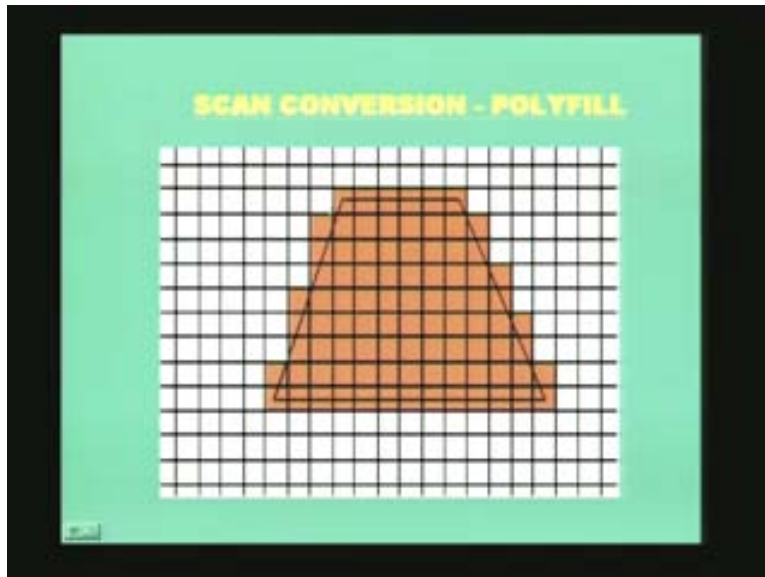


Computer Graphics
Prof. Sukhendu Das
Dept. of Computer Science and Engineering
Indian Institute of Technology, Madras
Lecture - 18
Polyfill - Scan Conversion of a Polygon

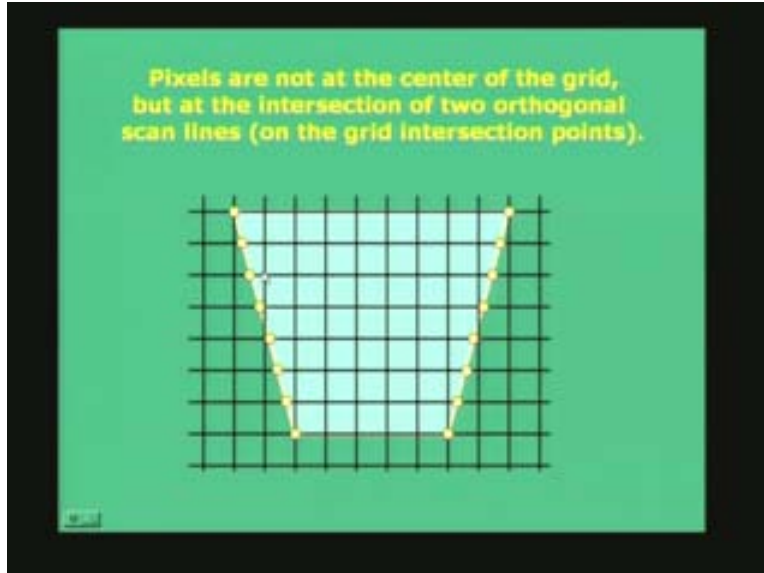
Today we will discuss the concepts of a polygon filling algorithm which is termed in some literature as polyfill or polygon filling and it is based on a method of scan line based drawing algorithm where the polygon is filled based on scan lines from the top to the bottom. So we will see the methodology behind polyfill or scan conversion of a polygon as the concept is called. Let us see a simple example of a polygon on a raster frame buffer or on a screen and see what the problem is when you need to fill the polygon with a particular color in this case or a particular shape.

(Refer Slide Time: 00:02:00)



Here we see an example of polygon filling with a polygon of four vertices or a quadrilateral. So there are four vertices forming this polygon or a quadrilateral and you can see that the region inside this polygon is filled with a different shade than the background color. I have chosen the background to be white color and the color to be filled inside is filled with red. You can see the effect of aliasing of the pixels which are at the center of the grid. They are not the grid intersection points but they are square pixels at the center of the grid and you can see the effect of aliasing as for non-horizontal lines in this case. So basically you can see here that the polygon filling is a case of filling horizontal lines from left to right and you will have the effect of aliasing as you had for drawing straight lines using Bresenham's algorithm.

(Refer Slide Time: 00:02:58)



Well this is a slightly different arrangement of this polygon where now we consider that the pixels may be at the intersection of the grids rather than or grid intersection points or the pixels rather than the center of a square. As you can see if I define to the polygon vertices as its integer coordinate values which lie on the intersection points it is possible that when you consider the intersection of a horizontal scan line, the scan line is a horizontal line with one of the edges of a polygon with a two vertical inclined edges and you find that the intersection point is basically a floating point number but we should not worry about this case because we already know from line drawing algorithm to select the nearest integer.

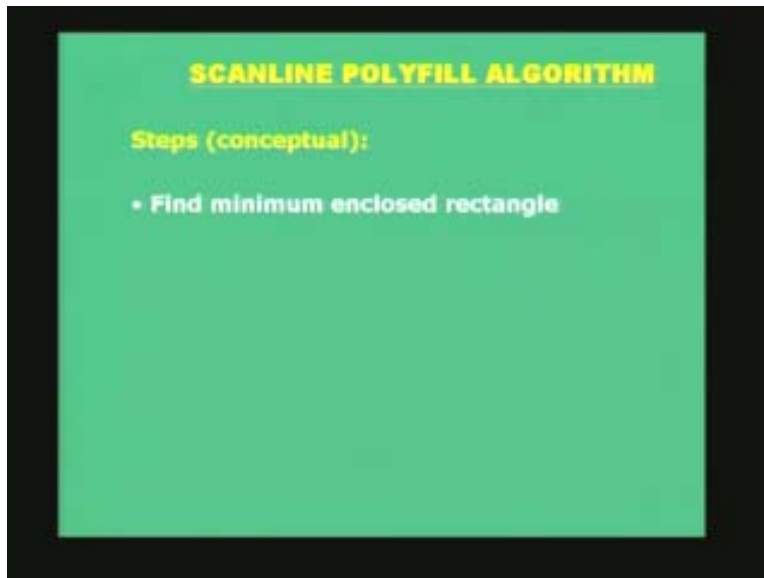
We know that we can actually draw a line with pixels which are the closest to a line or a straight line and well just find out which are the closest and which integer values are and we have also seen how an integer based algorithm can be used for drawing circles and ellipses. Almost a similar concept is used here but I must mention and specify here that we are not drawing lines. We are basically filling regions within a polygon but that concept of integer based algorithm will also be used here indirectly. So I must mention here with respect to this figure that although the intersection points with the inclined edges of the polygon and the horizontal scan lines could be floating point numbers.

But you have to actually select the nearest integer coordinates or pixels which are at the intersection of the grid lines and shade them. Change the color and of course all intersection points or pixels within this polygon must also be shaded. This is a clearer picture which illustrates or shows you how the polygon will appear after it is been shaded. This is the definition of the problem what we are going to do and let us assume to start with the input to the problem in terms of an algorithm is that the following input is provided to the algorithm that we have a set of integer coordinates which are the vertices of the polygon. We have vertices one, vertex two, vertex three and so on. There could be n different vertices and our job is to find out the pixels which lie inside the polygon and

also on the perimeter of the polygon. That is the scan conversion. Why the name scan conversion is used or what is called as scan line polyfill algorithm?

We will look at a very conceptual framework or the steps of the algorithm which is used to implement and then analyze these steps one by one and see with the help of algorithms how each step is implemented. We will have to get into concepts of graphics as well as mathematics to find out how this works.

(Refer Slide Time: 00:06:07)



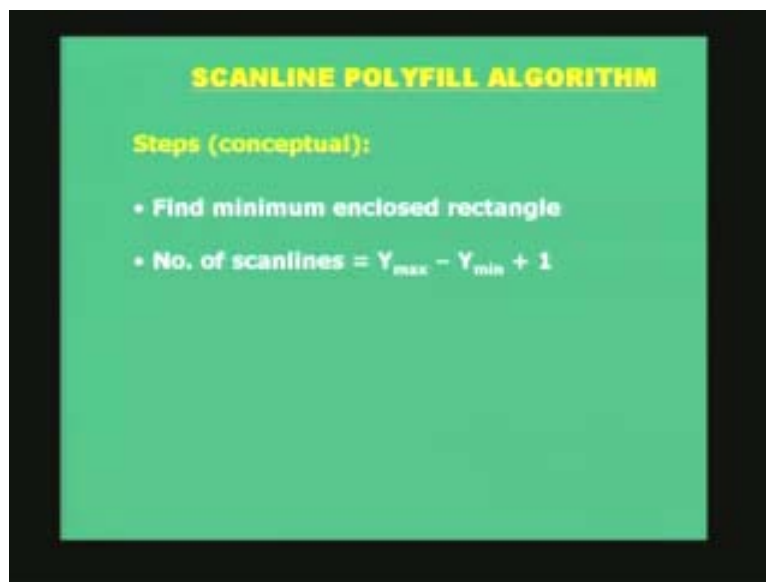
The first thing is when a polygon is defined we need to find the minimum enclosed rectangle. This minimum enclosed rectangle is a minimum rectangle which binds a polygon. So, when we have a polygon defined with a set of vertices one, two, three, four up to n and of course the last line connects the n th vertex to the first vertex or from a CS literature you can say vertex 0, 1, 2 and so on up to n minus 1.

We are talking of n different vertices defined on a polygon. Well we have to bother about certain set of pixels which lie inside or around these vertices. So basically we can concentrate to do a scan conversation on a minimum area which is defined by a minimum enclosed rectangle which is a bounding box within which the polygon lies. So you define a polygon and you find the minimum enclosed rectangle. That is a very simple problem as far as geometry is concerned because what we have to do basically is find out from the list of X and Y coordinates, the minimum and maximum value that is X_{\min} and X_{\max} from the X coordinates.

Where are the X coordinates? These are the coordinates of the vertices of the polygon. So take the X coordinates of the vertices of the polygon and find out the X_{\min} and X_{\max} which is the minimum and the maximum value and also take the sequence of Y values Y coordinates of the vertex of the polygon. Find the minimum and maximum value that gives you Y_{\min} and Y_{\max} . So X_{\min} X_{\max} and Y_{\min} Y_{\max} will give you the minimum enclosed rectangle. That is that minimum enclosed rectangle.

The minimum enclosed rectangle terminology means that, that is the rectangle with the minimal area which covers the entire polygon. Whatever shape the polygon may be you cannot probably find a rectangle smaller than that which just fits it. Of course you can take a larger rectangle but then you have to work hard and the time will be spent with the more. So we are looking into a smaller space which we have to analyze to find out which are the pixels you need to shade and which you do not want. So the first step involves, conceptually as we see here, find the minimum enclosed rectangle so that is the description of the first step where you find X_{\min} X_{\max} Y_{\min} Y_{\max} and that gives you the minimum enclosed rectangle.

