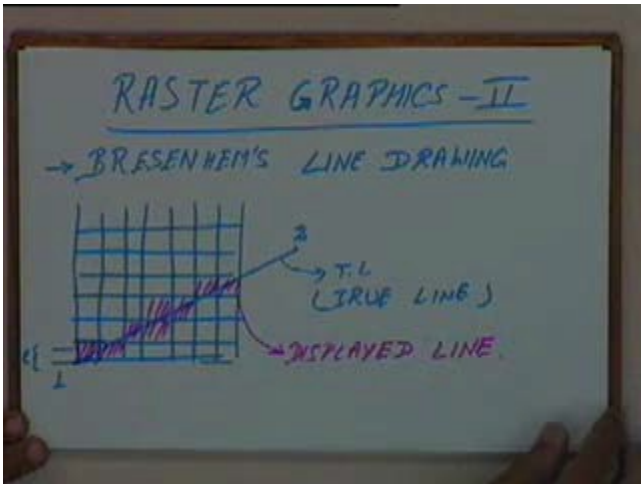
Computer Aided Design
Prof. Anoop Chawla
Department of Mechanical Engineering
Indian Institute of Technology, Delhi
Lecture No. # 04
Raster Graphics II

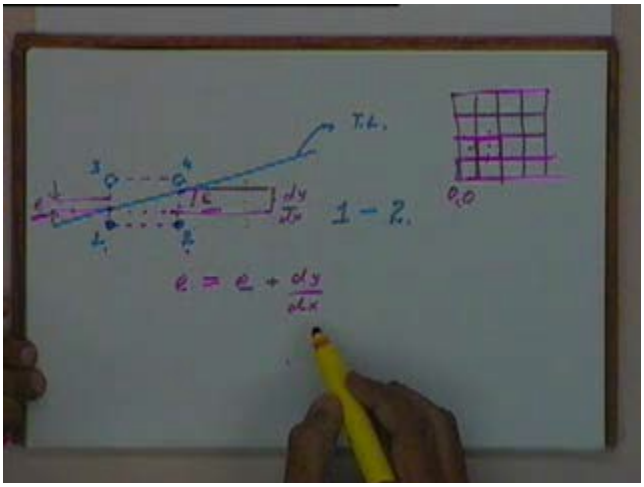
Today we will be having, continuing with the what we were doing last time and that I was essentially saying different kinds of line drawing algorithms.

(Refer Slide Time: 00:01:18 min)



Yesterday, I had introduced this Bresenhem's algorithm and we had said that if we have to draw a line between a point one and a point two, in this exact straight line between the two can be referred to as a true line. But in a Raster refresh display, the true line will never be displayed. What would be displayed would be a set of pixels which in this particular case might be pixels chosen something like this and so on. So, this set of pixels that are displayed will refer to these as a displayed line and this Bresenhem's line drawing algorithm works on minimizing the error between the true line and the displayed line and the error between the two is defined as the gap between the two. So if this particular pixel is being displayed, this pixel is actually let's say the center of the central here. Ok. And the line at the corner is supposed to pass through this point, so error is this error e. So we will go through this algorithm. The basic principle is that you have to minimize the error between the true line and the displayed line.

1
(Refer Slide Time: 00:02:54 min)



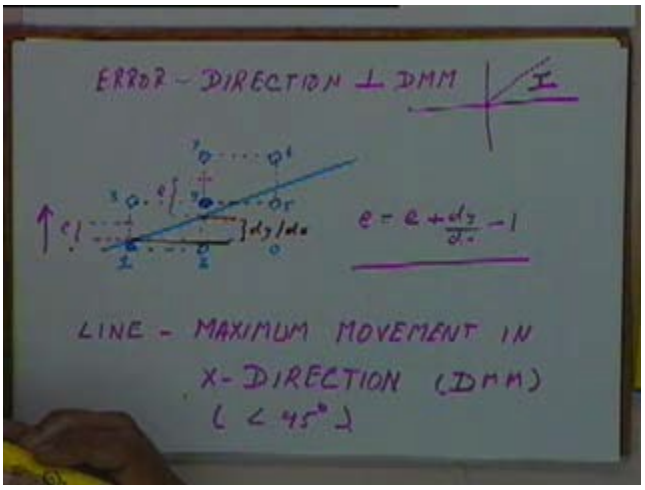
If we have a set of pixels like this, this is the true line. One pixel that has been displayed is this pixel that I have shown here in dark. You have already seen the previous line drawing algorithm, we display pixels let's say in a loop. The first pixel is this one, let's say the second pixel that is displayed is this pixel. So this is the first pixel, this is the second pixel to be displayed. We will number them. Ok. So if initially pixel number one is displayed and then the pixel number two is displayed, we will just see how the error would be defined in the two cases. If we look at this point, this pixel will actually correspond to the pixel something like this. The center of this pixel here will be at this level. So the gap between the two let's say this term, this much which is the error initially. At this point, the center for this pixel is that this level and the actual line is through this point. So the error here is this error e. So, initially this is the error. Finally this is the error.

