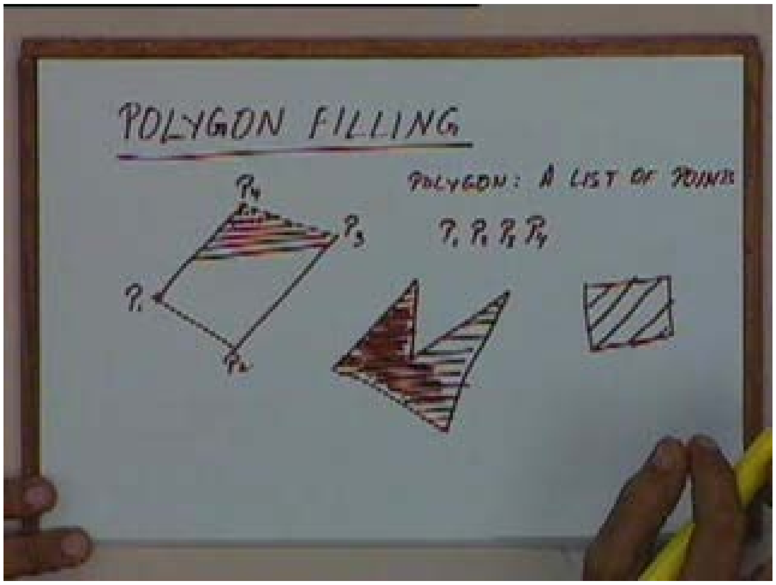
Computer Aided Design
Dr. Anoop Chawla
Department of Mechanical Engineering
Indian Institute of Technology, Delhi
Lecture No. # 05
Polygon Filling

Today we will be talking of some algorithms for what is termed as polygon filling. Last time we had seen some algorithms for drawing lines between two points.

(Refer Slide Time: 00:01:08 min)

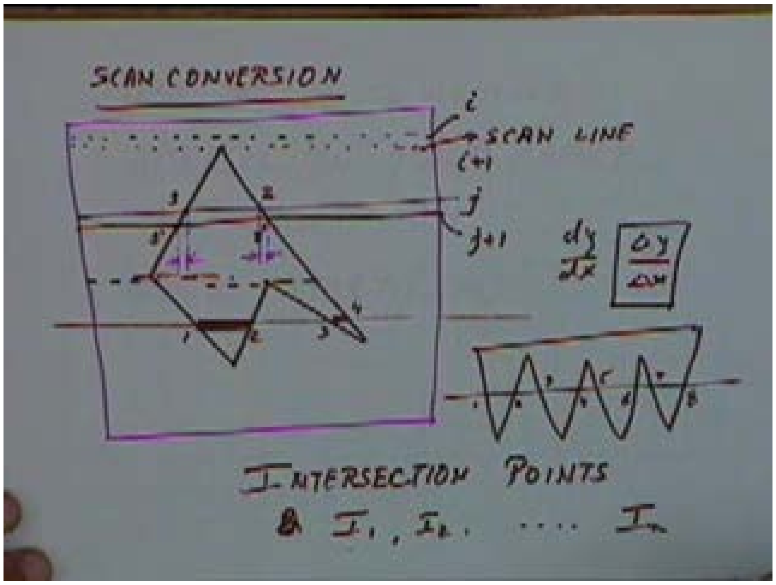


If you want to draw a line between two points, we can use either the DDA algorithm or the Bresenhem's algorithm. Now if you have to fill a polygon let's say if you have a polygon like this, filling the polygon means all the pixels that are inside the polygon they should be highlighted and all the pixels outside they should remain off. So it is essentially some form of either shading or you are familiar with hatching in drawings. If you want to hatch a given area or closed area that is done by using filling algorithms. So filling essentially means that if I cover this polygon row by row, if I take the first row of pixels these pixels which are inside they will be on and these pixels will be on, these pixels will be on and so on. So we have to decide which pixel is inside and which pixel is outside.

What we have given for this is a polygon. And what is a polygon? A polygon will be defined as a list of points. So if you have polygon like this, this would be represented as a list of points given like this. So if you have a list of points then a polygon filling algorithm will highlight all the points which are inside the polygon. The polygon need not be a simple convex polygon like this, it can also be a concave polygon like this. So in case of a concave polygon similar thing would be done. For the time being we will see algorithms which are going to fill the area by a set of highlighted pixels. We are not going to see algorithms which are going to hatch a given area.

If you have a let's say area like this and you want to do hatching, we will have to draw lines let's say at an angle of 45 or at some angle or some other hatch pattern inside this closed area. The same polygon filling algorithms can be modified for this purpose also. We won't go into that for the time being. Initially we will see algorithms for filling a area which means highlighting all the pixels and getting a complete filled or shaded view of this polygon. The type of algorithms that we will see in this are what are called as scan conversion algorithms.