(Refer Slide Time: 00:08:30)

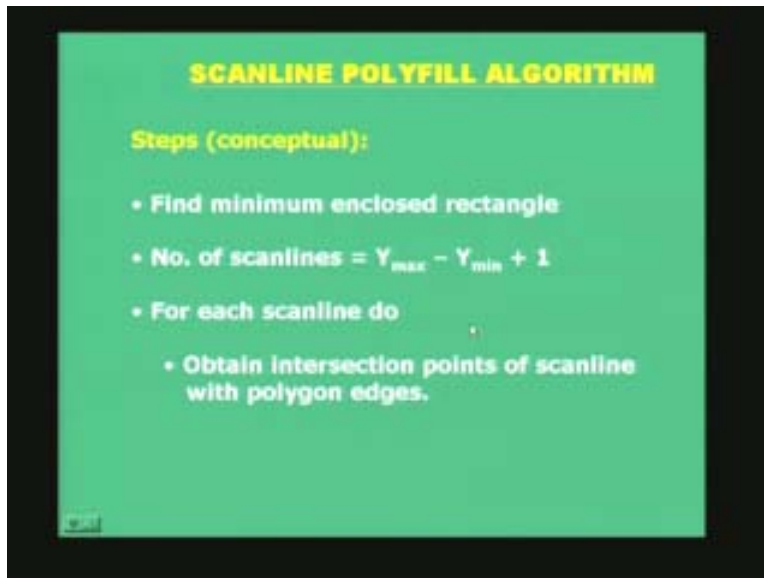


From the Y coordinates, maximum and minimum Y coordinates of the minimum enclosed rectangle you can find the number of scan lines you need to draw, that is the number of horizontal lines. I should say that you have a minimum enclosed rectangle, you have a X_{\min} to X_{\max} Y_{\min} to Y_{\max} as talked about and from Y_{\min} to Y_{\max} using the separation between the minimum Y coordinates and the maximum Y coordinates you can find the exact number of scan lines or horizontal rows of pixels which you have to analyze to find out if some pixel is inside the polygon or outside the polygon and then paint it or give it a color.

The number of scan lines means the number of rows of pixels which starts from the lowest row of the minimum enclosed rectangle to the topmost row of the minimum enclosed rectangle and so the Y_{\max} and Y_{\min} difference minus 1 of course becomes the integer we are talking about. The difference, the number of lines including the top one and the bottom one so you have to add 1 to that which will give you the number of scan lines from top to bottom or from bottom to top either way you can go it does not matter but you must be consistent that while you are moving at the from top to bottom or from bottom to top. So, number of scan lines is given by the difference in the Y coordinates of

the minimum enclosed rectangle Y_{\max} minus Y_{\min} minus 1. These are the first two small steps. Then, for each of these scan lines, how many number of scan lines you have to work on? It is given by the step number two that is Y_{\max} minus Y_{\min} minus 1. So, if Y_{\max} is 10 and Y_{\min} is 5 these number of scan lines will be 6. You can count yourself 5, 6, 7, 8, 9 and 10. So that makes it six scan lines. Let us say with an example. So for each scan lines you have to do the following steps.

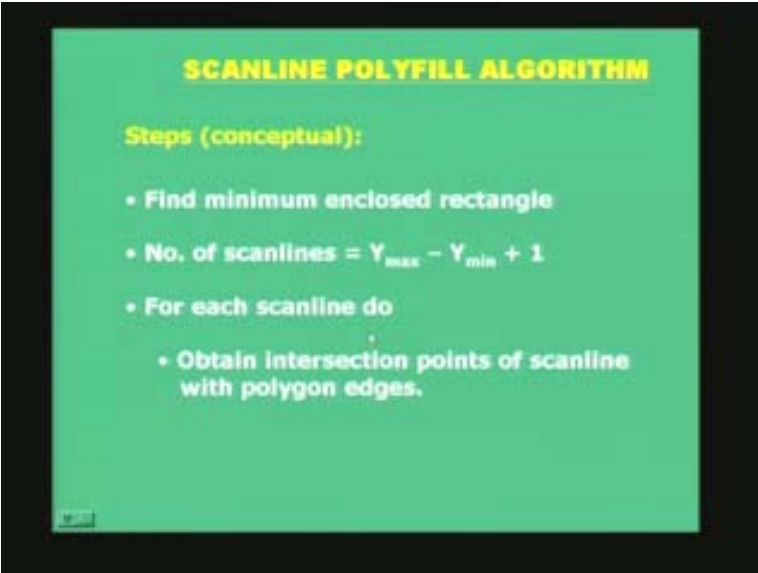
(Refer Slide Time: 00:10:23)



The first step is, obtain intersection points of scan line with the polygon edges. We will see this with an example in the next slide that means our idea is to find out how many pixels within each scan line lie inside the polygon. So what you can do is try to find out what are the edges which intersect with this particular scan line. From those intersections we have to process them to find out which of these regions within this intersection is a line within the polygon or outside. To visualize an arbitrary polygon not the simple polygon of a rectangle which we just talked about are just shown in the slide, one or two slides before that is a very simple or over simplistic case but we can assume something like a star type of a structure is an example of quite an arbitrary polygon. And if we take a scan line what you will find is that there are some sections of the scan line which lie inside the polygon.

We will take an example soon and there are some sections which lie outside. This is again we are talking of a scan line within the minimum enclosed rectangle. If you take a scan line some portions will be inside the area of the polygon some will be outside. You multiply such areas within a particular scan line which will be inside and some outside. So we have to find out which portions are inside and outside. But before doing or before finding out which regions or parts of scan line are within the polygon and which are outside, you must actually find out the intersection points of the scan line with the polygon edges.

(Refer Slide Time: 00:11:54)



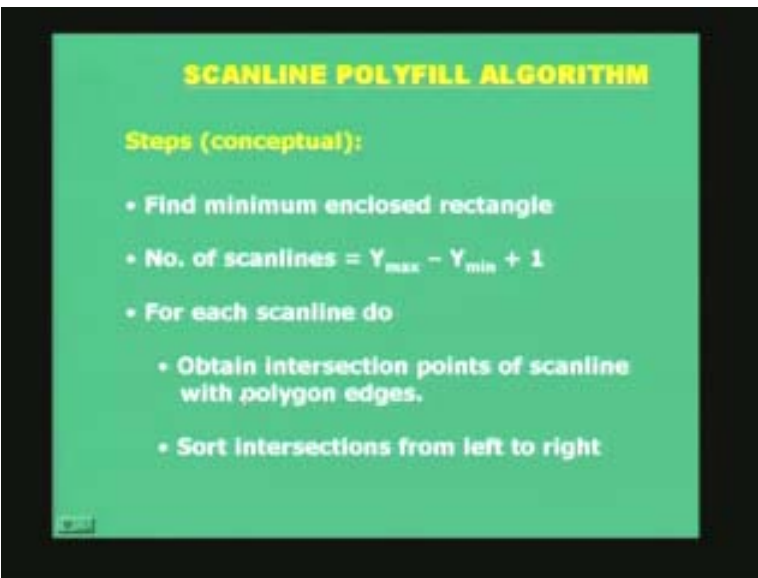
SCANLINE POLYFILL ALGORITHM

Steps (conceptual):

- Find minimum enclosed rectangle
- No. of scanlines = $Y_{max} - Y_{min} + 1$
- For each scanline do
 - Obtain intersection points of scanline with polygon edges.

So that is the first step of this fall loop where we say for each scan line we obtain intersection points of scan line with polygon edges and then sort these intersections from left to right.

(Refer Slide Time: 00:12:01)



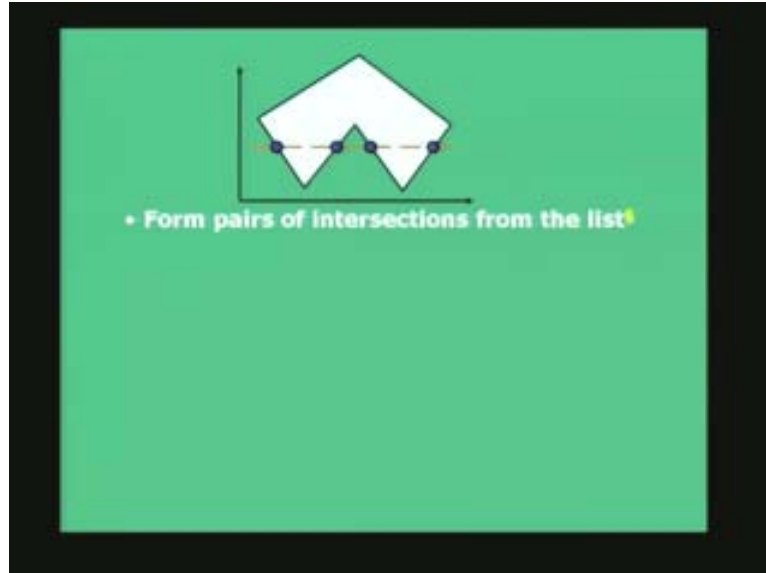
SCANLINE POLYFILL ALGORITHM

Steps (conceptual):

- Find minimum enclosed rectangle
- No. of scanlines = $Y_{max} - Y_{min} + 1$
- For each scanline do
 - Obtain intersection points of scanline with polygon edges.
 - Sort intersections from left to right

So in ascending order from left to right in terms of X coordinates you arrange these in ascending order and then you basically form pairs of intersection from this list.

(Refer Slide Time: 00:12:13)



This is an example of a figure which I was talking about. This dashed brown line talks of a particular scan line when you are within of course the rectangular and this is a polygon with 1, 2, 3, 4, 5, 6. There are 6 vertices and this is an example of an arbitrary scan line which intersects four different edges of the polygon, which are marked by the circular blue large dots. There are four intersections of these which you have to find first, that is the first step inside a fill loop for this particular scan line only and after you get these four intersection points you have to arrange them in ascending order from left to right in increasing X coordinate. So you will have X1, X2, X3 and X4, four different intersections and then you form pairs.

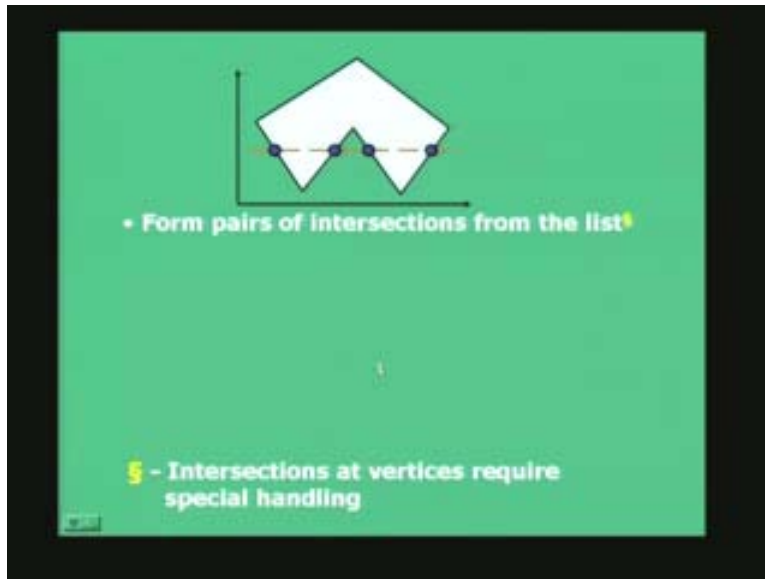
Now you see the interesting property of this polygon that if you form pairs of intersections from this list, that means you form a pair 1 and 2 and you form another pair 3, 4 and so on if necessary, then you will find that you have to only fill up pixels within pairs. If you shift the scan line somewhere here on the top then you will actually obtain one pair of intersections, so the numbers of intersections of the edges of the polygon with a particular scan line is always an even number and that is why you can form pairs.

