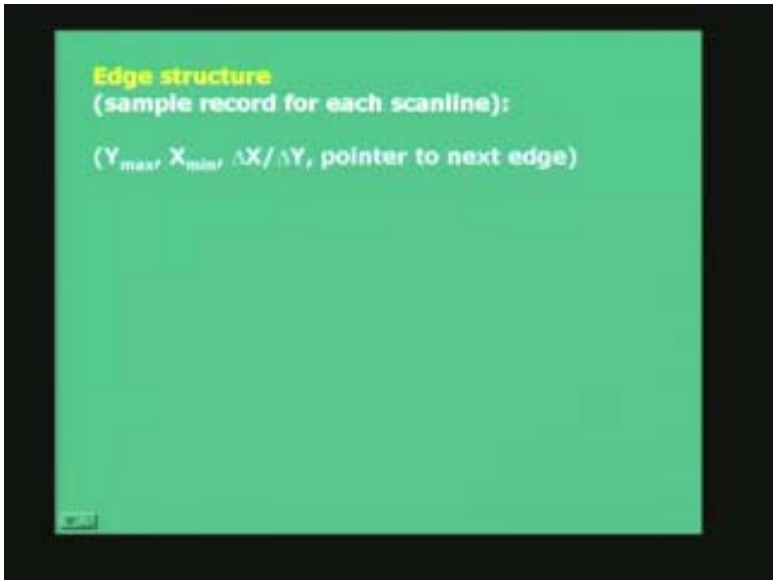
Remember these points and note it down now and verify. When you have an example of an SET within each bucket edges are stored by increasing X for Y_{min} point. You remember what is this minimum Y for a particular edge? And only non-horizontal edges are stored and store these edges at the scanline position in the SET.

(Refer Slide Time: 00:23:09)



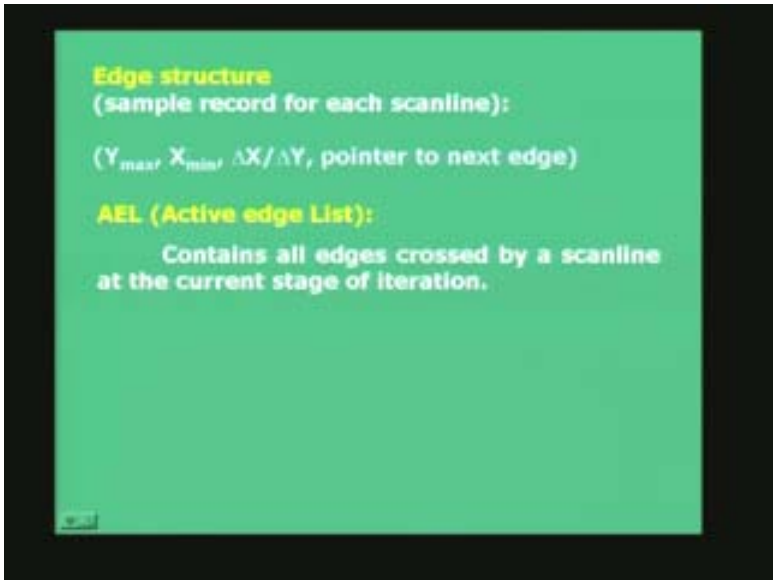
Let us look at a sample record for each scanline in the edge structure before looking into an example of a bucket sorted SET. The edge structure which is necessary to be stored for each scanline are the Y_{max} or the maximum value of Y. That means you need to know from your Y_{min} value for a particular edge till what value of Y_{max} you are going. So you need to store Y_{min} to Y_{max}, left to right and right to left, whatever the case may be and already it is sorted and kept in the bucket sorted SET with the Y_{min}. So you need to store the Y max, you also need to store the X_{min} because you are basically using the scanline coherence to increment the X. We have see seen that counter based integer algorithm which will help you to increase X. So you store X_{min} and also you need to store the value of delta X by delta Y or delta X and delta Y basically so pointed to next edge.

(Refer Slide Time: 00:24:00)



You can store delta X delta Y separately and of course the pointer to the next edge must also be given because the vertices have to be taken care of.

(Refer Slide Time: 00:24:07)

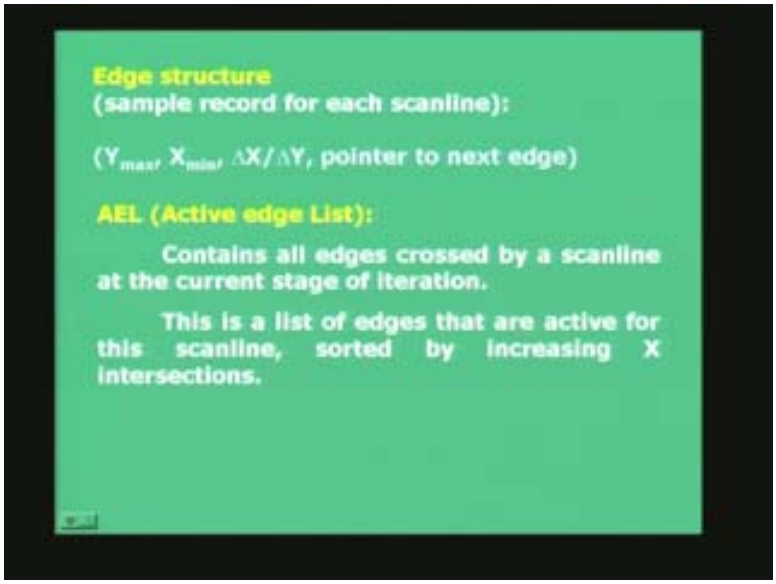


Active edge list is another data structure which must be used which contains all the edges which are currently active which means they are crossed by the current scanline at any stage of iteration or scanline at the current stage of iteration. Any scanline in general will not intersect at all the edges of a polygon. Some edges will start above the scanline and end and some will be finishing below that. But at any given scanline within a minimum

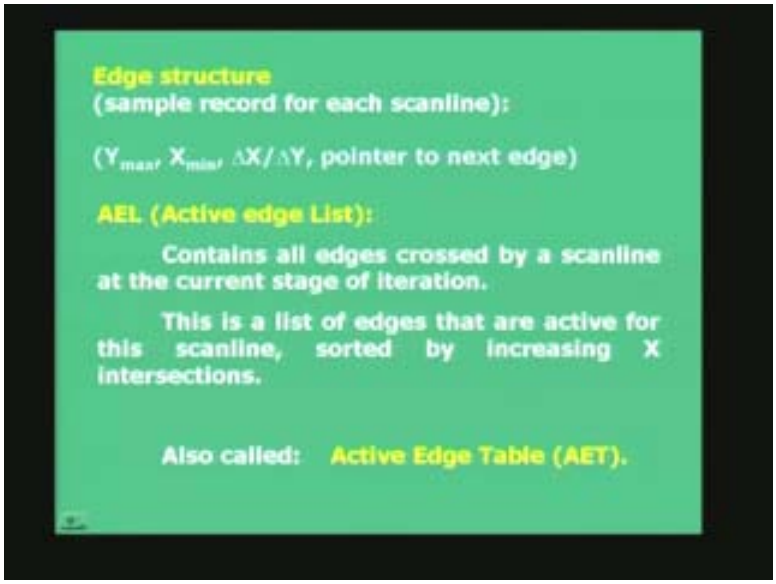
enclosed rectangle which covers the polygon, it will intersect a subset of the different edges of a polygon.

So you need to know what are the edges currently being intersected at a given scanline so the edge coherence, scanline coherence can be made to work. So that is stored in the active edge list. Remember edge structures store the information for each scanline. That will be updated for each scanline. But you also need to store an activate list which will tell you which are the edges which are being processed at any given point of time for a current scanline because when you reach a vertex what will happen is you will throw off an edge and bring in a new edge. Vertices are to be processed before the loop. Not only that, basically when you reach a vertex within the loop you throw off that edge which is over and bring in a new edge which starts with that corresponding value of Y. That is why you need an active edge list or AEL as it is called. So edge structure, active edge list and AEL is a list of edges that are active for this scanline sorted by increasing X intersections.

(Refer Slide Time: 00:25:26)

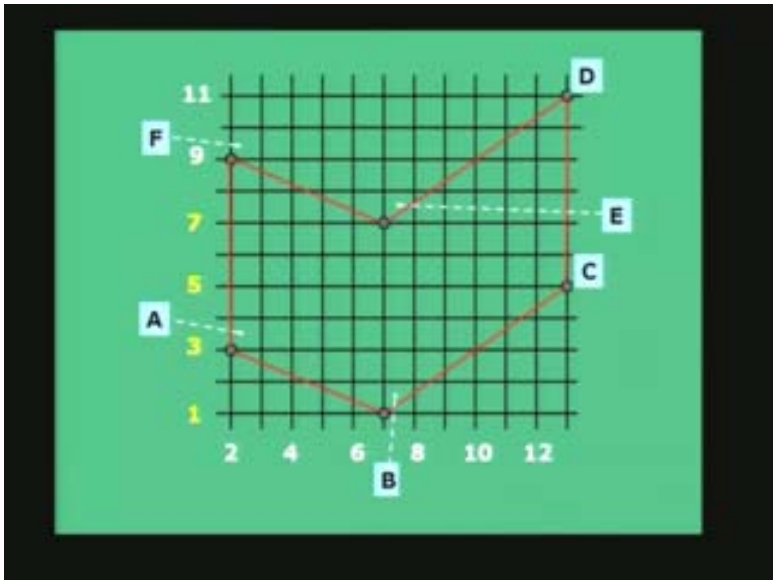


(Refer Slide Time: 00:25:32)



It is also called in some literatures as an active edge table or AET. Let us take an example in this class. This is again taken from the book.