So, what we say is as we move from this pixel to this pixel, the error is changing by this magnitude to this magnitude. And how much will be the difference in the two errors? The difference between these two errors will be equal to the slope of the line because the line has moved up by this much. Initially line is here then the line is here, so the line has moved up by this amount and this amount would be equal to the slope of the line. Slope of the line multiplied by unit pixel distance. Is that okay? So, we will say that as we move from one pixel to the second pixel, the error term e would change by an amount equal to $\frac{dy}{dx}$. Yeah what's that? Would we defined a pixel to be square or something?

I have we had covered it earlier. If this is the array of pixels, if you are addressing this particular pixel, it's actually this pixel, you are addressing it at this location. So if let's say for sake of argument I treat this as my 0 0, I have already said normally we treat the top left as 0 0. Now this is my origin, when I refer to a pixel 0 0, I am actually referring to this pixel. I am referring to pixel 1 1, I am referring to pixel which is this one. So we are actually referring to a pixel as the square entity and we are addressing it by the corner. And the center is the center of this point. and the center is the center of this point. And when you write point two, point, this point two? Yeah. Yes sir, where does it lie on the square?

Where does it lie on the square? Yes sir. This is the corner of the pixel. This will be in the next square. This actually would be a pixel like this. So the center is the center of the square. Yeah, actually the pixel is at the center but we are looking at the line at the beginning of that pixel, the position of the line at the beginning of that pixel. So, as we move from one pixel to the second pixel in this orientation, the error term would change by an amount equal to $\frac{dy}{dx}$. Is that okay? But in another way in which the pixels can be displayed, I have right now assumed that this was the first pixel and second pixel was this one, there is no reason for that to be true. The second pixel can actually be this pixel. You just see what happens in that case.

(Refer Slide Time: 00:08:20 min)



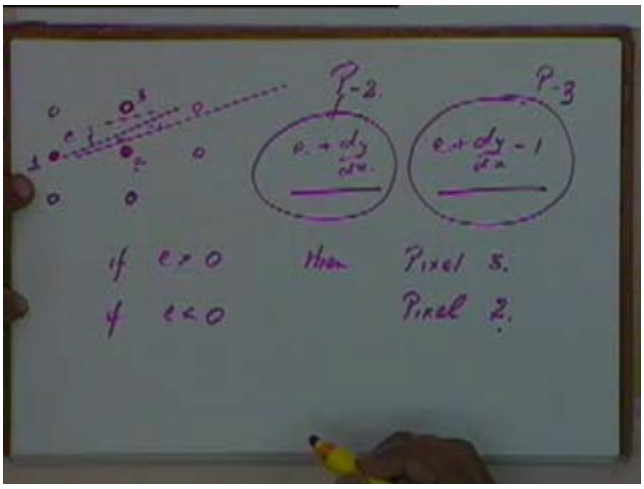
The first pixel that is displayed is the pixel number one, which is like this and the next pixel that we decide to display is this pixel number four which is. Ok. At this point my error term will be this much, at this point my error term would be this much. And by how much has my error term changed as I have moved one pixel in x the direction and say e will be equal to e plus, my line has moved up by an amount equal to, this amount is $\frac{dy}{dx}$. So this line has moved up by an amount equal to $\frac{dy}{dx}$ and if you look at this pixel, earlier to this pixel now the pixel being displayed is this one.

So the pixel to be displayed has also moved up by an amount of one unit. So the error term because of the line moving up would increase by an amount of $\frac{dy}{dx}$ and since the pixel being displayed has also shifted up, the error term will come down by one. Is that okay? So if the next pixel to be displayed is a diagonal pixel like this, then the error term would change in this manner. Is that all right? Now, right now we have not considered the case when the first pixel to be displayed is this one and the second pixel to be displayed is pixel number three, that can also happen. We are not taking that into account because what you will say right now is at this line has got a greater inclination towards the x axis. So this line has a maximum movement in x

direction or the direction of maximum motion is the x direction or the slope of this line is less than 45 degrees.

3
If you consider octants like this, this particular case is for the first octant. And in the first octant my direction of maximum motion is the x direction and the error data measuring is in a direction perpendicular to that. So, this error term is measured in the y direction. So the line of maximum moment in the x direction and error is measured in direction perpendicular to let's say the line of maximum motion or direction of maximum motion. So the error is measured in the direction perpendicular to the direction of maximum motion. Ok. Is this all right?

(Refer Slide Time: 00:13:43 min)



So now we have a scenario in which this particular pixel has already been displayed. Okay. The error here is this amount e, this is already displayed. Now I have to decide whether next pixel should be this one or this one. What should be the criteria for deciding that? One which give the lower error. The one which gives the lower error. Okay. If this is displayed, the error will be $e+\frac{dy}{dx}$. If this is displayed, the error will be $e+\frac{dy}{dx}-1$ and we can compare the magnitude of the two errors and find out which is lower.