(Refer Slide Time: 00:04:06 min)

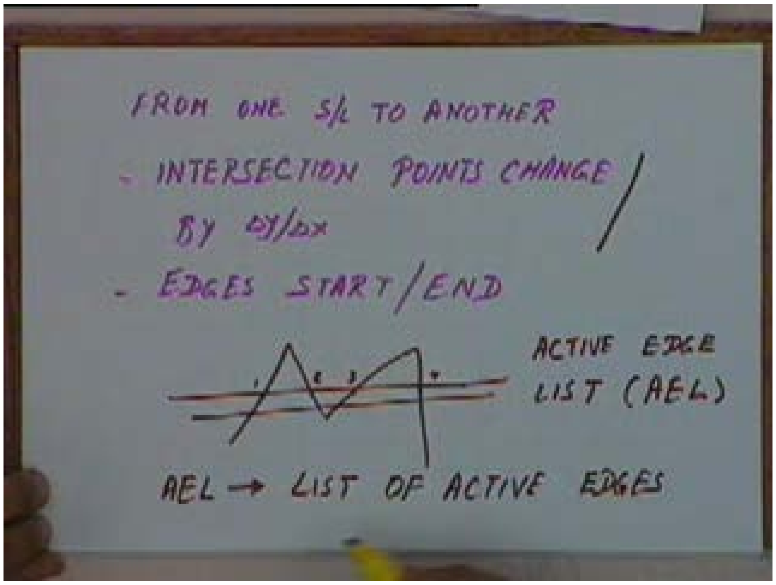


In these algorithms if you have a screen like this, we will start from the top row. Within the top row all the pixels that are there inside the polygon they will be displayed. From that top row we will go onto the next row and so on. So we will go through all the pixels but row by row. Row of pixels let's say this is a row of pixels is normally referred to as a scan line. So our job is now being that if you are talking of a particular scan line let's say you are talking of this scan line, in this scan line you have to find out which pixels or which sequence of pixels are going to be displayed. In this case we consider this point one and point two, all pixels from point one to point two will be displayed. If you are talking of a scan line over here, all pixels between 1 and 2 and between 3 and 4 would be displayed, so this portion. So often we can have on a single scan line, we can have more than one set of pixels being displayed because we are talking of concave polygons.

Now if we notice, as we move for one scan line to a second scan line, let's say initially we are at this scan line. This is my scan line number i and this is my scan line number i plus 1. What can happen as I go from one scan line to the second scan line? In scan line number i, no pixels are inside the polygon but on scan line i plus 1 we have a vertex which is there on this scan line. If I am on this scan line let's say which is scan line number j and I go into a new scan line which is scan line number j plus 1, this point one is being modified to one prime then this two will be modified to two prime. Initially from 1 to 2 all the pixels were on, now from one prime to two prime all the pixels will be on. What is the difference between lines j and j plus 1? The point one has been modified by an amount equal to the slope of this line.

The difference between 1 and 1 prime is this amount and similarly the difference between 2 and 2 prime is this amount. So in a case like this when we move from one scan line to the next scan line, the position of the edge shifts by an amount equal to dy by dx or delta y by delta x. So as we move from one scan line to a second scan line, one of the two things happen. First is for all intersection points, the intersection point would shift by an amount delta y by delta x or the other thing that can happen is as we move from one scan line to a second scan line new edges can start or an edge can come to an end.

(Refer Slide Time: 00:09:06 min)



FROM ONE S/L TO ANOTHER

- INTERSECTION POINTS CHANGE BY $\Delta Y/\Delta X$
- EDGES START/END

ACTIVE EDGE LIST (AEL)

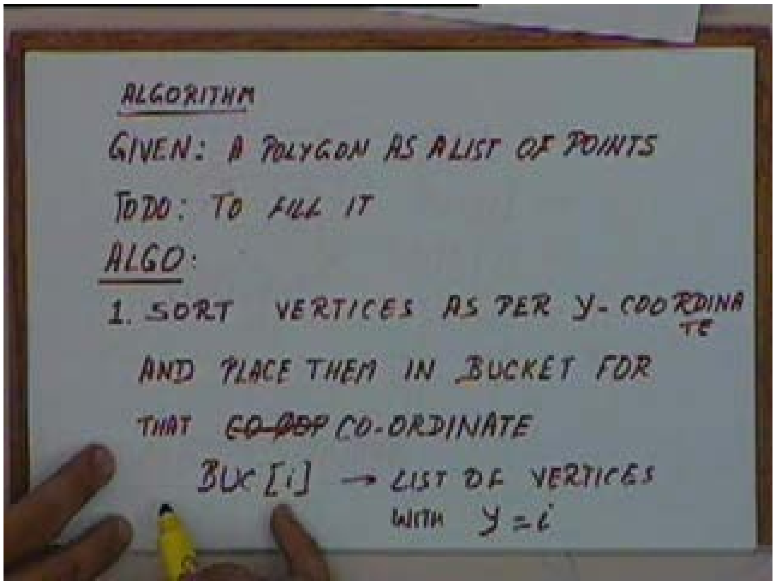
AEL → LIST OF ACTIVE EDGES

So we will say as we move from one scan line to another, the intersection points will change by an amount of delta y by delta x and the edges would start or end. These are two kind's things which can happen. Here two edges will start, at this point one edge is coming to an end and a second edge is starting. Similarly here two edges would start, at this point two edges will come an end. As we go from scan line in these positions from here to here, the intersection points will change by an amount equal to delta y by delta x. So if you are able to keep track of these changes in that case from one scan line to second scan line, we can keep deciding what are the pixels which are inside and what are outside. So if we know which are the pixels which are inside this scan line and the next scan line I will only change the intersection points by amounts equal to the slope of the edges.