I repeat again; you can try with arbitrary shape except there are certain exceptions we will talk about that very soon. In general if you take a polygon and draw arbitrary scan lines you will find that that the number of intersections formed are in general even numbers, you can try it in a piece of paper right now, that is the number of intersections which you have. And since the number of intersections are even numbers you can form pairs. As in this example we have seen, if you take a line, as we have taken in this figure there are four intersections so you can form two pairs.

If you shift this scan line to the top of the figure somewhere within the polygon of course you will get two intersections which will be only one pair. So, depending upon the nature of the polygon, the shape of the polygon or even the size, you may have different number

of intersections of the edges with this scan line. But in general typically we will find that the numbers of intersections are even numbers and if they are even numbers you can form pairs.

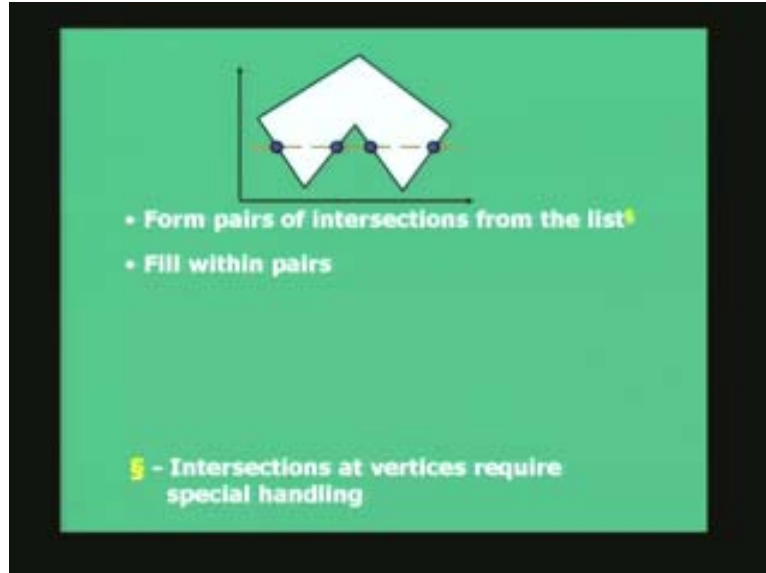
(Refer Slide Time: 00:15:05)



You see here you form pairs of intersections from the lists; they are exceptions which say that if a scan line passes through a vertex then you might have a problem. That means intersections at vertices may require special handling because the vertex is a combination of two different edges. So what you do? Should you take both the edges as intersections or which one should you take? So except the case when a scan line is passing through a vertex which would be could be the end here or somewhere at the middle you may have a problem.

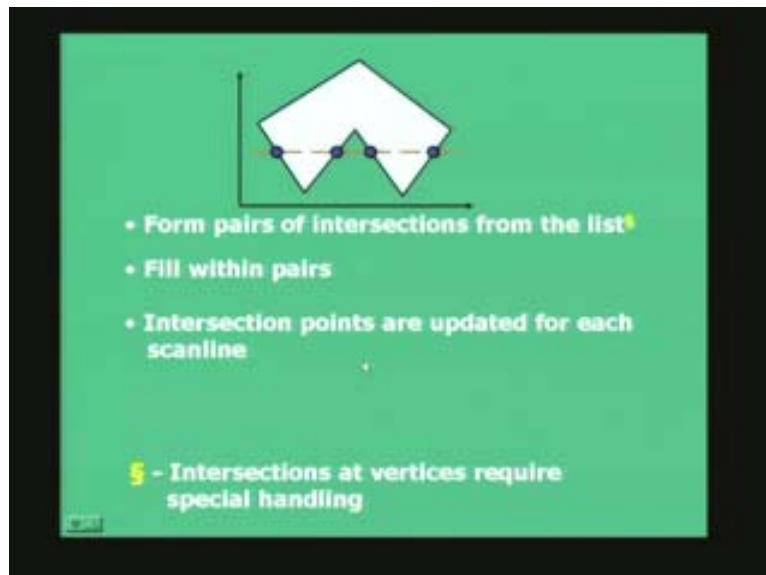
As for example if you take the top scan line it will have basically one intersection at this vertex but you can easily visualize that, that particular intersection is formed by two edges and actually there are two intersections. So we will see later on how to handle vertices because the vertices require special handling at the intersections otherwise typically in general will have pairs of intersections and you form pairs because the numbers of intersections are even numbers. So assuming that the number of intersections is an even number you form pairs and fill within pairs.

(Refer Slide Time: 00:16:01)



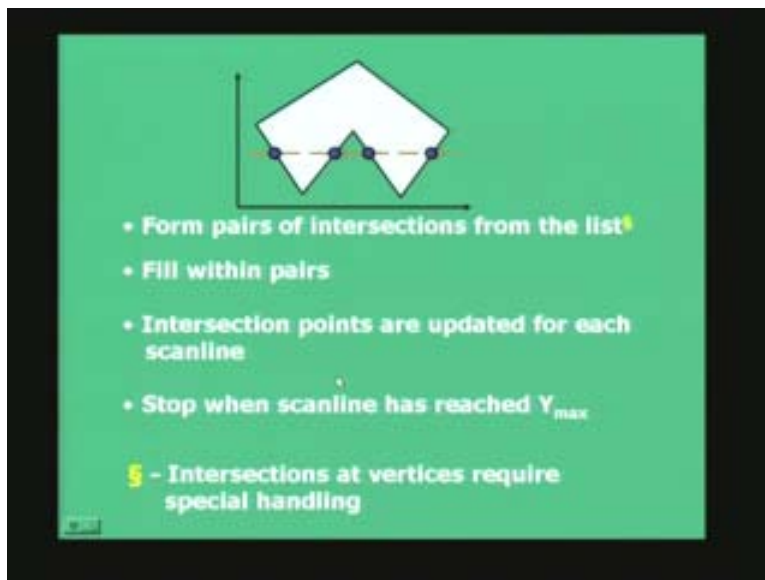
So fill the pixels within the first pair, leave between the second and the third, take the second pair which is between third and four filling pixels in between and you leave everything else which are next to four. So fill within pairs as you can see for this particular scan line or if you take somewhere in the top here you will form one pair. So fill means fill pixels equivalent to the draw pixel command which we have seen earlier in the case of a draw line draw circle we had a draw pixel command x, y. All those pixels which are within this pair or pairs is what you fill up.

(Refer Slide Time: 00:16:39)



Intersection points are updated for each scan line, that you have to do and that is an advantages because we will use a concept which we later on see as scan line coherence or edge coherence where we will find that if you know the set of intersections for a particular scan line which intersects edges at certain points you will find that for the next successive scan line which is at $Y - 1$ or $Y + 1$ the position of the intersection points do not change much. In fact the next intersection point can be obtained by the concept of what we have seen as integer based algorithms of Bresenham's to find the next pixel for the line. Thus, updating intersection points is very easy. The concept used here is exactly similar to the Bresenham's line drawing algorithm where you find the next pixel. Either you are moving from top to bottom or bottom to top does not matter, you find the next pixel of intersection for each edge with respect to the next scan line by a simple update based on Bresenham's algorithm. That is the point here which we were talking about saying that intersection points are updated for each scan line.

(Refer Slide Time: 00:17:53)



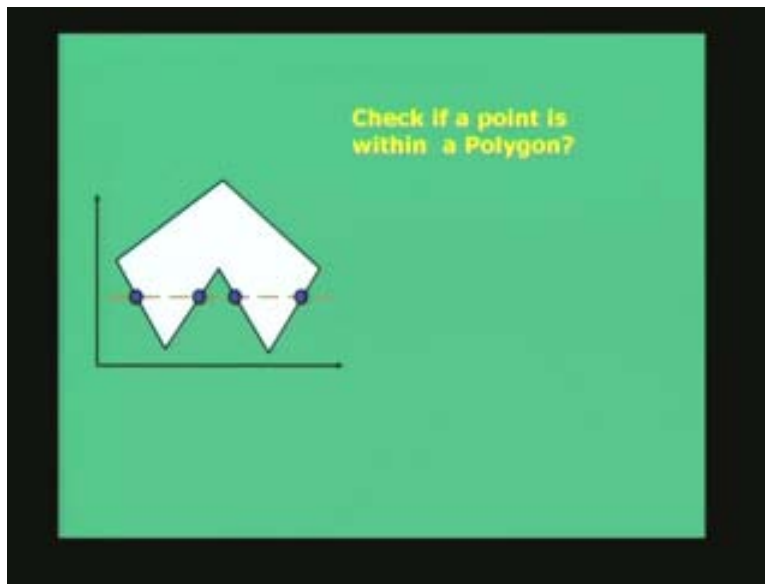
And you stop when a scan line has reached Y_{max} . So you start from Y_{min} and stop at Y_{max} . Y_{min} say if the bottom of the screen is 0 you start at the bottom and finish at the top or vice versa. So you basically start from the scan line Y_{min} and stop when the scan line as they reach the maximum value at Y_{max} . So I roll it back once again, the conceptual steps for implementation of a scan line, polyfill algorithm, find the minimum enclosed rectangle, find the number of scan line and you when know what are the numbers of scan lines for each scan line do the following steps. You have about five or six steps.

The first step is, obtain intersection points of scan lines with polygon edges short intersections from left to right, form pairs of intersections from the list. Of course intersections at vertices require special handling we know that now. Then fill within pairs and then intersection points are updated for each scan line. These are the conceptual steps of implementation of a scan line and you stop the scan line when the scan line reaches

Y_{\max} . So you started at Y_{\min} and of course at each point you do not have to recalculate the intersection points again for a scan line that is interesting.

You can make the algorithm work faster using the concepts of Bresenham's algorithm of looping for the east or north east pixel based on integer algorithm because basically intersection points are all lying on a line. So, if you know how to calculate the pixels on a line based on Bresenham's algorithm and given a pixel picked up at the current point of iteration you can always go to the next pixel by a simple update which we use for Bresenham's algorithm. That is what you need to do as an update rather than calculate a fresh for the next scan line new set of intersections, no. You need to start at some point, typically a vertex or vertices and then keep on updating them using the Bresenham's criteria which makes the algorithm very fast.

