# Computer Graphics and Image Processing

Slide Credit: Md. Neamul Haque, Lecturer, Department of CSE, Premier University

# Lecture : 01

## Pixel

- An image is composed of discrete pixels or picture elements.
- Arranged in Row and Column fashion to form a Rectangular picture area (Raster).
- Pixels are referred by their co-ordinates.
- Pixel at lower left corner of an image is considered to be at the origin (0,0) of pixel co-ordinate system.
- Find the co-ordinate of the pixel at the lower right corner of a 640x480 image.

Solution: (639,0)

- Find the co-ordinate of the pixel at the upper right corner of a 640x480 image.

Solution: (639,479)

## Rasterisation

- Rasterisation (or rasterization) is the task of taking an image described in a vector graphics format (shapes) and converting it into a raster image (pixels or dots) for output on a video display or printer, or for storage in a bitmap file format.
- This is also known as scan conversion.

# Resolution

- Number of pixels per unit length is known as resolution.
- How many pixels are availabe in a 3x2 inch image at a resolution of 300 pixels per inch?

Solution: Row, 3x300 = 900pixels

Column, 2x300 = 600pixels

Total = 900x600 = 540000 pixels

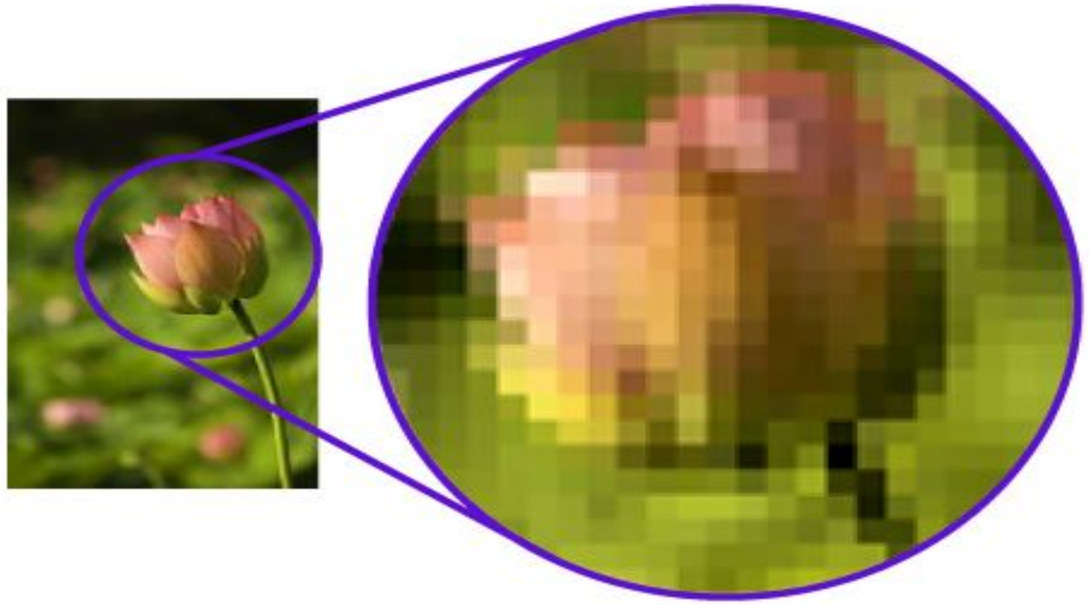
- Frequently image size is given as the total number of pixels in the horizontal direction ,times the total number of pixels in the vertical direction, such as 512x512, 640x480, 1024x768.

## Raster Image

- Also known as bitmaps comprised of individual pixles of color. Each color pixel contributes to the overall image.
- Raster images might be compared to pointillist paintings, which are composed with a series of individually-colored dots of paint.
- Each paint dot in a pointillist painting might represent a single pixel in a raster image. When viewed as an individual dot, it’s just a color; but when viewed as a whole, the colored dots make up a vivid and detailed painting.
- Raster images are capable of rendering complex, multi-colored visuals, including soft color gradients. Digital cameras create raster images, and all the photographs you see in print and online are raster images.

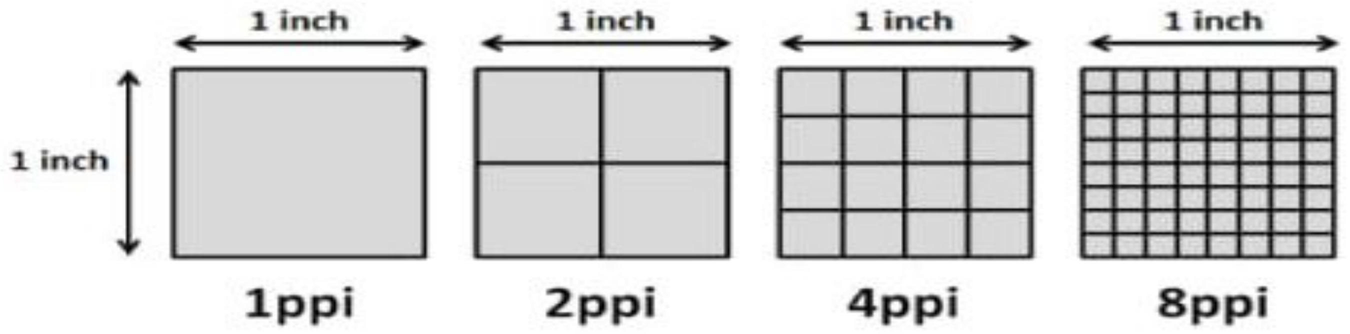
## Continue…

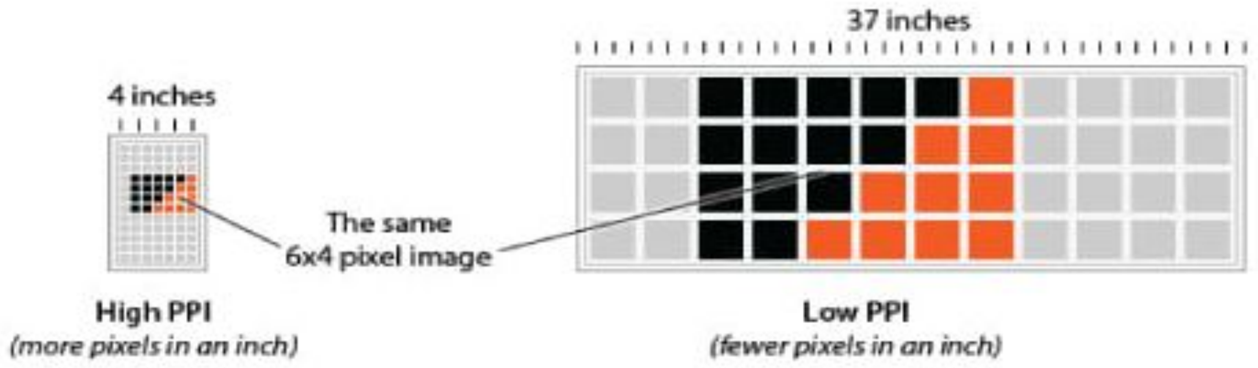
- There are different types of raster files: JPG, GIF, and PNG.
- Raster images are ideal for photo editing and creating digital paintings in programs such as Photoshop and GIMP , and they can be compressed for storage and web optimized images.



## Continue…

- Quality is often dictated by how many pixels are contained in an inch, expressed as pixels-per-inch or ppi.





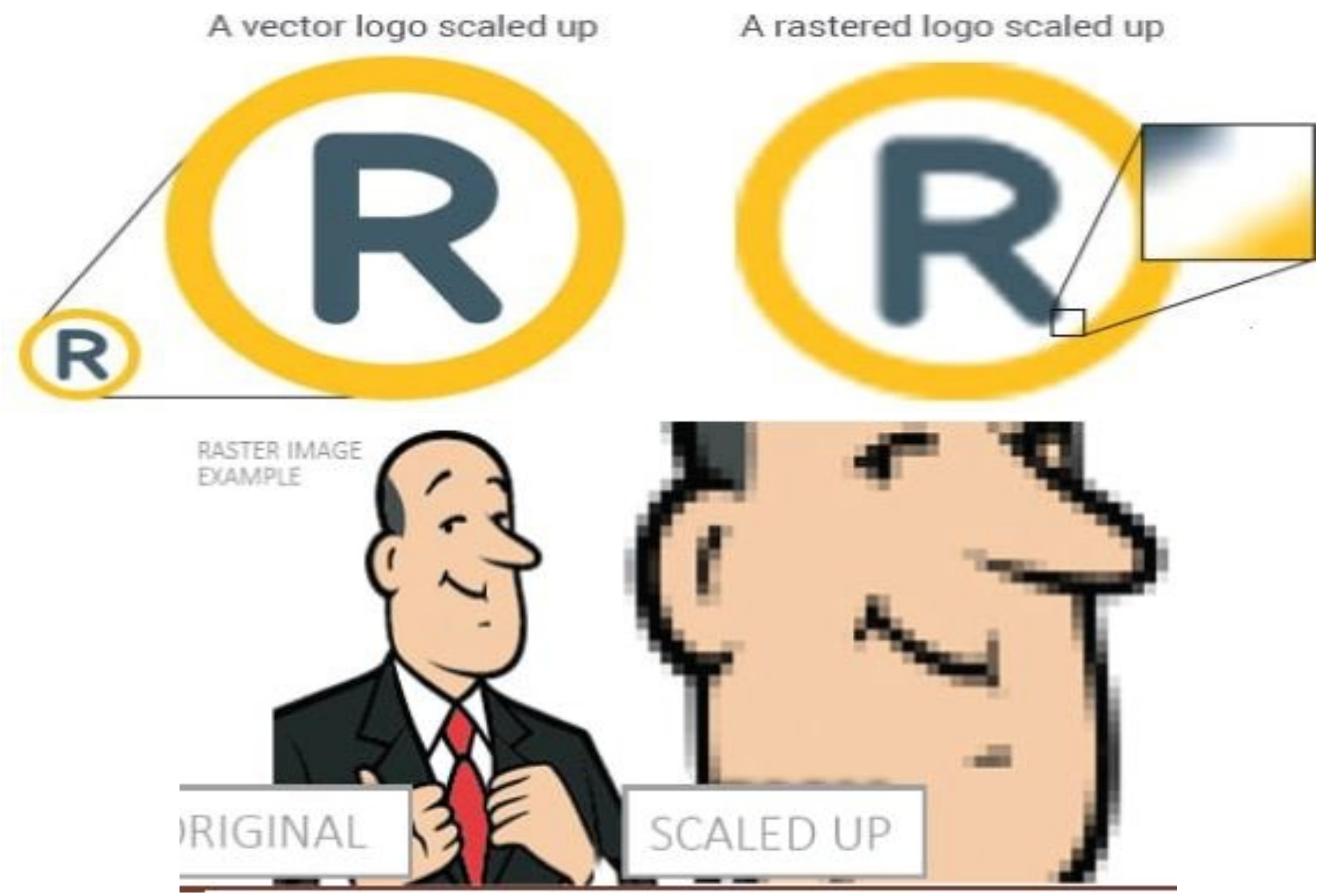
## Continue…

- The greater the ppi and dimensional measurements, the higher the quality.
- Let’s say you’re printing a brochure, and you need a background image to span the entire unfolded interior: 8.5”x11”.
- Your background image should be at least ______ pixels wide (8.5 inches) by ______ pixels tall (11 inches) with a resolution of 300ppi.

Solution: 2,550 and 3,300

- Anything less, and quality will be sacrificed, as raster images cannot be scaled to larger sizes. When they are scaled, quality is lost and they become blurry, as each pixel becomes larger.

## Continue…

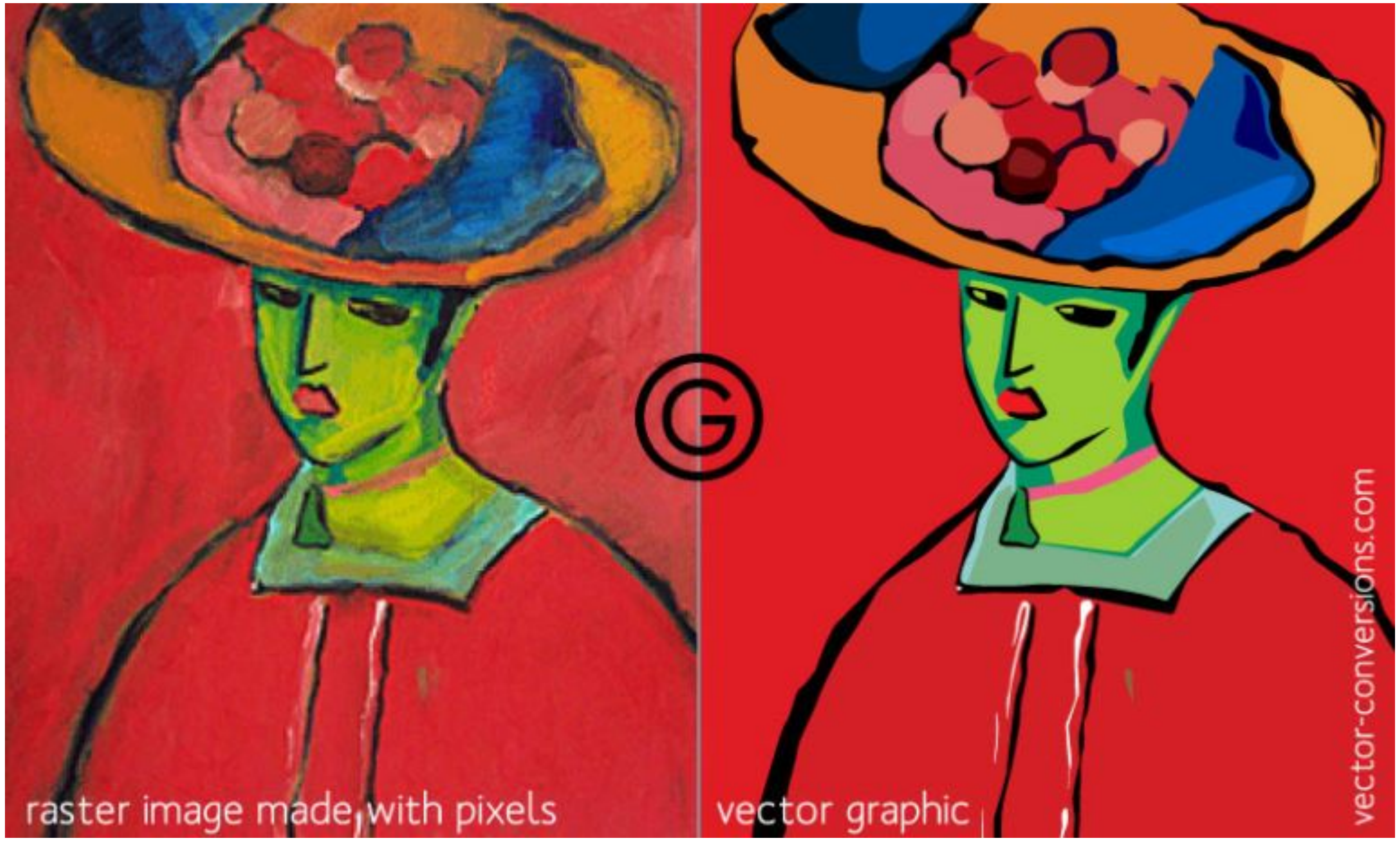


- Though raster images can’t be scaled up, they can be scaled down.