(Refer Slide Time: 00:25:43)

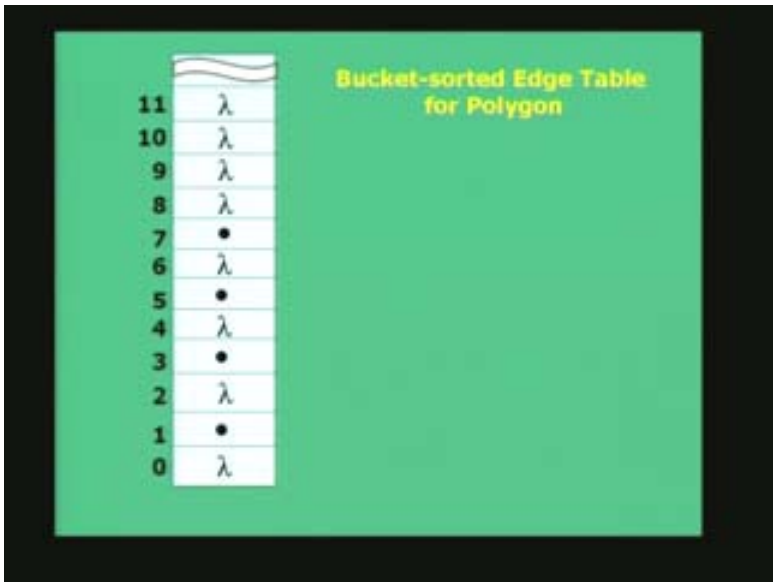


And let us consider this polygon which consists of 6 vertices. How many vertices are there in this polygon? Well, there are 6 vertices. So look at these grid structures where I have actually placed the pixel within the square area but will take the intersections of the grid as the integer coordinates. Pixel could be adjacent in this square so there are A, B, C, D, E and F. I repeat again this is A, B C D this is E here and F. So for an easier understanding we assume that the pixel is within this square grid but the intersection

point of the grid lines is our integer coordinates. And in this case the minimum enclosed rectangle will run from 2 to 13 along X direction and along Y direction from 1 to 11. This is the particular reason why I have put this 1, 3, 5, 7 in a different color yellow and others as non yellow that is 9, 11. We will see that very soon when we look at the data structure. I hope this polygon is very clear, it does not contain any horizontal edge but even if it was there we need not even process that.

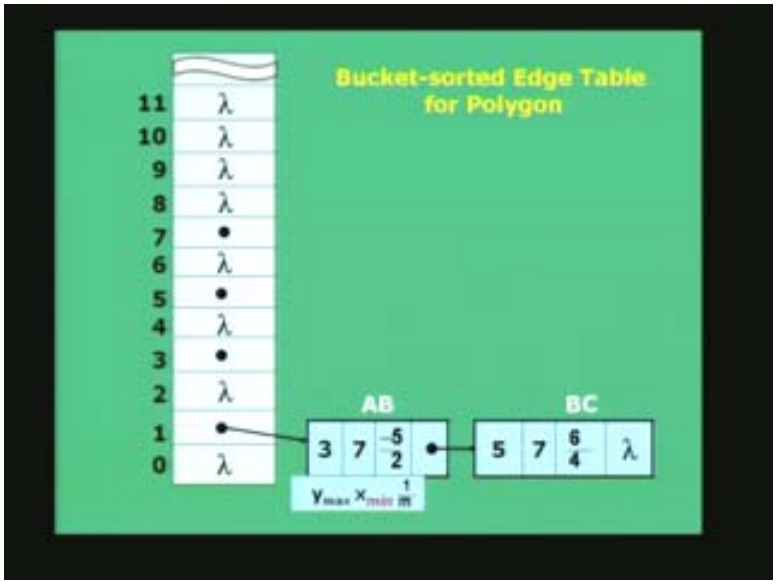
Only for non-horizontal edges which are all vertical or inclined except the horizontal one which process that and so in case since there are no horizontal edges we do not need to worry about that. So assuming that I will bring this figure again and again whenever you need and we will look at the data structures which are used for analyzing this polygon.

(Refer Slide Time: 00:27:14)



This is the bucket sorted edge table for polygon. What is the sorted edge table ACT? SET or ACT we talked about, sorted edge table, bucket sorted, you remember the properties which we just talked about. Each information of the scanline will hold of the Ymin value pointed to the next edge and so on. Hence only all these will be stored. So let us look at this bucket sorted edge table for the polygon which we have just seen. If necessary we will go back to the polygon again to have a look.

(Refer Slide Time: 00:27:46)



In the 0 scanline if you go and look back it does not contain any vertex. It starts with scanline number 1. So scanline number 1 has information about two edges AB and BC. Why it has information about two edges AB and BC? Let us go back to the figure and see. I will keep rolling back and forward between the polygon and the sorted edge table. You see at scanline number 1 there is a vertex where two edges that is AB and BC, remember we are talking about anti clockwise based representations so edges will be considered as AB BC CD DE and EF and finally FA. That is the way we will talk about a counter clock wise based rotation through the vertices and through the edges by which these edges will be represented.

So at scanline number 1 which is the bottom most scanline of the minimum enclosed rectangle which covers this polygon, the bottom scanline number 1 has two edges; one is AB another is BC. So that is what is there again AB and BC are the two scanlines of the polygon.

What was the data structure for the sorted edge table?
If you remember, what was the data structure? It is given at the bottom of that AB you need to store information in the sorted edge table, it is bucket sorted. But for each scanline for the edges which start with the minimum value of the Y_{min}, Y_{min} for both is 1 so that is why at that point AB and BC are there. You need to store the value of Y_{max}, the maximum value for the edge and then the minimum value of X where it starts for that vertex because the vertex is B. So we have to go back and see where is vertex B.

The vertex B has coordinates 7 and 1. X Coordinate is 7 and Y is 1. So X_{min} has to be put at 7 in both. What is Y_{max} for AB and what is Y_{max} for BC? Y max for AB is 3 that is the maximum and Y coordinate is 3. Look at the figure very carefully maximum Y value for the edge AB is 3. Once you follow the first entry of the sorted edge table the next few

things will be obvious. The edge AB has a maximum value of 3 for Y, maximum Y coordinate is 3.

What about for BC?
For the edge BC the maximum value of Y coordinate is at the vertex C which is 5. So 3 and 5 are the maximum Y value and X_{min} is 7. So let us go back that is very easy for you to visualize now that the X_{min} is 7 in both and the Y_{max} is 3 and 5 which we have just seen from the table. What is the value of del Y by del X for the sorted edge table? It is 5 by 2 del X and del Y. So del X is first stored which is basically 7 minus 2 is 5 for AB and del Y is 2 for AB in this case. And similarly for BC if you see it is 13 minus 7 which is 6 and 5 minus 1 which is 4. So 6 and 4 should be there obviously.

Actually instead of keeping as a fraction you can keep these two values separately because when you apply that scanline coherence you need that counter which will be incremented by del X and kept on observing till del Y is crossed beside the counter increment. So you basically need to store those values of del Y del X. So although this structure shows a floating point value of minus 5 by 2 and 6 by 4 but do not assume that you always need this fractional number because in fact you are not computing anything in floating point.

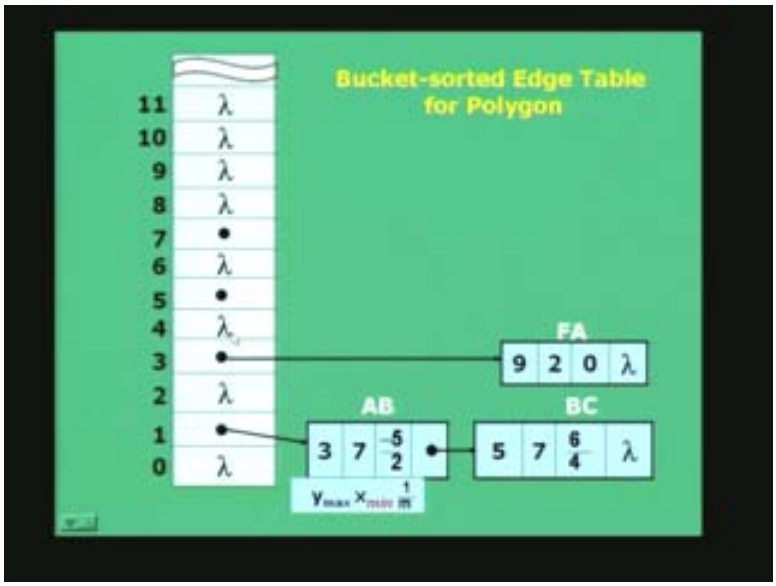
You are not computing anything in terms of floating point when you are looking at intersection from scanline Y_k to$Y_{k\ plus\ 1}$ to $Y_{k\ plus\ 2}$ and so on. So everything is done by integer based algorithm. Although this slope 1 by m which is del X by del Y is given as a floating point number you have to assume here that although I have given in this representation you are basically storing integer values of del X by del Y. So come back to that figure del X by del Y 5 and 2. Let us re look back into the edge AB and find out what is my del X and what is my del Y. Well you see again for the edge AB del X value is 7 minus 2 which is 5 and the del Y value is 3 minus 1 which is 2. Hence we have 5 by 2 or 5, 2 is also better to say.

