

Computer Graphics
Prof. Sukhendu Das
Dept. of Computer Science and Engineering
Indian Institute of Technology, Madras
Lecture - 13
Scan Converting Lines, Circles and Ellipses

Hello and welcome to the lecture on computer graphics. Today we are going to learn about algorithms by which lines, circles and ellipses are drawn in computer graphics.

The title of the lecture is scan converting lines, circles and ellipses. And the word scan converting itself gives you a meaning that we have to basically scan and convert the lines from the frame buffer on to the screen. So basically you have to fill up pixels in the frame buffer and automatically the lines and curves will be drawn on the screen. Now if you remember there are basically almost four different attributes of drawing in computer graphics, one is lines, second arcs, curves and ellipses. They fall under one category in fact circle is a special type of ellipse and arc is also a special type, it is a part of circle and ellipse. So, arc, circle and ellipse all fall under one category.

Straight lines fall under one category and then polygon drawing and its filling is the third type which we will see later on in due course of time. And of course those form a combination of line drawings. Using line drawings we basically draw polygons on the screen and also fill them with different types of patterns. So that is the third category and of course the fourth category is drawing text on the screen. So out of these four different attributes of drawing lines, arcs, circles, ellipses or polygons filled or empty and text, we are going to study the first one.

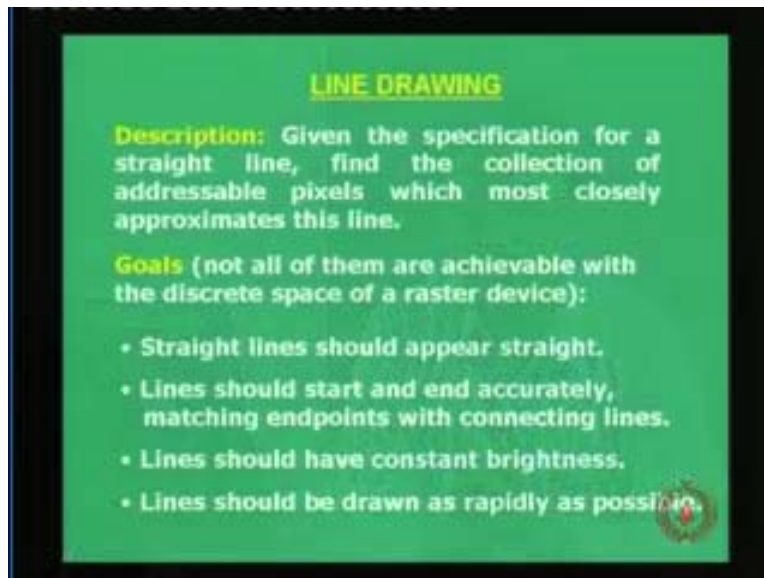
First let us see the method of drawing a straight line on the screen. And of course that seems to be the most easiest one out of these four different attributes, basic minimal attributes necessary for any graphics drawing package, software or standard whatever you mean. And we will see why for drawing lines you need an algorithm to draw a line, because the straight line is the easiest thing which a baby even can draw in a school in the primary section. If you give him a piece of paper and a pencil or a pen and give him a scale and he just draws a line, puts the scale and draws a line. And the equation of a line is also very straight forward, is the simplest one. When you read equations and read geometry, the equation of the line comes on, first linear expression and there should not be any problem.

So why do we talk of an algorithm for scan converting lines, we will see that. And before going into that of course I will giving a hint why you need an algorithm, because when we draw a line, when you think of a line in a graph paper or in a plain paper or whatever the case may be we are provably in fact drawing a line in an analog environment where all points are defined or for xy different coordinates.

But in the case of the computer graphics you have a digitized raster environment so xy are defined as integer coordinates and you need to actually find out what are those coordinates which fall on the line before you draw it. So that is the problem in the digitized space, we do not have continuous values of x and y like the analog world. And hence in the case we need an algorithm by which you draw a line.

What are the main points, what is basically aligned, what are the key issues in drawing a line? Well the problem now can be posed as that; given the specification for a straight line find the collection of addressable pixels which most closely approximates this line.

(Refer Slide Time: 9:36)



Now you can almost start to visualize, if you remember the lectures on display devices raster in all that you can visualize yourself the digitized space or a raster space as the matrix or an array of pixels and definitely if you draw an arbitrary line, not all of this square pixel blocks or raster position will fall on a line. So we are talking of approximately representing line in that sense truly of course in analog environment you can say there are basically infinite points which lie on the line. That is not the case, we have a finite set of pixels which fall on a line and you have to find out which are those finite pixels.

When you define a line you can think of the equation of a line or the starting point $x_1 y_1$ and the finishing point $x_2 y_2$ of a line and you can simply draw a line by a scale that is fine. But in the case of a graphic screen which you are viewing now in a TV or a CRT monitor you have to find out what are those pixels starting from $x_0 y_0$ and so on up to an end point x and y or x_1 and y_1 . What are those points or the addressable pixels is the term used here in the problem definition which falls in this line. So that is the problem posed now, I read it again. Given the specification of a straight line either in the form of an equation or the starting point and the end point, if that is given to you find the collection of addressable pixels which most closely approximates this line.

So that was the definition which was given and so if you look into the next line the goals of this solving the problem and of course I must say here that at given point of time not all of them may be achievable with discrete space of a raster device, so that is the discretization of the raster device is the one which is causing the problem. We will see lot of examples, what is this, illustrate with examples that are there in the next one hour or more where we will see what is meant by the problem approximating this line and drawing a line in a discretized space of a raster device.

Let us find out the goals of drawing. Straight line should appear straight. Now again you start thing in the rasterized scenario or you can wait till I bring up the illustration of drawing a line in a discretized environment because a straight line is a straight line.

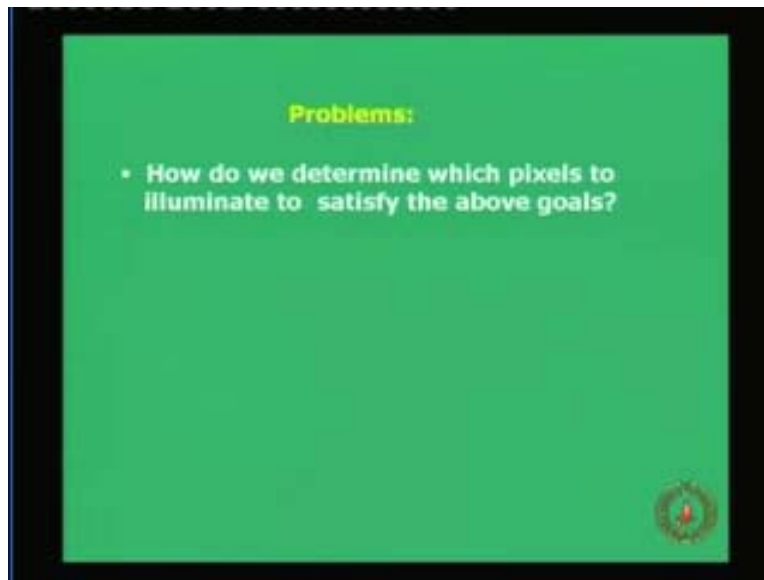
If you think from a mathematical perspective or from a child as we say we give a scale and you can draw a line, a straight line is a straight line. Unless the scale shifts or the paper shifts as long as it is a rigid environment you draw a line it always appears straight. So what is the issue here? And when we talk about straight line should appear straight, yes this is because the rasterized environment which we will see that it makes the line appear not crooked but something else which we will see. So our motivation or the goal here for the problem is to find out those addressable pixels from the starting point to the ending point in this discrete array of pixels where in such a manner that the straight line should appear as straight as possible, as linear as possible from the starting point to the finishing point.

