



Computer Graphics

Deepankar Sharma

Introduction to Computer Graphics:

- Definition,
- Applications,
- Graphics Hardware,
- Display Devices:

§ Refresh Cathode Ray Tube,
§ Raster Scan Display,
§ Plasma display,
§ Liquid Crystal display,
§ Plotters,
§ Printers.

Definition

Computer graphics is the most effective and commonly used way to communicate the processed information to the user. It displays the information in the form of graphics objects such as pictures, charts, graphs & diagrams in place of simple text.

Definition

It is the use of computers to create and manipulate pictures on a display device. It comprises of software techniques to create, store, modify, represents pictures.

Branch of computer science concerned with methods & technology to convert data to & from visual information using computers

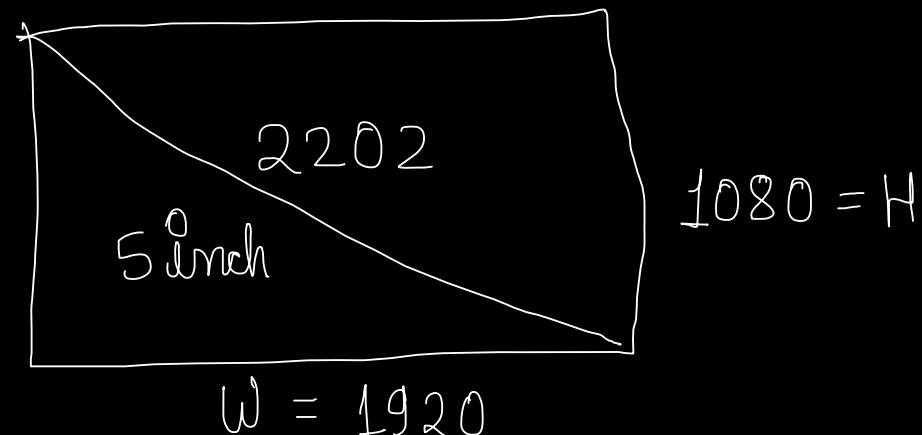
Applications

- ▶ User Interfaces
- ▶ Plotting Graphs and Charts
- ▶ Office Automation and Desktop Publishing
- ▶ Computer Aided Designing and Drafting
- ▶ Simulation and Animation
- ▶ Art and Commerce
- ▶ Process Control
- ▶ Cartography

Important Terms

- ▶ Pixel *any screen point is called pixel*
 - ▶ # pixels on entire screen
- ▶ Resolution $width \times height \Rightarrow 1920 \times 1080$
- ▶ PPI $\Rightarrow 2202/5 \Rightarrow 440 \text{ PPI}$
- ▶ Aspect Ratio $\Rightarrow 16:9$ ($width : height$)
- ▶ Frame Buffer (*yaha info store hoti h*)

$$\begin{array}{r} \cancel{1920} \ 48 \\ \cancel{1080} \ 16 \\ 270 \\ 9 \end{array}$$



Scan Conversion and Rasterization

Scan Conversion → process of determining appropriate pixels for representing a picture or graphics object.

Rasterization → process of representing continuous pictures or graphics object as a collection of discrete pixels.