(Refer Slide Time: 00:19:56)



Of course there are quite a bit of precautions necessary, the first precaution which we will study now is how to take care of intersections at the vertices. The question is, now we will all first study why the particular point inside the scan line if it is within a pair should be shaded with a pixel and if it is not within a pair it should not be shaded. So what is the check that if a point is within a polygon or not. So we conceptually look into it. If you look at the same polygon, the same scan line, there are four intersections and you see here if you can pick up a point somewhere in between or inside the polygon between first two intersections, let us take intersection one two three and four there are four intersections. There are four intersections particular in this scan line.

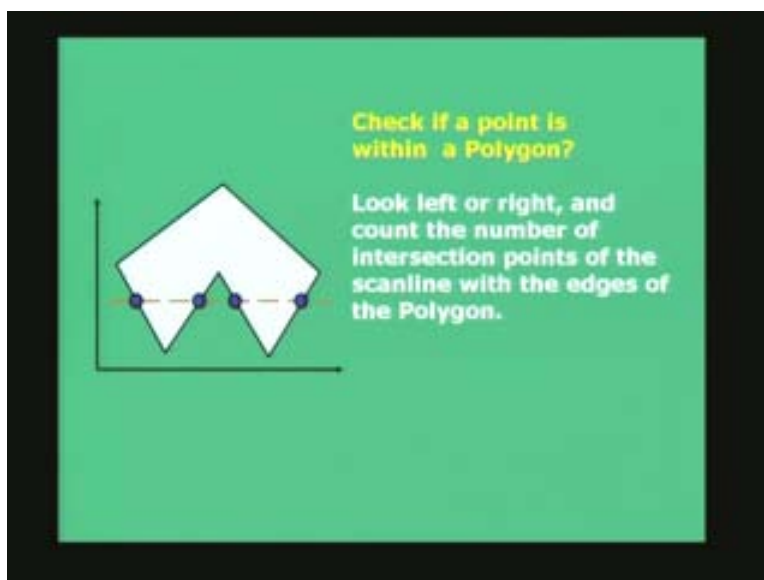
Now take a point which is between the first and the second intersection that is within a pair. Let us say this point is here somewhere. Now if you know the x y coordinates of that point and keep looking towards left or towards right you will find that the number of intersections of the polygon edges with this particular scan line is an odd number. If you look to the left how many do you have on the left? That is one, we have only one

intersection to the left of the point, the cursor is pointing to right now. When you look to the right you have 1 2 and 3. So either you look left or right for this intersection which is at the middle you have an odd number. The same is true for the second pair. If you take a point inside the polygon which is for this scan line which is lying within the second pair of intersections then you will have the same. If you look to the left you have 1 2 and 3 that means there are three intersections. To the right you also have one so it is basically an odd number. That means on a scan line if you select a point within a polygon and count the number of intersections either to your left or right, you can pick up any, any direction left or right and if the count of the number of intersections is an odd number, the pixel is guaranteed to lie within the polygon.

Again there are a couple of exceptions do not worry for the time being. That is the vertices I know but except the vertices, we will see how to handle the vertices anyway but let us only bother about scan lines which are not intersecting a vertex of a polygon. Otherwise you can apply this criterion to check if a point is within a polygon by the following manner. I repeat again, go to that particular point keep looking to the left or right not both, either it is only left or only right and count the number of intersections in that direction. If that count is an odd number then the point is within.

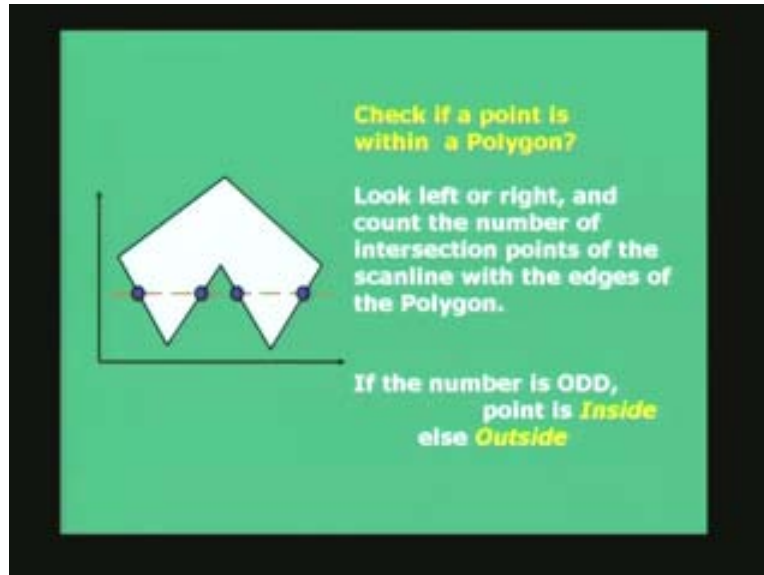
If the count is even, in any direction it is outside the polygon, let's us take an example with the same figure if I take a point in that scan line which is between intersection number two and three somewhere in the background here on this scan line and if you count the intersections to the left or right the number of intersections is a even number. It is basically two on either side. So the number of intersection to the left or right, if it is an even number then the point is on the background or outside the polygon. If the count is an odd number looking either left or either right then the point is within the polygon, remember this, even outside or inside something like a binary decision as we did for the east or north east pixel. We use this concept indirectly.

(Refer Slide Time: 00:23:17)



So look left or right and count the number of intersection points of the scan line with the edges of the polygon. So find out the intersection points of the scan line of the edges of the polygon. Once you do that then you count the number of intersection points to the left or right.

(Refer Slide Time: 00:23:31)



And the conditions that we have just described are very simple. If the number is odd the point is inside else it is outside the polygon. That is very simple, if the number is odd the point is inside otherwise it is outside the polygon.

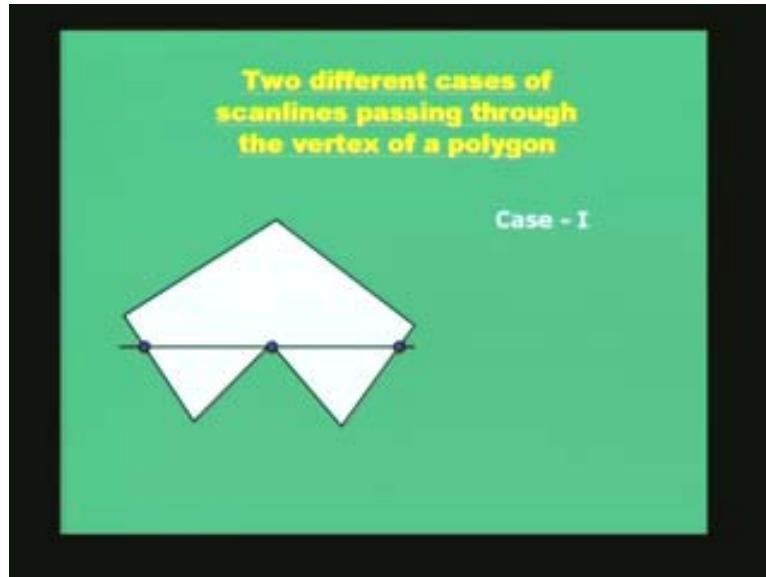
Now the special case, we have to deal with vertices. This is the most important part of the scan line algorithm. Of course there are other small mathematics involved in incrementing the intersection from one scan line to the other but that is within the loop but before you start the loop you actually have to take care of the vertices. Probably otherwise you might land up with odd number of intersection points.

We will see in some cases where the scan line is passing through a vertex you may land up with even number, usually it is even but when passing through a vertex in some cases you will find that the number of intersections is an odd number then you cannot find out pairs of intersection. Remember, you have to find out pairs of intersection and fill within pairs and this concept works because of that inside outside test of a polygon, which is odd and even. That is the basic idea, that odd and even test of the count helps you to form pairs of intersection and fill within pairs and that is how the fill polygon works.

But the other work to be done is, one is of course updating the scan line intersections incrementally by an integer based on Bresenham's criteria. But till before that, before entering the far loop itself you must take it to the vertices in such a manner that in each scan line you always get even number of intersections. And at each scan line as long as you are getting even numbers, somehow if you can make it even and then if the line is

passing through a vertex if you somehow ensure that the number of intersection points are even instead of being odd you can always form pairs and always fill and it always works. So we will find that there are two different cases of scan line passing through a vertex of the polygon and we will see with the case one.

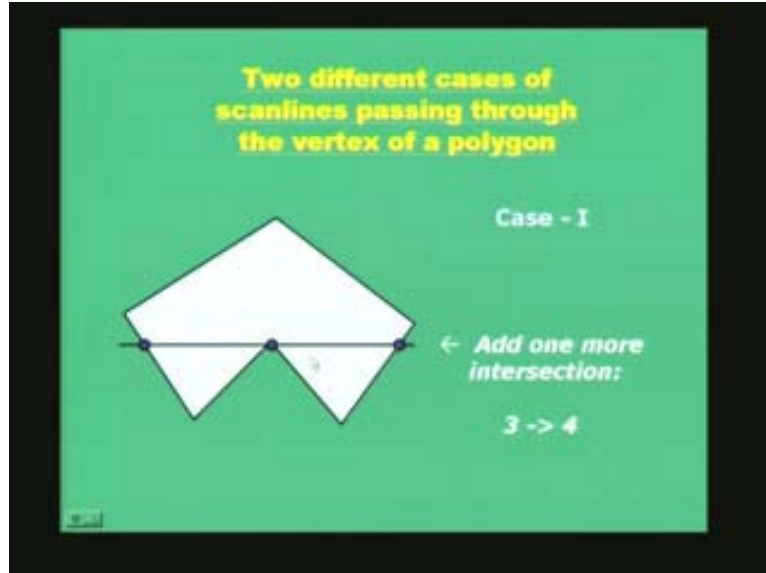
(Refer Slide Time: 00:25:29)



That is the typical example, it shows that the scan line is passing through a vertex of the polygon. And as I was talking before what happens is, the number of intersections of the scan line with the number of edges is showing as 3. But if you carefully look at this number three the center intersection point is the vertex where there are two edges of the polygon which are meeting or intersecting. And hence this count of the number of intersection at this point should conceptually be 2. So, if the central count is 2 minus 1 then it makes it 4. Actually you are having four intersection points but it shows three. And it happens in this particular case when the scan line is passing through the vertex every time.

You take the top scan line, as I was talking about a little bit earlier that it will show only one intersection point because that is again a vertex and a meeting point of two different edges and hence we have 1, but basically it should be 2. Similarly, you should have 1, 2 and 3 here and then a 4. And if you can do that somehow then what you have is you have two pairs of intersection and you can fill. Do not worry about what is done at that vertex but right now you visualize that the number is 3 but it actually should be 4. So that is case one.