So that is the first goal, a straight line should appear straight and you can hold on till the examples, come to find out what these goals basically mean, we will have lot of examples today. And the second goal as it appears on the screen now is that the line should start and end accurately, that is very very important. Actually we can start from a point and reach the other point and draw the line in the reverse fashion. But whatever the case may be you should draw the line accurately and should not expect to come up with an algorithm where you start at a starting point and finish somewhere near or close to the ending point. That will not solve the purpose and that is not the goal of computer graphics. It should start at the starting point and end at the finishing point that should be the case and matching endpoints with connecting lines. That is the case which is very important, two main goals are it should appear straight and it should be accurate.

The lines should have constant brightness that is also the case. And the question comes is when you probably take a pencil and draw the line it typically appears constant in a paper but what is the issue here when you talk of the line having constant brightness? Well you will see that the addressable pixels which you have drawn may not be uniformly spaced along the lines in terms of the distance between corresponding pixels as they come from the starting to the finishing point and if the addressable pixel if the distance between them start varying from small to large from the starting point to the finishing point, the pixels which are there, their gaps are non-uniform and what will happen? The line may appear slightly darker at certain small sections and **lighter in certain other.**

We will also try to see how to make a line with a constant brightness. And the last and most important point as far as Computer Science is concerned for any algorithm it should be as fast as possible, the complexity should be made as minimal, the cost of computation because you have to calculate certain value and it should be drawn as rapidly as possible such that the line is drawn very fast. So what are the problems associated with drawing a line?

(Refer Slide Time: 10:00)



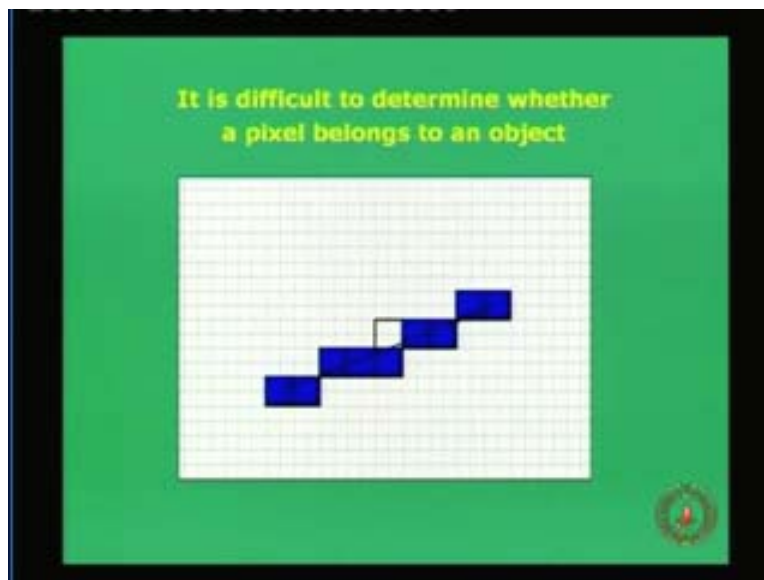
Basically the main goal of the algorithm or the problem is to determine which pixels to illuminate to satisfy the above goals. All the pixels are dark, if you want to just draw one line on the screen you have to eliminate a set of pixels and which are those pixels? Pixels from the starting to the finishing point are the pixels which you have to illuminate up, fill up the raster such that the straight line appears straight, accurate and uniform brightness.

Now this problem of the line appearing not very uniform in terms of brightness and straight does not happen in all cases of line. Out of all possible lines which you can draw in the screen, the lines which are vertical on the screen exactly vertical, horizontal it means typically we are talking about along the x axis of the screen and y axis of the screen and lines with slope plus and minus 1 that is the angle between the line on the x or y axis is 45 degree or $\pi/4$ radians. These are the three or four special cases where you actually get a straight line and they are the easiest to draw. We will see that as well.

For all other cases, for all other lines it creates problems. That means you have an effect what is called as a stair casing effect; it is also called jaggies or also called aliasing. These three terms will be used interchangeably throughout this course and we will soon see what is meant by the effect of stair casing of a line in general. Remember the second point which we discussed just before, the effect of stair casing, jaggies or aliasing do not occur in the line if the line is absolutely vertical, horizontal or to the slope of the plus or minus 1.

These lines appear absolutely straight. There is absolutely no problem. Of course in these cases also we have to find out using the same algorithm what are the points in between the addressable point which will be illumination. But the problem of the stair casing effect or the jaggies or aliasing together are the terms which are used interchangeably sometimes called as an aliasing or the stair casing effect, these problems will not be there for these three or four special cases whereas for every other line you will have this effect and the quality of the line drawn depends on the location of the pixels and their brightness. So that is also very important that the location on the pixels on the line determine the local brightness in the line and that local brightness should be uniform from the starting to the finishing point. So these are the various problems.

(Refer Slide Time: 12:41)



This is an example of an effect of stair casing, jaggies or aliasing which you are talking about. And therefore the problem comes because it is difficult to find out which are the addressable pixels which must be eliminated. In this case I am trying to draw a line from the left end corner or here at the bottom left of the line to the top right. So the starting and ending pixels are given and the question comes is which are the pixels in the line you can almost visualize that the line does not actually pass the center of any pixel so you have to eliminate a set of pixels to make it appear very straight and you can see here in the middle there are two options for you, you can eliminate the top white portion which is either left or the bottom blue one. In both cases it is the same because the line passes through in the center of these two pixels and you can actually have two different lines drawn for the same straight line, same specification of the starting point or ending point or slope and in set of the line. We will see the equations of the line.

But in this case you can see the effect of stair casing as you move from left to right, left bottom to top right it appears as a stair case and that is why it is called as a stair casing effect. Or the line appears as jaggies, have you moved through the analog line, analog

line means a straight line exactly as you have drawn in a graph paper, the line appears very crooked and jaggy from left to right and it oscillates and that is the effect which is predominant in almost any line except special lines. And then again now you can almost visualize, you basically need a very fast algorithm to find out which are these lines from the bottom left to the top right of the screen to be drawn here are illuminated such that you have the uniform straight line. That is the problem.

Of course we will take numerical examples to find out how to calculate and get those pixels. But this is an example which illustrates two facts, the one fact is that there is an effect of stair casing or jaggies as you move the line dances up and then again there are two options in certain cases, you have two options of choosing this or that pixel and it is sometimes difficult to find out which pixels you will draw, actually you will have uniform illumination and that is the main problem. As far as the third and last goal of the algorithm is concerned is, now the question comes is, how fast you can draw because the algorithm must be so fast that you must be able to draw a line instant in a scale.