Graphics Hardware

Input Devices

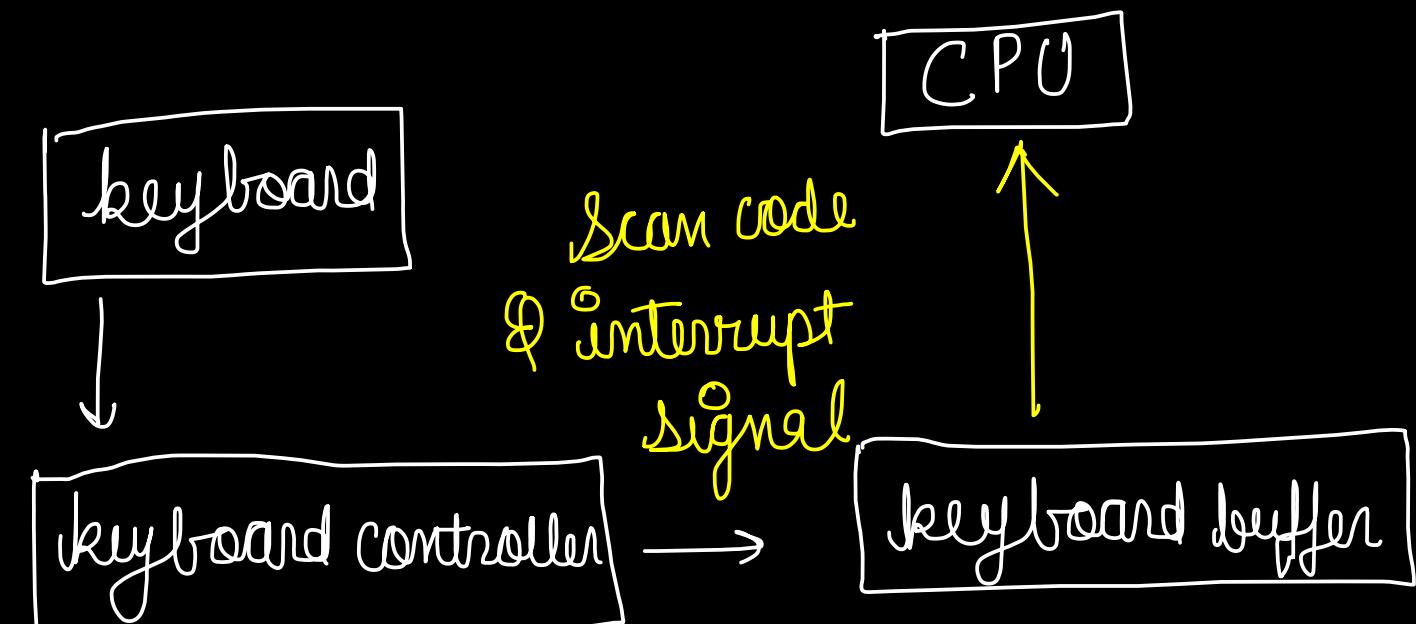
- ① Keyboard
- ② Mouse
- ③ Trackball/Spacball
- ④ Joystick