(Refer Slide Time: 00:27:05)



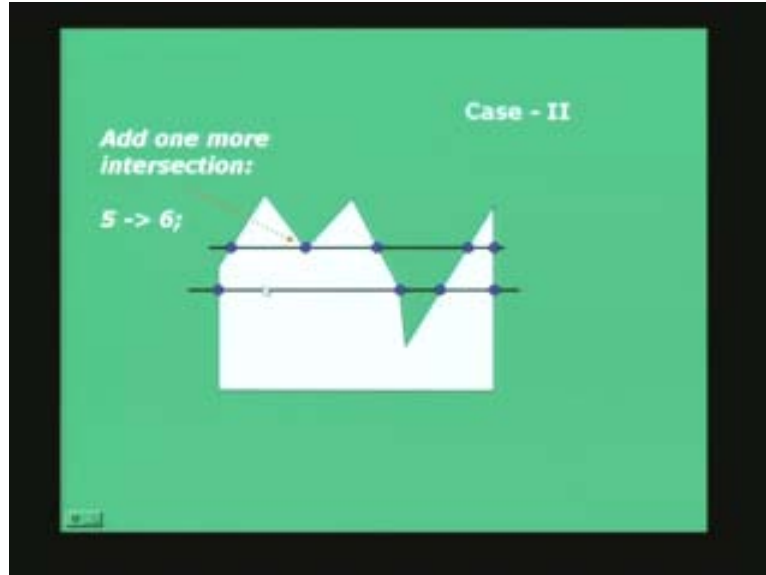
So in this case basically we just put a note here that in such a case of this type of a vertex we should add one more intersection where the scan line is passing through and make the number of intersections up from 3 to 4. Just increment the number of intersections from three, you can see three intersections but if we consider two only at the center, the number of intersection should be four. **I hope the idea is clear**, this is the first case of a scan line passing through a vertex.

(Refer Slide Time: 00:27:40)



Let us look at the second case. Well I have to take a different polygon to illustrate a slightly different concept for intersection of a vertex with a polygon.

(Refer Slide Time: 00:27:54)



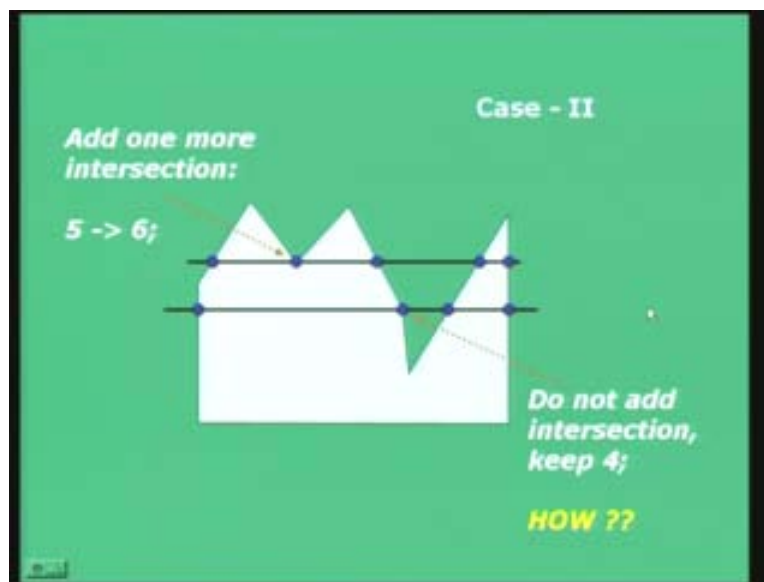
And what we will see here is the first scan line which shows that there are 1 2 3 4 and 5. There are basically five intersections and this concept is the same as the case one. Some concept is same in some cases as case one where you just add 1, you add just 1, increment the number of intersections and make the number of intersections from 5 to 6. In this case if you make from 5 to 6 what happens? Let us sequentially look at the numbering; 1, 2, 3, 4 and 5. So, that is an odd number and you cannot form pairs. If you put the number of intersections at this point as two then you can have 1 and 2 as one pair, a 3 and a 4 as another pair and a 5 and a 6 as another pair. So you have three pairs, you basically have the number of intersections as 6 which is an even number and you can form hence three pairs of intersections. And as long as you have three pairs of intersections you can fill in three different parts of this scan line from 1 to 2, 3 to 4 and then 5 to 6. That is how adding one more intersection works here.

Look at this particular case, this is very very interesting, the second scan line. You see the number of intersections is; 1, 2, 3 and 4, it is anyway an even number. What is the problem? It is because only when you have a odd number you should start worrying. You have an even number so you can form two pairs and fill. There should not be any problem for the second scan line. But why is it drawn? You can easily guess that one of the intersection points is a vertex of a polygon but still the number of intersection points is an even number. It is very very interesting. So you have 1 2 3 and 4 even number of intersections, you can form two pairs but the vertex is an intersection point. Intersection point is a vertex of the scan line. And if you apply this same logic as we have done in the previous two cases and add one more intersection point from four you will land up with five intersections. Then you have an odd number. You cannot add 1 2 and 3 here 4 and 5, you will have odd number of intersections. This is a very peculiar case where we just say that the number of intersections should be an even number in general.

When you are passing through a vertex you have the number of intersections being odd. So you increment and make it even and form pair; that is fine. But the third case we are seeing, a very very special case where the scan line is passing through a vertex but the number of intersections is even. Even though it is passing through a vertex we have even number of intersections.

Now, if you follow the previous logic that whenever you find the vertex please add on intersections. You are making the number of intersections from even to odd and then you cannot run the algorithm because you cannot form pairs out of an odd number, odd number of intersections. So you have to do a special arrangement here to keep the number of intersections even, as an even number in this case, even though it must be an even number even though the scan line is passing through a vertex. So come back to this figure. As we are saying there are one two three and four keep the even numbers. So this is a special case where we do not have to add the intersection.

(Refer Slide Time: 00:31:34)

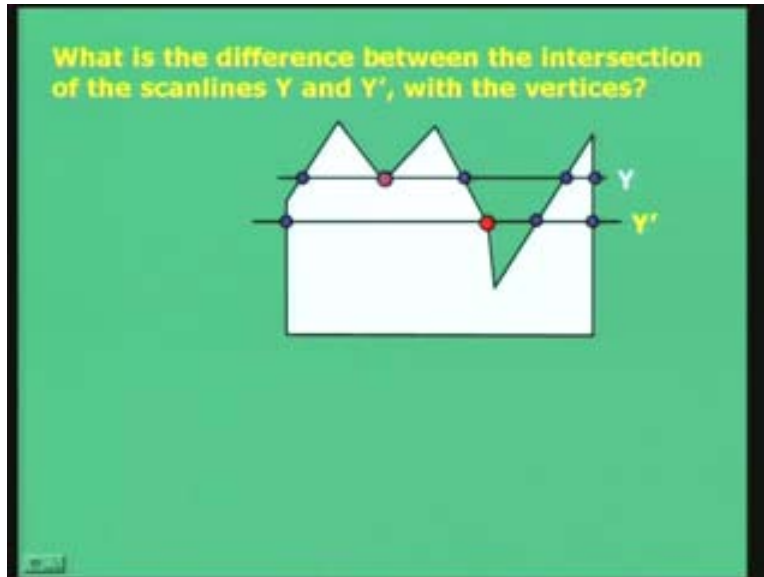


You keep the number four here as the number of intersections, keep four means you do not have to add one and make the four as five as you have done here by adding one and making five as six intersections for the top scan line. For the bottom scan line you have to keep the number of intersections as four. But how this is possible? Because if you put forward a theory and a programming step inside an algorithm, is that algorithm step inside a program basically which says that whenever you find a vertex or a scan line passing through the vertex just add one more intersection, that will make your odd number even. But if the total number of intersection itself is even you cannot add to make it an odd number. So you have to keep the number of intersections even and that is the tricky part in the algorithm which we will see next in the remaining part of the lecture today.

How to do this?

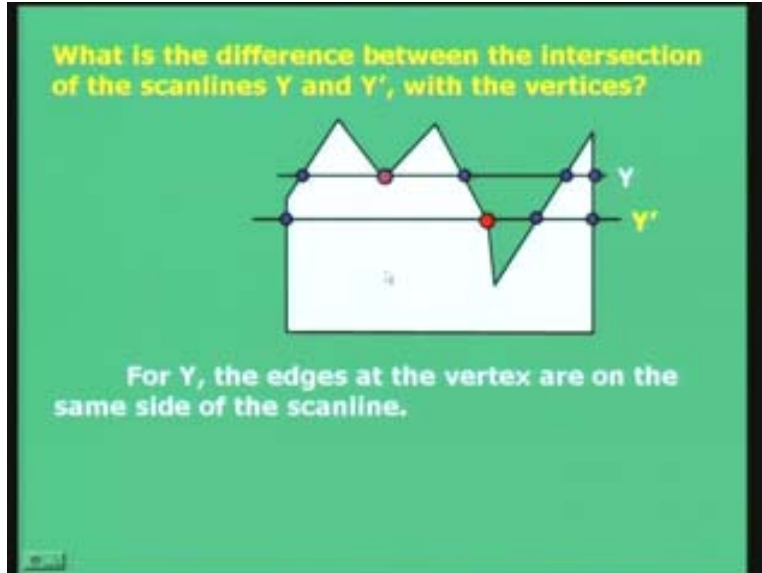
You do not need to add an intersection point here but in this case you need to add an intersection point here. You can already see that the vertices are of two different types. This is what we exploit to find out that, if this is the type of the vertex we add. If this is the type of vertex we do not add. Add means, adding the intersection point to the scan line. So how do you quantify this? Quantify that this is one type of a vertex and this is the second type of vertex? This is what we are going to study next.

(Refer Slide Time: 00:33:01)



So what is the difference between the intersection of the scan lines Y and Y prime with the vertices which we have been discussing now? I label the two scan lines as Y and Y prime. And we know from the previous figure, this is the same figure but only the vertices have been labeled with the slightly different color wherever the scan line is intersecting that is number one and the two scan lines are being labeled Y and Y prime. We know that the intersection of scan line Y with the vertex needs to increment the intersection by 1 such that out of five intersections or from five intersections we have six. Whereas in the case of the scan line Y prime where it intersects the vertex in this red color dot we do not need to add the intersection point and we need to keep the number of intersection points as four.

(Refer Slide Time: 00:33:58)



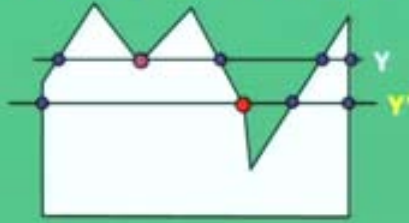
So it means, for the scan line Y, for Y means scan line Y, the edges at the vertex are around the same side of the scan line. We are trying to distinguish the vertex here of scan line Y and the other vertex which is scan line Y prime intersection. Remember, all the other blue dots are intersection points of the scan lines with the edges of the polygon. These are not vertices anymore. None of them, these blue dots is the vertex of the polygon. These are edges, intersection of clean edges with the scan line.

The two non-blue dots one is about purple in color and the red ones are the two intersection points which are vertices of the polygon and intersection point of the scan line with the vertex is what is specific in beginning of the problem. So, if you look at this vertex of scan line Y which intersects at this point in purple color, top scan line Y you will find that the adjacent edges which forms this vertex or meets at this vertex are on the same side which means either top or bottom.