## Vector Image

- vector graphics are made up of paths, each with a mathematical formula (vector) that tells the path how it is shaped and what color it is bordered with or filled by.
- Since mathematical formulas dictate how the image is rendered, vector images retain their appearance regardless of size.
- They can be scaled infinitely.
- True vector graphics are comprised of line art, sometimes called wireframes, that are filled with color.
- Because vectors can be infinitely scaled without loss of quality, they're excellent for logos, illustrations, engravings, etchings, product artwork, signage, and embroidery.
- Vectors should not be used for digital paintings or photo editing; however, they're perfect for projects such as printing stickers that do not include photos.

# Raster Image vs Vector Image



# Continue…

- **Raster Image:**
  - Pixel-based.
  - Raster programs best for editing photos and creating continuous tone images with soft color blends.
  - Do not scale up optimally - Image must be created/scanned at the desired usage size or larger.
  - Large dimensions & detailed images equal large file size.
  - It is more difficult to print raster images using a limited amount of spot colors.
  - Some processes cannot use raster formats.
  - Depending on the complexity of the image, conversion to vector may be time consuming.
  - Raster images are the most common image format, including: jpg, gif, png, tif, bmp, psd, eps and pdfs originating from raster programs.
  - Common raster programs: photo editing / paint programs such as Photoshop & Paint Shop, GIMP (free).

## Continue…

- **Vector Image:**
  - Shapes based on mathematical calculations.
  - Vector programs best for creating logos, drawings and illustrations, technical drawings. For images that will be applied to physical products.
  - Can be scaled to any size without losing quality.
  - Resolution-independent: Can be printed at any size/resolution.
  - A large dimension vector graphic maintains a small file size.
  - Number of colors can be easily increased or reduced to adjust printing budget.
  - Vector art can be used for many processes and easily rasterized to be used for all processes.
  - Can be easily converted to raster.
  - It is not the best format for continuous tone images with blends of color or to edit photographs.
  - Common vector graphic file format: ai, cdr, svg, and eps & pdfs originating from vector programs.
  - Common vector programs: drawing programs such as Illustrator, CorelDraw, Inkscape (free).

## DPI

- DPI - Dots per Inch.
- This is the amount of ink dots the printer will put on each pixel of your image.
- The DPI is set by the actual printer device and it is not something in the image for the graphic designer to manipulate.

## PPI

- PPI - Pixels per Inch.
- Digital raster images are measured in pixels, or picture elements.
- How many pixels per inch is determined by the device you create the digital image with: camera, scanner, or graphics software and can be modified with a photo/paint editing software.

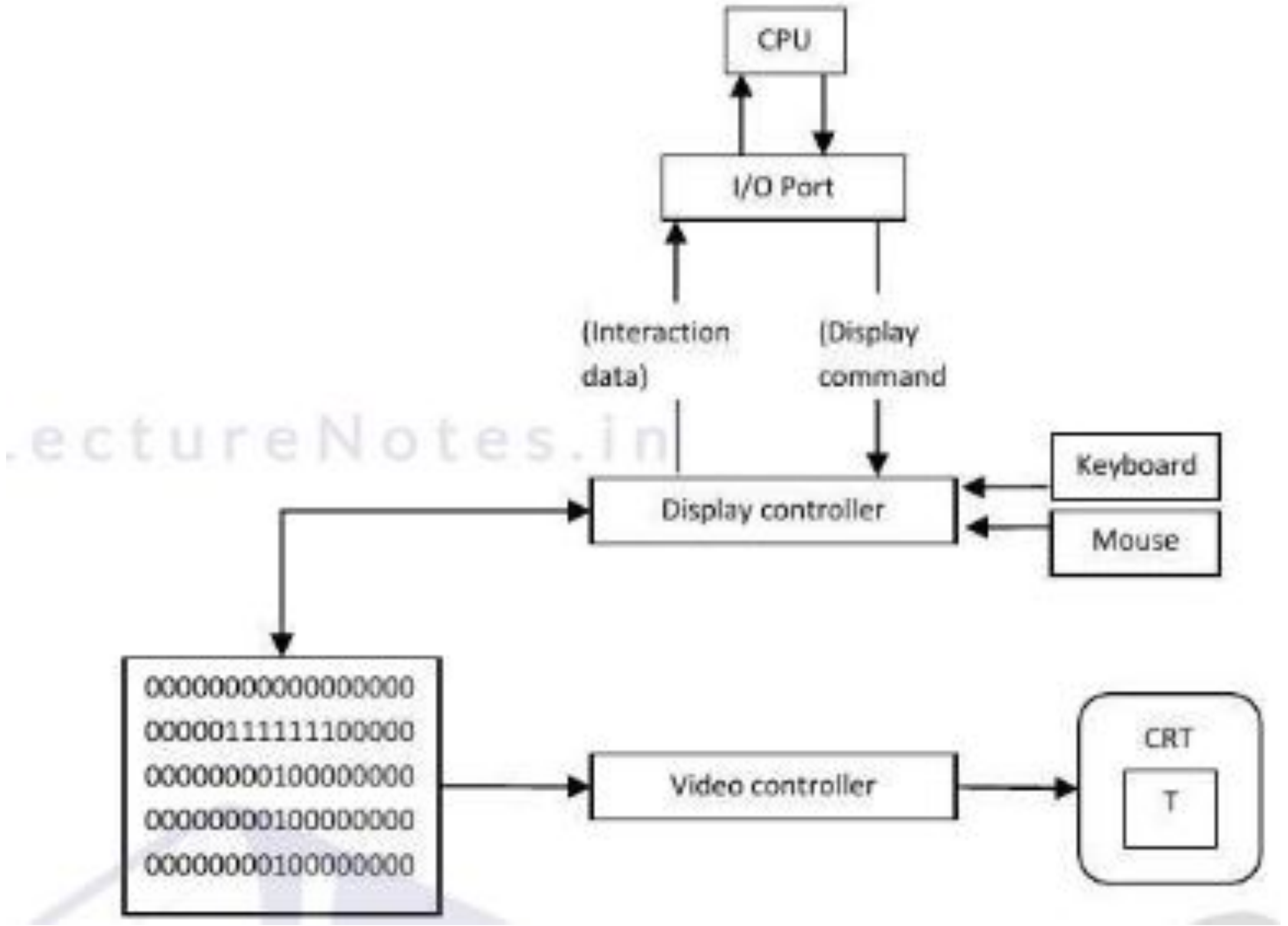
# Computer graphics vs Image processing

- Computer Graphics:
  - Computer graphics traditionally referred to the process of creating images from abstract models.
  - A computer game, for example, might internally keep track of Mario as a large list of points, where each point has three numbers representing its (x, y, z) coordinates.
  - Then, given the coordinates of the camera, and the direction its facing, the computer will calculate the color at each row and column in the final image of Mario that you see on your screen.
  - Computer Graphics: Convert Model to Image.

## Continue…

- Image Processing:
  - Image processing refers to the process of starting with an existing image and refining it in some way to obtain another image.
  - For example, if you take a picture with your camera, you would use an image processing algorithm to try and make the colors more vibrant, or remove the blur, or increase the resolution.
  - The output of an image processing algorithm is another image.
  - Image Processing: Refine Image to Image.

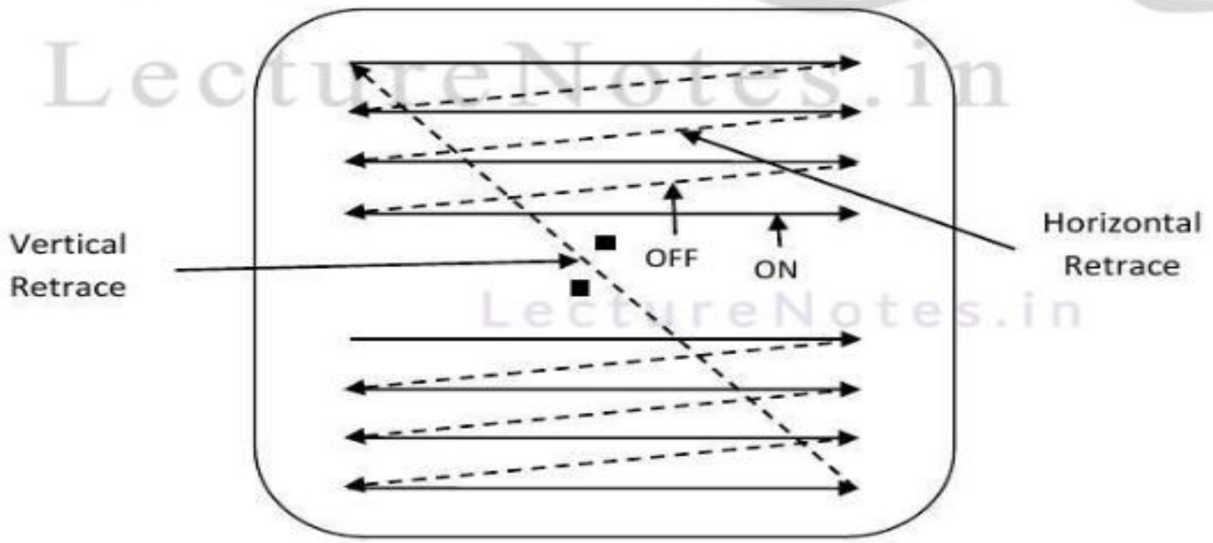
## Raster Scan Display



.ectureNotes.in

## Continue…

- Fig. 1.3 shows the architecture of Raster display. It consists of display controller, CPU, video controller, refresh buffer, keyboard, mouse and CRT.
- The display image is stored in the form of 1's and 0's in the refresh buffer.
- The video controller reads this refresh buffer and produces the actual image on screen.
- It will scan one line at a time from top to bottom & then back to the top.