Output Devices

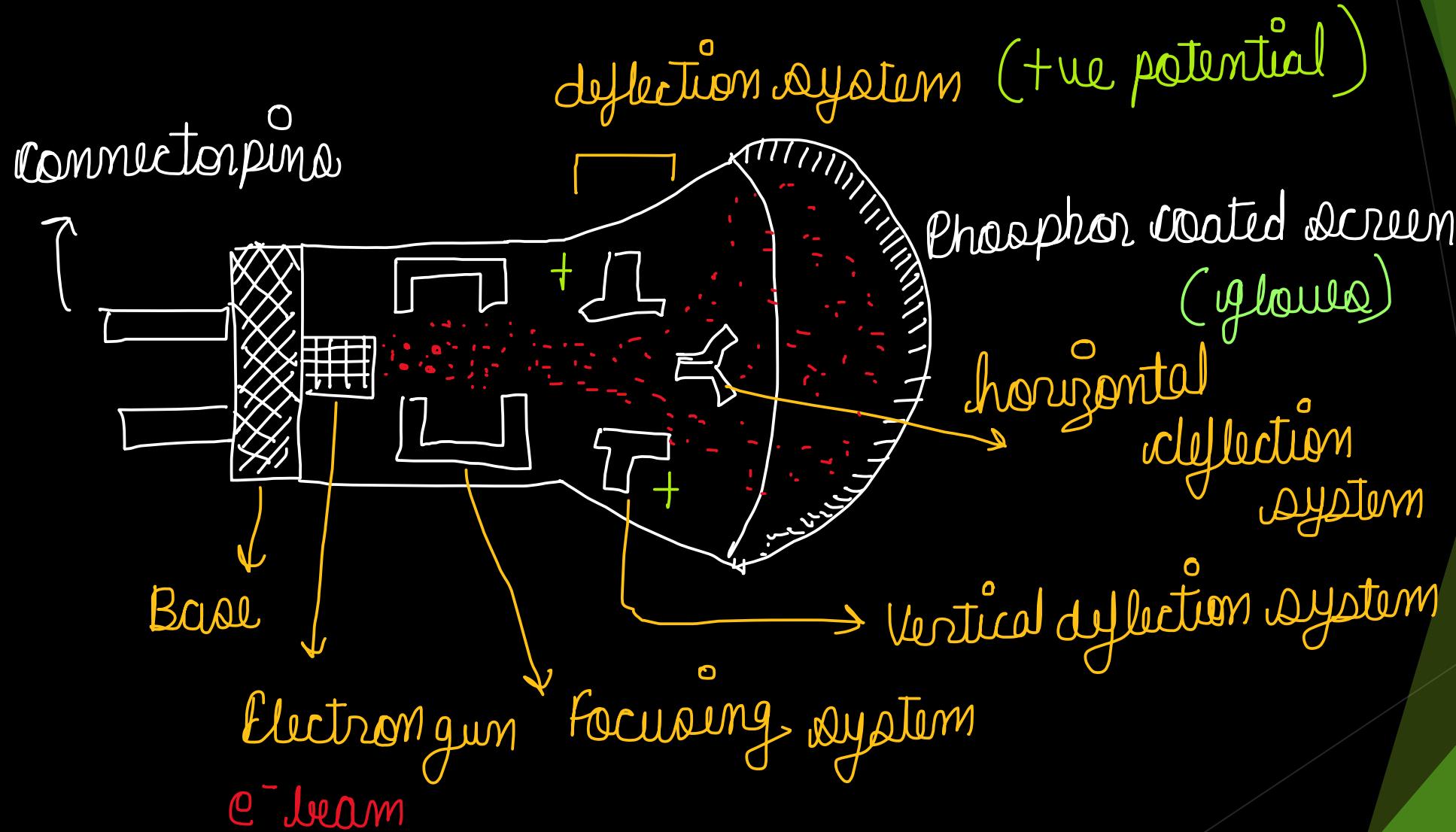
- ① CRT
- ② LED, LCD, TFT
- ③ Plasma
- ④ Colored Monitor