If the algorithm is slow you are illuminating one pixel after another and nobody will purchase your graphics software because you are not going to draw only a single line, you are going to draw an entire picture which consists of lines, arcs, text, polygons, shading, textures and all that. And to draw a line itself if it could be a very slow process, to a user basically he sees one line drawn nearer then another line and so on. To complete one step itself if you take a second or more and the entire picture a few minutes then nobody in fact will draw a line. In fact you should draw a line as fast as possible so that the user cannot perceive that the line is being drawn. The algorithm must be very fast. So you need to do some computation very fast and in the integer space in the discretized raster space such that the algorithm is drawn very fast.

I repeat again that for an analog case as far as mathematics is concerned, given the specification of a line you can always draw a line in a graph paper. There is absolutely no problem. But that is not a simple transformation form, the analog graph paper or a white paper in fact to the discretized skin where we have an array of pixels, discretized environment and you have to select certain pixels, certain addressable pixels in the raster map and it should be drawn. So these are the two problems which hamper us and of course there are good algorithms which we will see. We will see a slightly simpler algorithm to start with and then we will come up which may not be that fast then we will see how fast an algorithm can be designed. Let us look at a solution for trying to draw a line. The way we know we have drawn a line in a graph paper.

(Refer Slide Time: 19:13)

Direct Solution:
 Solve $y=mx+b$, where $(0,b)$ is the y -intercept and m is the slope.
 Go from x_0 to x_1 :
 calculate $\text{round}(y)$ from the equation.
 Take an example, $b = 1$ (starting point $(0,1)$) and $m = 3/5$.
 Then $x = 1, y = 2 = \text{round}(8/5)$
 $x = 2, y = 2 = \text{round}(11/5)$
 $x = 3, y = 3 = \text{round}(14/5)$
 $x = 4, y = 3 = \text{round}(17/5)$
 $x = 5, y = 4 = \text{round}(20/5)$
 For results, see next slide.

We look at the equation of the line in a screen where we say that the direct solutions say the simple expressions of the line which gives, you try to solve the equation y equal to mx plus b where m is the slope of the line and $0, b$ is the y intercept of the line, we all know this, m is the slope b is the y intercept and you can pick up any values of x substitute on the right hand side of the expression mx equal the plus b and compute the value of y which we will get on the left hand side, so that is very simple.

You can substitute any value of x but unfortunately x values are discrete set of values and let us say my x_0, y_0 is the starting point and x_1, y_1 is the finishing point and end point of the line. So here what you must keep doing is substitute values from x_0 x_0 plus 1 and so on up to x_1 and substitute in the equation mx plus b here and get the value of y . Even if x is the integer value the problem which comes here is the value of m and b , b could be an integer location but the value of m is definitely in general a floating point number. And f is a floating point number and even in spite of the fact that x and b , I can put a constrain that x and b are integer values because I am drawing it on a graph paper with integer coordinates, let us say and origin screen with the integer coordinates pixel values.

So, if x and b could be integers and m is the floating point, that is enough to give us a value of y which is a floating point number. So you have to round off the value of y , y round off? You can usually guess, you will see that you do not round off the value of y or take the next highest or the next lowest then the line does not appear well. You need to round off the value of y from the above equation mx plus b or the value of mx plus b which is the floating point number must be rounded off to get the actual integer of the value of y .

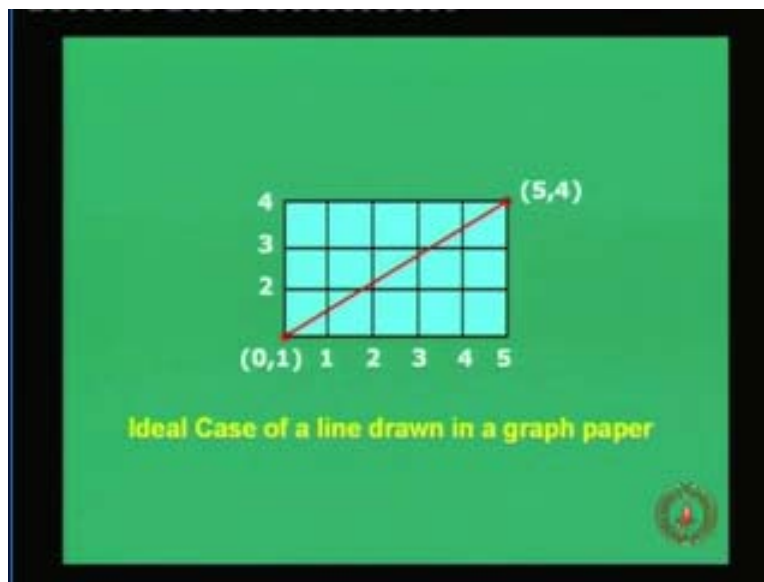
We will take an example and compute certain values of integer coordinates. Let us say I take a simple value of b is equal to 1, so my starting point is 0, 1 for the line and the slope is a fractional number say 3 by 5 which could be point 6 which is in fact point 6. I represent it as 3 by 5 or you can write it as point 6. And try to calculate that if I start to obtain the value of successive values starting from the location 0, 1 and the next

successive five values, you see here they are given in this table and the bottom where if I substitute x equal to 1 in the expression mx equal to b where mx is equal to 3 plus 5 and the value of b is equal to 1 you get a y which is round of 3 by 5 plus 1 which is 8 by 5 so round off 8 by 5 which is equal to 2.

So you rounded off and get the integer value of 2. You can keep on substituting, next take the value of x equal to 2, what you will get is under mx plus b you will get 6 by 5 plus 1 which gives you round of 11 by 5, round off 11 by 5 is 2 for x equal to 3 you get round of 14 by 5, why? Because 3 into 3 by 5, 9 by 5 plus 5 so it gives you 14 by 5, x equal to 4 gives you 12 by 5 plus 1 which is 17 by 5 and x equal to 5 gives 15 by 5 plus 1 divided by 5. So it is 20 by 5 so round off 20 by 5 which is exactly 4. And so the ending point is fine but if you see the other values around 17 by 5 will give 3 round 14 by 5.

These are now the integer coordinates obtained by using the calculation mx plus b where b is equal to 1, m is equal to 3 by 5, starting point is 0, 1 of the line, finishing point is 5, 4 and these are the integer coordinates 1, 2, 2, 2, 3, 3 and 4, 3. These are the addressable pixels or integer coordinates of the line which we obtain by this calculation. So let us look at the result, actually if you have given a starting point 0, 1 and finishing point is 5, 4 you can easily calculate yourself the equation of the line and say that the intercept b is equal to 1 and the slope is 3 by 5 because it starts at 0, 1 and ends at 5, 4 or it starts at 1, 2 and ends at 5, 4 and you can draw the line in this form.

(Refer Slide Time: 21:21)



So 0, 1, so there it is basically 1 2 3, so 3 by 5 is the slope of the line because the vertical disparity is exactly 3 and the horizontal disparity is 5, slope of the line is basically 3 by 5, 4 minus 1 the numerator 5 minus 0, so it is 3 by 5 as you can see. And if you have given a graph paper to a boy in a school or a child in a school you would have drawn this line from the starting point 0, 1 to the finishing point 5, 4. So this is an example of not the

picture of the raster line drawn on the graphics screen, this is the ideal case of a line drawn on a graph paper, you just have to worry about 0, 1 and 5, 4 or vice versa.