Alternatively at a previous level whichever pixel was displayed whether this one or this one, I could have computed the error at the end of the line. Okay. And if this line is passing below this midpoint then, I will display this pixel. If it is passing above the midpoint like let's say somewhere from here then, I will display this point. If this true line is passing above the midpoint then this pixel will give a lower error compared to this pixel.

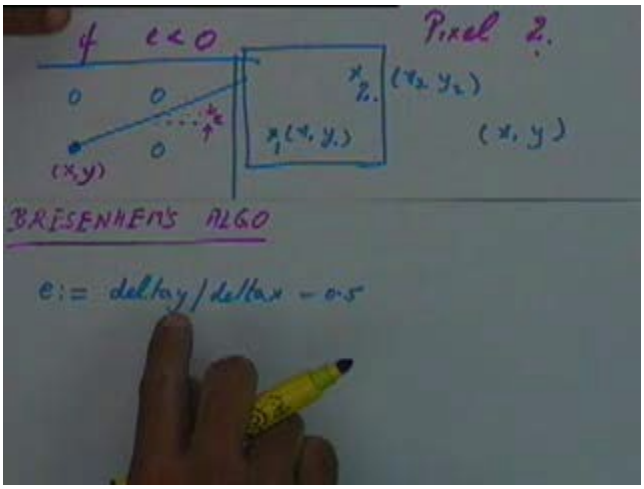
Now I am looking at one step before I am not looking at these two pixels, I am looking at these two pixels. If the line is passing exactly through the center how much is the error? Zero. The error is zero. If the line is passing below that then how much is the error? Whatever amount it is passing. Positive or negative? Negative. The error will be negative. If the line is above that the error is positive. So if my line is above this I will display this pixel, if my line is below this I will

display this pixel or in other words we will say that if e > 0 then we will display the upper pixel. So let's say this is number two and this is number three. If e > 0 then pixel number three and if e < 0 then pixel number two.

4

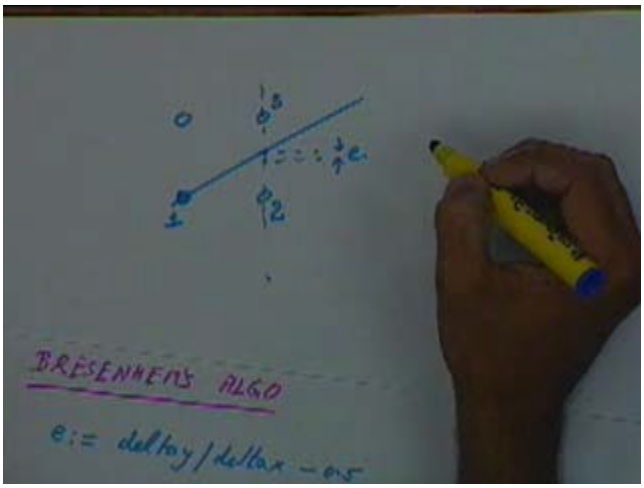
So what we will basically do is that we will find out the error at this line. If the error is positive we will display this pixel, if the error is negative we will display the lower pixel and depending on which pixel is being displayed, the error in the next term will be computed either according to this or according to this. If, I am considering these two, the previous pixel has to be this one. Pixel number one has been displayed, if at the end of that pixel the error is positive then we will display the pixel number three. If at the end of that the error is negative I will display pixel number two. If pixel number two is being displayed, my error will be computed according to this formula, this is for pixel number two. If pixel number three is being displayed, I will use this formula. So according to this method, we will write down the Bresenhem's algorithm.

(Refer Slide Time: 00:18:48 min)



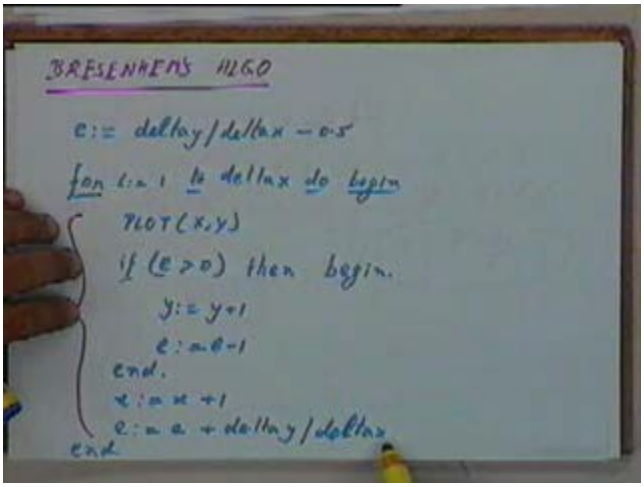
Again we are taking the case where this is the screen and we have to a draw line from a point one to a point two. The point one let's say is x_1 y_1 and the point two is let's say x_2 y_2. The first point that we displayed that will be the pixel x y which will be initialized to x_1 y_1. So, and this start of the algorithm we will compute the error term which initially when we display the pixel x y that is the starting pixel. The x y is the starting pixel that is being displayed. At the end of this pixel how much is the error term? The line has moved up by an amount of $\frac{dy}{dx}$ and the center of this pixel is at 0.5. So at this location the error will be $e=\frac{dy}{dx}-0.5$. This amount of the error that I have shown in this part of the figure is $\frac{dy}{dx}-0.5$. Is that okay? So the initial error that will be there at the end of the first pixel will be $\frac{dy}{dx}-0.5$. So when I have to decide, I will draw the figure clearly here.