Graphics Software

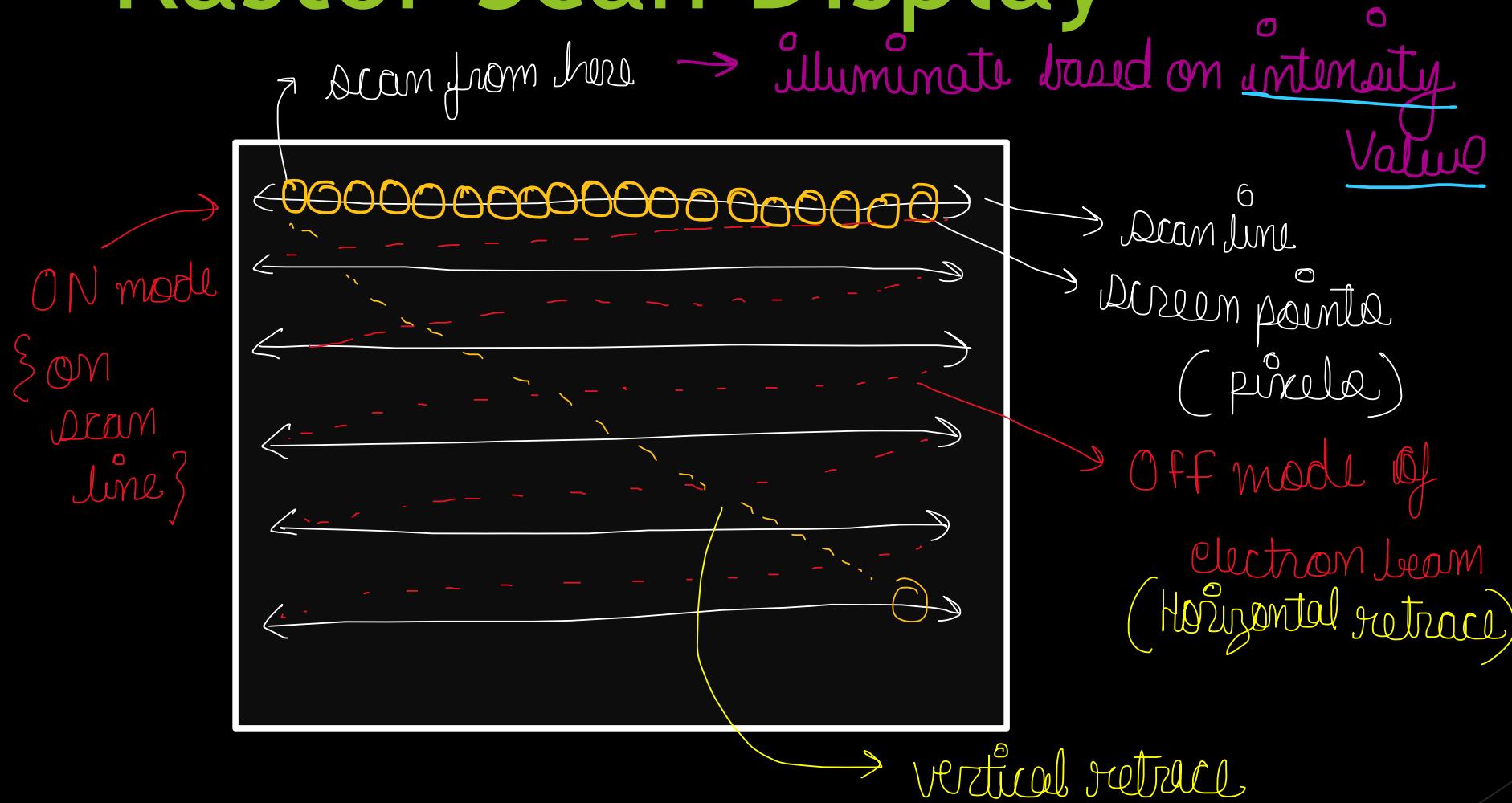
- ① Photoshop
- ② Maya 3D
- ③ Cad S/F
- ④ Corel Draw



Cathode Ray Tube(CRT)