If I know that at this point two edges are going to start, I will just take care of these two edges and I will compute the intersection points and display all the points which are inside them. At any point if I know that the set of intersection points on that scan line are let's say 1 2 3 4, if I know the set of intersection points, the intersection points are let's say its column Q now or I, I_1 I_2 so on till I_n that is 1 2 3 and 4. These are the 4 intersection points on a given scan line then which are the pixels which are going to be displayed. Pixels lying between I_1 and I_2 they have to be put on, between 1 and 2 they will always be on, between 2 and 3 they will always be off. Alternate pairs of pixels or alternate sets of pixels will be on and off.

So this is a first set of pixels, this will be on, the second set of pixels this will be off, the next set will be on and so on. If you have a polygon which is like this you are considering a scan line at this point 1 2 3 4 6 7 and 8 between 1 and 2 they will be on, 2 and 3 will be off, 3 and 4 will be on, 4 and 5 will be off, 5 and 6 will be on, 7 and 8 will be off. So alternate pairs of or alternate sets of points will be on and off. So what we will do is at every scan line, we will try to keep track of these intersection points. How do we do that?

(Refer Slide Time: 00:13:52 min)



ALGORITHM

GIVEN: A POLYGON AS A LIST OF POINTS

TODO: TO FILL IT

ALGO:

1. SORT VERTICES AS PER Y-COORDINATE
AND PLACE THEM IN BUCKET FOR
THAT ~~CO-ORD~~ CO-ORDINATE

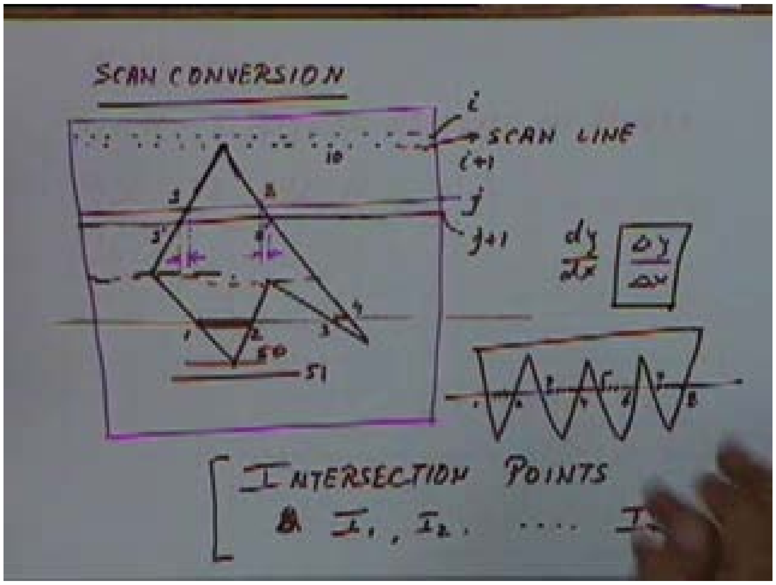
$BUK[i] \rightarrow$ LIST OF VERTICES WITH $y = i$

Let’s see the algorithm for scan conversion of polygons. What is given to us? A polygon as a list of points. And what we have to do, what is the aim? To fill the polygon. The way we will do it, we will first explain the data structures involved.

We know this set of vertices, we know that at every vertex there will be two edges which will be coming. Out of these two edges we can have number of cases either both the edges are just going to start at that scan line like in this case this edge is also starting, this edge is also starting at this scan line. At this scan line i plus 1 let’s say this edge as well as this edge both the edges are starting. If I consider this scan line, one edge is ending the other edge is starting. At this point both the edges are coming to an end. So at each vertex we will always have two edges and they can either be starting or ending. So what we will do is we will keep track of the vertices or the scan lines on which the vertices are occurring. If I know that on scan line number let’s say 10 a particular vertex is starting. The moment I come from scan line 9 to 10, I will take up all the edges which are on scan line number 10 which are starting at scan line number 10 and I will start taking care of them.

So if as I come from let’s say 9 to 10 and nowhere at 10 two edges are going to start so that will help me in taking care of the intersection of these two edges. Similarly let’s say from as I go from scan line 50 to 51, I know that two edges are going to end in that region. So when I go from 50 to 51 I will delete the intersection of those edges and not bother about them anymore. So at every scan line we will keep track of the vertices that are there in that scan line and we will also keep track of the set of intersections on that scan line so that will be done in two different structures. The first step in this algorithm would be that we will sort the vertices as per the y coordinate and what this means is I will just put that back.

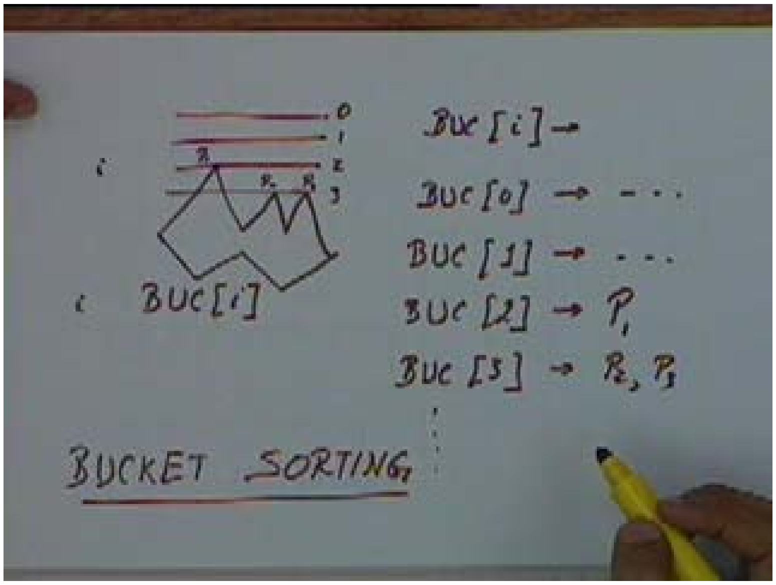
(Refer Slide Time: 00:14:56 min)