5
(Refer Slide Time: 00:21:24 min)



This is the initial error at the end of this pixel. So at this level, at this position I have to decide, whether this is my pixel number one, whether the pixel number two is to be displayed or the pixel number three is to be displayed. In order to decide that I will look at whether this e is positive or negative. If this e is positive, I will display this, if e is negative, I will display this. So accordingly we will write the algorithm.

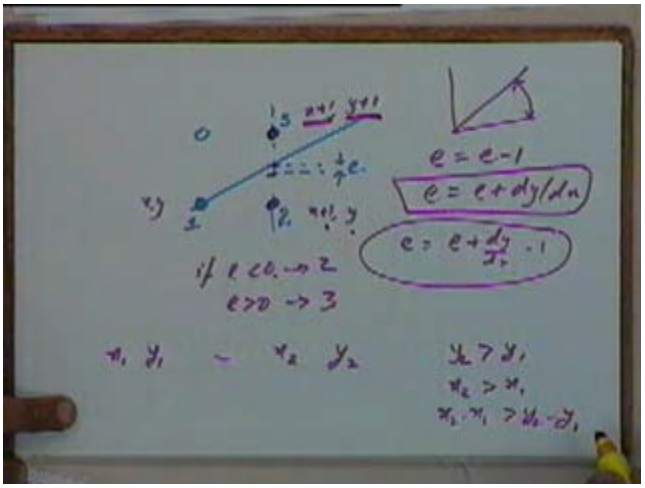
(Refer Slide Time: 00:22:07 min)



We initialize e that is the error term to be $\frac{dy}{dx} - 0.5$ and then we have a loop. Okay. So, we have initialized the error term to be $\frac{\Delta y}{\Delta x} - 0.5$ and this as I said would be the

error term or would be the error at the end of the first pixel that is at this location. Then we are going, repeating this part in a loop, we will first plot the first pixel, we will plot the pixel x y.

(Refer Slide Time: 00:23:58 min)



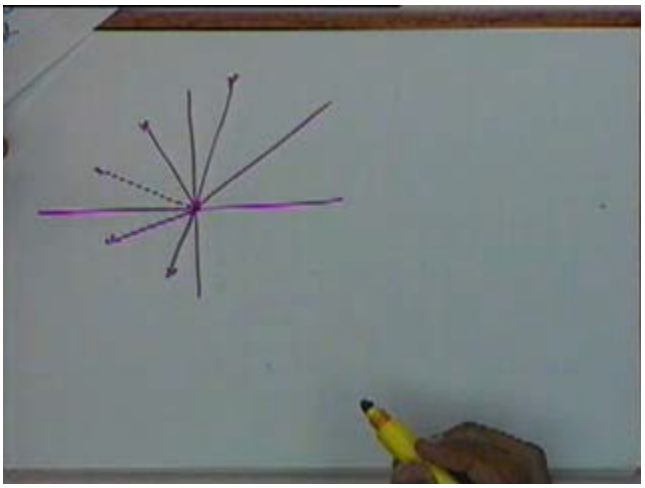
Then we will say if e > 0. If e > 0 then the next pixel to be displayed is the pixel number three. If pixel number one is x y, pixel number three is x + 1, y + 1 and pixel number two would be x + 1 and y. So we have plotted the first pixel x y, if e >0 then you are saying y is equal to y + 1 and e =e - 1 that means we are changing the value of y to y + 1 and the error term e, you are saying e= e - 1 and then immediately out of this if statement we are saying x =x + 1. So, x is also being made x + 1 and error term is being changed to $e = e + \frac{\Delta y}{\Delta x}$ e. We have already said e = e - 1 and after that we are saying $e = e + \frac{dy}{dx}$ e. So effectively $e = e + \frac{dy}{dx} - 1$. So if my error term is greater than 0 then we are changing x to x + 1 and y to y + 1 and e to $e + \frac{dy}{dx} - 1$.

So if the error is positive now y is becoming y + 1, x is becoming x + 1 and e is becoming $e + \frac{dy}{dx} - 1,$ which is effectively what we want to do if the pixel number three is to be displayed. If the pixel or if the error term is less than zero in that case if e <0, these two steps will not be

done, we will just say x is equal to x + 1 and e will be equal to $e+\frac{dy}{dx}$ which is what we want to do for pixel number two. The x will change to x + 1 and y will remain as y and e will change to $e+\frac{dy}{dx}$. So effectively if e <0 then pixel two, if e >0 then pixel three. Okay. And mind you all this we are doing when our line is in the first octant, in this part. If we have to draw line from x_1 y_1 to x_2 y_2, when we say that the line is in the first octant, we are saying $y_2 > y_1$, $x_2 > x_1$ and $x_2 - x_1 > y_2 - y_1$. So this algorithm that you have written right now is for the condition, when these conditions are satisfied.