So you can start from 5, 4 and end at 0, 1 also. The starting and ending point could be swapped but still you get the same line and you draw the line on the graph paper. But this will not suffice in a raster screen, why? As you see the integer position, the pixels, the horizontal grids, those intersection points are the points where the pixels lie. Of course the starting point 0, 1 at the bottom left and 5, 4 on the top right at the starting and finishing point. So those pixels of course have to be switched on, that is fine.

What about the rest in the middle? As you can see the intersecting grid lines in the locations of the pixels which include of course the starting and finishing point also, where is a starting point? Here 0, 1 and the finishing point 5, 4, but if you take the other integer coordinates which are the intersection points of the grids the line does not pass through any of the intersection points, it is bound to happen in general for almost all lines except the lines which are vertical, horizontal or of course bearing the slope of plus and minus 1, those are special cases.

Otherwise in general a line is almost, it is not guaranteed but in general what will happen is it will not pass through any intersection point. This is what we see in this example that these line does not pass through the intersection point and now the question comes is which of these intersection points which you saw on that screen just now have to be illuminated. They may not exactly lie on the line but you actually have to find out the closest of those integer pixels or addressable pixels or points on the screen which has to be switched on to give you a feeling that you are basically drawing a line which is almost straight.

Now you will ask a question, when I see a straight line on a screen I usually see it straight, I do agree with you in fact the screen which you have just seen now, I had drawn a line which is shown on a graphic screen, why does it appear straight enough. Well I must admit that you are seeing a line in a very high resolution graphics monitor and that monitor typically may have thousand pixels or by about few hundreds, by thousand grid and you have seen that line will not consist of only three pixels which can be illuminated on the screen, now we have just seen the red line. Now as an example, the continuous line may consist of several hundred pixels and those several hundred addressable points, when they are so very close and dense and lie almost close to the line you have the feeling that you are basically drawing a straight line and that is the advantage of perception of the eye which you try to bypass or cheat a little bit to give you an impression that all the pixels are not exactly lying on the line. It is so dense and so close and if you drawing a line of length of several hundred pixels let us say one hundred or more it will appear to be a straight line. But you can find the effect of stair casing and jaggies by two methods which are following.

When you are watching through your TV monitor or a computer screen, take a lens and try to observe a line very closely, a very thin line, even it could be a thick line. We will see the problems of thick lines later on in this screen. Let us worry about the line but the

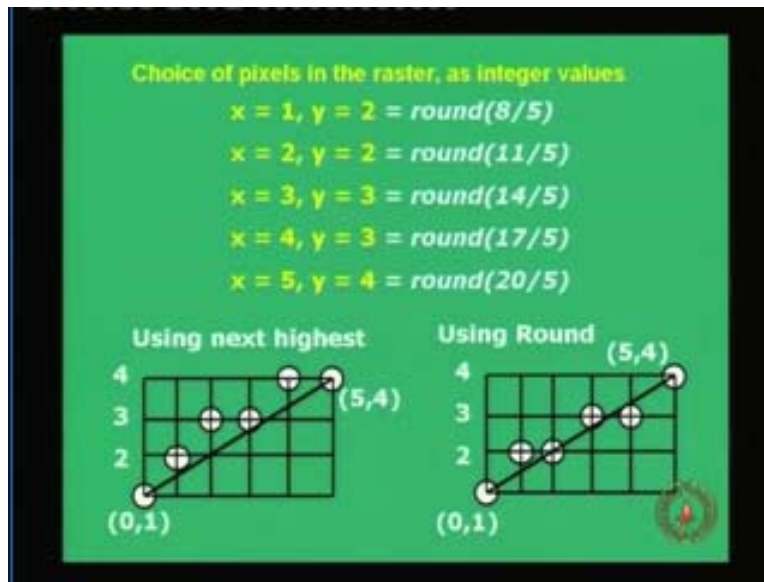
width is just one pixel. If you zoom on to that screen either using a lens or through some photo shop or paint brush type of software you will find that the pixel will start to glow up as square blocks. We will see with an example and I have got an example where I will show when you zoom close to a line how the pixels start appearing. And you will start to see the effect of jaggies from the starting point to the finishing of the line. So you blow out instead of blowing in and the line appears dense and condensed and it appears in the straight line. **So please do not have a feeling in mind that I am just saying** that the line which appears straight, why I am I saying that the line will appear straight, it has to appear, the stair casing effect has to appear because you are working in a rasterized digitized environment. When the pixel addressable points are very close they will appear almost in a straight line but still we should try to make an attempt to draw a straight line as possible by selecting the addressable points which are very close, that is the problem which I have just shown now.

And of course the second problem is how fast you can draw the line. That is the most important part where we need the computer scientist and the computer science based students will be very happy to come up with a algorithm where we were in need of processing things very fast, come up with optimal algorithm followed to solve that. But we are still in the environment where we substitute the value of m into mx plus b , m is the floating point number, x and b could be integers, so the resultant value is the floating point number used on operation to calculate the y and x , y integer value is u select.

Let us see with the example which we have just solved now. This was the example we just solved now, the starting point was obtained as 1, 2, the finishing point of the line was 5, 4 the rest of the values in between where obtained using round functions. So these are the examples which we just solved about one or two slides back. And why the round function is important I did say that, that is probably the closest integer pixel which you get.

If you do not use round function then what will you use? Of course you cannot get a pixel at the fractional numbers which are put on the right hand side that is 8 by 5, 11 by 5, 14 by 5, 17 by 5 these are all floating point numbers. There are no addressable pixels available at any arbitrary floating point numbers. So you will say instead of rounding I will choose the next highest number or the next lowest number. If you choose the next highest number let us see what happens. If you choose the next highest number which is starting from 0, 1 and then 1, 2 that is the next highest instead of element by 5 the next highest number will be 2, 3 or 3, 3 of course the next highest then you get 4, 4 and 5.

(Refer Slide Time: 28:54)



You can see these are the next highest. Instead of using the round function what has been done here is you have chosen the next highest integers with respect to the values given on the right hand side. The next highest of 8 by 5 is 2. So you have 1, 2 the next highest of 11, 5 is 3. So you have 2, 3 instead of 2, 2 and 14 by 5 the next highest gives you 3 that is fine. But in 17 by 5 the next highest is 4. So these are the pixels which you will get if you chose the next highest you can visualize yourself that if you choose the next lowest instead you can just almost get the mirror image of what are the pixels which we have just seen on the screen. And as you can see on the screen the pixels which you have seen they are all on one side of the line and that is not what we are looking for, the line will appear to be shifted away little bit only the starting and ending point will be same and all the pixels appear to be outside.

If you use the round function what happens, this is now the figure which we see here instead of next highest, if we use the round functions and we already have used it and obtained the integer coordinates of x and y as 1, 2 2, 2 3, 3 4, 3 and 5, 4 and they have been drawn on the screen starting from 0, 1. The starting point to 5, 4 1, 2 of course is the next addressable point. Then 2, 2 here 3, 3 here 4, 3 here 5, 4 is the last finishing point.