And for the edge BC the del X value is 13 minus 7 which is 6 and the del Y value is 5 minus 1 which is 4. That is what is put into the first scanline. Hope it is clear. Y_{max} X_{min} 1 by m. These are the three elements of the data structure for each scanline stored in a bucket sorted edge table and each entry corresponding to a scanline which is passing through a vertex. That is what you can visualize is a sorted edge table. And you can see there are two, the last pointer lambda typically indicates the null pointer and the last entry of the field for your AB entry will point to the next stage BC. So now you can see, this is done. There is no vertex at scanline number 2.

Go back, this is scanline number 2, one above one does not have. There is a vertex at scanline number 3, 5, 7, 9 and 11. You just see the vertex A is at Y is equal to 3, the vertex C is at scanline 5, the vertex E is at scanline 7, vertex F is at scanline 9 and vertex D is at scanline 11. Those are the entries, if you see here in fact you can only see that the entries at 3, 5 and 7 because we need to store the bottom most. Up to 7 you need to store that is why I have put different colors, you do not need to enter anything at 9 and 11 because those are the Y_{max} entries for the respective edges. We are actually looking at Y

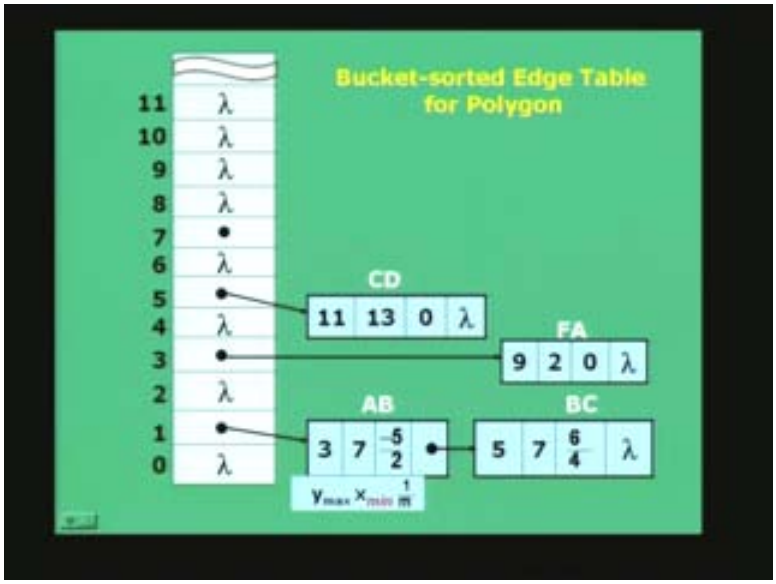
min. So 1, 3, 5 and 7 are the entries in the bucket sorted edge table which will have non 0 entries or it will not be null. The rest of them are all lambda which indicates null entries. So 1 is there. Lets us look at 3. It contains only one entry the edge FA.

(Refer Slide Time: 00:34:23)



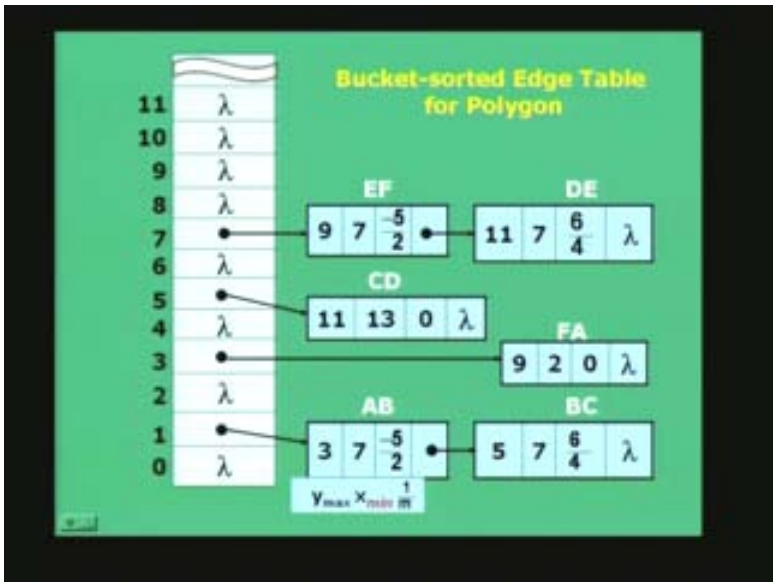
What is the Y_{max} value 9? We will go and check it. The X_{min} is 2 and Y is the value 0 here because the edge we will see is vertical. If the edge is vertical basically the del Y is 0. So you do not need to store anything in there basically you have to only keep incrementing the X incremental as you go but do not increment X at all and that is the basic idea. First look at Y_{max} and X_{min} for edge FA. Look at edge FA we transform 3 to 9 so 9 is the Y_{max} entry and X max is 2. So that is what is entered here, 9 and 2 and the last pointer is null. So I have drawn the diagram already for the polygon. I will not go back.

(Refer Slide Time: 00:35:16)



You can see that the next entries for the scanline 5 is D with Y_{max} at 11 X_{min} is 13 and the last entry is lambda and that 0 entry in the 1 by m shows that again it is a vertical edge CD and FA you can see in your notes that what we have seen from the polygon are vertical edges. So absolutely there is no problem. You do not have to provide a 1 by n value.

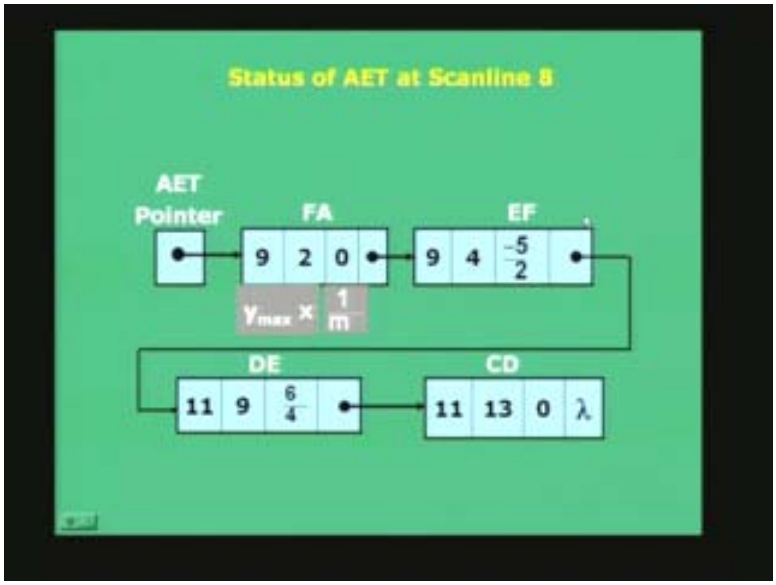
(Refer Slide Time: 00:35:40)



The top most entry comes at scanline number 7. We will go back to the figure and see that there are basically two edges EF and DE here. As you can see here EF and DE are the two entries at scanline number 7. This is a new diagram do not worry about the pixels

that are in between. We are going to analyze them later on but a scanline number 7 for the scanline EF and DE. Let us look at DE and EF and it is sorted with increasing X Y Y values typically. So as you see here the Y_{max} entry is 9 in one, 11 in the other one, X_{min} is both 7 of course and the slope is 5 by 2 and 6 by 4 minus 5 by 2 for EF because it is just 2 along delta Y and it is 7 along del X. In this case for the edge DE X is about 13 minus 7 which is 6 and 4 for the Y. This is the entry for the bucket sorted edge table for the polygon.

(Refer Slide Time: 00:36:53)



This is the example of how you construct your sorted edge table after making the minimum enclosed rectangle and going through the edges and then this is how you have to write an algorithm to construct the sorted edge table before you start and initialize and before you enter the loop. Of course we are assuming here that all vertices to form pair wise intersections are all handling special cases so that you will always end up with even number of intersections to be filled up. So let us look at this example, why I am talking of this example of scanline number 8.

This is of course a very simple case of intersection. I will teach a simple example which is again given from the book and you see here that for scanline number 8 the pixels which you have to fill in are pixels 2, 3 and 4 between edges FA and EF and then again pixels 9, 10 up to 13 for edges BD and DE between CD and DE. So you get four intersections or two pairs and you fill in between pairs so that is what you will fill. So how do you get those intersections for scanline number 8. The status of the active edge table AET if you remember, the active edge table will contain the edges which are active.