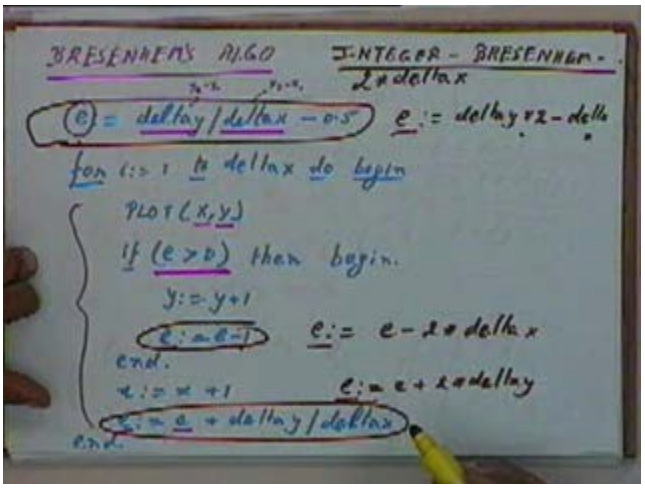
7

(Refer Slide Time: 00:27:34 min)



If our line is in some other quadrant instead of the line being here, if our line is to be drawn in this octant okay, if we have to draw line starting from this point with the slope like this and the direction of maximum motion is the y direction. So if the direction of maximum motion is the y direction then my algorithm will change accordingly. What will be the changes? Interchange x and y. Again? Interchange x and y. We will essentially interchange x and y, so we will have to rewrite the algorithm with x and y interchanged. Similarly we have to make similar changes if my line is like this or like this or in any other octant.

(Refer Slide Time: 00:28:32 min)



So the algorithm, the way I have written right now is only for drawing lines in the first octant and in this, $\Delta y = y2 - y1$, $\Delta x = x2 - x1$ and of course e is the error term, x and y are the coordinates of the current pixel being displayed. Of all these x_1 x_2 y_1 y_2 they have to be integers because we are talking of drawing a line from one pixel to a second pixel. Pixels will always be addressed by integers, so x_1 x_2 y_1 y_2 they are all integers. Therefore Δ x and Δy they are also integers. Okay. x and y will also be integers. The only thing that is not a integer here is this error term e. e has to be a real number. Similarly all these are the places where e is being used, e has to be a real number because we are dividing two integers and we have a real number 0.5 coming in the picture, so e has to be a real number. And if you notice in this complete algorithm e is important only for finding out the sign of e, we are only bothered about whether e is positive or whether e is negative, we are not bothered about the actual magnitude of e.

Since we are not bothered about the actual magnitude of e, we will say that instead of coding the algorithm on the basis of e, we will code it in terms of e multiplied by x multiplied by two. What I mean is, if I consider, if I take this statement and multiply it by 2*Δ x, if I multiply 2*Δ x, my statement will become $e = \Delta y * (2 - \Delta x)$. sorry. Now my error term e is not the error term which we had shown in the figure earlier but it is that error term multiplied by 2*Δ x. So error is equal to or $e = \Delta y * (2 - \Delta x)$. Though other places where I am changing e or this line and this line, these statements also I will multiply it by the same amount 2*Δ x. So this statement will become, e = e -2*Δ x and this statement will become e = e +2*Δ y.

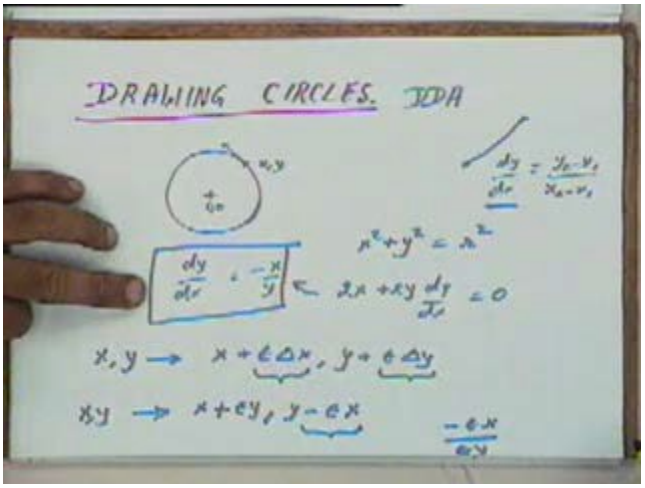
If I make these changes, my program will still be valid because I am only bothered about the sign of e. So wherever I am using e I have multiplied it by a constant amount, so sign of e will still remain the same. Mind you, Δ x is a positive number because in this case I have taken $x_2 > x_1$ so Δ x is a positive number, so I have multiplied my term e by a positive number, by a positive constant. So the sign would remain the same throughout which effectively means this statement will be replaced by this one, this will be replaced by this one and this statement will be replaced by this one. E will also become an integer. Pardon; now e will also become an integer. Now e

will also become an integer. The reason why I am doing this is after this all my variables will become integers.