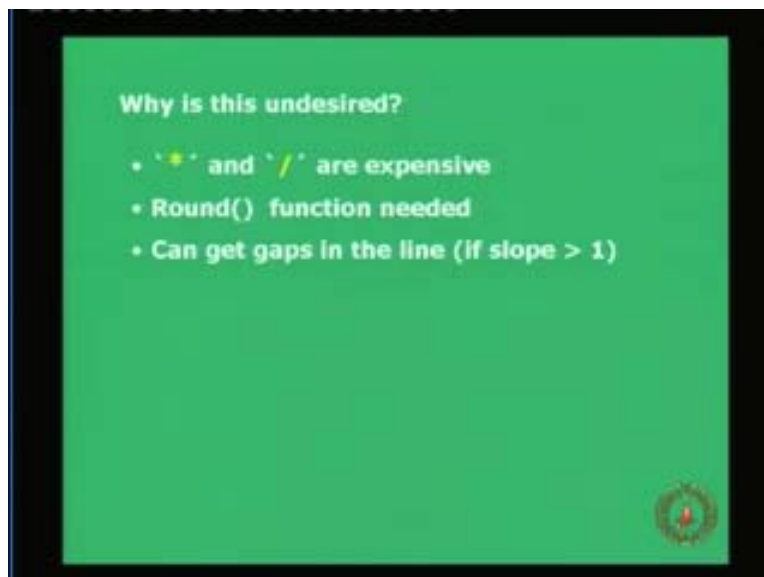
Now you can see there is a lot of balance, there are few pixels on the top of the line, some on and some at the bottom of the line. This is what is going to happen and you will have a uniform split of a view point on both sides of the line and typically this is scenario which you will get a numerical company exercise of what is called as least square equation of a line where these points are given you need to find the equation of a line. But that is a different problem altogether and in this case of computer graphics you are expected to find out those points which are distributed over the line and the most closest. That one is most important that you need to find out the point which is the most closest and these are the points which are the closest by using the round function.

What we have done in the calculations so far? We know the starting point and the finishing point of the line from that we compute the slope of the line which is the m and of course we know the y intercept b that is fine and you know the starting point. So from the starting point to the finishing point x coordinates we keep on talking the next successive integer values that means keep incrementing by one and for each such step of after incrementing one what you have to do is you have to substitute in the equation mx plus b , get a floating point number rounded off, get a y value which is an integer and that is how you compute this. So this is not an algorithm but the method by which you get the nearest addressable one.

Why will I say that this is not an algorithm? It works fine, in this case it will give you the correct addressable lines, we can guarantee that, it is absolutely no problem. But there are almost two issues here in fact one issue, the problem comes because that means I say the method, why this is undesired or not wanted for, I do not want to use operators like multiplication and division which are expensive and also the round function at all.

At each step I have to multiply m by x and then add up to b , adding is fine and it is an ordinary floating point space, floating point based operations multiplication, division, addition, subtraction are most costly than integer operation. We know this from basics of computer science of arithmetic, the floating point arithmetic is much more costlier than the integer arithmetic because the hardware is suited, there are very fast algorithms and hardware available to do integer computations in terms of addition, subtraction, multiplication. In the floating point it takes about more than double the time, it could be even four times the amount we need to do an addition or multiplication using integer arithmetic. So all these floating point operations multiplication, addition and division are expensive operations and most over that at each step you need to have a round function.

(Refer Slide Time: 32:05)



At each step you need to have a round function plus integer arithmetic so that is where you lose time. You cannot generate the points very fast to draw the line and you know that you cannot generate points very fast you cannot fill up the raster space with addressable points and the line drawing process will be very slow. You need a very fast algorithm where given the starting point and the finishing point of a line or the starting point could be here and the finishing point could be there whatever the case may be you need to actually get the addressable integer locations very fast that the line is drawn almost instantaneously not like a pixel, then the second pixel, then the third pixel and so on.

If you have an algorithm by which each of the computation produce a significant amount of delay in generating all these integer pixels one after another from the starting to the integer point nobody will purchase your software, nobody will use your computer graphics program and it will not be approved by software specialists and specialist in the field of computer graphics and they will say we are sorry it is a true a slope program. It works fine, it gives you the exact position of addressable points, those addressable points are the closest to the line drawn, that is all fine but it is too slow, this is too slow because of these mathematical operations which take time multiplication, addition, division and the round operation.

Look at the n number of loops depending upon the number of points which you need to draw and for each loop you have a multiplication operation floating point, a floating point addition and a round function. Three floating point operations out of them the round is the most costly one and of course multiplication as well and they are the wants which is consuming time. And the other fact which you see now is that if you use this round function you can get gaps in the line if the slope is more than 1. Let us take this example as you see on the screen.

(Refer Slide Time: 33:59)

Why is this undesired?


- '*' and '/' are expensive
- Round() function needed
- Can get gaps in the line (If slope > 1)

Take another example:

$$y = 10x + 2$$

$x=1, y=12;$

$x=2, y=22.$



You can get gaps in the line if the slope is much more than 1. Let us take an example, the slope is very high where m the slope of the line is equal to 10 and the slope b is equal to 2, now the equation of the line y equals mx plus b is written as y equals $10x$ plus 2. Substitute x equal to 1, what you get is given already, y is equal to 12 substitute the next integer point x equal to 2 what do you get 20 plus 2 is 22. Now you can see you have two address, next addressable points differ by a wide margin x equal to 1 and here you have y equal to 12 and x equal to 2 and y equal to 22, you cannot draw a line like this.

You cannot draw a line like this. There are methods by which these sort of problems are also handled and addressed while drawing a line. So you move towards the first algorithm which is incremental in nature, it tries to reduce the computational burden and tries to increase the speed of computing the integer addressable pixel occasions.

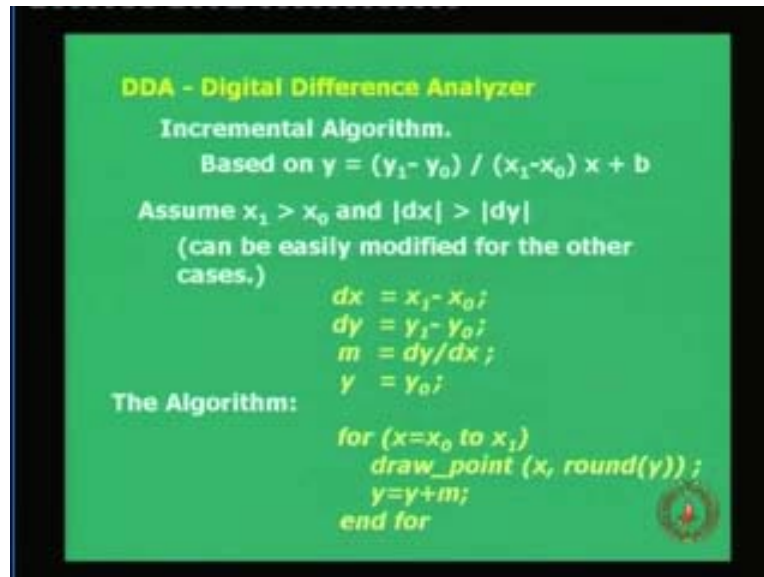
The first of the most primitive algorithm you will find them in some old books mostly than the current books which is called the digital differential analyzer or digital difference analyzer algorithm popularly called as the DDA for computing the integer locations of the line. It is based on incremental arithmetic, incremental algorithm and what is basically done is the same equation of the line y equals mx plus b is used and the m is obtained by the ratio of the y coordinates of the starting and finishing point and also the x coordinates of the starting and finishing point, that ratio can be taken and we are substituting that in place of m , the starting and finishing point x and y coordinate to give you the m . That is the equation of the line.