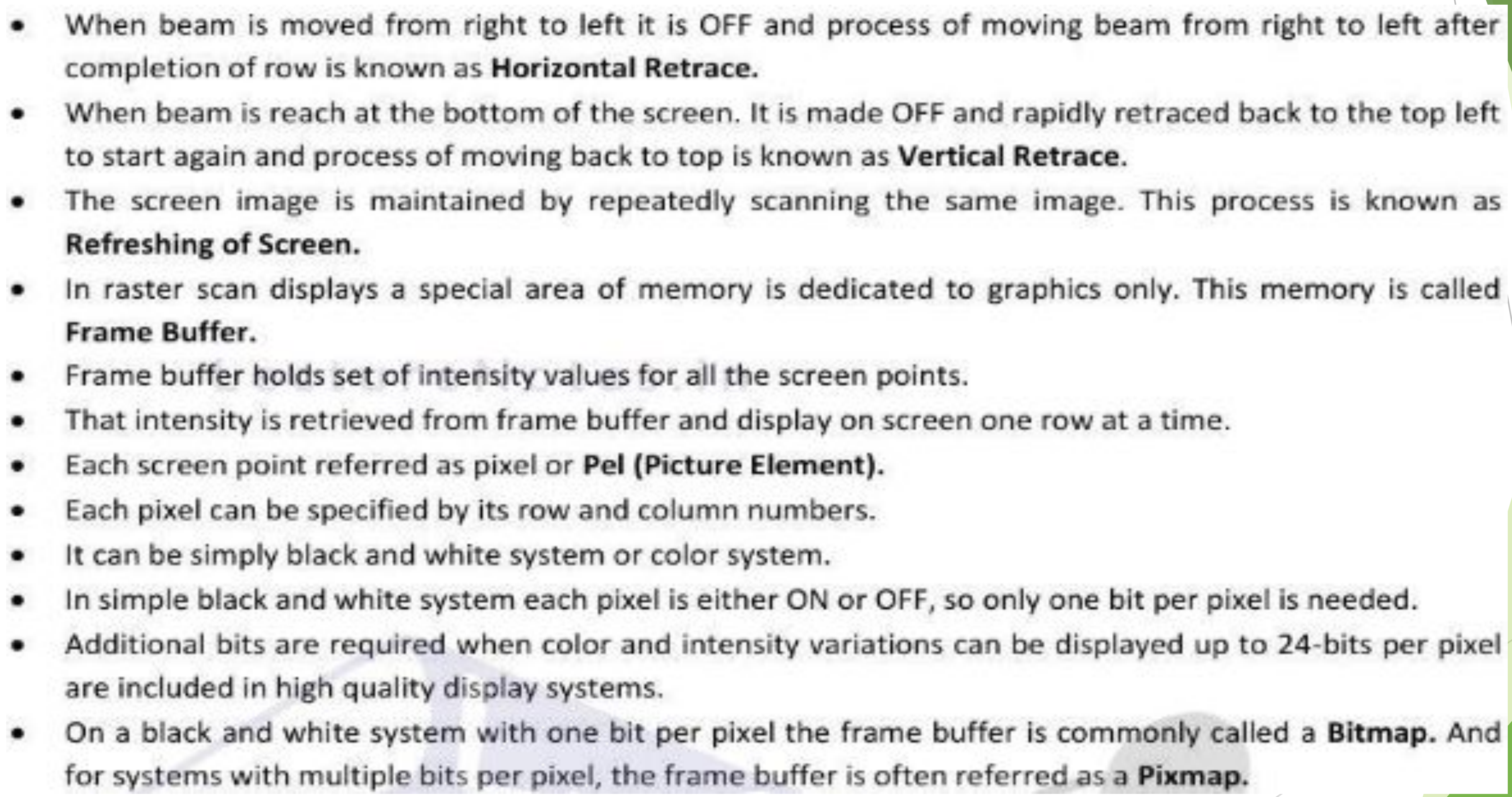


LectureNotes.in

Fig. 1.4: - Raster scan CRT.

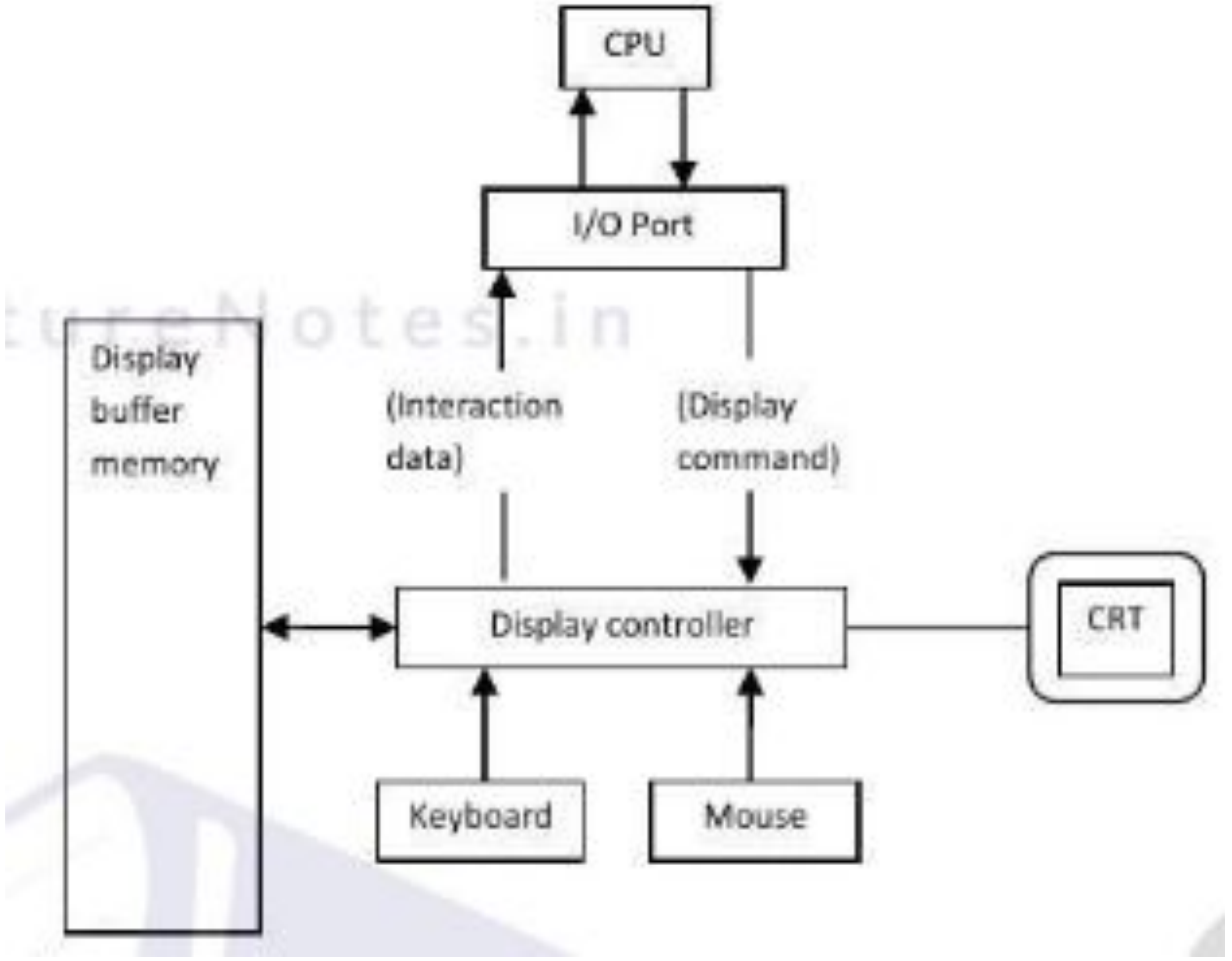
- In this method the horizontal and vertical deflection signals are generated to move the beam all over the screen in a pattern shown in fig. 1.4.
- Here beam is swept back & forth from left to the right.
- When beam is moved from left to right it is ON.

## Continue…

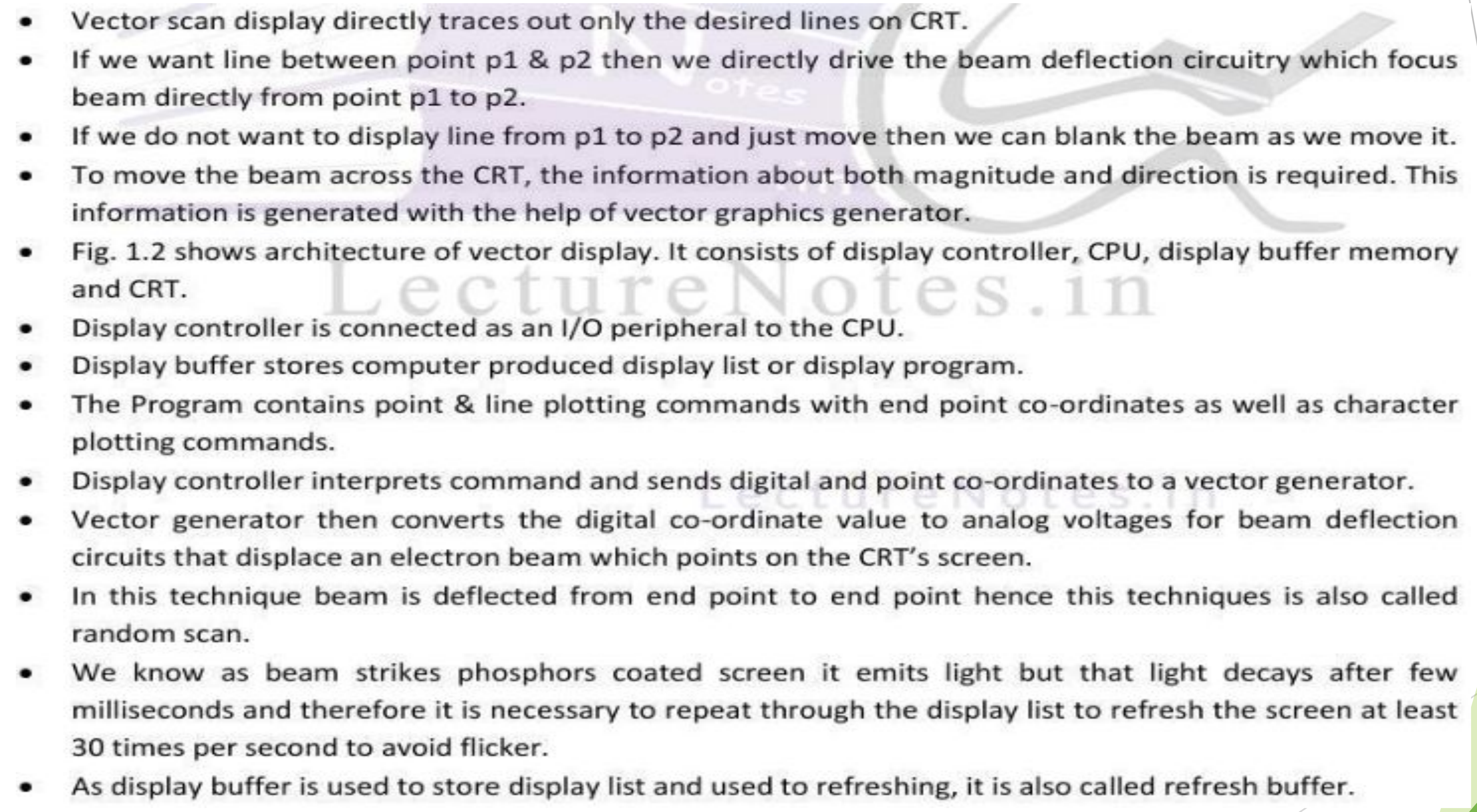


- When beam is moved from right to left it is OFF and process of moving beam from right to left after completion of row is known as **Horizontal Retrace.**
- When beam is reach at the bottom of the screen. It is made OFF and rapidly retraced back to the top left to start again and process of moving back to top is known as **Vertical Retrace.**
- The screen image is maintained by repeatedly scanning the same image. This process is known as **Refreshing of Screen.**
- In raster scan displays a special area of memory is dedicated to graphics only. This memory is called **Frame Buffer.**
- Frame buffer holds set of intensity values for all the screen points.
- That intensity is retrieved from frame buffer and display on screen one row at a time.
- Each screen point referred as pixel or **Pel (Picture Element).**
- Each pixel can be specified by its row and column numbers.
- It can be simply black and white system or color system.
- In simple black and white system each pixel is either ON or OFF, so only one bit per pixel is needed.
- Additional bits are required when color and intensity variations can be displayed up to 24-bits per pixel are included in high quality display systems.
- On a black and white system with one bit per pixel the frame buffer is commonly called a **Bitmap.** And for systems with multiple bits per pixel, the frame buffer is often referred as a **Pixmap.**