What do you mean by active? Those are the edges which are active or available or the scanline intersects that. So if you look at that these are the set of intersections by the AET pointer which will point to FA which will in turn point to EF, EF will point to DE and DE

will point to CD. Just note down this order FA pointing to EF, EF pointing to DE, DE pointing to CD.

We look at the other values later on. They are the same as Y_{max} X_1 by m entries. But you look at the order of the pointer from the AET pointer to the last entry of CD the null pointer here FA to EF to DE to CD, FA to EF and then you have DE to CD. That is the sequence and you look at the intersections of the values of X in increasing order. You have an intersection from 2 to F, two pairs for 2 to 4 and then 9 to 13. So you fill up pixels from 2 to 4 and then 9 to 13 that is what we have done. And the scanline will be filled with exactly those pairs of intersections and nothing will be filled outside the polygon.

So, if you can maintain this loop and reach a particular point where the active edge table contains the exact pair of intersections at any given point of time then you can form pairs and just fill in between and your algorithm will work. Now we have to of course see how you reach this position starting from any vertex. And of course at any vertex you need to throw out the active edge. The inactive edges are over and they bring in new edges. That is another point which we need to look at.

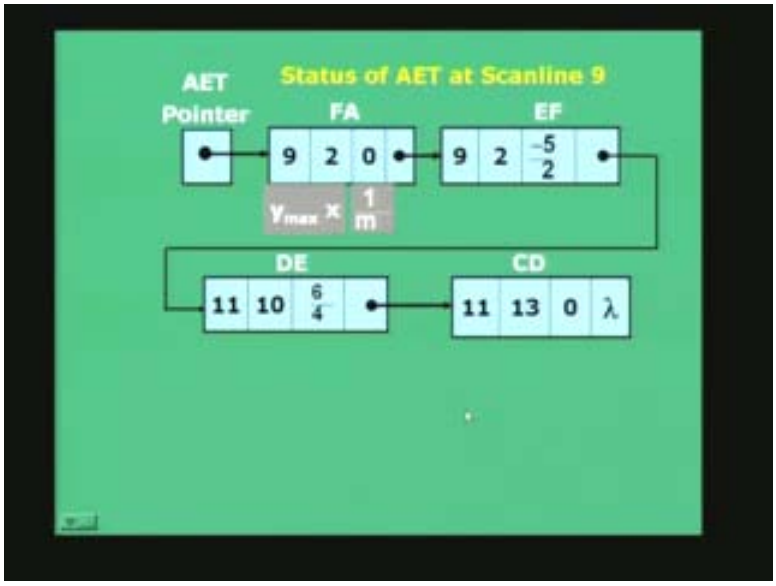
The status of AET active edge table as I hope is clear for scanline 8. As you see before in the Y_{max} of FA is kept as 9 that is not interchanged because this active edge table entries are brought in from the bucket sorted edge table. Eight entries are brought in from the ACT put it as sorted edge table for any given scanline, only when you reach a particular vertex what is done is for that particular scanline when you find that the Y_{max} value of an entry has reached that Y_{max} for that scanline you throw it out and bring in a new vertex from the sorted edge table.

You started a vertex, you have a set of active edge tables with the new one brought in, you keep on going till you reach a scanline which is the vertex pointed out into one by a ACT that means the new edge has come throughout the old edge and bring in anyone. That is what it typically keeps doing, in fact sometimes you bring in two edges, it is possible. But typically you bring in one edge and throw out one edge because at any given point of time you need to have pairs of entries and that will help you to obtain pairs of intersections for a scanline in a sorted edge table and to fill pair wise intersections.

So at any given point of time if you have an even number which helps you to form pairs then when reach a vertex basically if you throw out two edges you need to bring in two edges otherwise you will have an odd number of entries and that is not possible. Of course vertex manipulation and all that which we discussed in the earlier class, you remember the special case of vertex handling has all been done. You reduce an edge so that you basically bring in only one edge and throw out one edge. If you need to throw out two edges then the same two edges must be brought in. All those adjustments have been done, a prairie. So you always maintain the active edge table eight entries as even number of fields. Because whenever you have even number of fields you will have an even number of X intersections and you can form pairs otherwise you will have a

problem. So you see for the scanline 8, we have 4 or even number of entries and even number of intersections and in this specific case it is two pairs of intersections.

(Refer Slide Time: 00:42:08)

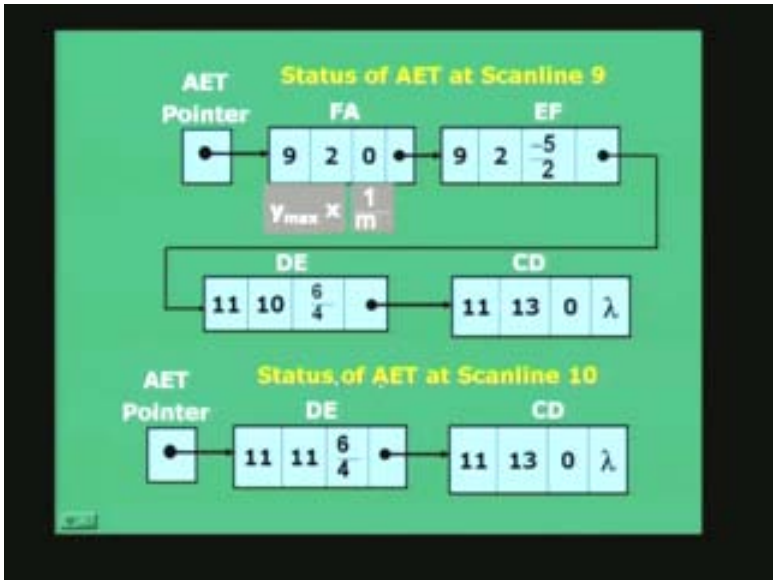


Look at the very interesting case of the status of the active edge table at scanline 9. Remember you rollback this is scanline number 8 compare this with scanline number 9, you see that the entries for the X has changed. Since FA is a vertical line X does not change because the increment is 0. EF is different; it changes from 4 to 2. Now this change of the X entry for the EF edge is done by that counter which I am not showing the calculation but I hope you will understand. I will leave it as an exercise for you to verify that the counter will work for EF edge starting from the vertex E to the vertex F as we go along.

But we assume that with the help of that dY dX deciding counter to 0 incrementing by dX till you reach maximum value of del Y and all those calculations which we have studied today. With the help of counter and counter increment all have to take place from one scanline to another to bring in that scanline coherence and reduce the calculation based on integer based arithmetic. That helps you to reduce the X intersection from EF which was at scanline 8, it was 4 for EF and at now for scanline 9 it has become 2. So you fill up just one pixel at 2 and then for the DE and CD. If you see DE intersection was 9, CD intersection was 13, DE changes to 10 but CD is to remain at 13.

Why CD remains at 13? It is like FA, CD is also vertical and if you see 1 by m entry it is 0. So the X intersection for FA and CD does not change because the 1 by m entry is 0 and since it is a non 0 entry for del Y by del X, del X basically to be very precise for EF and DE the X intersection changes. So what is the intersection at scanline 9? It is 2 to 2 that is one pair and another pair is from 10 to 13. We will see this intersection for scanline 9, the figure will come back.

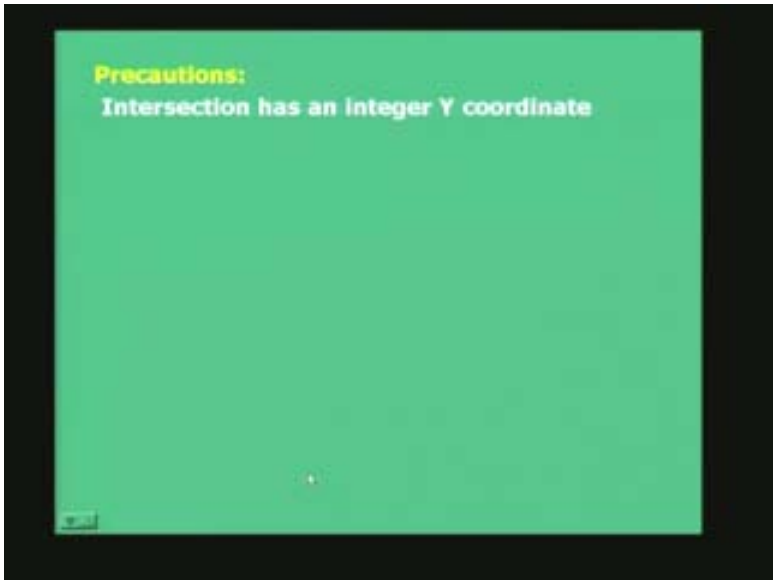
(Refer Slide Time: 00:44:02)