We have a set of scan lines number 0 1 2 and 3 so on. We want to know whether there is any vertex starting on the scan line 0, any vertex on scan line 1 and so on. So we will maintain an array which is conceptually something similar to a bucket. In that array or in that data structure whenever there is a vertex on a particular scan line, like this vertex is on scan line number two, so this vertex will be kept in the bucket number two. Again in some sense it is going to be a queue of events. So let's say we have an array called bucket of i and bucket of i is going to contain a list of vertices at scan line number i, at row number i. So let's say if we have a case like this, this is p_1, this is p_2 and this is p_3. So what you want is bucket of 0 should be empty, bucket of 1 should also be empty, bucket of 2 would contain one vertex which is P_1, bucket of 3 would contain two vertices P_1 and P_3 and so on. So if the y quadrant of a particular point is i, that point or we will keep track of that point in bucket of i.

So if you have a let's say n vertices, each vertex will be stored in some bucket, so n location and these buckets are going to get occupied. Is that clear? The basic idea is that at scan line number i, I should know what are the set of vertices starting on that particular scan line. [Conversation between Student and Professor – Not audible ((00:21:57 min))] again please. Isn't it memory deficient, if we have so many buckets in a bit most? Isn't it memory inefficient? It is time wise very efficient because you are able to find out on which particular scan line which points are there and for doing this kind of sorting, you are able to do in time which is proportional to the number of points. This process is a referred to as what is called bucket sorting. The amount of memory used will be equal to the number of scan lines that will be the array size. The speed will be proportional to the number of points otherwise if you do sorting in any other way, the speed will be much slower than that, the time required will be more.

(Refer Slide Time: 00:18:54 min)



BUC[i] →

BUC[0] → . . .

BUC[1] → . . .

BUC[2] → P_1

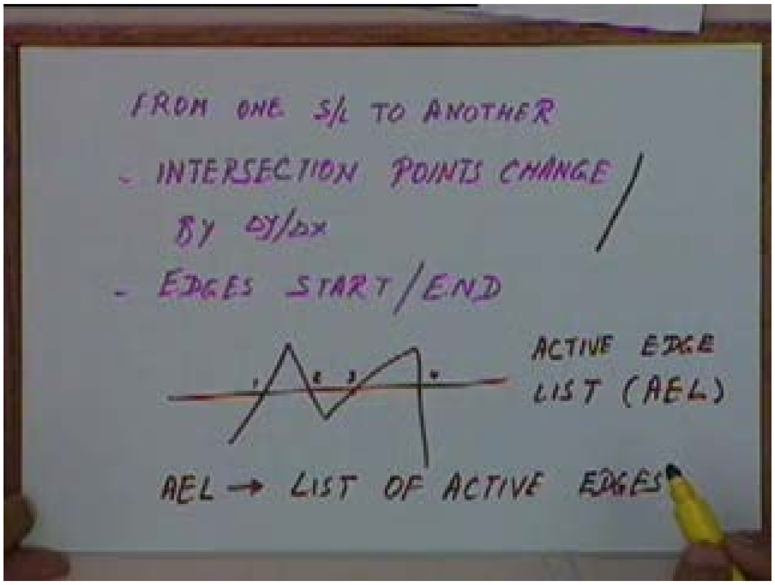
BUC[3] → P_2, P_3

BUC[i]

BUCKET SORTING

So in each bucket we will have the set of point starting on that particular scan line. That’s what is a sort vertices as per y coordinate and place them in the bucket for that coordinate. So bucket of i would have list of vertices with y coordinate equal to i. Then I mentioned earlier that we need to keep track of two things, one is on a particular scan line which are vertices which are there and the second thing that we need to keep track of is on a particular scan line, what are the sets of intersections, intersections or what are sets of edges which are active. If a particular edge is active that means that edge is intersecting the scan line. So the first part of the algorithm is taking care of telling us on particular scan line which are the vertices which are occurring. The second part of the algorithm now that will tell us the intersections on every scan line.

(Refer Slide Time: 25:35)