I don't, I will not deal with any real numbers and in addition to that I will not have any division in my algorithm. There will be no integers and no divisions which means the algorithm can be very easily implemented on hardware and the algorithm will be very fast because integer numbers will take less memory and integer addition and multiplication are easier to carry out, there is no division involved. And this version of the algorithm is referred to as integer Bresenhem's algorithm, integer Bresenhem's algorithm, okay, the integer version of the Bresenhem's algorithm. Any question on the integer version of the Bresenhem's algorithm? This algorithm is superior to the previous algorithms primarily because you are dealing only with the integers and we don't have any division in the algorithm. If we don't have divisions, the computations are faster, divisions are very expensive. So this is the integer version of the Bresenhem's algorithm.

9

(Refer Slide Time: 00:35:25 min)

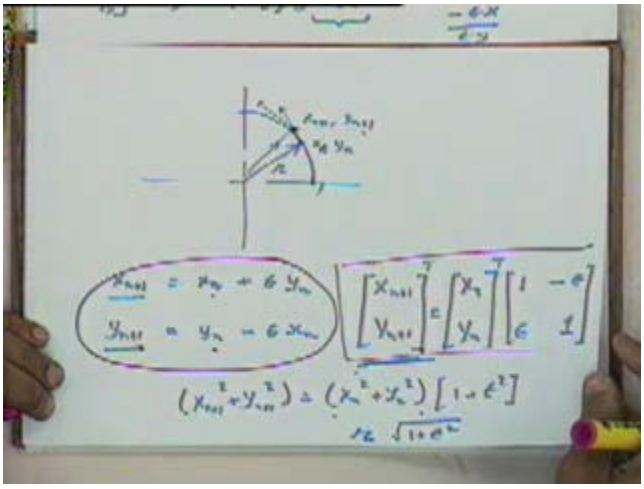


Now the next thing that we will take up is how to draw curves, how to start with, drawing of circles. For drawing circles we will again use what is referred to as a DDA algorithm, digital differential analyzer based algorithm. In this DDA algorithm when we are drawing a straight line, we had said that the slope was equal to a constant which was $y_2 - y_1 / x_2 - x_1$. If we have to draw a circle then let's say this is the origin which is 0 0, at any point x y what is the slope? For a circle how much is $\frac{dy}{dx}$? [Conversation between Student and Professor - Not audible ((00:36:51 min))] it will be minus x / y. You can verify it very easily, you know the equation of a circle is $x^2 + y^2 = r^2$. The circle is passing through the origin, differentiated with respect to x

we will get $2x+2\frac{y*dy}{dx}=0$ and this will give us this, $\frac{dy}{dx}=\frac{-x}{y}$. So basically we were using this fact that $\frac{dy}{dx}=\frac{-x}{y}$.

If you remember the line drawing DDA algorithm, we had said that if x y is a pixel that has been drawn at a particular stage, the next pixel to be drawn will be x + ε*Δx, y + ε*Δy. So this increment was proportional to $\frac{dy}{dx}$. We will do the same thing now, we will say that from x y the next pixel to be drawn will be by an increment proportional to this $\frac{dy}{dx}$. So if you are taking increment proportional to this $\frac{dy}{dx}$, my x = x + ε*y and y =y-ε*x , where $\frac{dy}{dx}=\frac{-x}{y}$. Okay. My increment in the y direction is –ε* x, my increment in the x direction is ε* y, so the ratio is - $\frac{x}{y}$. So we are taking an increment in both x and y, a very small amounts so that the next pixel to be drawn will be in the direction of the tangent, will be in the direction of the slope at that point. So you just look at this figure.

(Refer Slide Time: 00:39:40 min)



This is the pixel x y, we are talking in a. This is the initial pixel being drawn. The next pixel to be drawn is along the slope of the curve, the slope of the circle at that point. That will be a pixel let's say somewhere on this tangent at a point like this. So if I call this pixel, this particulate pixel

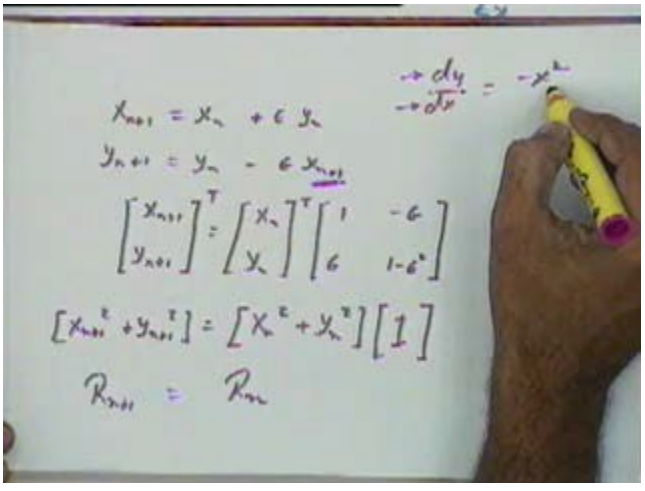
as x_n y_n and the next pixel to be displayed as x_{n+1} y_{n+1}. I can write $x_{n+1} = x_n + \varepsilon * y_n$ and $y_{n+1} = y_n - \varepsilon * x_n$. Sir if we want to draw n+2 then x_{n+1} y_{n+1}, the slope at that point will not be defined because it is not lying on the circle. I am just coming to that.