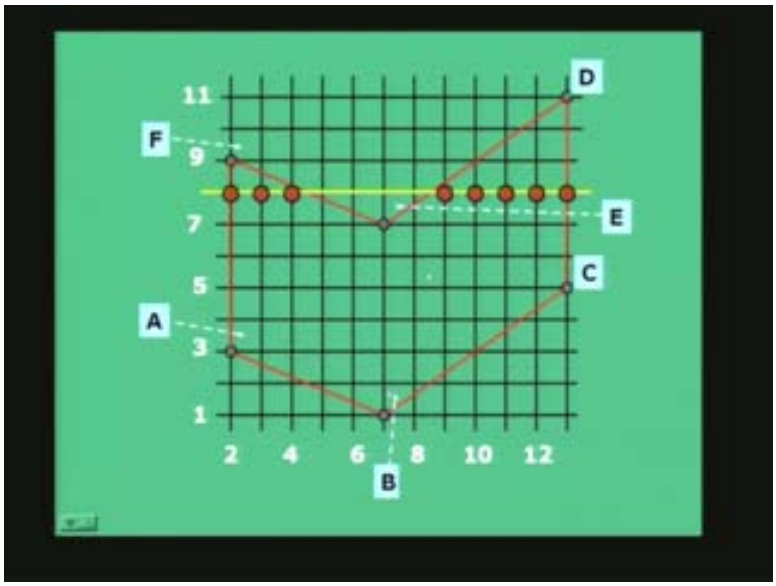
Status of AET at scanline 10, it is very very interesting. From 8 we have moved to 9 and from 9 to 10. When we move from 9 to 10 you find that for the edges FA and EF the Y_{max} value has been reached for that scanline. So throw out FA and EF and keep those edges for which the Y_{max} value is more than the scanline number since Y_{max} entries for DE and CD are 11 which is more than 10 you keep those entries and if necessary bring in new entries from the sorted edge table. But at scanline 10 if you remember the bucket sorted edge table, look into that entries which you have copied from that slide just now and there are no new entries in the sorted edge table. There is no edge that starts at 10. So there is no need to bring in anything. So it is basically showing out two edges.

If you throw out these do not bring 1. So the number of entries is still an even number. So look at that, in the scanline number 10 the entry of DE moves up from 10 it changes from previous entry 10 and then it moved to 11. CD is still at 13 because the value is 0. So this is 8, 9 and 10.

(Refer Slide Time: 00:45:07)



(Refer Slide Time: 00:45:10)

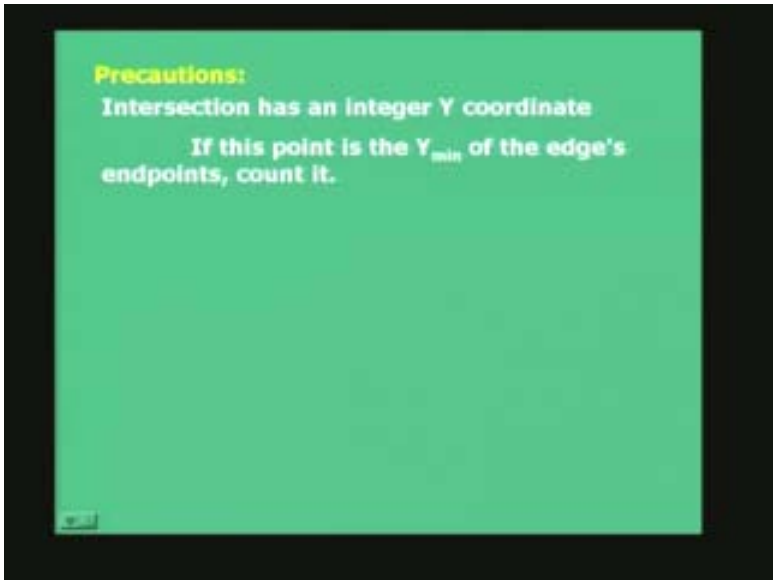


So you will just look at these things. See 8 we had filled from 2 to 4 and the yellow line shows the scanline number 8 where you had intersections from 2 to 4 and then from 9 to 13. Check it out again it is from 2 to 4 and 9 to 13. Look at the edge values for FA and EF DE and CD. I repeat 2 to 4 and then 9 to 13 X coordinates of intersection. When we move to 9 you will have this intersection only at 2 and then you will have from 10 to 13. Let us check the values which I gave you; 2 to 2 then again from 10 to 13. Let us look at scanline number 9 2 to 2 and 10 to 13. Roll back the figure and at scanline number 10 you will have only two intersections which is basically at its midpoint, you will basically get 11 or 12 to 13.

Basically you will move from scanline number 9 which was intersection 10 to 11 here, 11 to 13 a little bit of Bresenham's algorithm, either this pixel or the next one will be picked up, I think it is from 11 to 13. Let us go to scanline number 10. It is 11 to 13. You look at the X value for DE and CD for scanline number 10 at the bottom of your screen, intersections are 11 and 13 respectively. Precautions you need to do to make this algorithm work.

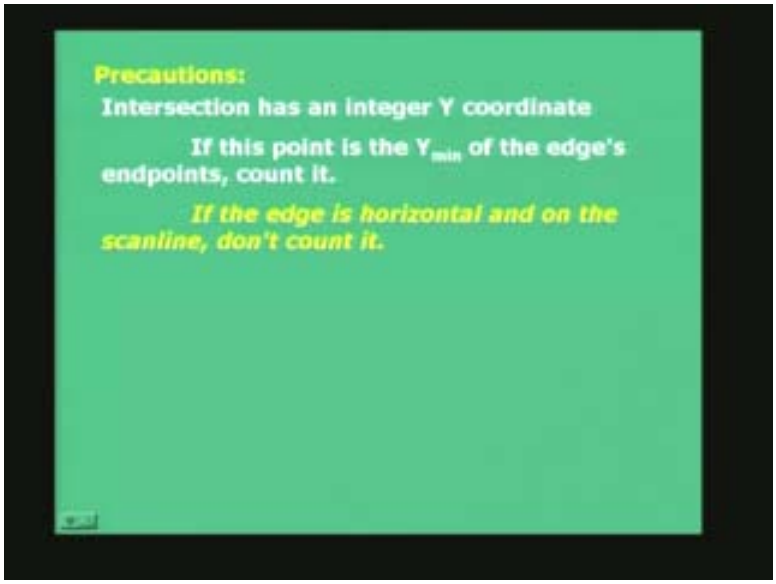
Now, we talked of 2 to 3 different data structures. One is SET a sorted edge table that has information about each edge to be stored in the bucket sorted edge table and also in the active edge list. It is the base data structure for each edge which has to be plugged out from the sorted edge table and brought it with the active edge based on the scanline entries. As you keep incrementing the scanline throughout and if you bring in new edges ensure the number of intersections is always even, you do that. Hence, this is all done. Precautions about vertices already taken up prairie and then these are the precautions which we revisit again. Intersection has an integer Y coordinate which is always true.

(Refer Slide Time: 00:47:07)



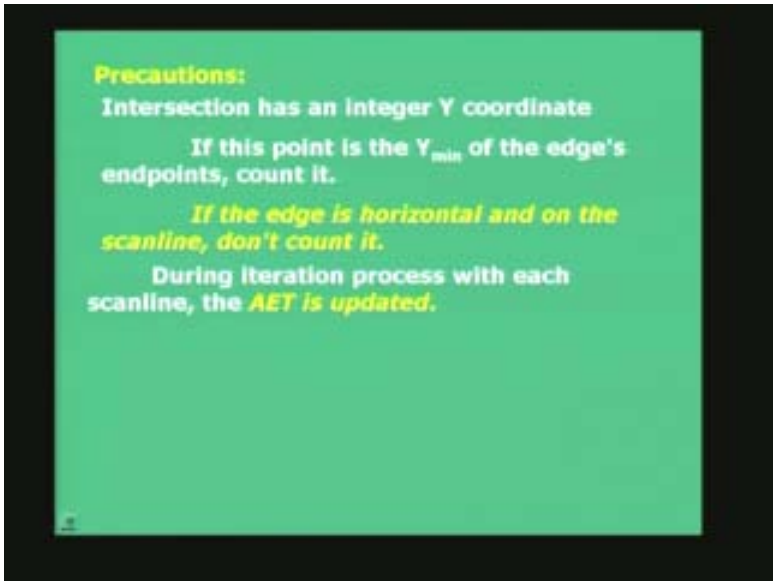
If this point is the Y_{min} of the edges endpoint you also take that as the Y coordinate.

(Refer Slide Time: 00:47:16)



If the edge is horizontal and on the scanline do not count it. In this example we have taken only non-horizontal edges but the edge is horizontal and on the scanline we never count it.