FROM ONE S/L TO ANOTHER

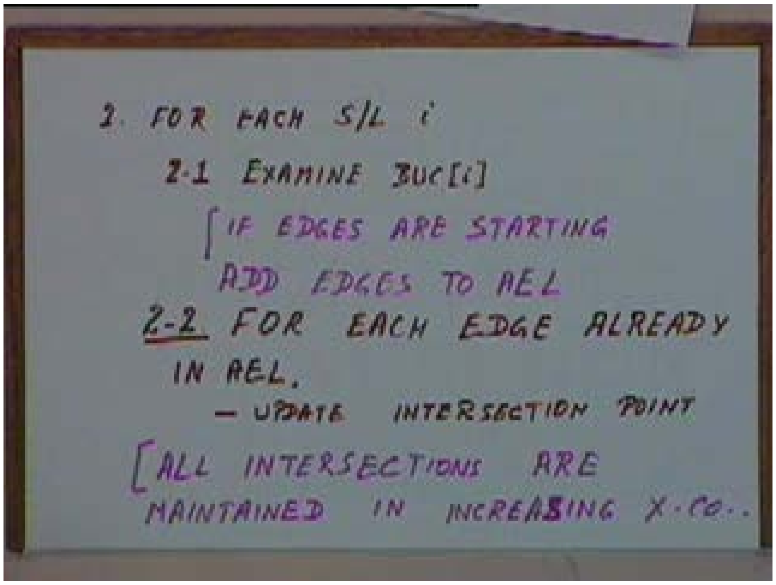
- INTERSECTION POINTS CHANGE BY $\Delta y/\Delta x$
- EDGES START/END

ACTIVE EDGE LIST (AEL)

AEL → LIST OF ACTIVE EDGES

What we will do is if we have a set of intersections, we will maintain what is called an active edge list or we will refer to as by AEL. So this AEL active edge list will store the list of active edges and their intersection on a particular scan line. So in this case our intersections are 1 2 3 4. This active edge list is going to maintain that these four edges are intersecting and the intersections are so and so. And as we go from one scan line to second scan line, we will keep updating this active edge list every time. So the steps in an algorithm would be like this.

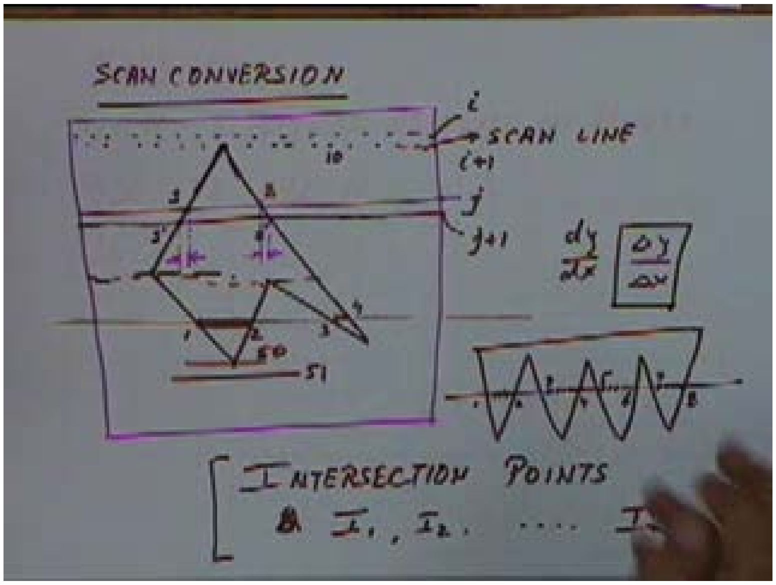
(Refer Slide Time: 00:25:53 min)



```
2. FOR EACH S/L i
    2-1 EXAMINE BUC[i]
        [IF EDGES ARE STARTING
        ADD EDGES TO AEL
    2-2 FOR EACH EDGE ALREADY
    IN AEL.
        — UPDATE INTERSECTION POINT
    [ALL INTERSECTIONS ARE
    MAINTAINED IN INCREASING X-CO..
```

We have already placed the vertices in a bucket. Now step two is for each scan line, SL I am keeping for scan line, for each scan line i the first thing we will do is examine bucket of i. We will first check whether there are any edges which are starting in that particular scan line or if any edges are being or coming to an end at that scan line. So if there are any edges which are starting at that scan line, what will we do? If edges are starting then we will add the edges to active edge list. If there is any edge which is staring on that particular scan line that will be added onto the active edge list. Then for each edge which is already in the active edge list, what should be we doing? These are edges like this.

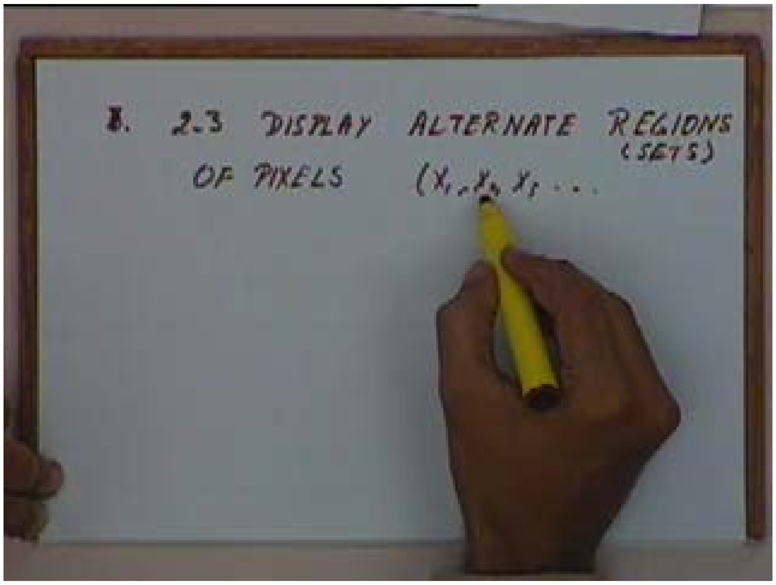
(Refer Slide Time: 00:28:09 min)



This edge and this edge. At this particular scan line, this edge is already in the active edge list and the intersection point for this is 1. This edge is already in the active edge list and the intersection is point 2. As we go onto the next scan line, we should change the intersection points, we should modify the point of intersection by an amount equal to delta y by delta x. So for each edge already in active edge list update intersection point. Incidentally, when I mention here that this amount is equal to delta y by delta x, it will actually be delta x by delta y because we are moving in a y direction and moment in the y direction is equal to one pixel. So the moment in the x direction is going to be delta x or delta x by delta y then basic idea is that the moment will be proportional to the slope of the line. So we said for each edge already in the active edge list update the intersection paths.