## Vector Scan Display



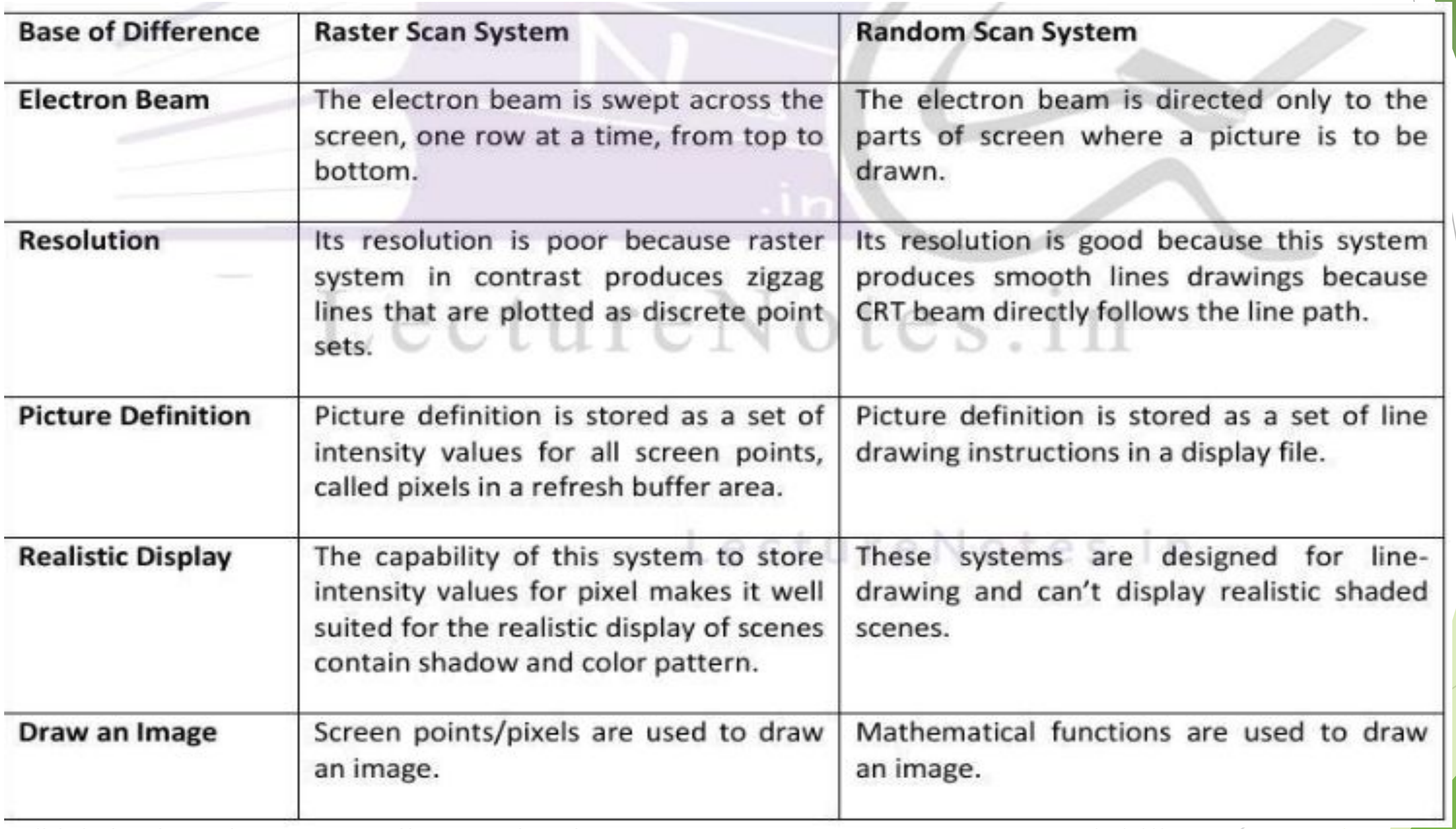
## Continue…



- Vector scan display directly traces out only the desired lines on CRT.
- If we want line between point p1 & p2 then we directly drive the beam deflection circuitry which focus beam directly from point p1 to p2.
- If we do not want to display line from p1 to p2 and just move then we can blank the beam as we move it.
- To move the beam across the CRT, the information about both magnitude and direction is required. This information is generated with the help of vector graphics generator.
- Fig. 1.2 shows architecture of vector display. It consists of display controller, CPU, display buffer memory and CRT.
- Display controller is connected as an I/O peripheral to the CPU.
- Display buffer stores computer produced display list or display program.
- The Program contains point & line plotting commands with end point co-ordinates as well as character plotting commands.
- Display controller interprets command and sends digital and point co-ordinates to a vector generator.
- Vector generator then converts the digital co-ordinate value to analog voltages for beam deflection circuits that displace an electron beam which points on the CRT's screen.
- In this technique beam is deflected from end point to end point hence this techniques is also called random scan.
- We know as beam strikes phosphors coated screen it emits light but that light decays after few milliseconds and therefore it is necessary to repeat through the display list to refresh the screen at least 30 times per second to avoid flicker.
- As display buffer is used to store display list and used to refreshing, it is also called refresh buffer.

LectureNotes.in

# Raster Scan vs Vector System



| Base of Difference | Raster Scan System | Random Scan System |
|---|---|---|
| Electron Beam | The electron beam is swept across the screen, one row at a time, from top to bottom. | The electron beam is directed only to the parts of screen where a picture is to be drawn. |
| Resolution | Its resolution is poor because raster system in contrast produces zigzag lines that are plotted as discrete point sets. | Its resolution is good because this system produces smooth lines drawings because CRT beam directly follows the line path. |
| Picture Definition | Picture definition is stored as a set of intensity values for all screen points, called pixels in a refresh buffer area. | Picture definition is stored as a set of line drawing instructions in a display file. |
| Realistic Display | The capability of this system to store intensity values for pixel makes it well suited for the realistic display of scenes contain shadow and color pattern. | These systems are designed for line-drawing and can't display realistic shaded scenes. |
| Draw an Image | Screen points/pixels are used to draw an image. | Mathematical functions are used to draw an image. |

LectureNotes.in

## CRT(Cathode Ray Tube)

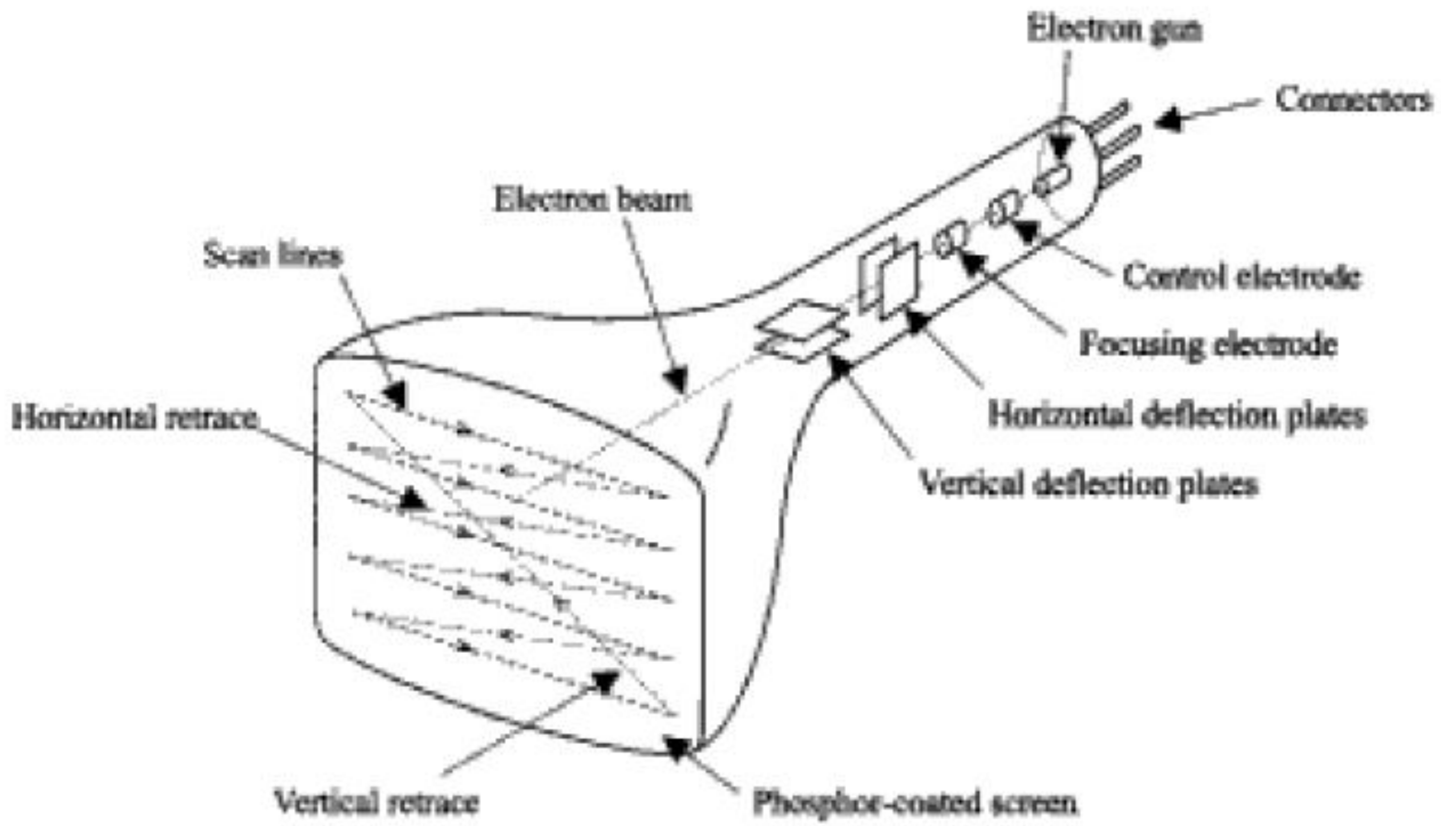
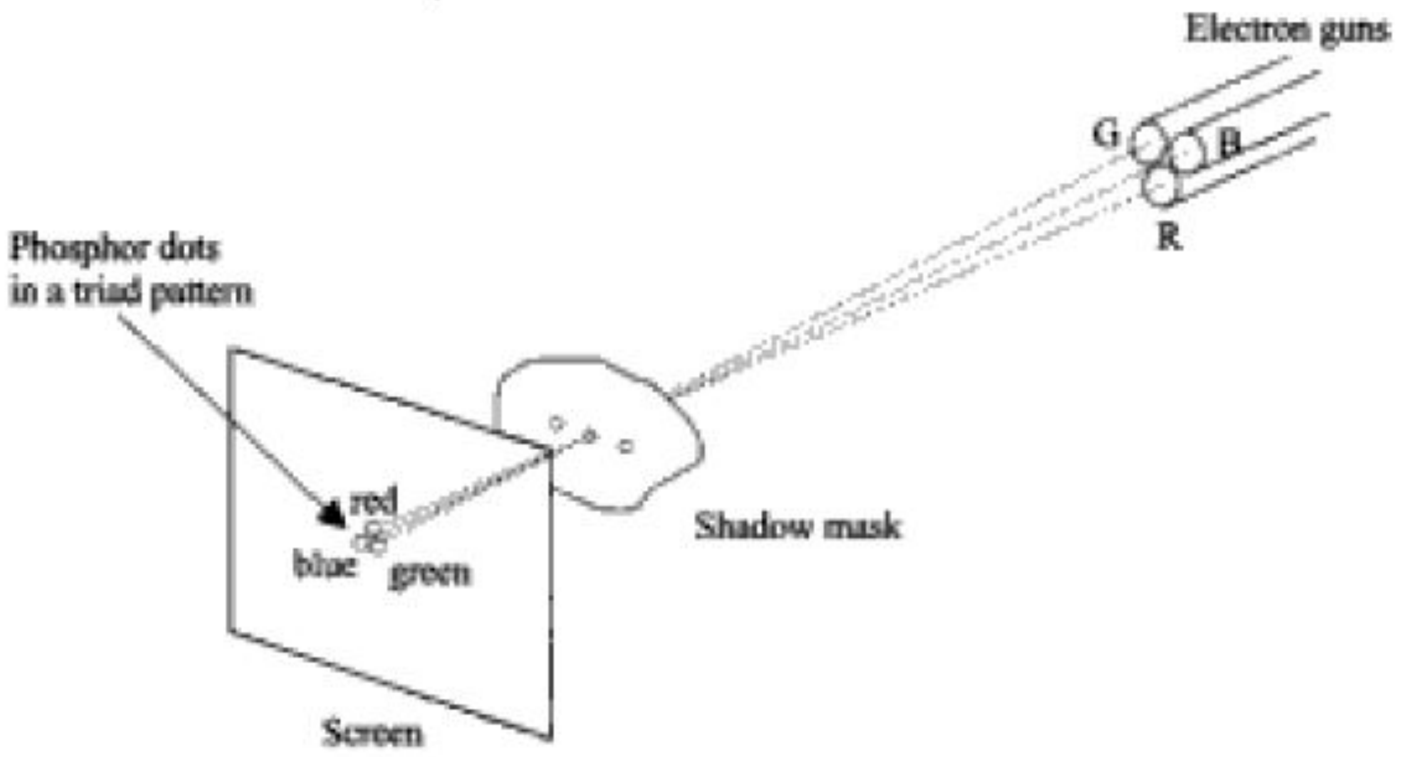


**Fig. 2-5** Anatomy of a monochromatic CRT.

# Continue…

- **Color Display**

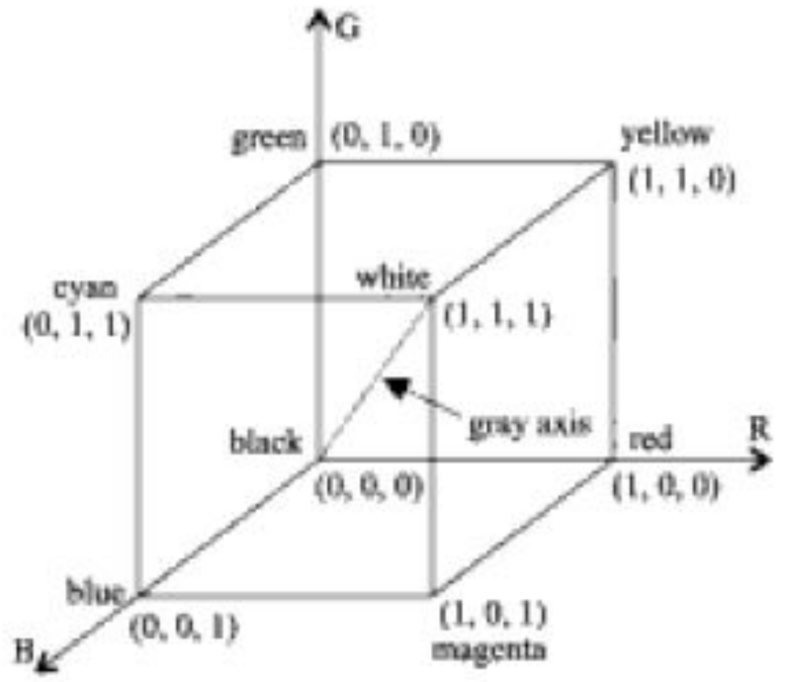


THANK YOU

# Lecture: 02

## RGB color model

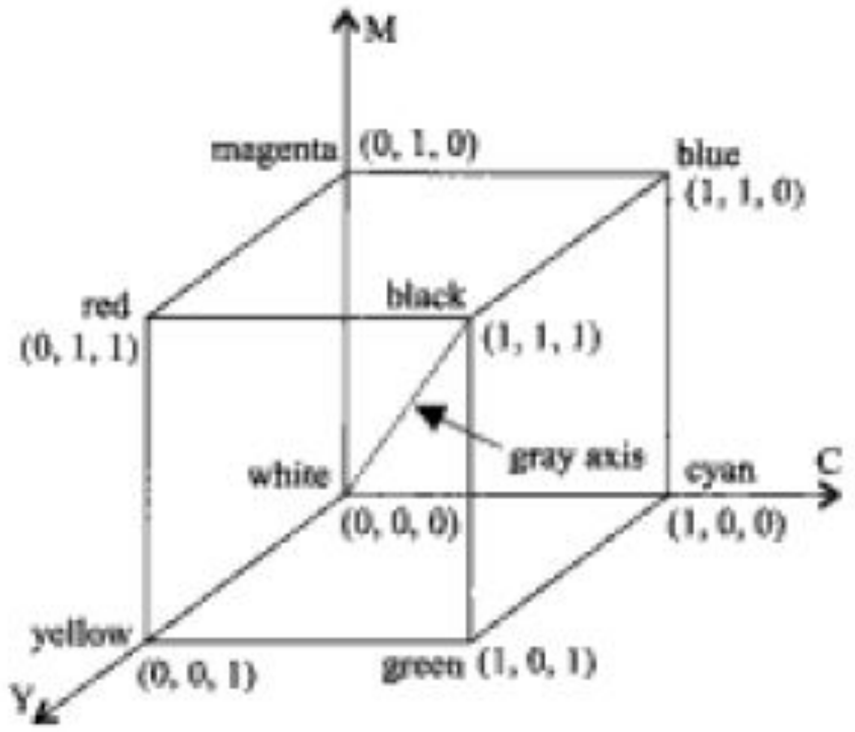
- A color co-ordinate system with 3 primary colors: R(Red), G(Green), B(Blue).
- Each color intensity range is from 0 to 1.
- Origin (0,0,0) indicates black color and starts from here.
- Additive process.