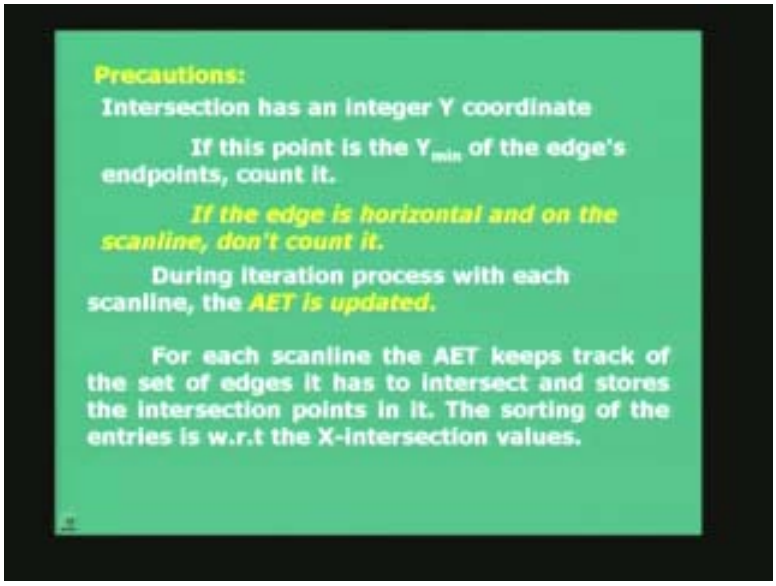
(Refer Slide Time: 00:47:27)



During iteration process with each scanline the AET is updated nothing is done to the SET. SET is constructed before, after the vertex cases have been handled you construct the sorted edge table and only start with the active edge table, AET constructed from the base bottom scanline and keep updating the AET. The X intersections are brought in, get

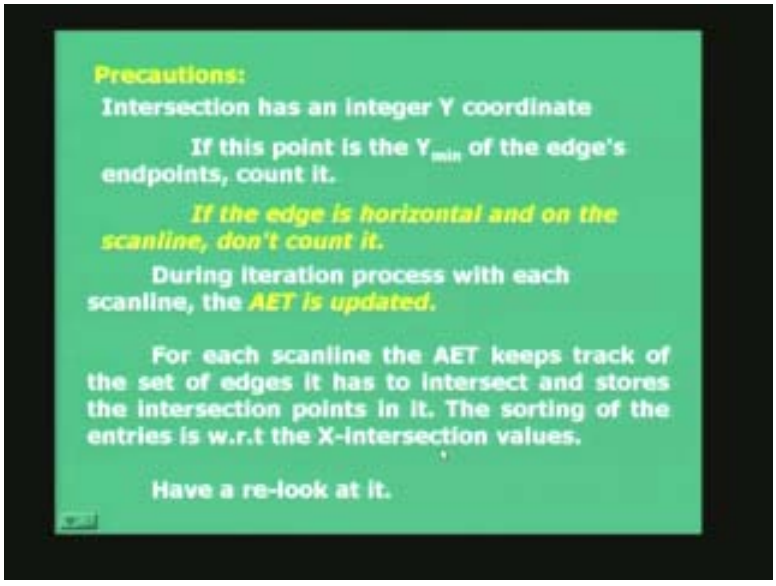
a vertex, look in the sorted edge table, bring in new entries and throw out if necessary and this is what you do at the active edge table, that is updated.

(Refer Slide Time: 00:47:55)



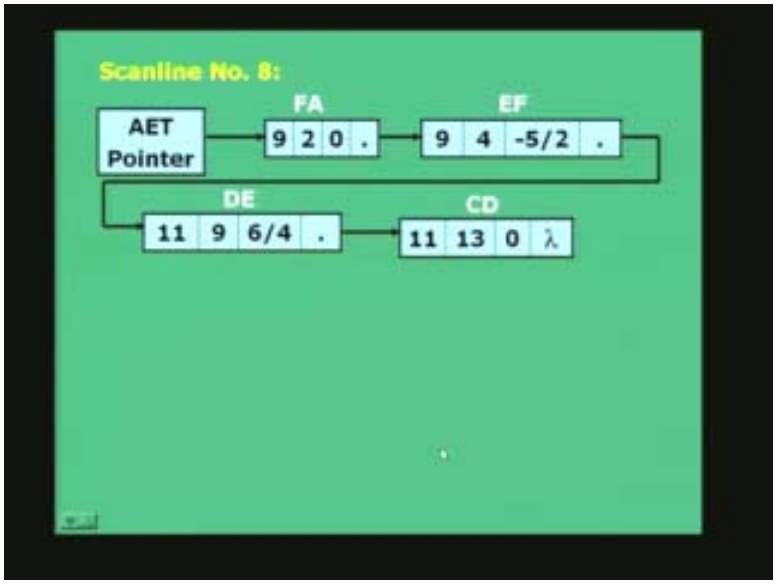
And for each scanline the AET or the active edge table keeps track of the set of edges it has to intersect and stores the intersection points in it. The sorting of the entries is with respect to the X intersection values.

(Refer Slide Time: 00:48:22)



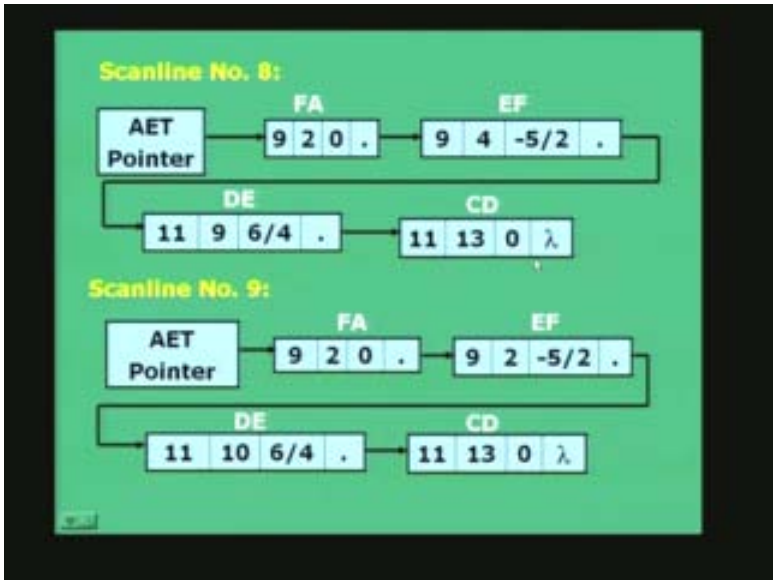
We have a re look at it, scanline number 8.

(Refer Slide Time: 00:48:23)



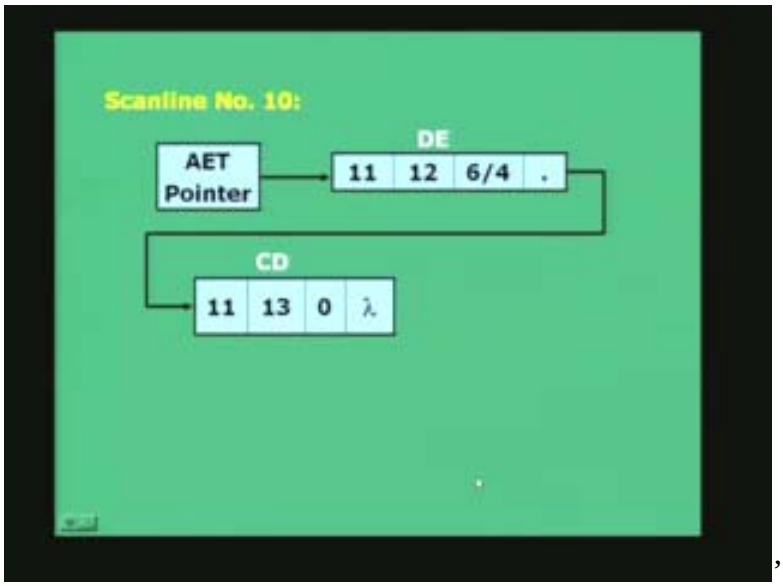
You see I have intersections now from 2 to 4 and 9 to 13, FA EF DE and CD last null pointer.

(Refer Slide Time: 00:48:35)



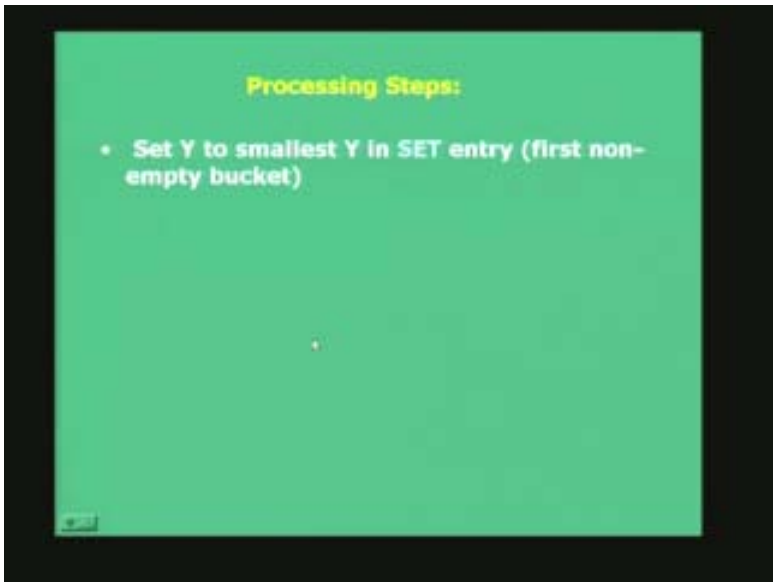
After this when you move to 9 and this is what we have, we find intersection from 2 to 2 for FA and EF and then from 10 to 13 for DE to CD. Do not fill in anything between.