Now if x_n y_n is a particular pixel that has been drawn, the next pixel to be drawn we are giving by x_{n+1} y_{n+1} or I can write it in the form of a matrix x_{n+1} y_{n+1} is equal to x_n y_n multiplied by, what will it be? x_n and y_n, so 1 epsilon minus epsilon 1. Is that okay? is that okay? Thanks. So x_{n+1} y_{n+1} can be found out by this system of linear equations. Is that all right? Now if I keep applying this relation consecutively this is the first pixel that has been drawn, next pixel is this. The next pixel to be drawn would again be found out by the same relation, so my x_{n+1} y_{n+1} will become x_n y_n and then at from this point I will take a tangent like this and I will continue in that process. But effectively what is happening is this point x_{n+1} y_{n+1} is slightly further off from the center than the point x_n y_n. It is not exactly at the same radius. The difference will be very small but there will be a difference. This distance would be r and how much would this distance be? We can find out from here. If you want to find out the magnitude of or the radius of this point x_{n+1} y_{n+1}, we will get $(x_{n+1})^2 + (y_{n+1})^2 = [(x_n)^2 + (y_n)^2] * (1+\varepsilon^2)$, that is the magnitude of this determinant.

So the distance of this point x_{n+1} y_{n+1} from the origin will be, if this distance is $r = r*(1+\varepsilon^2)^{1/2}$ or the new $r = r*(1+\varepsilon^2)$, even though epsilon will be a very small number. But even then as we keep repeating this process throughout the circle, by the time I come back here I will not close here but I will close slightly further off. Some error will creep in because the same process is being

repeated throughout. So effectively we will get a spiral and not a circle because the radius will keep increasing by an amount of $(1+\varepsilon^2)^{1/2}$ or by a factor of $(1+\varepsilon^2)$ every time. To avoid that instead of using this relationship, instead of using this relationship or this system of equations what we will do is instead of using it this way we will make a small change so x_n we will make it x_{n+1}. And if we represent this in a similar matrix notation like last time you will get. you can verify the relationships.

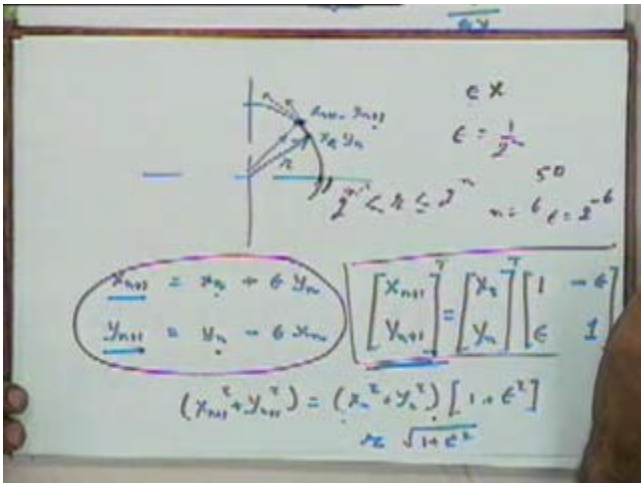
(Refer Slide Time: 00:45:23 min)



Essentially instead of x_{n+1} we put in this expression and then again we write it in the matrix form. Now if I look at the radius at this point I will get this multiplied by magnitude, the value of this determinant. What is the value of this determinant? one into one minus epsilon square minus this, which is 1one. So the radius here it is R_{n+1} will be equal to the radius at R_n. So just by making this small change, we will be able to get a circle accurately.

12

(Refer Slide Time: 00:47:13 min)



So the next point instead of being here will be at the circle boundary and the circle instead of becoming a spiral and ending here will end at this location. So this is how we can draw a circle using the DDA algorithm. If we want to draw any other curve, if we know the equation of the curve you can again find out the slope. For any curve if you know about the value of $\frac{dy}{dx}$ then we can draw that curve by taking increments proportional to dy and proportional to dx. Let's say

if I know the $\frac{dy}{dx} = -x^2$, this is for the sake of argument. Then I will give an increment in the y direction which will be proportional to x^2 and we can continue that way. What will be the exact proportionality constant?