Now while we are doing this, we should take care that all intersections are maintained in an increasing x coordinate or remains sorted as per x. If we have intersection points 1 2 3 4, we should always maintain them in the increasing x order because we have seen that alternate sets of pixels are going to be on and alternate sets would be off. So you should take care that these intersections are maintained in an increasing order of x. So whenever we add a edge to it, we added such that it remains in the increasing order of x. So at this point, at this scan line we had an intersection here 1, intersection here 2. Now two edges are being added here at this point. So let’s say these edges are a and b, so we wont the set of intersections should not be 1 2 a and b, it should be 1, a, b and 2. They should remain sorted as per x coordinate. So for each edge already in the active edge list update the intersection points and while maintaining the intersection points we have to be careful about keeping them in the increasing order of x, keeping them sorted as per x then the third step of the algorithm or sorry not the third step, for each scan line we have done this, we have done this so 2 3, the third of the part of the second step itself.

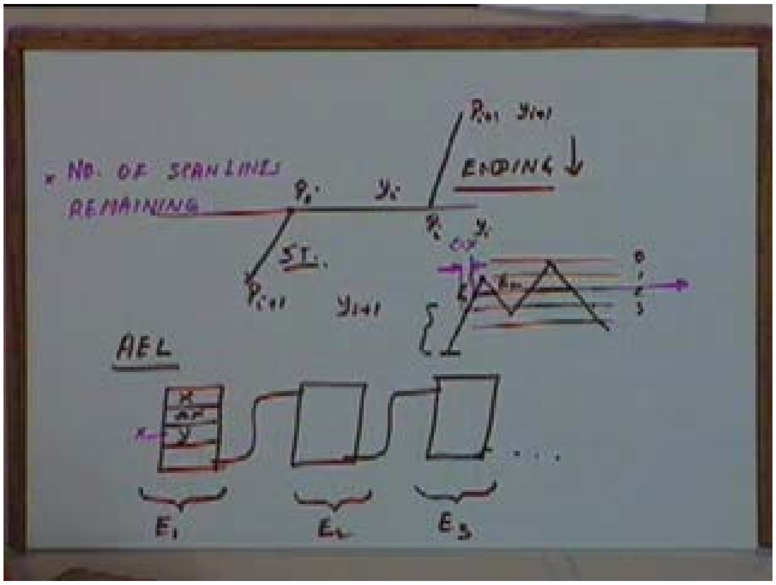
(Refer Slide Time: 00:32:16 min)



3. 2.3 DISPLAY ALTERNATE REGIONS (SETS) OF PIXELS (X_1, X_2, X_3 . . .

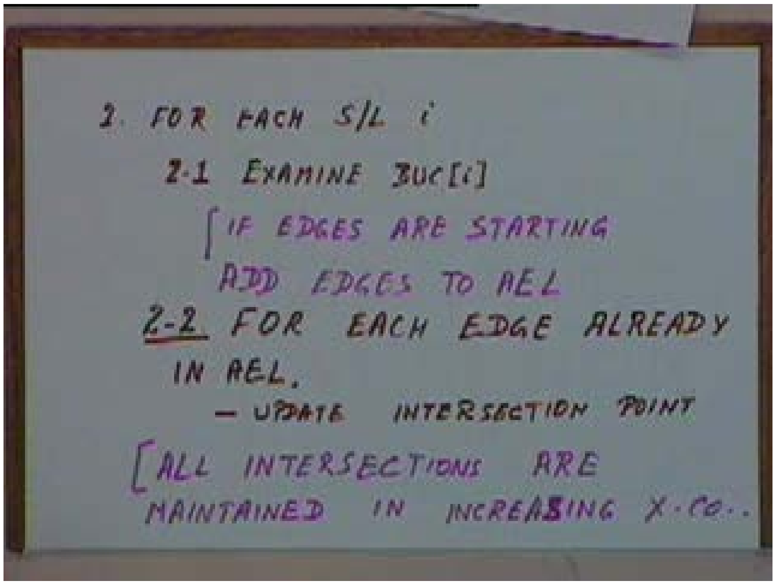
At this stage we know the set of intersections on a particular scan line, the set of intersections is known. So now we will say display alternate regions of alternate regions or alternate sets of pixels. So in the active edge list if we have intersection points given by x_1, x_2, x_3, so on and between x_1 and x_2 we should display, x_3 and x_4 we should display, x_5 and x_6 we should display and so on. And this process will keep repeating for every scan line, for every scan line we will keep repeating this till we reach the end of the screen. This way we will be able to fill the polygon such that all the pixels inside would be highlighted and all outside would not be. Is that okay? Any questions on this up to what I have covered so far, then we will see some more details of this algorithm. What you find out in step 2 1 you said for an x starting in step 2 1. You said the edges are starting. If edges are starting what happens when edges are ending? How do you test for the edges are starting or ending? How do you test whether the edges are starting or ending? that is quite straightforward.