(Refer Slide Time: 00:48:44)



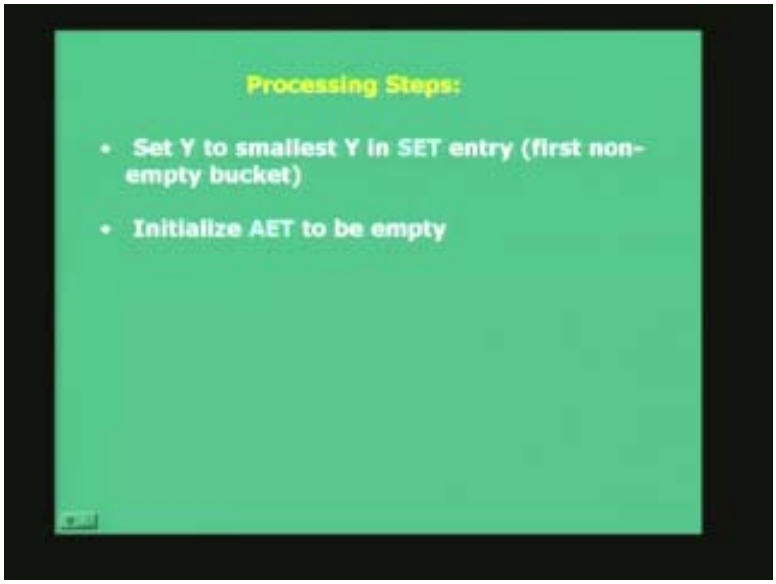
When we move to scanline number 10 you only fill in points between 12 and 13 and that is what we do for 8, 9 and 10.

(Refer Slide Time: 00:48:50)



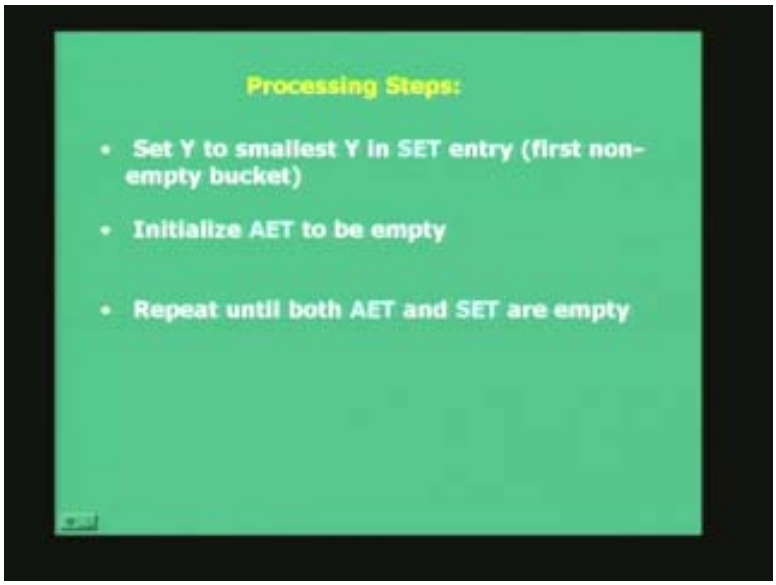
So the processing steps again revisited, set Y to the smallest Y in SET there is the first non-empty bucket in the sorted edge table, this is what you have to set your Y.

(Refer Slide Time: 00:49:01)



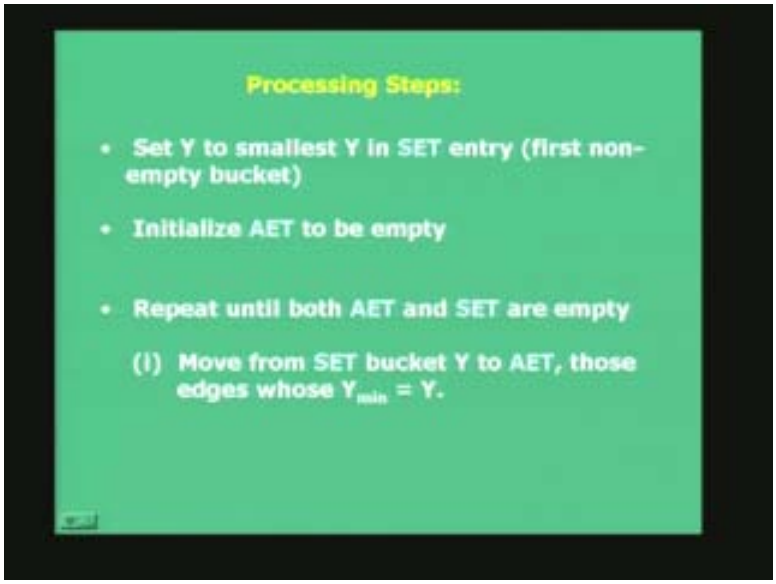
And initialize active edge table to be empty.

(Refer Slide Time: 00:49:06)



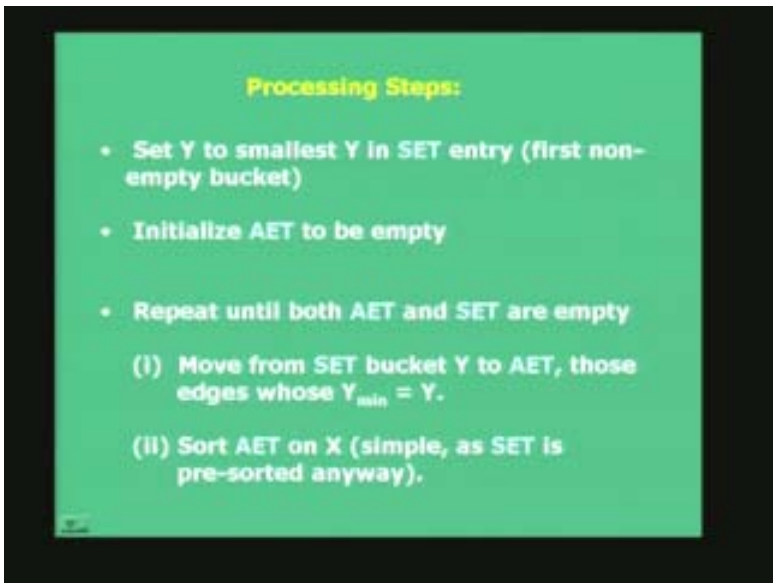
And then repeat until both active edge table and sorted edge table are empty, both have to be empty so keep on repeating this one.

(Refer Slide Time: 00:49:12)



Move from the sorted edge table the bucket Y to active edge table those edges whose Ymin is equal to Y. You start with the minimum value of Y and at that point move from the SET a bucket entries the Y entries to the active edge table. So you have a non entry into the active edge table now and the ACT does not change.

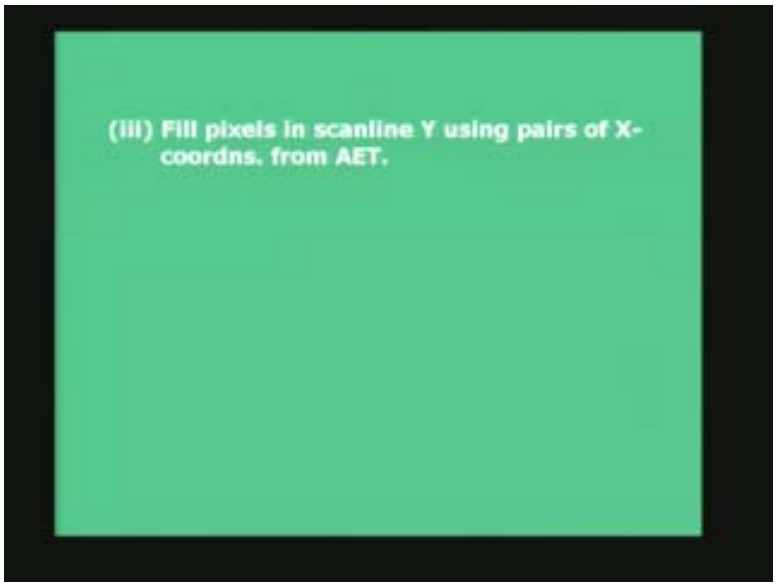
(Refer Slide Time: 00:49:36)



Sort AET on X this is simple as sorted edge table is presorted anyway so you do not need to sort anything on the active edge table. Look into whatever you have copied for the active edge table it is already presorted because you are bringing in values. Anyway at the start they are presorted so you put it when active. At any vertex later on when you