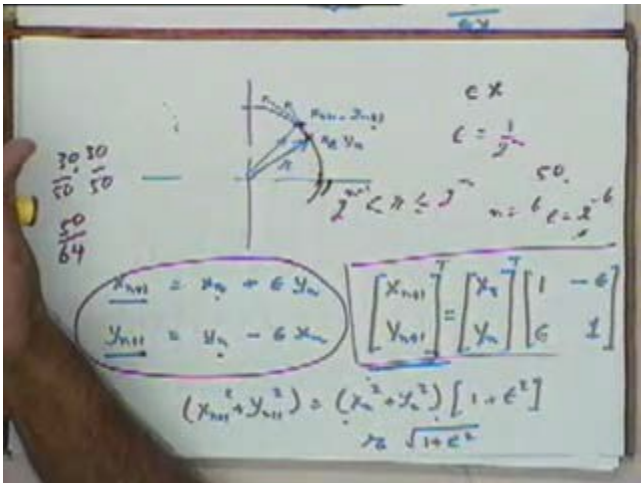
Now in the case of drawing the circle, I am multiplying ε by y_n. Here I am multiplying $\varepsilon * x_n + 1$. Instead of both y_n and x_n will be in the range of - r to + r. If you are drawing a circle x_n and y_n will always vary from - r to + r, so $\varepsilon * x_n$ or $\varepsilon * x < 1$ pixel that is our initial criteria. If it is very much less than one pixel then we do wasteful computation. If it is more than 1 pixel then we don't draw all the pixels, so epsilon x should be just less than 1 pixel. So again one criteria we can use is we will take epsilon to be equal to one over two to the power n such that two to the power n will just be greater than r. If $2^n > r$ and $r > 2^n - 1$. So let's say if the radius is 50 then our n will be 6 , ε will be 2^{-6}.

So if you are drawing a circle with a radius of 50 pixels then our epsilon is one by 64. So every time we give an increment that increment will be of 50 by 64 or minus 50 by 64 that's the maximum increment will be given. Why can't we put epsilon straight away 50? Because then at this point in the y direction my increment will become one pixel, it won't be and in the x direction of course at this point the increment would be zero. So here the increment in the y direction will be one complete pixel, we might not want that. We will have to do that if that is the case sometimes the circle at the end of the circle, we get a straight line sort of four quadrants.

13

Even in this case we would, even in this case you would get that because I don't know whether I have figure for that. Probably no. Anyway the pixels that would be displayed will look like this.

(Refer Slide Time: 00:51:12 min)

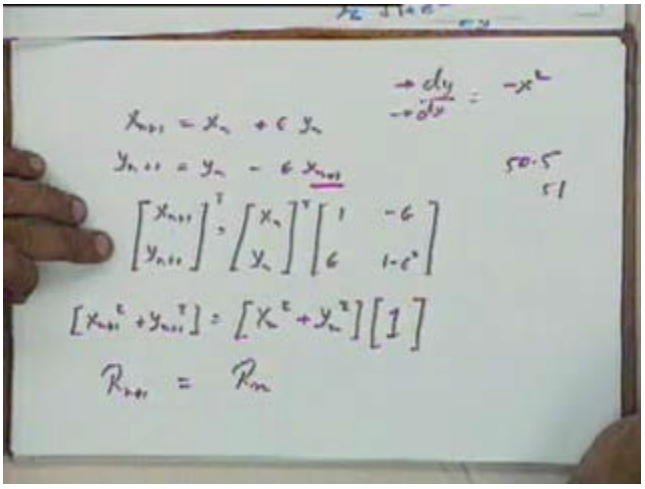


I mean a few straight then here like this, at the end will almost look vertical. Sir? Yeah. That is also have to do with division by power of two being easier. Division by power of two being easier. Yes sir. that is one because I am dividing by 2 every time, so if you have any number

dividing it by 2 means 6 right shifts, I mean in this case 6 right shifts or n right shifts but as if you are going to divide it by 50 every time at this point the value of x and y is let's say 30 and 30. We have to divide 30 and 30 by 50 which is you are going to take more time than just doing n right shifts, so that won't be there. So, for finding out the value of epsilon this is one way of doing it.

The algorithm will work equally well if you just take a random value of epsilon as choosing 1 by 50 or something like that. We will still get a circle, algorithm would work. Sir what would happen if epsilon x is equal to 0.5, epsilon into x. It can become 0.5 so which pixel would we choose or next pixel or would we still remain on the same. We are rounding it off. If you remember in the DDA algorithm earlier we were rounding off whatever value we were computing, so whatever value for x_{n+1} and y_{n+1} we get, we will round off that value. If you want to change it to truncation you will have to add 0.5 every time and so on, you will add 0.5 once in the beginning and take care of that but nevertheless effect is of rounding it out.

14
(Refer Slide Time: 00:53:17 min)



So if x_{n+1} comes out to be 50.5, we will take it to be 51. Any other question on this? In that case I think we will end today, we will end here today and in the next class we will go onto some other algorithms for display purposes specifically algorithms with respect to may be clipping and windowing and transformation and so on. Okay. That's all.