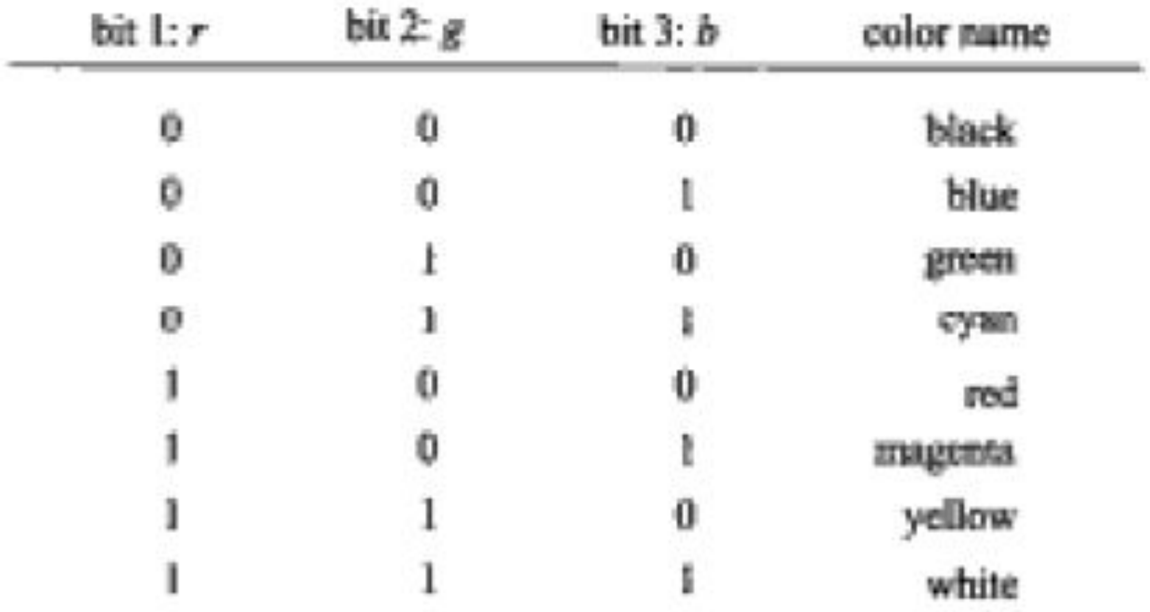
# CMY color model

- C: cyan ; M: magenta ; Y: yellow;
- Subtractive process.
- Begin with white(0,0,0).

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} C \\ M \\ Y \end{pmatrix} \qquad \begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$



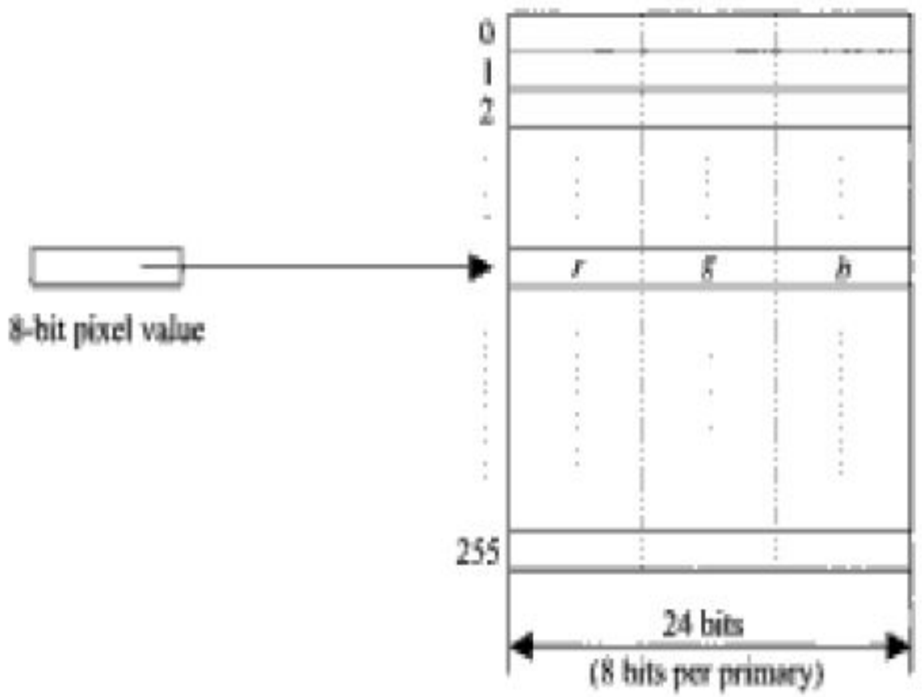
## Direct coding



| bit 1: *r* | bit 2: *g* | bit 3: *b* | color name |
|---|---|---|---|
| 0 | 0 | 0 | black |
| 0 | 0 | 1 | blue |
| 0 | 1 | 0 | green |
| 0 | 1 | 1 | cyan |
| 1 | 0 | 0 | red |
| 1 | 0 | 1 | magenta |
| 1 | 1 | 0 | yellow |
| 1 | 1 | 1 | white |

## Lookup Table

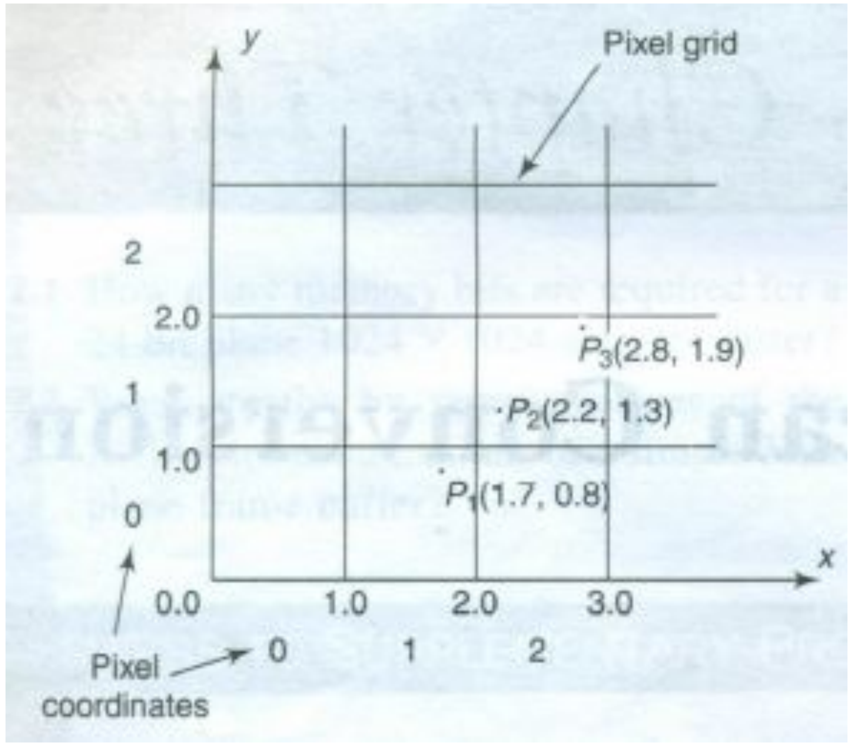
-  A compromise between our desire to have a lower storage requirement and our need to support a reasonably sufficient number of simultaneous colors.
-  256 entries, each entry has 24 bit RGB color value.
-  Pixel value is now 1byte or 8 bit.



THANK YOU

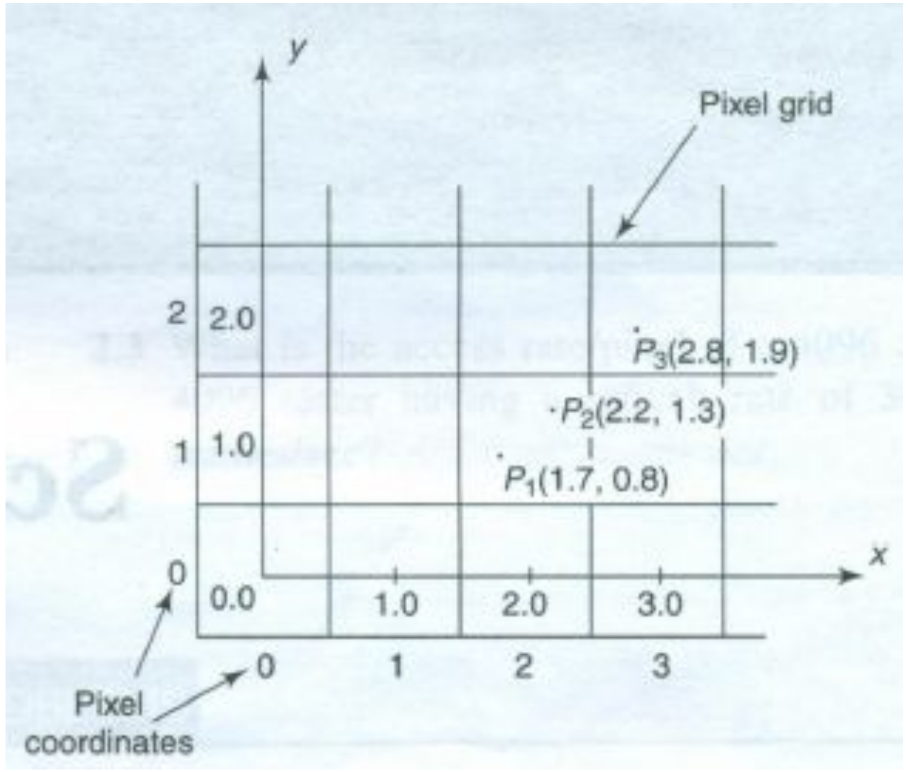
## Lecture : 03

# Scan Conversion of a Point



- A point ($x$, $y$) within an image area, scan converted to a pixel at location ($x'$, $y'$).
- $x'$ = Floor($x$) and $y'$ = Floor($y$).
- All points satisfying $x' \leq x < x' + 1$ and $y' \leq y < y' + 1$ are mapped to pixel ($x'$, $y'$).
- Point $P_1(1.7, 0.8)$ is represented by pixel (1, 0) and points $P_2(2.2, 1.3)$ and $P_3(2.8, 1.9)$ are both represented by pixel (2, 1).

## Scan Conversion of a Point…



- Another approach is to align the integer values in the co-ordinate system for (*x*, *y*) with the pixel co-ordinates.
- Here *x*' = Floor(*x* + 0.5) and *y*' = Floor(*y* + 0.5)
- Points $P_1$ and $P_2$ both are now represented by pixel (2, 1) and $P_3$ by pixel (3, 2).