In this case they are on the top side of scan line Y, it could be at the bottom as well but they are on the same side. So that line which says that, for Y the edges at the vertex that means the two edges which is meeting that vertex which are those edges you are only bothered about and those edges at the vertex are on the same side of the scan line. Since they are on the same side of the scan line you need to add the intersection point if the edges are on the same side. Just visualize it or note this down, whereas for Y prime the edges are on either sides or on both the sides of the vertex. One is on the top and another is at the bottom.

(Refer Slide Time: 00:35:43)

What is the difference between the intersection of the scanlines Y and Y', with the vertices?



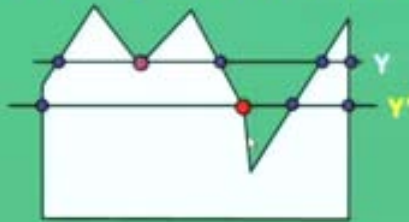
For Y, the edges at the vertex are on the same side of the scanline.

Whereas for Y', the edges are on either/both sides of the vertex.

If you take the scan line Y prime and see the intersection of this red marked by a red dot of this vertex of the polygon with Y prime, the two edges which meet to form this particular vertex through which the Y prime scan line is passing through, the two edges which meet to form this vertex are on two different sides of Y prime. One is on the top and one is at the bottom whereas for the scan line Y both the edges were on the top. They were on the same side, I mean at the top but it could be at the bottom also. So whether it is bottom or top the case is the same but this is a different case where we will say that the edges are on both sides of the vertex.

(Refer Slide Time: 00:36:36)

What is the difference between the intersection of the scanlines Y and Y', with the vertices?



For Y, the edges at the vertex are on the same side of the scanline.

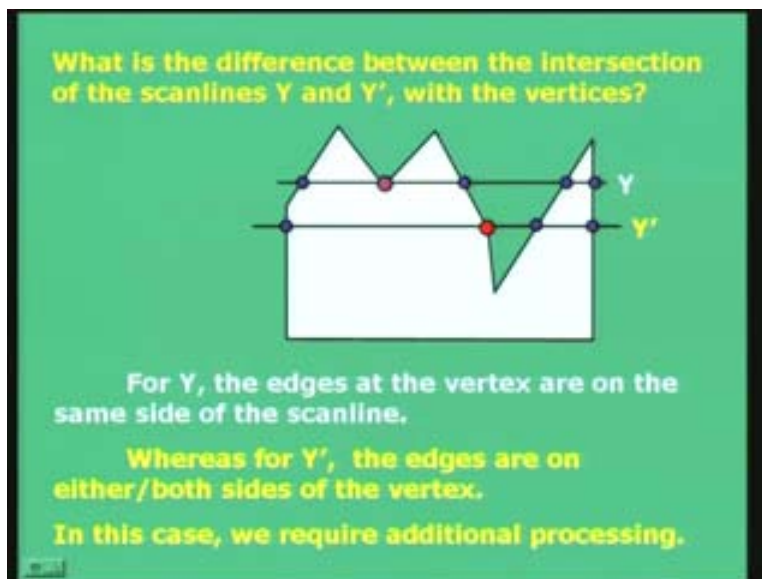
Whereas for Y', the edges are on either/both sides of the vertex.

In this case, we require additional processing.

And in this particular case when edges are on either side or both sides of the vertex we require some additional processing to ensure that we do not increment the number of intersections. **I hope it is clear,** we are talking about intersection of scan lines with vertices and whenever there is a case that the two edges which meet to form the vertex and the scan line is passing through that vertex. So, if the two edges are on the same side on the top or it could be at the bottom two edges could be at the bottom or the top of the scan line then you increment the intersection and visualize that intersection to be of intersection of two edges with the scan line.

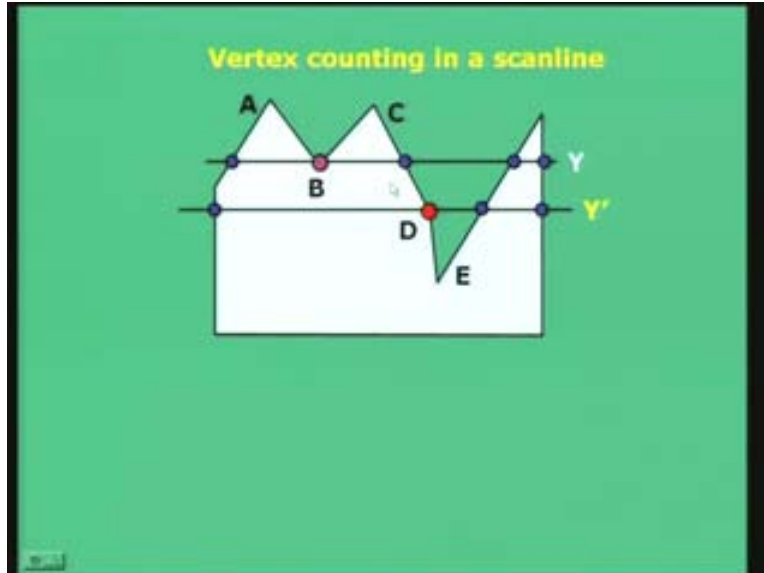
Just increment the number of intersections that will make the number of intersection an even number from odd then you can form pairs and fill, that is not a problem. In the case of the second line Y' , we had a scan line and then of course there was an edge coming and meeting the scan line and then going down, so there was a vertex, there was a vertex and a scan line came, it should come from this side and go down it does not come and go up, the polygon vertex is such that the edge does not come and go up but it travels into this scan line and goes down which is possible. In that case we do not, we have to do something where you keep that intersection only as one. Not in the not in the previous case where you increment and visualize it or consider it to be two intersections for scan line Y . But in Y' you take that intersection as one only. So I go back to this particular figure.

(Refer Slide Time: 00:38:11)



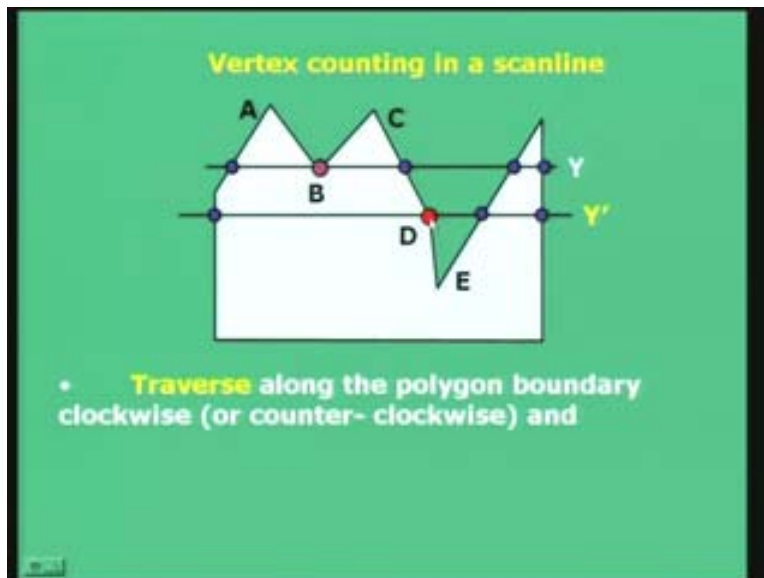
So for this intersection for scan line Y , I must consider this to be two intersections whereas for this particular red color intersection for scan line Y' I must consider it as only one intersection. How to do this additional processing such that I can stop making my algorithm, consider this to be one intersection and this to be two intersections. Vertex counting in a scan line is what we are bothered about.

(Refer Slide Time: 00:38:42)



Just to elaborate once again I have labeled the vertices A B C D and E. Let us say there are five vertices, in fact this scan line Y passes through point B which is a vertex and scan line Y prime passes through point D which is another vertex of the polygon.

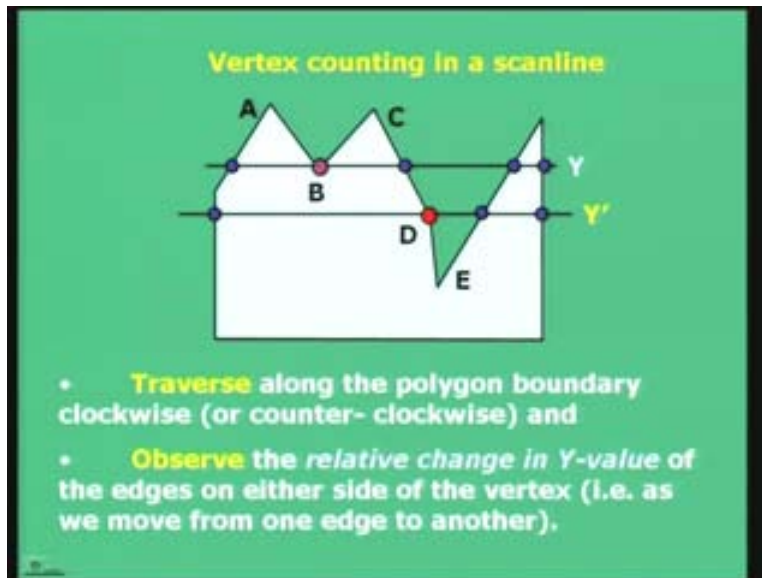
(Refer Slide Time: 00:39:01)



If that is so, to find out whether the edges are on the same side or on both sides of the polygon, you traverse along the polygon boundary clockwise or counter, you have to be consistent, you either move in a clockwise type of arrangement or you can move in anti clockwise arrangement, there is absolutely no problem. But you must be considerate, you cannot move clockwise for half of the algorithm and then go back. So you can trace the

polygon boundary either anti clockwise or you can go clockwise as well. So follow that, to traverse along polygon boundary clockwise or counter clockwise, the typical example of clockwise could be in this case A B C D E and so on, that is clockwise. Anti clockwise could be the other way where it is E D C B A and so on. Of course I have labeled only five vertices out of about eight. About nine vertices exist in this polygon and I have labeled only five which are necessary for my vertex processing.

(Refer Slide Time: 00:40:20)

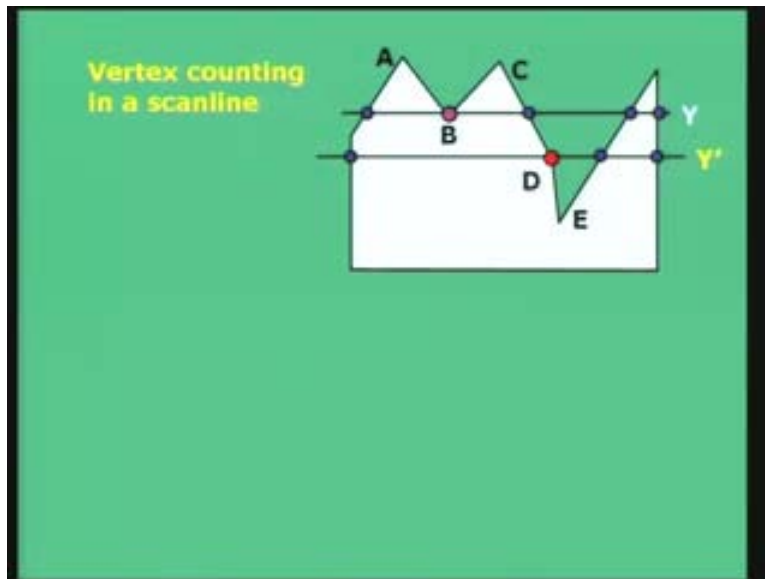


I hope this concept is clear about what you mean by traverse along the polygon boundary. We have to do this, clockwise or anti clockwise before the iteration starts for each scan line what I will call as a preprocessing step or stage before the iteration for each scan line starts, you traverse along that and observe, what you observe? Observe the relative change in Y value of the edges on either side of the vertex. That is, as you move from one edge to another, from edge AB to BC you find out what is the relative change in the Y value with respect to B. With respect to B this vertex are A and C on one side or not. So you actually have to look at the vertices only and find out if A and C, the relative change, that means $A - B$ if it is positive $C - B$ should also be positive. That is another way of looking at it.