(Refer Slide Time: 00:34:45 min)



See this is a particular scan line, you have an edge given by P_i, P_{i+1}. The y quadrant of this is let's say y_i, this is y_{i+1}. If y_{i+1} is greater than y_i the edge is starting. If y_{i+1} is less than y_i the edge is coming to an end. Is that okay? Because we are moving in this direction. If a line is like this, this is P_i and this is P_{i+1} or i minus 1. In this case y_{i+1} is less than y_i, so the edge is coming to an end. Here y_{i+1} is greater than y_i, so the edge is starting. This is a case of the edge starting, this is the case of ending. Is that okay? Any other point?

(Refer Slide Time: 00:36:07 min)



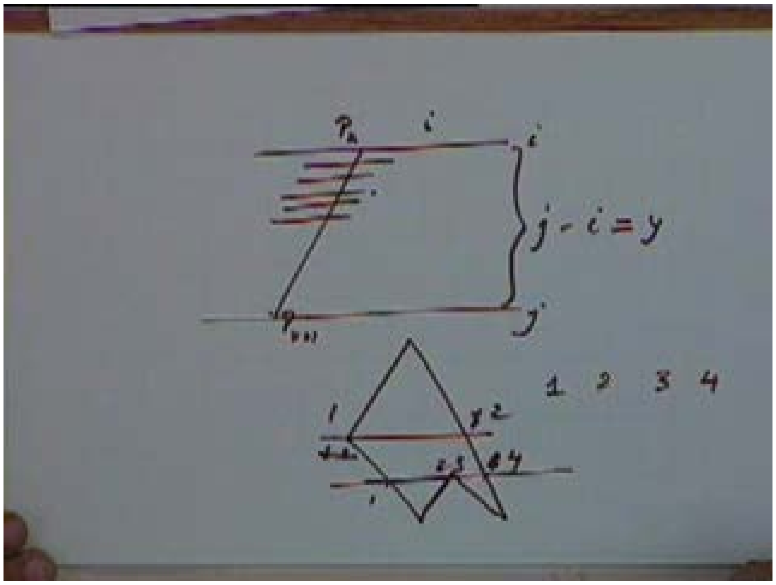
```
2. FOR EACH S/L i
    2.1 EXAMINE BUC[i]
        [IF EDGES ARE STARTING
        ADD EDGES TO AEL
    2.2 FOR EACH EDGE ALREADY
    IN AEL.
        – UPDATE INTERSECTION POINT
    [ALL INTERSECTIONS ARE
    MAINTAINED IN INCREASING X-CO..
```

Now in this algorithm if you noticed we are maintaining this active edge list. In the active edge list we are computing the intersection point of every edge and as we move from one scan line to the next scan line, this intersection point is being modified. So this active edge list is going to contain a list of edges, a list of entries. So we will have something like this and so on. This is going to contain all the information for the first edge, edge one this contains all the information for edge two, this contains all the information for edge three. We said for each edge we need to know what is the intersection or the x intersection at that particular scan line. We also need to know what is the slope of the edge. So this entry in the active edge list needs to contain some information. The first information would be the x intersection. So this is let's say scan line numbers and on this particular scan line or let's take something like this, we have entries for this is edge E_1, this is edge E_2 and so on. For this edge E_1 we need to know the x intersection of edge E_1 with this scan line. We also need to know that as we move from this scan line to the next one by how much is this intersection changes.

As I said at this particular, on this scan line, on this line this is an intersection. On the next scan line this would be the intersection and the amount that is this difference will be how much? It will be some delta x. So for each edge in the active edge list we will also have to store the delta x. Delta x is the amount by which the intersection would change as you move from one scan line to the second scan line. Is that okay? So we need to know x and we need to know delta x. Do we need to know anything else? The end points. You need to know when the edge is coming to an end. How do you keep track of that? We have that list of points. You have the list of points. P_1, P_2, P_3. Any other way in which you can keep track of the end point of this line?

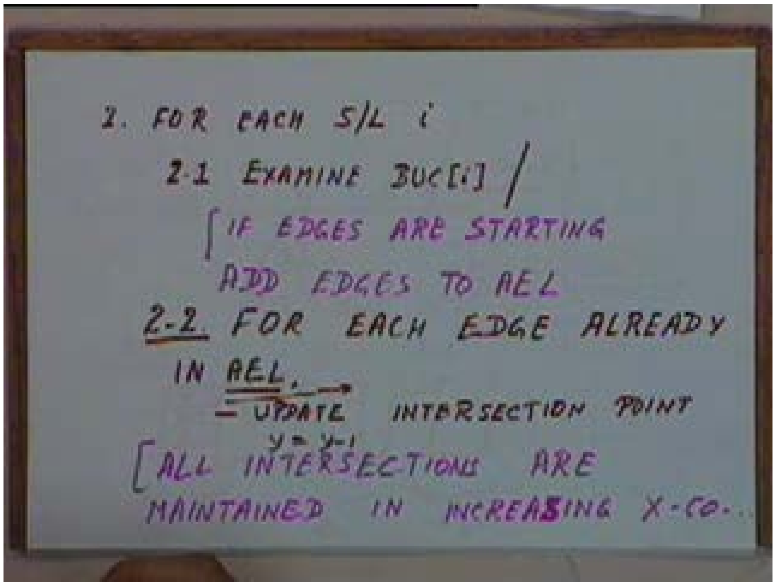
Delta x is zero for some particular point, delta x is zero, delta x depends on the slope of the edge. I mean it shows that there is no other point. There is no other point? Like the line is ending and there is no point, delta x will be zero for that. We store y_i. You store y_{i+1} yi. [Conversation between Student and Professor – Not audible ((00:40:13 min))] you store the end point, so this is the starting point, you also store the end point and every time you will keep comparing whether the end point has come or not. Instead of that what you do is you store the number of scan lines remaining, so at this point scan line has started. You can compute the difference between the y values between the starting and the end point and keep that y value over here. So this y value is the number of scan lines, the number of scan lines remaining. Number of scan lines remaining that means this is the line, it has started at this particular scan line which is scan line number i. Let's say this is point p_k, this is point p_{k+1}. The y quadrant of this point is i and the y quadrant of this point is let's say j.