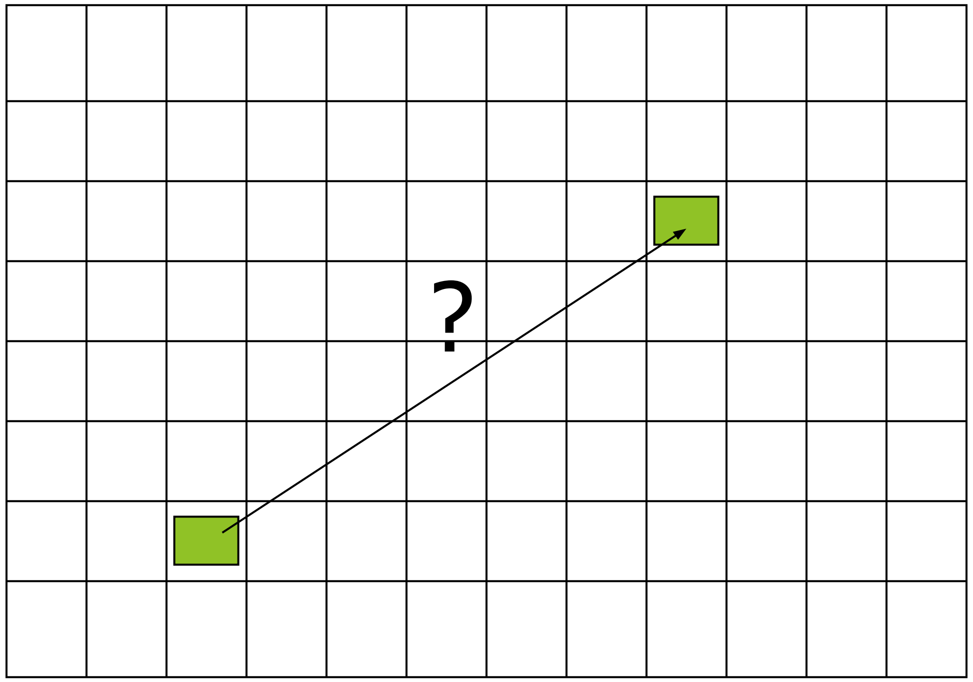
# Lecture: 04

## Line drawing algorithm

- Need algorithm to figure out which intermediate pixels are on line path
- Pixel ($x$, $y$) values constrained to integer values
- Actual computed intermediate line values may be floats
- Rounding may be required. Computed point

  (10.48, 20.51) rounded to (10, 21)

# Line Drawing Algorithm



Line: (3,2) -> (9,6)

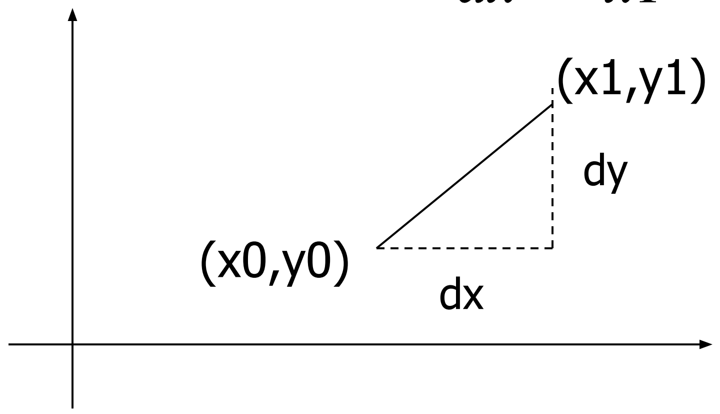
Which intermediate pixels to turn on?

## Line Drawing Algorithm…

- Slope-intercept line equation
  - y = mx + b
  - Given two end points (x0,y0), (x1, y1), how to compute m and b?

$$m = \frac{dy}{dx} = \frac{y1 - y0}{x1 - x0}$$

$$b = y0 - m * x0$$



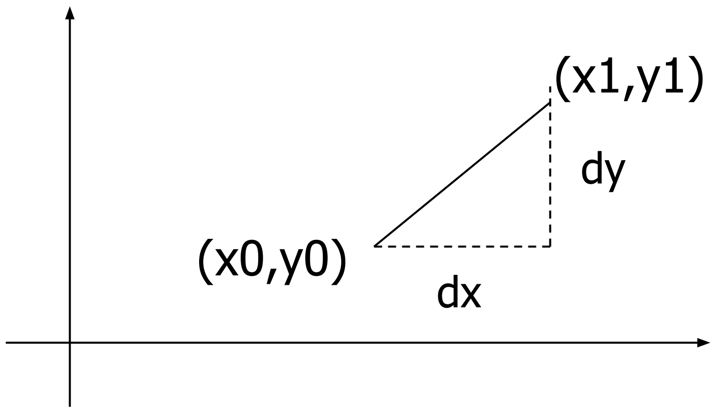
## Line Drawing Algorithm…

- Numerical example of finding slope m:
- (Ax, Ay) = (23, 41), (Bx, By) = (125, 96)

$$m = \frac{By - Ay}{Bx - Ax} = \frac{96 - 41}{125 - 23} = \frac{55}{102} = 0.5392$$

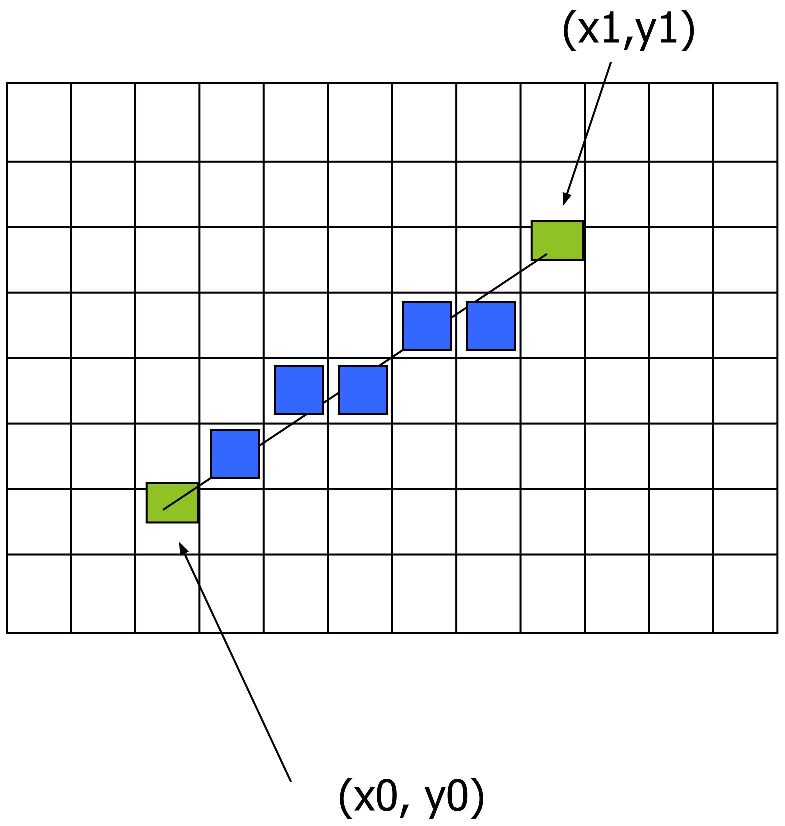
# Digital Differential Analyzer (DDA): Line Drawing Algorithm

- Walk through the line, starting at (x0,y0)
- Constrain x, y increments to values in [0,1] range
- Case a: x is incrementing faster (m < 1)
- Step in x=1 increments, compute and round y
- Case b: y is incrementing faster (m > 1)
- Step in y=1 increments, compute and round x



## DDA Line Drawing Algorithm(Case a:m<1)

$$y_{k+1} = y_k + m$$



x = x0 y = y0

Illuminate pixel (x, round(y))

x = x0 + 1 y = y0 + 1 * m

Illuminate pixel (x, round(y))

x = x + 1 y = y + 1 * m

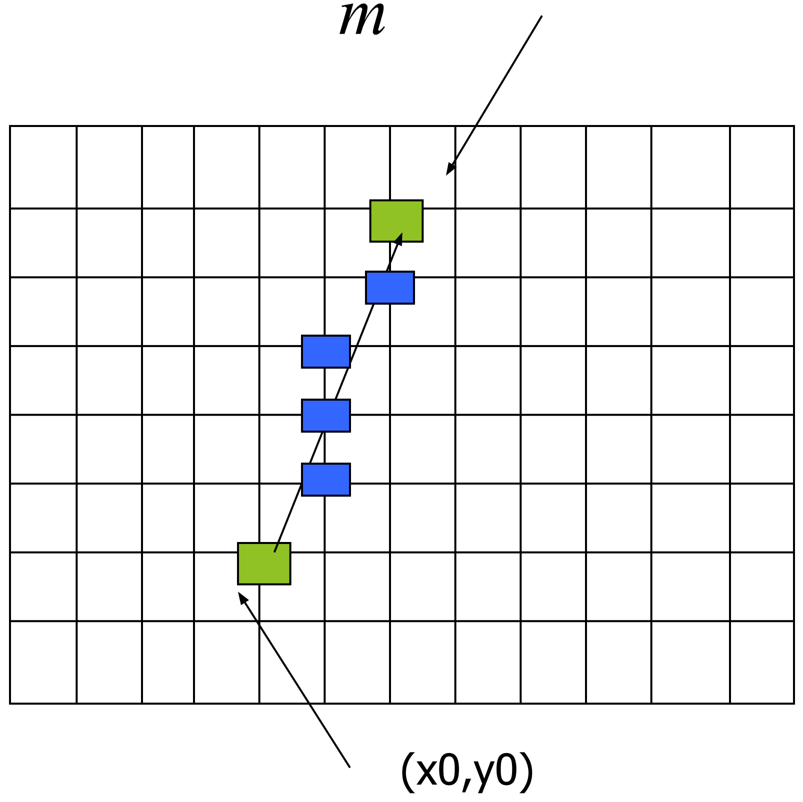
Illuminate pixel (x, round(y))

…

Until x == x1

# DDA Line Drawing Algorithm(Case b:m>1)

$$x_{k+1} = x_k + \frac{1}{m}$$



x = x0 y = y0

Illuminate pixel (round(x), y)

y = y0 + 1 x = x0 + 1 * 1/m

Illuminate pixel (round(x), y)

y = y + 1 x = x + 1 /m

Illuminate pixel (round(x), y)

…

Until y == y1

## DDA Line Drawing Algorithm Pseudocode

```
compute m;
if m < 1:
{
   float y = y0;        // initial value
   for(int x = x0;x <= x1; x++, y += m)
                setPixel(x, round(y));
}
else // m > 1
{
   float x = x0;        // initial value
   for(int y = y0;y <= y1; y++, x += 1/m)
                setPixel(round(x), y);
}
```

- Note: **`setPixel(x, y)`** writes current color into pixel in column x and row y in frame buffer

# DDA Example (Case a: m < 1)

- Suppose we want to draw a line starting at pixel (2,3) and ending at pixel (12,8).
- What are the values of the variables x and y at each timestep?
- What are the pixels colored, according to the DDA algorithm?

| t | x | y | R(x) | R(y) |
|---|---|---|---|---|
| 0 | 2 | 3 | 2 | 3 |
| 1 | 3 | 3.5 | 3 | 4 |
| 2 | 4 | 4 | 4 | 4 |
| 3 | 5 | 4.5 | 5 | 5 |
| 4 | 6 | 5 | 6 | 5 |
| 5 | 7 | 5.5 | 7 | 6 |
| 6 | 8 | 6 | 8 | 6 |
| 7 | 9 | 6.5 | 9 | 7 |
| 8 | 10 | 7 | 10 | 7 |
| 9 | 11 | 7.5 | 11 | 8 |
| 10 | 12 | 8 | 12 | 8 |

## DDA Algorithm Drawbacks

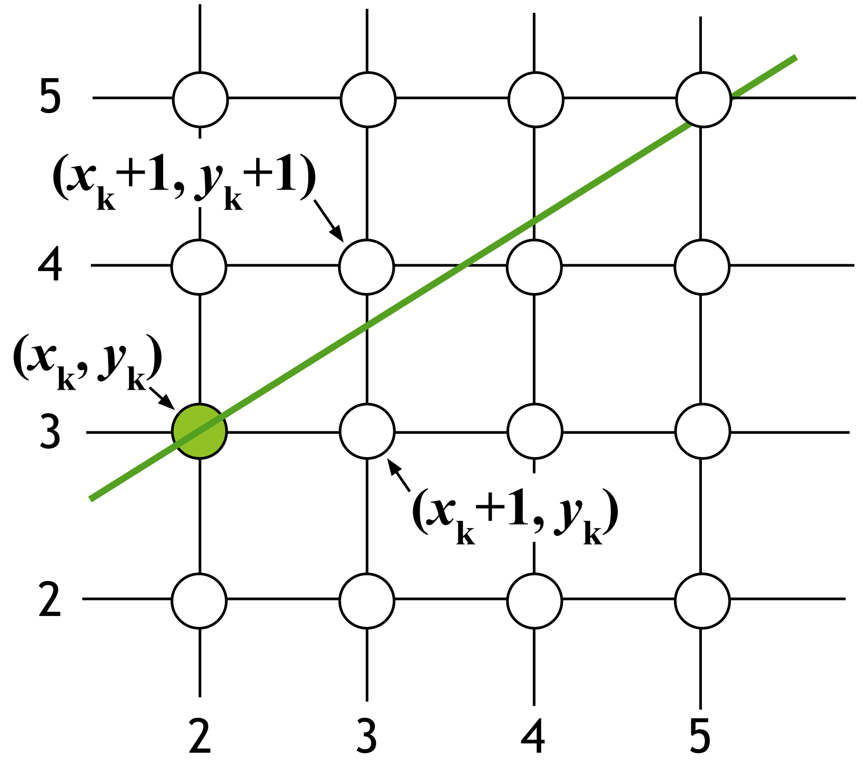
- DDA is the simplest line drawing algorithm
  - Not very efficient
  - Floating point operations and rounding operations are expensive.

## The Bresenham Line Algorithm

- The Bresenham algorithm is another incremental scan conversion algorithm
- The big advantage of this algorithm is that it uses only integer calculations: integer addition, subtraction and multiplication by 2, which can be accomplished by a simple arithmetic shift operation.