So you look at that change and if that change is on one side then of course you had an intersection whereas when you are traveling from C to D to E, which C that the change is unidirectional. That means you are coming down in Y coordinate from C to D and then the Y coordinate also keeps going down from D to E further. So that's one type of a change where you do not need to increment and in the case of B, A to B and back to C, the Y coordinate has fallen down and then gone up. So that is how you find out whether the edges AB and BC with respect to the vertex V are on one side of the scan line or for the scan line Y prime the edges CD and DE are on two different sides of the scan line. So you basically have to compare the Y coordinates of the three vertices A B C to test for vertex B. And for vertex B compare the Y coordinates of vertex C D E. When you

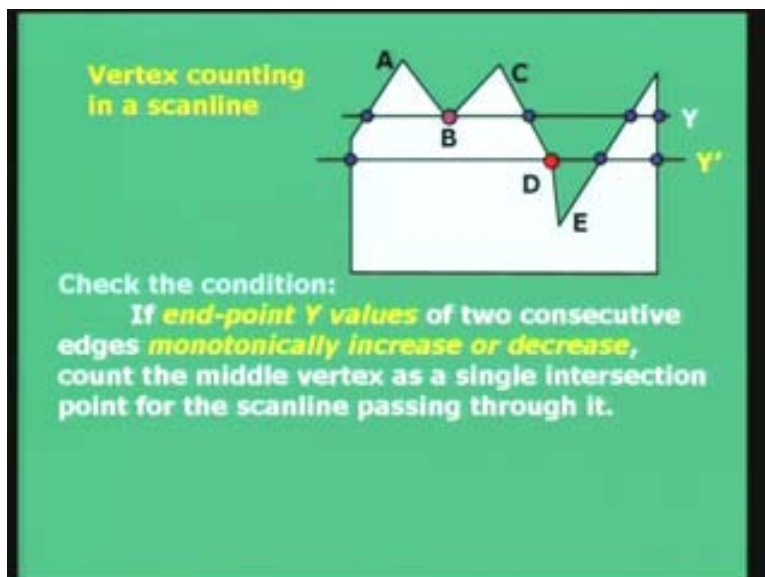
compare the Y coordinates of the vertices you can easily find out whether the edges or the vertices themselves are on the same side of the polygon. Both could be on the top or both could be at the bottom or each one is on the other side. That means the two vertices are respectively on opposite sides with respect to the scan line and this is the relative change in Y value which you have to keenly observe.

(Refer Slide Time: 00:42:21)



So vertex counting in a scan line that is we continue with the same figure to discuss this.

(Refer Slide Time: 00:42:26)



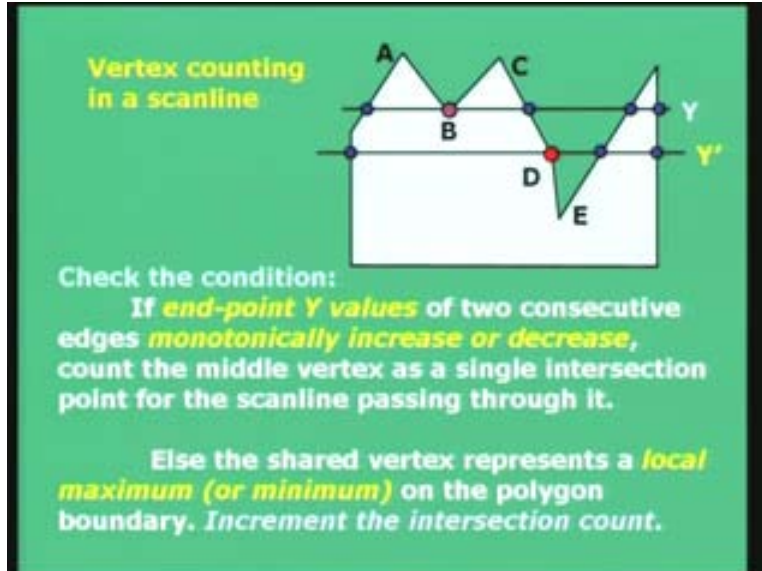
Check the condition that if endpoint Y values of two consecutive edges monotonically increase or decrease. Then you count the middle vertex as a single intersection point for the scan line passing through it. This is the condition which is true for intersection of vertex D with scan line Y prime. Scan line Y prime is passing through vertex D and what do you do? You check the endpoint Y values of two consecutive edges. That means you look into the Y value of vertex C, you look into the Y value of vertex D, you look into the Y value of vertex E. What will you find? For these two consecutive edges the endpoint Y values will be monotonically decreasing if you are traveling from C to D to A for clockwise traversal along the polygon. Or even if you do counter clockwise, that means you are traveling from E to D and then to C, you will find that the endpoint Y values for these two consequent edges are monotonically increasing from E to D and then to C.

I am assuming that the origin is the left bottom on corner of the screen and Y value is increasing upwards, X value is increasing to the right, that is what is my assumption. And if you do that, when we travel from E to D to C the Y value has been increased monotonically or if you travel clockwise from C to D to E the value has been decreased monotonically. If that is so then you count the middle vertex D as a single intersection for the corresponding scan line Y prime. And that is enough for you to show that you have even number of scan lines. So, for D you count it as one intersection and you have even number of scan line intersections for that scan line Y prime and you can form pairs. Once you have even numbers you form pairs.

This condition of monotonically increasing or decreasing Y values, remember the term, monotonically increasing or decreasing Y values will not be true for vertex B because as you check the Y coordinates from vertex A to vertex B to vertex C the values will go down and then go up if we are moving in a clockwise. If we are moving counter clockwise from C to B to A the value will decrease and then increase again.

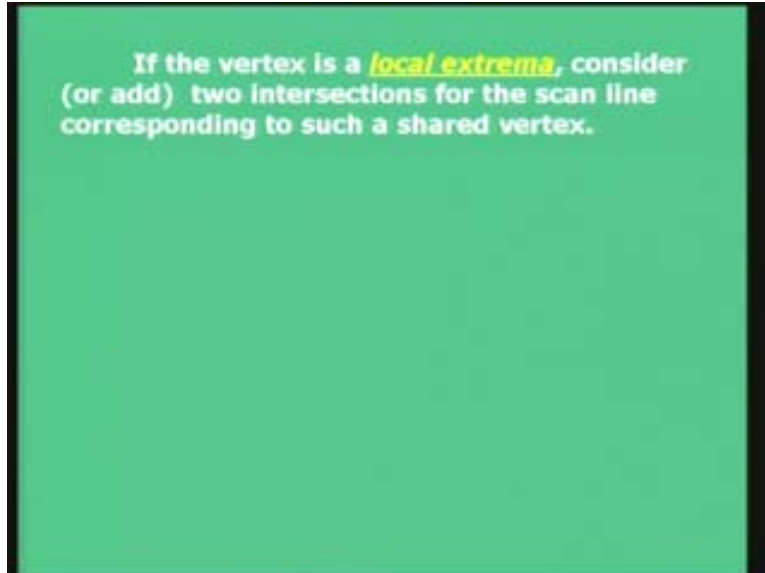
So, this condition of endpoint Y values monotonically increasing or decreasing will not be satisfied for the vertex B for scan line Y. It will only satisfy for the vertex intersection D for scan line Y prime. So hence, I keep and I say that when this condition is satisfied for vertex D, I consider only one intersection whereas for vertex B since my condition is not satisfied I increase the number of intersections by 1 and hence odd number of intersections for scan line Y becomes an even number basically from exactly 5 you go to 6 whereas for scan line Y prime the number of intersections remain exactly at 4 because we are not incrementing.

(Refer Slide Time: 00:45:44)

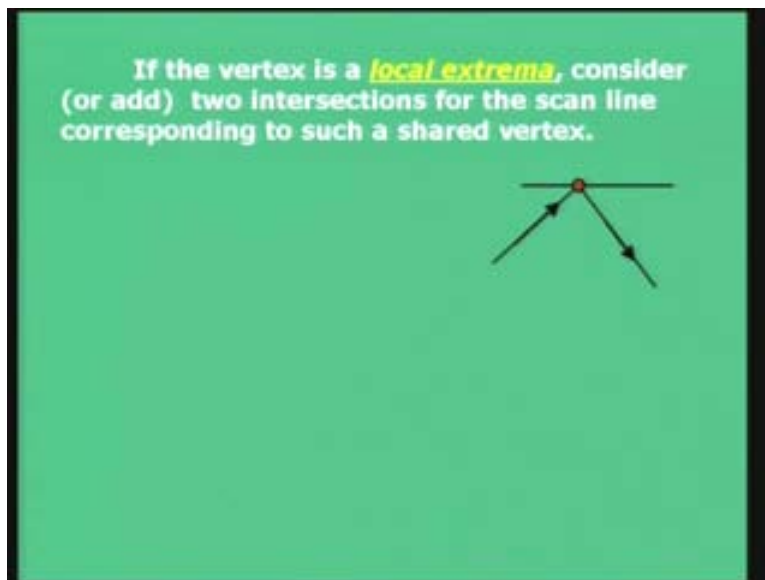


Else, the shared vertex that is this first condition was for D and it is valid for vertex B when the shared vertex represents what we call as a local maxima or minimum, there is a term called extremum which could also be used, it is a local extrema which we will use now, often used in literature and we have a local minimum or maximum on the polygon boundary and we increment the intersection count and this second paragraph is true for B vertex, the first paragraph condition holds good for vertex D. And the else condition means the top condition for monotonically increasing or decreasing, it does not satisfy for vertex B it only satisfies for vertex D. So I do not increment for D but I increment the number of intersection by one more for B. So instead of 5 I have now 6, I can form three pairs, I can fill in the second case for one line scan line Y prime, I anyway have four intersections, I keep four intersections because I am not incrementing and when I have four intersections I can form two pairs and I fill within pairs to obtain the region to be filled in within the polygon for scan line Y and Y prime respectively. **I hope the idea is getting clear.**

(Refer Slide Time: 00:47:03)