Based on this equation we assume for the time being that the value of x_1 is more than x_0 , that means you must start from the left hand side of the screen and moving towards right. The starting point is less than the finishing point that is why x_0 is less than x_1 and x_1 is more than x_0 and dx by dy which are nothing but the y_1 minus y_0 as given in the expression of m as y_1 minus y_0 assume that to be the dy and dx is nothing but x_1 and x_2 . We assume that dx is more than dy basically means that the value of m will be, if dx is more than dy and m is dy by dx this value of the slope of the line is less than 1.

We assume these constraints, now you need to draw a line from anywhere to anywhere on the screen, that is true. But what may happen is it not always the case that you start from the left in most points and go to the right and the slope is less than 1 and all that.

But in the other case you need to draw a line from here to there and all sorts of positions are bound to happen but we will tackle all the situations later on but right now let us assume that from your left most point to the right top point is what you are drawing a line. You are selecting the addressable pixels from left bottom to top right, you need a good algorithm to get these integer locations number 1 and the slope is less than 1, it is not 0. These are the two conditions we will worry about, the rest in a couple of slides as we go ahead.

(Refer Slide Time: 37:41)



So come back to the slides and we assume that x_1 is more than x_0 , dx is greater than dy that means the slope is less than 1 and we will see that we can easily modify the other situations to handle all the other possible cases of drawing the line. What is the DDA algorithm? As I said before that the dx is defined to the first four statements of this algorithm here on the right bottom of the screen gives the initializing condition, that is not the loop, dx is what I just said sometime back x_1 minus x_0 , dy is y_1 minus y_0 , the slope of the line m is given as dy by dx .

Now since x_1 x_0 y_1 y_0 are integer coordinates dx dy will be also integers. But the ratio of two integers is not guaranteed to be an integer quantity. In general it will be a floating point, so just remember here that the value of m or the slope of the line is going to be floating point number, y is equal to y_0 that is the starting point, you know it starts at x_0 y_0 and so what we do basically is, you actually draw the first point x_0 y_0 and then start, you make an iterative loop, actually I have written this in pseudo code, not follow any syntax of languages like c, it could be Pascal like Algol like, so you make a loop from x_0 to x_1 . You start from the x coordinates of the first point of the starting point and wind up and finish at and the end point of the x coordinates and at each point you draw the lines.

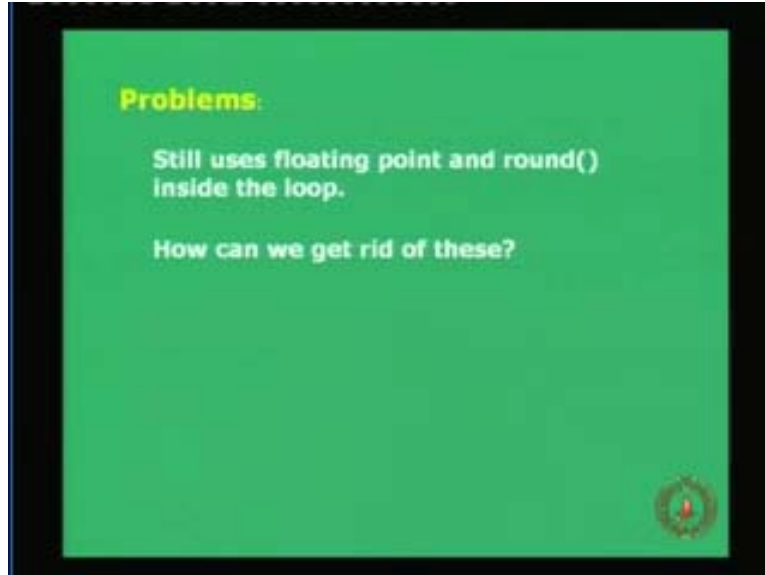
The first point of course will be y is equal to y_0 , so you draw the first point which is x_0 and round of y_0 which is nothing but the y_0 itself. Then what you do is this is the incremental algorithm that is why you increment y by the value m . You increment y by the value m , remember y is an integer but adding a floating point number will actually return a floating point value on the y . So in the next loop what will happen is in the draw point command the round function will return an integer point or integer value of y and that is what the draw point commands basically illuminates, let us assume that this draw point is a very low level liability function where given two integer points x , y it draws a point on the screen.

One addressable point given by the integer coordinates x , y both are assumed to be integers. In the next calculation in each loop y becomes the floating point but the round y will return an integer. So that is what the end of the loop, so you see the end of the loop now, you have a floating point addition and a round function. The incremental algorithm has abolished the floating point multiplication. So that is the small advantage, so it will become a little bit faster but not that much because there are still two costly functions within your loop, the loop has n number of steps, n number of steps depending upon the number of points to draw on the screen x_0 to x_1 , that is the integer difference.

There are two costly functions; one is the round operation, another is the floating point addition. If you can eliminate these two, of course you cannot eliminate both cannot draw a line, how to eliminate? If you can eliminate the round function which is the costliest and still exists in the DDA or the incremental algorithm and replace the floating point arithmetic of addition by an integer arithmetic which will be several times faster, then eliminate some of the round operation. You can almost imagine that for a line to be drawn with several hundreds of points on the screen the speed up will be tremendous, many many times more than you would like to have for a DDA type of an algorithm which has floating point addition and the round function. So DDA is an incremental algorithm, it has speeded up the algorithm a little bit but it is not the most efficient one.

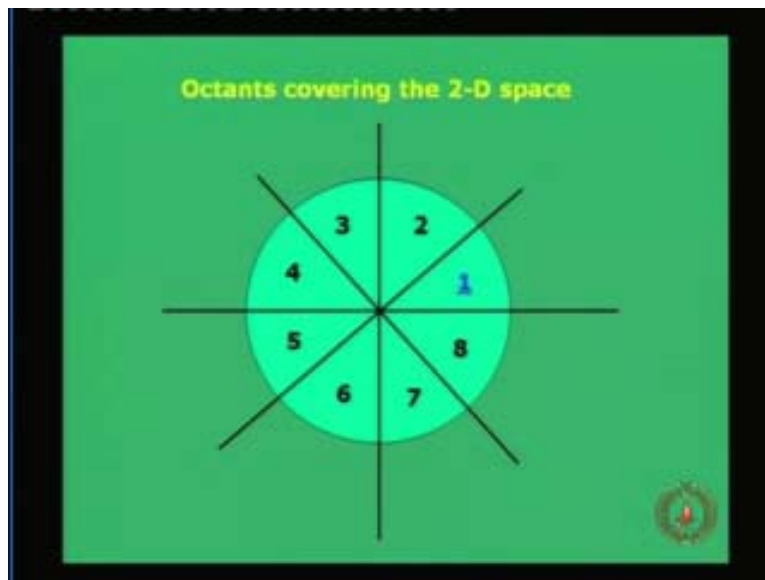
The most efficient one which is going to be discussed now in the next few minutes left in this class. So we carry over also to the next class because it is a very important concept which you must know is the method of the mid point criteria of drawing a line and it is based on the idea or the concept developed by Bresenham's, it is also called the Bresenham's line algorithm which is integer based, fully on integer calculations, integer based arithmetic is what you use and it is based on mid point criteria for drawing lines, so that is in it.