## The Big Idea

- Move across the *x* axis in unit intervals and at each step choose between two different *y* coordinates

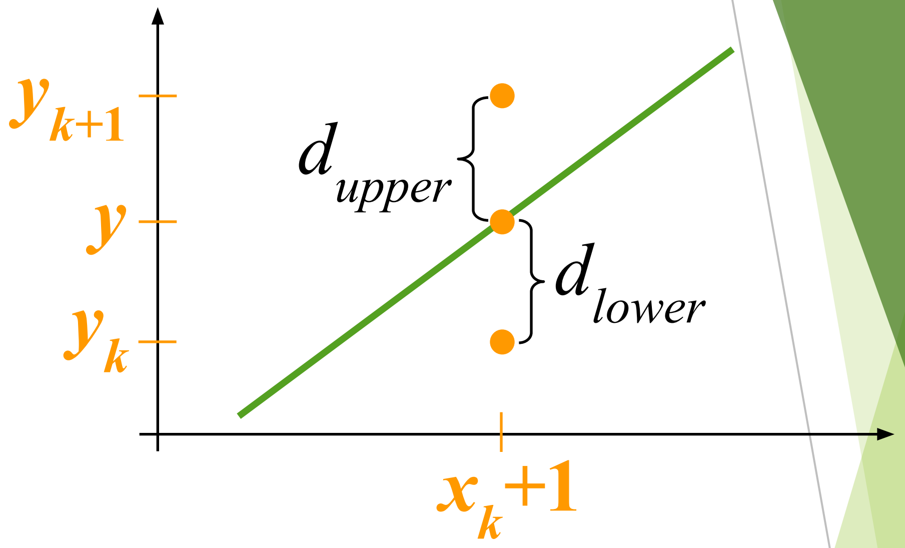


For example, from position (2, 3) we have to choose between (3, 3) and (3, 4)

We would like the point that is closer to the original line

# Deriving The Bresenham Line Algorithm

- At sample position $x_k+1$ the vertical separations from the mathematical line are labelled $d_{upper}$ and $d_{lower}$



The $y$ coordinate on the mathematical line at $x_k+1$ is:

$$y = m(x_k + 1) + b$$

## Deriving The Bresenham Line Algorithm…

- So, $d_{upper}$ and $d_{lower}$ are given as follows:

$$d_{lower} = y - y_k$$
$$= m(x_k + 1) + b - y_k$$

- and:

$$d_{upper} = (y_k + 1) - y$$
$$= y_k + 1 - m(x_k + 1) - b$$

- We can use these to make a simple decision about which pixel is closer to the mathematical line

## Deriving The Bresenham Line Algorithm…

- This simple decision is based on the difference between the two pixel positions:

$$d_{lower} - d_{upper} = 2m(x_k + 1) - 2y_k + 2b - 1$$

- Let's substitute $m$ with $\Delta y / \Delta x$ where $\Delta x$ and $\Delta y$ are the differences between the end-points:

$$\Delta x(d_{lower} - d_{upper}) = \Delta x(2\frac{\Delta y}{\Delta x}(x_k + 1) - 2y_k + 2b - 1)$$

$$= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + 2\Delta y + \Delta x(2b - 1)$$

$$= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$$

## Deriving The Bresenham Line Algorithm…

- So, a decision parameter $p_k$ for the $k$th step along a line is given by:

$$\begin{aligned} p_k &= \Delta x(d_{lower} - d_{upper}) \\ &= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c \end{aligned}$$

- The sign of the decision parameter $p_k$ is the same as that of $d_{lower} - d_{upper}$
- If $p_k$ is negative, then we choose the lower pixel, otherwise we choose the upper pixel

## Deriving The Bresenham Line Algorithm…

- Remember coordinate changes occur along the $x$ axis in unit steps so we can do everything with integer calculations.
- At step $k$+1 the decision parameter is given as:

$$p_{k+1} = 2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + c$$

- Subtracting $p_k$ from this we get:

$$p_{k+1} - p_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$$

## Deriving The Bresenham Line Algorithm…

- But, $x_{k+1}$ is the same as $x_k+1$ so:

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

- where $y_{k+1}$ - $y_k$ is either 0 or 1 depending on the sign of $p_k$
- The first decision parameter p0 is evaluated at (x0, y0) is given as:

$$p_0 = 2\Delta y - \Delta x$$

## The Bresenham Line Algorithm…

BRESENHAM'S LINE DRAWING ALGORITHM
(for $|m| < 1.0$)

1. Input the two line end-points, storing the left end-point in $(x_0, y_0)$
2. Plot the point $(x_0, y_0)$
3. Calculate the constants $\Delta x$, $\Delta y$, $2\Delta y$, and $(2\Delta y - 2\Delta x)$ and get the first value for the decision parameter as:

$$p_0 = 2\Delta y - \Delta x$$

4. At each $x_k$ along the line, starting at $k = 0$, perform the following test. If $p_k < 0$, the next point to plot is $(x_k+1, y_k)$ and: $$p_{k+1} = p_k + 2\Delta y$$

## The Bresenham Line Algorithm…

Otherwise, the next point to plot is $(x_k+1, y_k+1)$ and:

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

5. Repeat step 4 $(\Delta x$ - 1) times

- The algorithm and derivation above assumes slopes are less than 1. for other slopes we need to adjust the algorithm slightly

## Bresenham’s Line Algorithm ( Example)

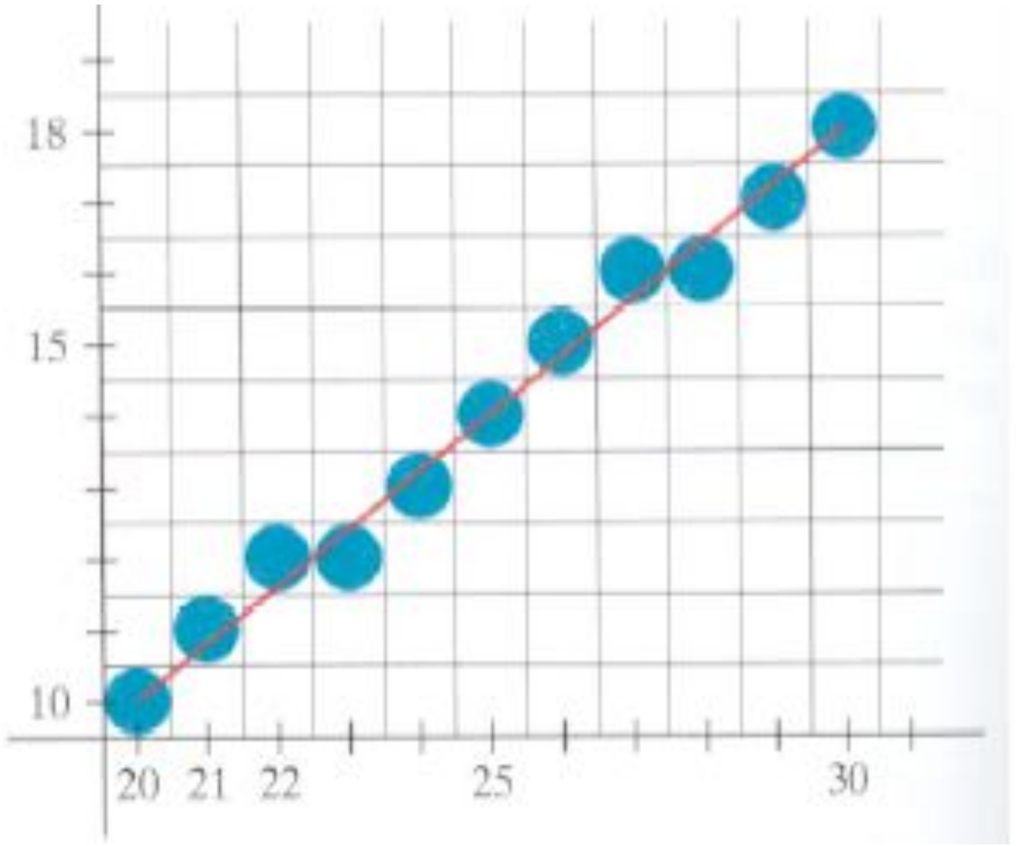
- using Bresenham’s Line-Drawing Algorithm, Digitize the line with endpoints (20,10) and (30,18).

- $\Delta y = 18 - 10 = 8$,
- $\Delta x = 30 - 20 = 10$
- $m = \Delta y / \Delta x = 0.8$
- $2*\Delta y = 16$
- $2*\Delta y - 2*\Delta x = -4$
- plot the first point (x0, y0) = (20, 10)
- $p0 = 2 * \Delta y - \Delta x = 2 * 8 - 10 = 6$ , so the next point is (21,11)

## Example (cont.)

| $K$ | $P_k$ | $(x_{k+1}, y_{k+1})$ | K | $P_k$ | $(x_{k+1}, y_{k+1})$ |
|---|---|---|---|---|---|
| 0 | 6 | (21,11) | 5 | 6 | (26,15) |
| 1 | 2 | (22,12) | 6 | 2 | (27,16) |
| 2 | -2 | (23,12) | 7 | -2 | (28,16) |
| 3 | 14 | (24,13) | 8 | 14 | (29,17) |
| 4 | 10 | (25,14) | 9 | 10 | (30,18) |

## Example (cont.)



## Bresenham's Line Algorithm (cont.)

- Notice that bresenham's algorithm works on lines with slope in range $0 < m < 1$.
- We draw from left to right.
- To draw lines with slope $> 1$, interchange the roles of x and y directions.

## Code (0 < slope < 1)

```
Bresenham ( int xA, yA, xB, yB) {
    int d, dx, dy, xi, yi;
    int incE, incNE;

    dx = xB – xA;
    dy = yB – yA;
    incE = dy << 1;              // Q
    incNE = incE – dx << 1;          // Q + R
    d = incE – dx;           // initial d = Q + R/2
    xi = xA; yi = yA;
    writePixel(xi, yi);
    while(xi < xB) {
         xi++;
         if(d < 0)               // choose E
       d += incE;
         else {         // choose NE
       d += incNE;
       yi++;
      }
          writePixel(xi, yi);
     }}
```

# Bresenham Line Algorithm Summary

- The Bresenham line algorithm has the following advantages:
  - An fast incremental algorithm
  - Uses only integer calculations
- Comparing this to the DDA algorithm, DDA has the following problems:
  - Accumulation of round-off errors can make the pixelated line drift away from what was intended
  - The rounding operations and floating point arithmetic involved are time consuming

# Lecture: 05

## A Simple Circle Drawing Algorithm

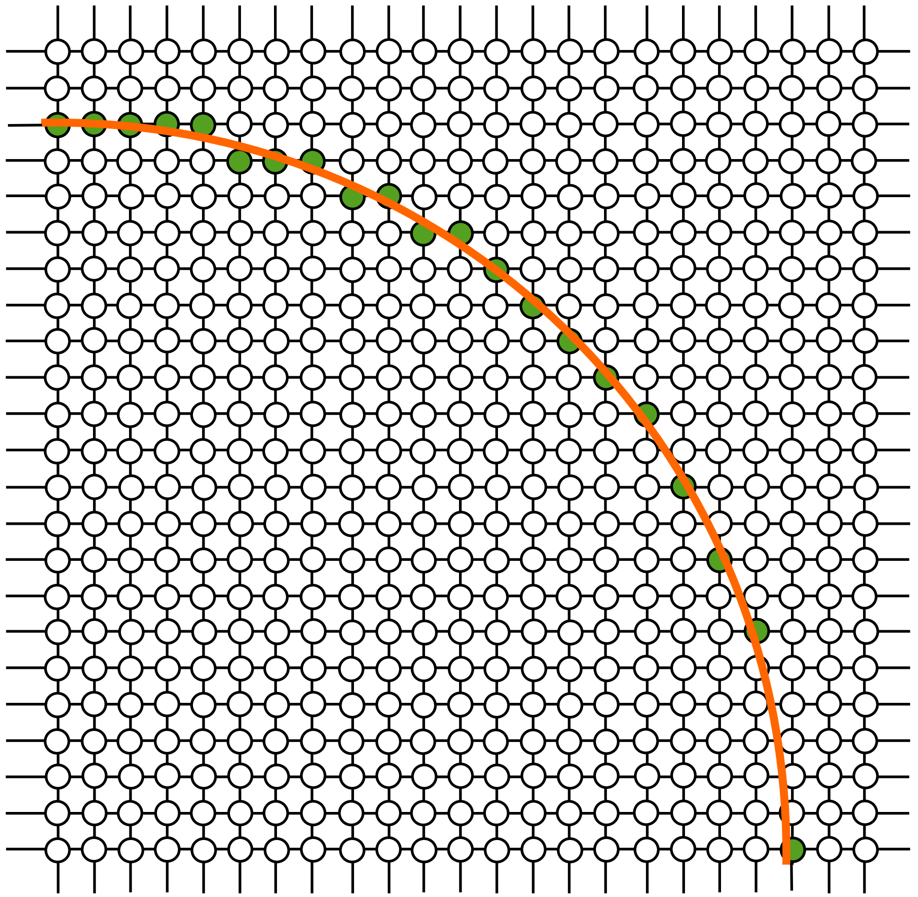
- The equation for a circle is:

$$x^2 + y^2 = r^2$$

- where $r$ is the radius of the circle
- So, we can write a simple circle drawing algorithm by solving the equation for $y$ at unit $x$ intervals using:

$$y = \pm\sqrt{r^2 - x^2}$$

## A Simple Circle Drawing Algorithm (cont…)



$$y_0 = \sqrt{20^2 - 0^2} \approx 20$$

$$y_1 = \sqrt{20^2 - 1^2} \approx 20$$

$$y_2 = \sqrt{20^2 - 2^2} \approx 20$$

$$\vdots$$

$$y_{19} = \sqrt{20^2 - 19^2} \approx 6$$

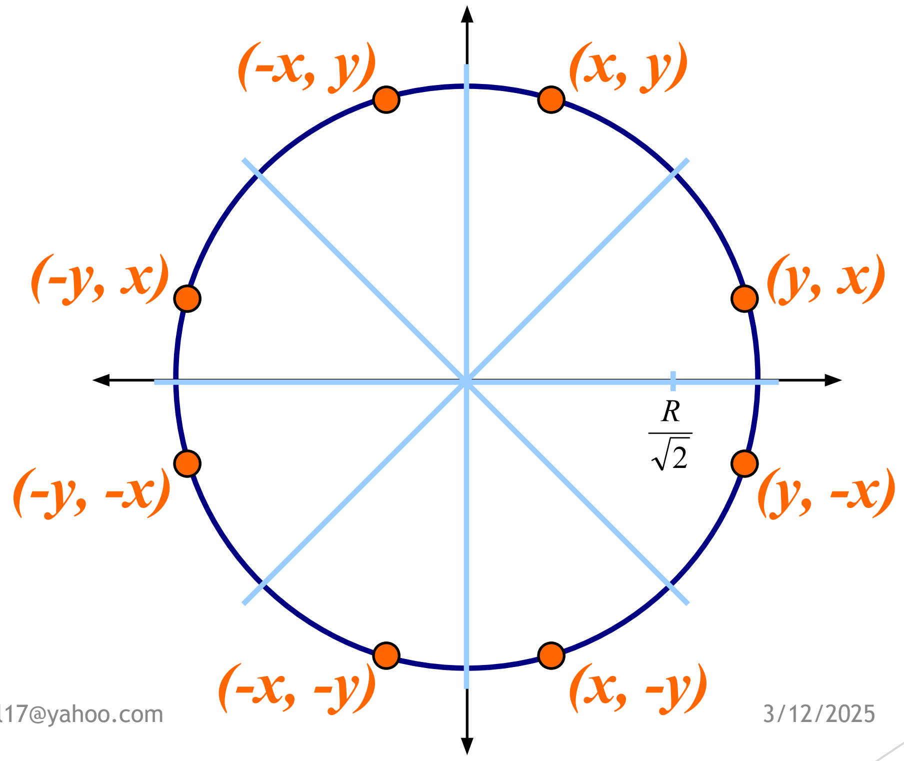
$$y_{20} = \sqrt{20^2 - 20^2} \approx 0$$