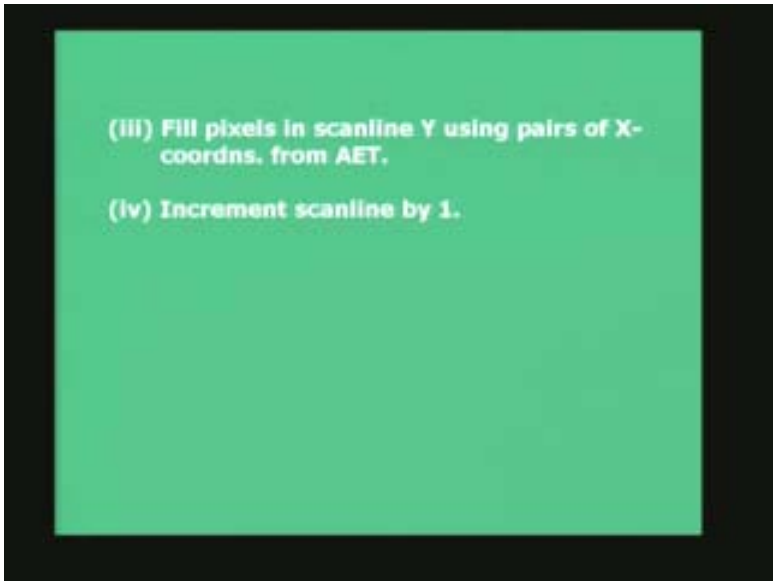
throw away a set of vertices and bring in new you might have to do a little bit of adjustment in the number of pointers and entries. What you need to do is when you throw out you readjust the pointer and bring it something towards the end or may be at the beginning. That is where you have to do when you reach another vertex later on. But initially it is just presorted, you just float from the sorted edge table the bucket entries corresponding to Y in the active edge table.

(Refer Slide Time: 00:50:19)

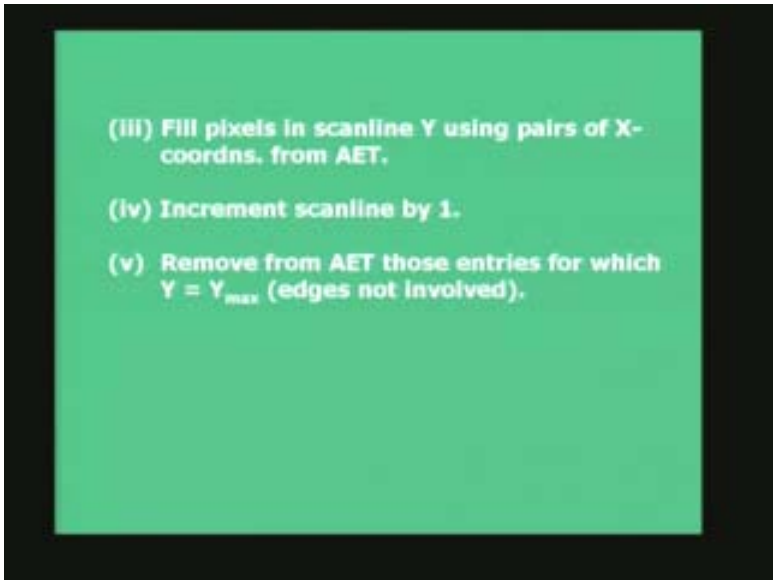


Once this is all done fill pixels in scanline Y, look at the screen. The third point in loop is fill pixels in scanline Y using pairs of X coordinates from active edge table.

(Refer Slide Time: 00:50:33)

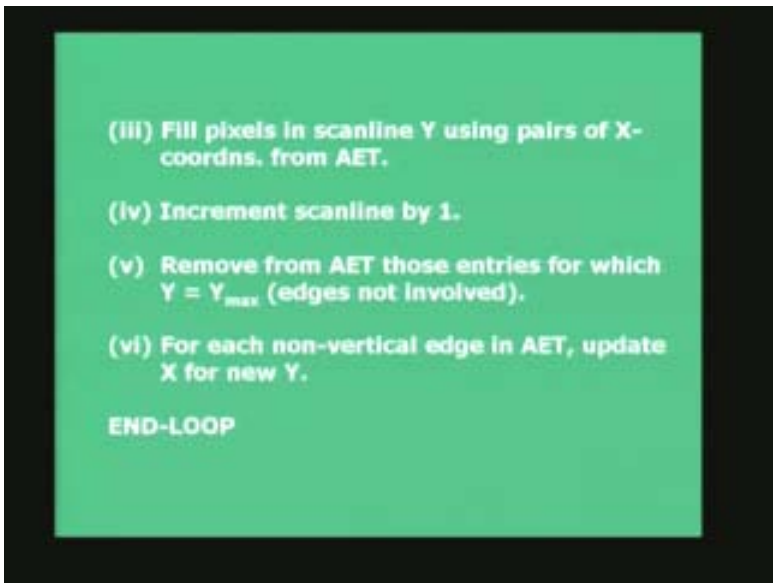


(Refer Slide Time: 00:50:38)



Increment scanline by 1 and then remove from active edge table those entries if you find that there is a Y_{max} where edges are involved where there is a Y_{max} entry into the active edge table for that corresponding scanline which we have reached and for that scanline Y if Y is equal to Y_{max} for certain edges in the active edge table you remove those entries.

(Refer Slide Time: 00:50:38)



And for each non-vertical edge in active edge table and but you anyway do not have horizontal edges so all edges are non-vertical anyway. Update X for new Y as you know how to do that using counter. So we will go through this loop once again which is very short. You actually set Y to smallest Y, initialize AET to be empty, fill up SET, repeat

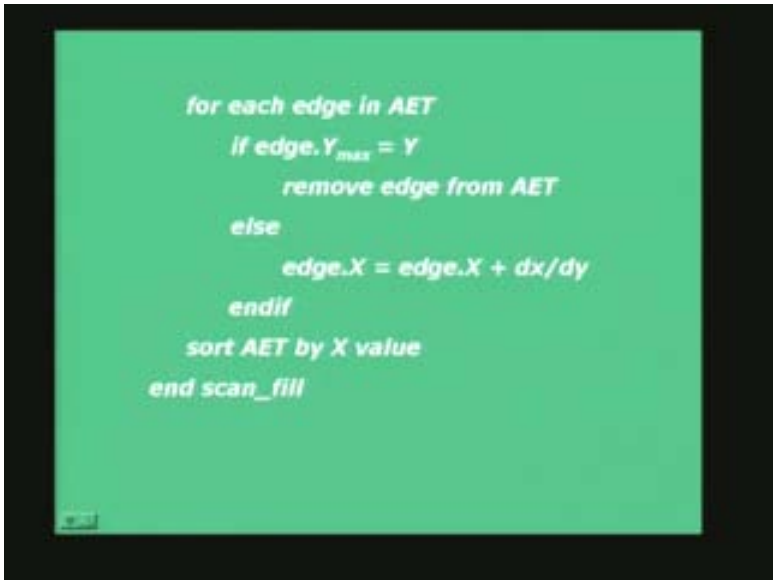
until both AET and SET are empty. What do you repeat? It is a loop which says move from SET bucket Y to active edge table those edges for which Y_{min} is equal to Y. Sort AET is simple and sorted anyway at the start later you have to do that sorting when you encounter in a vertex.

Fill pixels on scanline Y using pairs of X coordinates from active edge table increment scanline by 1. Then remove from active edge table those entries for which Y is equal to Y_{max} and edges are not involved. So you can move that. For each non-vertical edge update X for new Y and that finishes the loop till you complete this cycle. So this is in short a sort of a pseudo language, pseudocode of scanline filling algorithm, construct edge table, Y_{min} is equal to Y for all Y in edge table, active edge table is null and you run from Y_{min} to Y_{max}, merge sort edge table Y into active edge table Y.

(Refer Slide Time: 00:52:01)

I hope, those with CS background will know what is a merge sort edge table of a Y entry from the sort edge. Construct the edge table means the sorted edge table. Merge sort these entries of Y of the sorted edge table into the active edge table by X value. Fill between pairs of X in edge table and for each edge in active edge table if Y_{max} is equal to the current Y scanline remove the edge from AET else increment the edge by dX. Although this floating point equation is given and we know that this is done by integer algorithm. So do not take this by heart that actually you are doing a floating point addition. This equation on that 1 by m whenever it is given in the entries in the active edge table example you must know that we are always using that counter based integer algorithm to get to the next X intersection.

(Refer Slide Time: 00:53:07)



And finally sort active edge table by X value and that is the scanline filling algorithm. So this ends the series of lectures on polyfill scan conversion of a polygon or a scanline polygon filling algorithm and I hope you enjoyed the example as I also did delivering it to you.

Please work out that example yourself or take a set of polygons, you can work with groups of two students each. Take a graph paper, draw a polygon, find out the set of vertices, construct a sorted edge table, start with a empty active edge table, fill up from the minimum Y the entries into the active edges table, keep on incrementing the scanlines, when you enter the vertex remove the required number of edges, brings in new edges if required and keep on doing this till you reach Y_{max}. So work out different example, tally your results with each other and see if you have understood the algorithm and the data structure which has been used and the integer based algorithm which exploits the scanline and edge coherence and things will be verified.

So please try an example to understand this method and then you can try to implement this using a program based on segment. Thank you very much.