(Refer Slide Time: 41:58)



Now let us go back to the problems which we had in DDA.

(Refer Slide Time: 42:24)

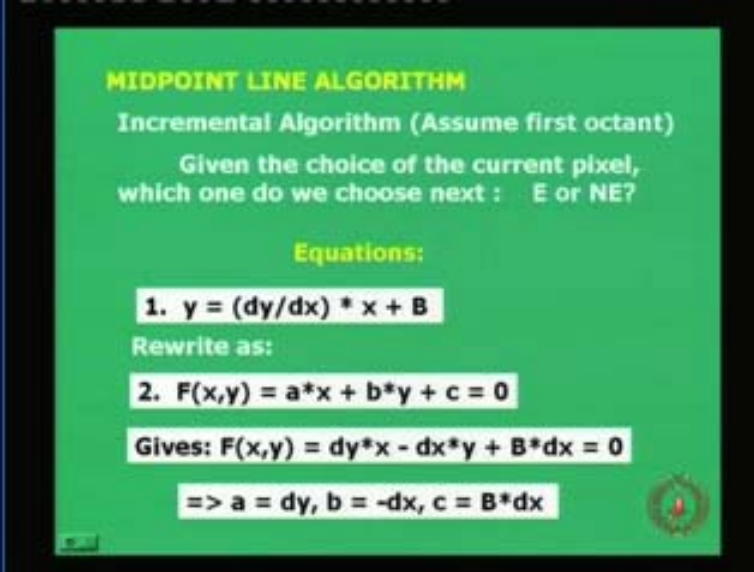


Again that it uses a floating point and a round inside the loop. So the DDA in spite of doing incremental has the problem is still slower because of these two draw backs, the drawbacks are the floating point and the round function inside the loop and you have to get rid of this. Of course you have to get rid of the round function. The addition floating point can be made integer; you have to add anyway, you cannot escape that. But once you make the floating point operation integer and throw off the round operation to **speed up will get the image**. The other part is which we talked in the DDA algorithm is the octants

covering the 2D space. We assumed that you are drawing a line from the left hand side of the screen to right hand side, the slope is less than 1. So if you see all possible lines in the 2D space and there are eight octants, each with angles subtending of π by 4, that means you are dividing the 2D space into eight different octants.

We are basically solving the algorithm for the first octant only that means the slope is less than 1 from 0 to 1. And if the line lies in any of the other octants, that means the slope is more than 1 or the starting point is to the right and the finishing point is to the left, that means the line could be in the eighth or seventh octant or even third or fourth, then we will see what to do, because those are simple manipulation of certain numbers which you have to manipulate and get the environment done, solve it in the first octant space and then draw the line back. So we are assuming that the line to be drawn for the time being is drawn in the first octant only. That is one and we will see how the other octants are also handled. So we move on to the last few minutes of this lecture on what is called the Bresenham's algorithm or I will use the technical term of mid point line algorithm.

(Refer Slide Time: 45:59)



MIDPOINT LINE ALGORITHM
Incremental Algorithm (Assume first octant)
Given the choice of the current pixel,
which one do we choose next : E or NE?

Equations:

1. $y = (dy/dx) * x + B$
Rewrite as:
2. $F(x,y) = a*x + b*y + c = 0$
Gives: $F(x,y) = dy*x - dx*y + B*dx = 0$
 $\Rightarrow a = dy, b = -dx, c = B*dx$

It is also an incremental algorithm like the DDA algorithm but it is based on integer arithmetic and we assume that we are only working in the first octant. That means we are drawing a line from the left bottom to top right and the slope is less than 1. That is the first octant, we had seen the picture of eight octants covering the entire 2D space. Only the first octant is what we are looking through and once you are able to solve that, all the other seven octants also will be handled. So we look back into the slide and given the constraint that we are drawing a line only in the first octant.

Given the current choice of the pixel which could be the starting point or any other point, integer point at any given point of iteration we will see that the choice of the next pixel based on the assumption of the constraint of the first octant is only between two pixels which you will be labeling as the east pixel or the north east pixel. So the lines state that

given the choice of the current pixel which one do we choose next, east or north east? We will see what do you mean by the east and north east pixels as we go and look into the figure.

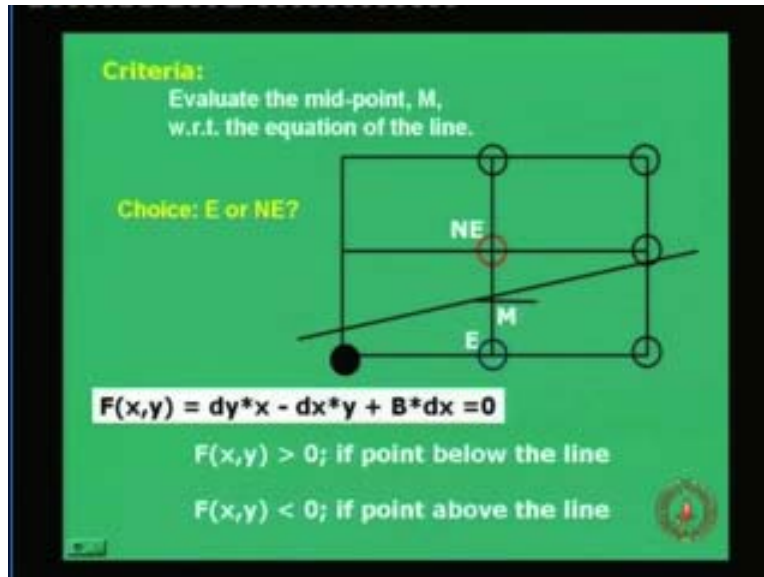
Let us look into the couple of equations today at least to visualize this configuration of east and north east and understand the mid point criteria at least today. The equation of the line as given in this parametric form is y equals $\frac{dy}{dx}$ multiplied by x plus B while $\frac{dy}{dx}$ is nothing but the slope of the line, $\frac{dy}{dx}$ you know is given by y_1 minus y_0 , x_1 minus x_0 , I think you have noted down those equations by now, otherwise please note down dy is y_1 minus y_0 , the difference in y coordinates of the starting and finishing points and dx is x_1 minus x_0 the difference in x coordinates of the starting and finishing point of the line and so this is the equation of the line, we know that. Now we rewrite the equation of this line y equals $\frac{dy}{dx}$ multiplied by x plus B as a function of $F(x, y)$ equal to 0.