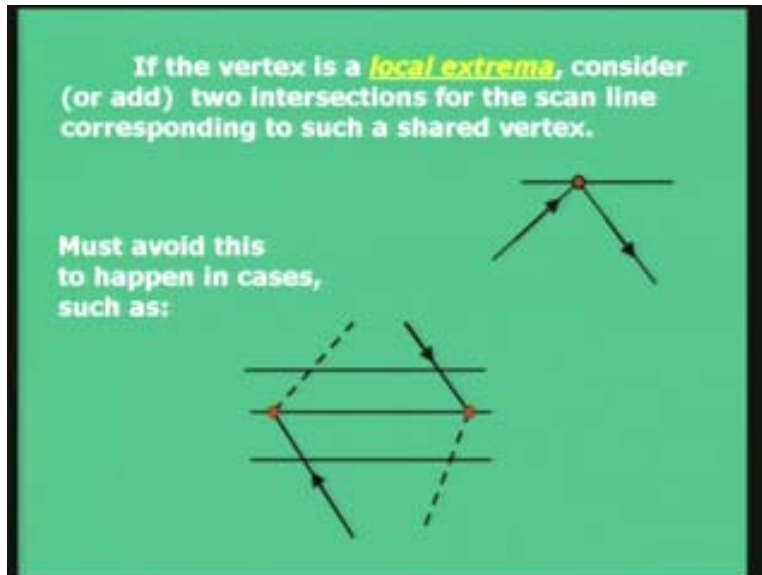


(Refer Slide Time: 00:47:06)



So if a vertex is a local extrema, this is an example of two edges meeting at a vertex on one particular side and both on one side and in fact it is below the scan line. In some of the previous examples both had been on the top or some of them had been on the bottom. So, if the vertex is a local extrema, that means you either have a maximum or minimum at that point, something like a peak. Consider or add two intersections for the scan line corresponding to such a shared vertex. The shared vertex is a local extrema which basically means the condition that both the edges are on the same side of the scan line then we increment by 1 or consider that vertex to be two intersections.

(Refer Slide Time: 00:47:44)



But however you must avoid this case when such a thing happens that the shared vertex is not a local extrema. That means whether you are traveling counter clockwise or clockwise, the edges shows that they are on either side of the polygon.

In this figure there are basically three scan lines and you are now at scan line number I and scan line I minus 1 is shown in dashed one. There is a vertex intersection and this is the case of a vertex where you should not add an intersection and consider it to be only one intersection point. So you must avoid this case to add an intersection. If you add you will have a problem because you will change the number of intersection points from even to an odd number. So you must not add in the second case, in the first case of a local extrema you have to add.

(Refer Slide Time: 00:48:32)

To implement the above:
While processing non-horizontal edges (generally) along a polygon boundary in any order, check to determine the **condition of monotonically changing (increasing or decreasing) endpoint Y values.**
If so:

To implement the above the following thing is done while processing the non-horizontal edges generally because we do not process horizontal edges. We will see why it is not necessary later on. Thus, while processing non-horizontal edges along a polygon boundary in any order, check to determine the condition of monotonically changing that means increasing or decreasing endpoint Y values. So you try to observe while processing the edges and keep checking at the vertex for this monotonically changing values of endpoint Y values.

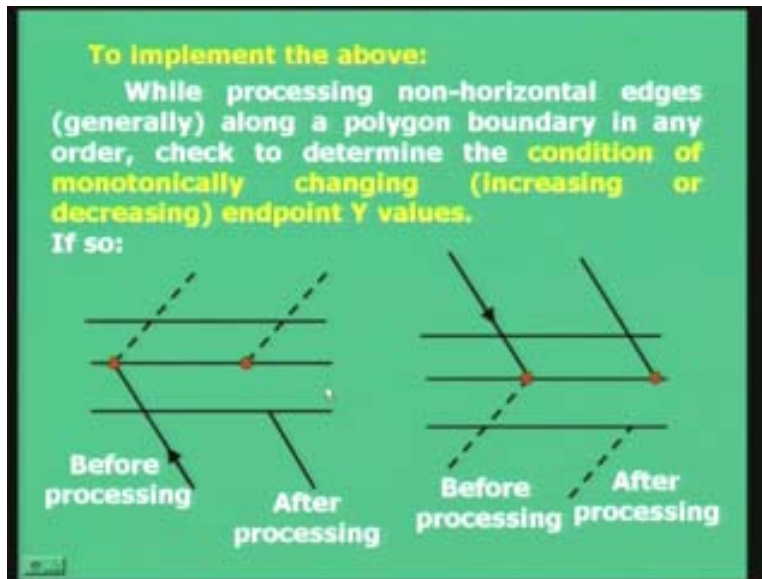
(Refer Slide Time: 00:49:05)

To implement the above:
While processing non-horizontal edges (generally) along a polygon boundary in any order, check to determine the **condition of monotonically changing (increasing or decreasing) endpoint Y values.**
If so:

Before processing After processing

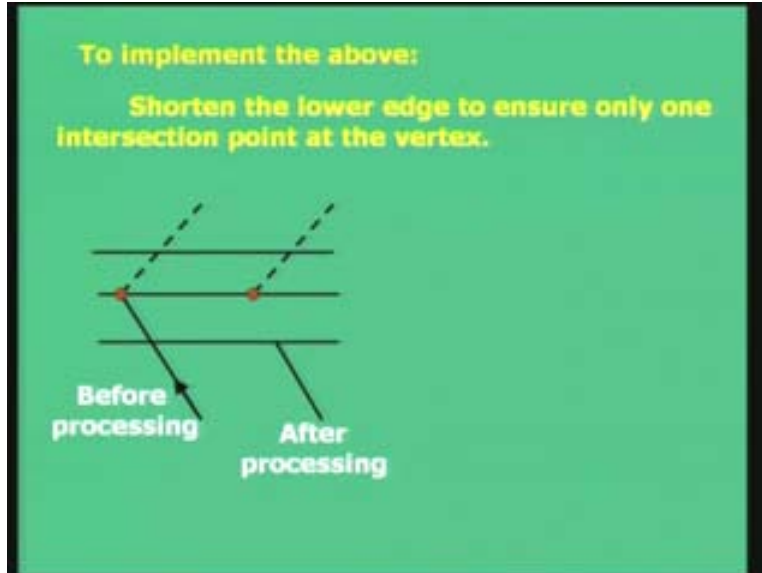
And if you do so then what the algorithm basically says is, you treat this vertex in such a manner that you reduce one of the edges by one particular scan line. If this is the vertex before processing where two edges were intersecting what you do after processing is reduce this edge by one unit in length and allow the edge to only stop here and not to go up to this point. And that is sufficient to provide you that this point is not a vertex but the end point of one edge of a polygon and you will intersect the scan line only with one edge but could not **double it**. This is a sort of processing which could be done.

(Refer Slide Time: 00:49:46)



This is another scenario, either you are traveling counter clockwise or clockwise or whatever the case may be, when you find such monotonically changing values where the edges are on both sides you reduce the edge for processing. So, after processing you reduce the edge when you are going from top to bottom, or bottom to top, it does not matter, you reduce the edge on one side and that will show that the scan line has only one intersection point.

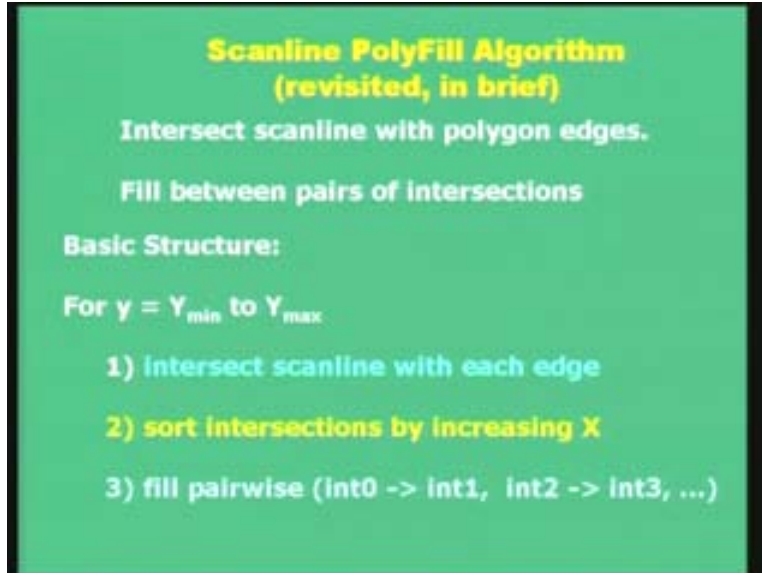
(Refer Slide Time: 00:50:15)



So it basically says that the shorter and the lower edge is to ensure that you have only one intersection point at the vertex. So before processing you had both the edges meeting at this point. When you have shortened one of the edges after processing then at this vertex you have only one edge coming, the other edge does not reach the vertex. And henceforth you will land up with one intersection point or not. That is what you do.

We will wind up this class and continue with the remaining order where we revisit this scan line polyfill algorithm in brief. If you visit it, intersect scan line with polygon edges filled between pairs of intersection and that is the basic structure from Y_{\min} to Y_{\max} . That is from scan line top to bottom, intersect scan line with each edge, short intersections by increasing X and fill pair wise.

(Refer Slide Time: 00:51:06)



So these are the three important steps of the basic structure of the scan line filling algorithm. But there is a lot of overhead to handle the vertices. So while you actually intersect scan line with polygon edges filled between pairs, that is the basic idea or the basic structure of the algorithm. But what you basically need to do is do a lot of preprocessing. Do lot of preprocessing prior to this such that you can handle this vertex problem inside the loop. Inside the loop the algorithms says that you just keep on incrementing the scan line from one to another, from either top to bottom or bottom to top whatever the case maybe. And if the vertex problem has been handled earlier by shortening the edges as may be necessary in monotonically increasing or decreasing Y values. If that is already been done then what it basically means is you have to just increment the scan line intersections and go from one scan line to another along an edge, each vertices come up with new vertices and so on and that is how you basically implement. But there is a lot of preprocessing before which one has to do earlier in terms of vertex manipulation. That is how the incremental scan line based algorithm can work smoothly and it is guaranteed that at each scan line you get even number of intersections, form pairs and fill.

So we can wind up this class where we will start again from the next class from this point where we will revisit the polygon algorithm in brief. The basic idea is intersect scan line with polygon edges and then fill between pairs of intersection. And the three steps is that from scan line Y_{\min} to Y_{\max} you intersect scan line with each edge, short intersections by increasing X and fill pair wise.

Actually again I must repeat here, there is an overhead prior to this loop between scan lines that is number one and then within each loop you are not freshly calculating the intersections, you never do that. You always increment along an edge to get the next intersection point from the current point and use properties similar to Bresenham's algorithm midpoint criteria, similar to that which are called edge coherence and scan line

coherence. We will start with this basic structure and loop into this concept in simplified calculations to compute these intersection points very fast and also look at the algorithm if we will handle the vertex problem.

We know how to handle that vertex problem. We will see what data structure is used to store the polygon vertices and handle these problems of vertices by shortening edges and then of course at each scan line from one to the other, how to increment the intersection points and get into the next scan line intersection points. This will be the main treatment for the next class for scan line filling algorithm. We have studied the concepts and the geometry behind it today and we will move towards the algorithm in the next class. Thank you very much.