## A Simple Circle Drawing Algorithm (cont…)

- However, unsurprisingly this is not a brilliant solution!
- Firstly, the resulting circle has large gaps where the slope approaches the vertical
- Secondly, the calculations are not very efficient
  - The square (multiply) operations
  - The square root operation – try really hard to avoid these!
- We need a more efficient, more accurate solution

## Eight-Way Symmetry

- The first thing we can notice to make our circle drawing algorithm more efficient is that circles centred at $(0, 0)$ have *eight-way symmetry*

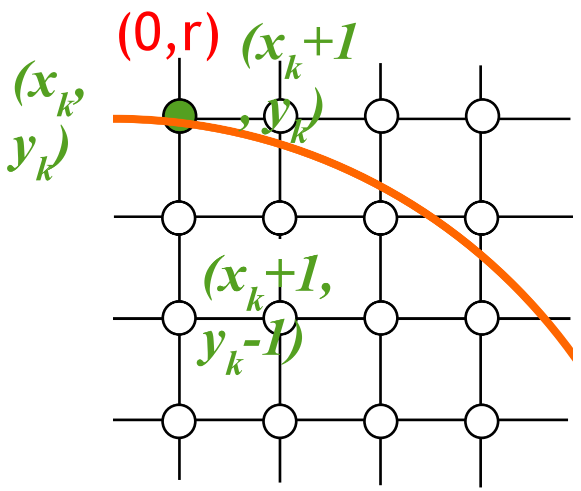


## Mid-Point Circle Algorithm

- Similarly to the case with lines, there is an incremental algorithm for drawing circles – the *mid-point circle algorithm*
- In the mid-point circle algorithm we use eight-way symmetry so only ever calculate the points for the top right eighth of a circle, and then use symmetry to get the rest of the points

## Mid-Point Circle Algorithm (cont…)

- Assume that we have just plotted point $(x_k, y_k)$
- The next point is a choice between $(x_k+1, y_k)$ and $(x_k+1, y_k-1)$
- We would like to choose the point that is nearest to the actual circle
- So how do we make this choice?



## Mid-Point Circle Algorithm (cont…)

- Let’s re-jig the equation of the circle slightly to give us:

The equation evaluates as follows: $f_{circ}(x, y) = x^2 + y^2 - r^2$

- By evaluating this function at the midpoint between the candidate pixels we can make our decision

## Mid-Point Circle Algorithm (cont…)

- Assuming we have just plotted the pixel at $(x_k, y_k)$ so we need to choose between $(x_k+1, y_k)$ and $(x_k+1, y_k-1)$
- Our decision variable can be defined as:

$$\begin{aligned} p_k &= f_{circ}(x_k + 1, y_k - \tfrac{1}{2}) \\ &= (x_k + 1)^2 + (y_k - \tfrac{1}{2})^2 - r^2 \end{aligned}$$

- If $p_k$ < 0 the midpoint is inside the circle and and the pixel at $y_k$ is closer to the circle

- Otherwise the midpoint is outside and $y_k-1$ is closer

# Mid-Point Circle Algorithm (cont…)

- To ensure things are as efficient as possible we can do all of our calculations incrementally
- First consider:

$$p_{k+1} = f_{circ}\left(x_{k+1}+1, y_{k+1}-\tfrac{1}{2}\right)$$

$$= [(x_k+1)+1]^2 + \left(y_{k+1}-\tfrac{1}{2}\right)^2 - r^2$$

- or:

$$p_{k+1} = p_k + 2(x_k+1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

- where $y_{k+1}$ is either $y_k$ or $y_k-1$ depending on the sign of $p_k$

## Mid-Point Circle Algorithm (cont…)

- The first decision variable is given as:

$$p_0 = f_{circ}(1, r - \tfrac{1}{2})$$

$$= 1 + (r - \tfrac{1}{2})^2 - r^2$$

$$= \tfrac{5}{4} - r$$

- Then if $p_k$ < 0 then the next decision variable is given as:

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

- If $p_k$ > 0 then the decision variable is:

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_k + 1$$

## Mid-point Circle Algorithm - Steps

1. Input radius **r** and circle center **($x_c$, $y_c$ )**. set the first point **($x_0$ , $y_0$ ) = (0, $r$ ).**

2. Calculate the initial value of the decision parameter as **$p_0$ = 1 - r.**

   **($p_0$ = 5 /4 - r ≅ 1 - r )**

3. If **$p_k < 0$,**

   plot **($x_k$ + 1, $y_k$ )** and **$p_{k+1} = p_k + 2x_{k+1} + 1$,**

   Otherwise,

   plot **($x_k$+ 1, $y_k$ - 1 )** and **$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$,**

   where **$2x_{k+1} = 2x_k + 2$** and **$2y_{k+1} = 2y_k - 2$.**

## Mid-point Circle Algorithm - Steps

4. Determine symmetry points on the other seven octants.

5. Move each calculated pixel position ($x$, $y$) onto the circular path centered on ($x_c$, $y_c$) and plot the coordinate values: $\boldsymbol{x = x + x_c}$ , $\boldsymbol{y = y + y_c}$

6. Repeat steps 3 though 5 until $\boldsymbol{x \geq y}$.

7. For all points, add the center point ($x_c$, $y_c$ )

## Mid-point Circle Algorithm - Steps

- Now we drew a part from circle, to draw a complete circle, we must plot the other points.
- We have $(x_c + x , y_c + y)$, the other points are:
  - $(x_c - x , y_c + y)$
  - $(x_c + x , y_c - y)$
  - $(x_c - x , y_c - y)$
  - $(x_c + y , y_c + x)$
  - $(x_c - y , y_c + x)$
  - $(x_c + y , y_c - x)$
  - $(x_c - y , y_c - x)$

# Mid-point circle algorithm (Example)

- Given a circle radius r = 10, demonstrate the midpoint circle algorithm by determining positions along the circle octant in the first quadrant from x = 0 to x = y.

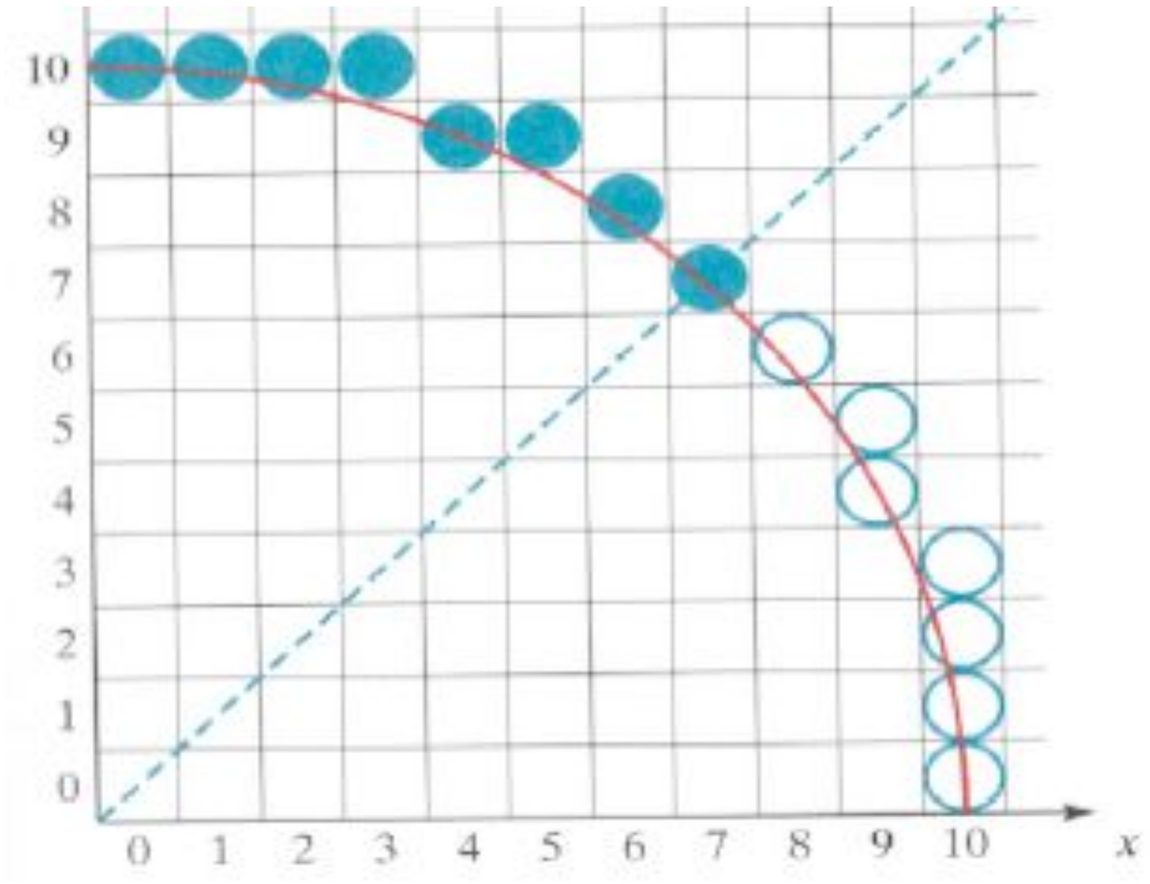
Solution:

- $p_0 = 1 - r = -9$
- Plot the initial point $(x_0, y_0) = (0, 10)$,
- $2x_0 = 0$ and $2y_0 = 20$.
- Successive decision parameter values and positions along the circle path are calculated using the midpoint method as appear in the next table:

## Mid-point circle algorithm (Example)

| $K$ | $P_k$ | $(x_{k+1}, y_{k+1})$ | $x_{k+1}$ 2 | $y_{k+1}$ 2 |
|---|---|---|---|---|
| 0 | 9 – | (10 ,1) | 2 | 20 |
| 1 | 6 – | (10 ,2) | 4 | 20 |
| 2 | 1 – | (10 ,3) | 6 | 20 |
| 3 | 6 | (9 ,4) | 8 | 18 |
| 4 | 3 – | (9 ,5) | 10 | 18 |
| 5 | 8 | (6,8) | 12 | 16 |
| 6 | 5 | (7,7) | 14 | 14 |

## Mid-point circle algorithm (Example)



# Mid-point Circle Algorithm – Example (2)

- Given a circle radius r = 15, demonstrate the midpoint circle algorithm by determining positions along the circle octant in the first quadrant from x = 0 to x = y.

<u>Solution</u>:

- $p_0 = 1 - r = -14$
- plot the initial point $(x_0 , y_0) = (0, 15)$,
- $2x_0 = 0$ and $2y_0 = 30$.
- Successive decision parameter values and positions along the circle path are calculated using the midpoint method as:

## Mid-point Circle Algorithm – Example (2)

| $K$ | $P_k$ | $(x_{k+1}, y_{k+1})$ | $2x_{k+1}$ | $2y_{k+1}$ |
|---|---|---|---|---|
| 0 | – 14 | (1, 15) | 2 | 30 |
| 1 | – 11 | (2, 15) | 4 | 30 |
| 2 | – 6 | (3, 15) | 6 | 30 |
| 3 | 1 | (4, 14) | 8 | 28 |
| 4 | – 18 | (5, 14) | 10 | 28 |

## Mid-point Circle Algorithm – Example (2)

| $K$ | $P_k$ | $(x_{k+1}, y_{k+1})$ | $x_{k+1}\ 2$ | $y_{k+1}\ 2$ |
|---|---|---|---|---|
| 5 | 7 – | (6,14) | 12 | 28 |
| 6 | 6 | (7,13) | 14 | 26 |
| 7 | 5 – | (8,13) | 16 | 26 |
| 8 | 12 | (9,12) | 18 | 24 |
| 9 | 7 | ( 10,11) | 20 | 22 |
| 10 | 6 | (11,10) | 22 | 20 |