(Refer Slide Time: 00:41:25 min)



So the moment this edge starts, at this point the number of scan lines remaining would be these many which is j minus i. So this would be the value of y that is the number of scan lines remaining for this edge. [Conversation between Student and Professor – Not audible ((00:42:13 min))] let me just complete that then I will answer your question. So this is the number of scan lines remaining. As we go from one scan line to the next, we will keep decreasing this number by 1, so number of scan lines remaining would become 1 less every time. The moment this number becomes equal to zero or becomes less than zero you will delete that edge. So with the every edge we will also keep track of the number of scan lines remaining and every time we go from one scan line to a second scan line will be decreasing the value of y by 1.

(Refer Slide Time: 00:43:31 min)



```
2. FOR EACH S/L i
    2.1 EXAMINE BUC[i] /
        [IF EDGES ARE STARTING
        ADD EDGES TO AEL
    2.2 FOR EACH EDGE ALREADY
    IN AEL,
        – UPDATE   INTERSECTION POINT
          y = y-1
    [ALL INTERSECTIONS ARE
    MAINTAINED IN INCREASING X-CO..
```

And if this happens to become negative at any point that edge has to be deleted. So the corresponding modification you can make in the algorithm that is we have said update intersection points along with that we will also have to update y, we will have to say y will be equal to y minus 1. For each edge in the active edge list we will have to say y equal to y minus 1 and in the beginning when we are examining the bucket for new edges, in addition to this we will also have to examine whether some edge has come to an end or we can examine it write here before updating. We can examine whether y coordinate is less than zero, if it is less than zero you can simply delete it, at this point we can do that. So we are adding edges in the active edge list according to the bucket, we will be deleting the edges if the value of y becomes less than zero and as we go from one scan line to a second scan line we will update the intersection points and we will delete edges which have come to an end. We will make sure that all intersections are maintained in the increasing order of x and after that we will display alternate sets of pixels, alternate set should be on, alternate set should be off.

Now coming back to the questions you had asked, any reason for maintaining y and not just the end point j. See if you are maintaining the end point j, at every scan line you have to compare the y value with j but now you will be subtracting one every time and the moment the number becomes negative you can delete that. Checking that is computationally easier. [Conversation between Student and Professor – Not audible ((00:45:36 min))] comparisons are quite expensive and if statement is much more expensive than a [Conversation between Student and Professor – Not audible ((00:45:43 min))] subtraction [Conversation between Student and Professor – Not audible ((00:45:46 min))] no comparison with zero, so you are only checking for a sign bit. [Conversation between Student and Professor – Not audible ((00:45:51 min))] decrement decrement by unit amount [Conversation between Student and Professor – Not audible ((00:45:59 min))] decrement by unit amount decrement by unit amount plus comparing the sign bit would be cheaper. Basically your efficiency consideration. Conceptually you can always compare it with j every time. By comparing two abstract numbers I mean arbitrary numbers is more expensive. So we normally avoid that. So this is as far as the algorithm is concerned.

Just one more thing and that is consider a situation like this and consider a scan line like this. At this point we have one intersection here and we have one intersection here. Both the edges are just starting at this point, so we have an intersection over here. You can see the problem. If you consider this only as one intersection then as list of intersections would be 1 2 and 3. So all pixels between 1 and 2 will be highlighted, between 2 and 3 they will off and after 3 they will become on, so these pixels will become on and these pixels will become off. This is not what we want. So at this point we should consider it has two intersections because there are two edges, we should consider them as two intersections. So 1 2, this would be 3 and this would be 4 then there will be no problem.

In such cases you should consider this as two intersections. But what happens when you add this end point? At this end point, you should now consider this as has two intersections here 1, 2 and 3. The two edges, do you consider them as two intersections or one? If you consider them as two intersections again you will have a problem. So here it should be considered as one intersection. So when it should be considered as one intersection and when it should be considered as two intersections? [Conversation between Student and Professor – Not audible ((00:48:58 min))] I will leave it up to you to decide that and give it some thought and then decide what will be the criteria? It’s a simple criteria, you can give it some thought and get it easily, under what situations at a vertex you should consider two intersections or you should consider it as one intersection. You will have to draw number of figures and see that whatever criteria you are suggesting whether that is giving you correct results in all the cases. You can do that bit of thinking. That’s all for today. We will start in the next class.