It is possible to manipulate and put all the terms on one side and write the equation in the form as $ax + by + c$ equal to 0. This is also an equation of a line. You can actually find out that the slope of this line is going to be minus a by b . We can visualize this. You can easily manipulate this and this given in the bottom. If you rewrite the equation number one and try to put that in the form of equation number two, you will rewrite as $fx + dy$ multiplied by x minus dx multiplied by y plus Bdx equal to 0.

You please verify that yourself right now that you can rewrite the equation number one, then you try to write it as in the form of equation number two. In the process of doing so you get this $F(x, y)$ equal dyx minus $dx y$ plus Bdx equal to 0. And this equation is in the form of equation number two now as $F(x, y)$ equal to 0 where the parameters of the line now instead of being m and b only are capital B of course you must be careful between small b and capital B in this case which are different, in two different forms.

We can just simply substitute this equation number two and one given at the bottom, a is your dy which is given here, b is minus dx this also given here, remember this b minus dx and small c will be capital B multiplied by dx . So these are the three parameters of the line which we are used to and we will use now and what is the advantage of using this equation? Well you can use this equation to find out where is the mid point of the next choice of the line with respect to the equation of the line.

(Refer Slide Time: 52:05)



So you can see that this is where I describe what are meant by east and north east pixels. The two possible choices left to me, let us assume from this figure if you look that the black pixel on the bottom left of the screen, the black pixel is the current pixel or it could be the starting pixel also, of course it is the starting pixel, the line will start from here so I have taken a general case where you have used this mid point criteria to iterate and got a few points already. And at any given stage of iteration this is the current point which you have already selected. The next choice will be between these two pixels north east level as red and east leveled as blue, why? This has to be because of the constraint that you are working on the first octant and the slope of the line is less than 1. If the slope of the line is more than 1 it would have been a different case but since we do not consider the situation right now, the choice is between east and north east only. The m is the mid point between east and north east. If you draw a line east to north east, a vertical line the mid point is the bisector of the line.

So what you try to do is basically with the equation $F(x, y)$ is equal to 0 is, you evaluate the mid point m with respect to the equation of the line. So that is the criteria which is suggested by Bresenham's mid point algorithm here and the m is between east and north east and you need to find out whether this m is above the line or below the line because that gives you justification. As you see in these figures now if m is below the line, the line is closer to the north east and you will be choosing the north east. If the line was below m which you will see in another figure similarly, then of course you will now choose east. So the choice is now clear which we just discussed sometime back that you have to choose between two pixels north east or east. This is the current pixel at any given point of time, the next choice is between north east and east and you evaluate the mid point with respect to the equation of the line and find out where is this mid point located and based on this criteria choose east or north east.

How do we use the equation to find out whether the mid point is below or on the top of the line. What is the equation of the line? This was the equation of the line as given in this form because we started with y equals mx plus b and then we rewrote in the form of $F(x, y)$ equals to 0 as ax plus by plus c , c is b times dx , b is minus dx and a is dy . So this form we got in the previous slide. We had written this equation already.

Now what this $F(x, y)$ equal to 0 means? If you take a point which lies exactly on this line that means I select an integer coordinate or any fractional number in fact x , y and substitute in this expression, those coordinates of a point which lie on the line or exactly on the line, anywhere on the line but on the line is very important, then if you substitute assuming the other parameters of the line dy dx capital B and so on, the value of $F(x, y)$ will be equal to 0. That means the point satisfies the criteria that is on the line and the value of the function is exactly equal to 0 on the line whereas if you substitute the values for a point which is on the top of the line you are on the bottom and you will get the value of the function is not equal to 0. On the line the function is equal to 0 on the top and the bottom it is not equal to 0. But if it is not equal to 0 then it gives you two different conditions.

We can almost visualize from the equation of the line that the value of the function f , if the point is above the line, assume this to be a line, if the point is above the line in 2D space then you will get a positive or negative value and complete with reverse and the point is below the line. Let us look at this figure and try to see this is the basic concept here that, if you substitute a point which is below the line, in this case take the value of coordinates of m where m is below the line, if you substitute the xy coordinates of this point m and substitute in the equation of f what basically will happen is, this term minus dx multiplied by y will start dropping from the case if the point m would have been exactly on the line.

If the point m is below the line, this negative quantity will start reducing and force the value of f to grow positive. So the value of f will be positive, we just have to remember that the value of f becomes positive if the point is below the line and if the point is above the line, assume the scenario where m goes above the line or take the value of xy coordinates from the north east substitute from the value of m , what will happen? Is this dx multiplied by y and term will grow more forcing the value of f to be negative. So this is the main criteria that the value of f becomes positive for a point below the line and for the point above the line the value will be negative. It is negative on top of the line and positive below the line.

You can visualize that the function is equal to 0 exactly on the line, it is negative and positive, the entire two dimensional space on the screen is split into two halves where exactly on the line you have all 0 values, anything on the top you have negative, anything on the bottom you have positive. So it is positive below the line, negative on top of the line exactly 0 in the middle. So you are dividing the space equally into two different halves, the positive half and negative half and that is the concept which is used to evaluate the mid point. Now you can visualize that if the m is below the line you will get the f function to be positive and if m is on top of the line you have the value to be

negative above the line. So just looking into the sign of f itself will tell you whether m is on the top or below the line and that is sufficient for you to tell whether you choose east or north east so that is the criteria which we say.

We stop here saying that this is the mid point criteria. We will start from here in the next hour and move towards forming in the algorithm. But the mid point criteria is the following: I repeat again if you look into the slide the criteria for Bresenham's mid point algorithm is evaluate the mid point m with respect to the equation of the line. So, all points on the line will give a function value 0. Points below the line as this figure shows for the m will give positive values, points on top of the line will give negative values.

Look into the sign of f and decide whether the point is below or on top of the line. So in this figure as you see here, since m is below the line, if you substitute the m into the equation f you will get a positive sign and the sign of the function f is itself a sufficient clue to tell you whether the point is below or above the line or in other words the line is closer to north east or east. In this case since m is below the line, the line will be closer to north east so you will select the north east pixel as your next iteration. And the case is there where the point is below the line f is positive. The reverse would have happened if m would have gone on top of the line, f would have been negative and the line would have passed between m and E or it is closer to these pixels and in this case you would have considered the east pixel. So this is the criteria, a sort of the Boolean logic, look into the sign of f positive negative.

For positive take one decision of choosing east or north east and negative you choose the other one, a sort of a bi polar logic or Boolean logic and that gives you a very good algorithm to select the line. That part is done. But of course we have not seen where the integer arithmetic is coming from. Where the integer arithmetic is coming from? Throughout the round function will go away, the choice criteria you want is clear. We will start exactly from this mid point criteria and move forward and see how the integer algorithm is brought on with the help of this mid point criteria. The choice becomes very easy between east and north east, how the other octants are also taken into account and that completes the entire line drawing algorithm.

We will move to the next lecture and of course then we move a head with curves, equations of curves, circles and ellipses. Thank you very much.