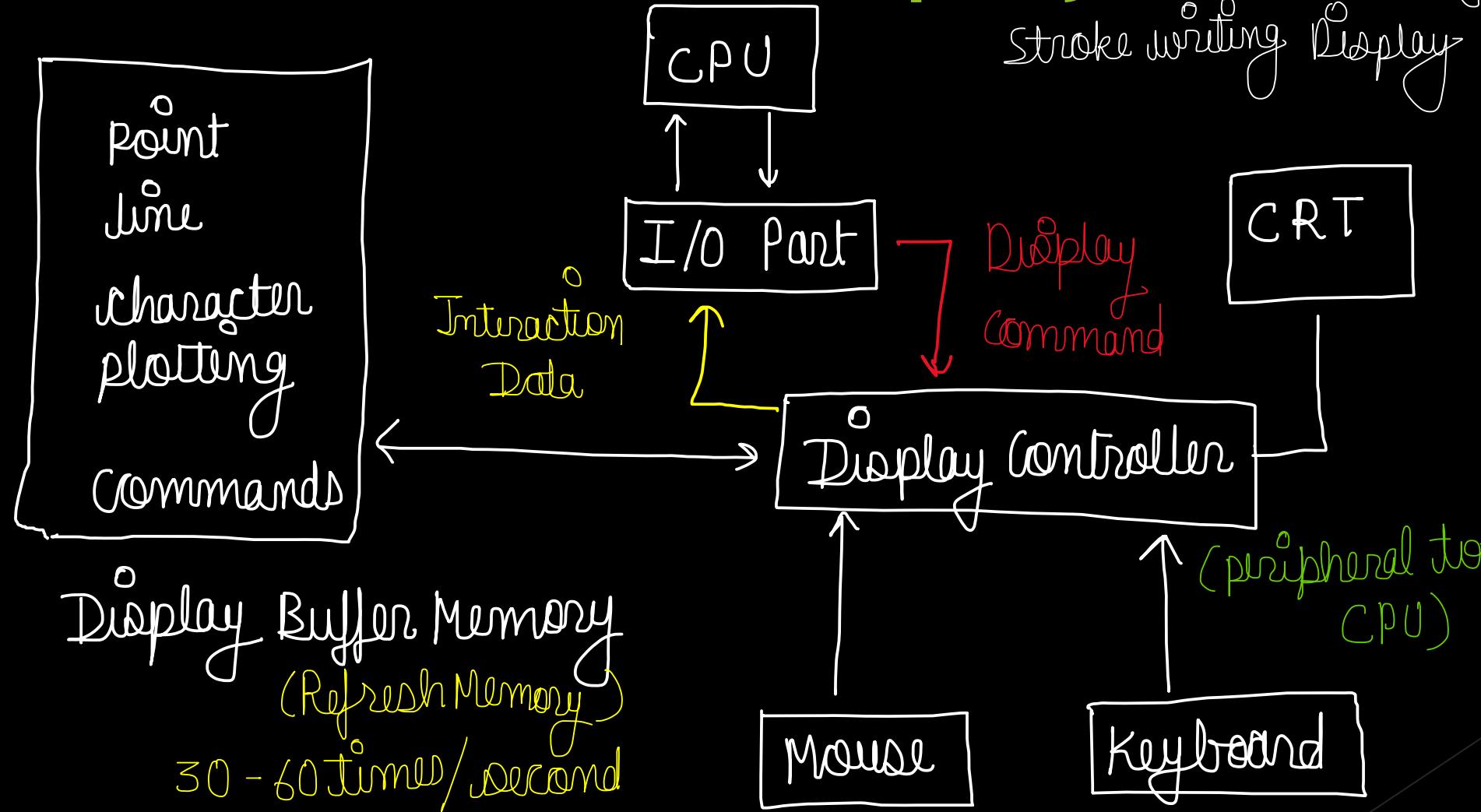
Raster Scan Display



Intensity value → refresh / frame Buffer

Vector Scan Display

Random Scan Display
Calligraphic Display
Stroke writing Display



Display Controller \Rightarrow directs electron beam via deflection system

Vector Scan Display

Random Scan Display

- ① high resolution
- ② more expensive
- ③ Modification is easy
- ④ Solid pattern is tough to fill.
- ⑤ Refresh rate depends on resolution
- ⑥ only screen with view is displayed
- ⑦ Beam Penetration Technology
- ⑧ doesn't use interlacing method
- ⑨ Restricted to line drawing applications

Raster Scan Display

- ① low resolution
- ② less expensive
- ③ Modification is tough
- ④ Solid pattern is easy to fill.
- ⑤ Refresh Rate depends on the picture
- ⑥ Whole screen is Scanned
- ⑦ Shadow Mask technology
- ⑧ Uses interlacing
- ⑨ suitable for realistic display

Properties of phosphor

→ color

→ persistence → kitni der?

method of incrementally displaying a visual on a CRT.

On some Raster Scan Systems, each frame is displayed in two passes using an interlaced refresh procedure.

In the first pass, the beam steps across every scan line from top to bottom. Then after vertical retrace, the beam sweeps out the remaining scan lines.

→ provides only four colour { Red, Green, Orange, Yellow }

Beam Penetration (Random Scan Display)

2 layers of phosphor (Red and green) are coated onto the inside of CRT screen. The displayed colors depend on how far the e^- beam penetrates the phosphor layers.

fast e^- beam → excites red average e^- beam → excites combination of Red & green

slow e^- beam → penetrates red and excites green { yellow & orange }

Shadow Masking (Raster Scan Display)

→ gives more color than beam penetration

→ 1 pixel → 3 color dots (RGB) → 17 billion different colors in full color system.

→ This type of CRT have 3 e^- guns for each color dot

→ A shadow mask grid is installed behind phosphor coated screen

Flat Panel Display

(volume, weight, power requirement) < CRT
→ TV, Monitor, Calculator, Laptop, etc.

2 categories:

① Emissive Display

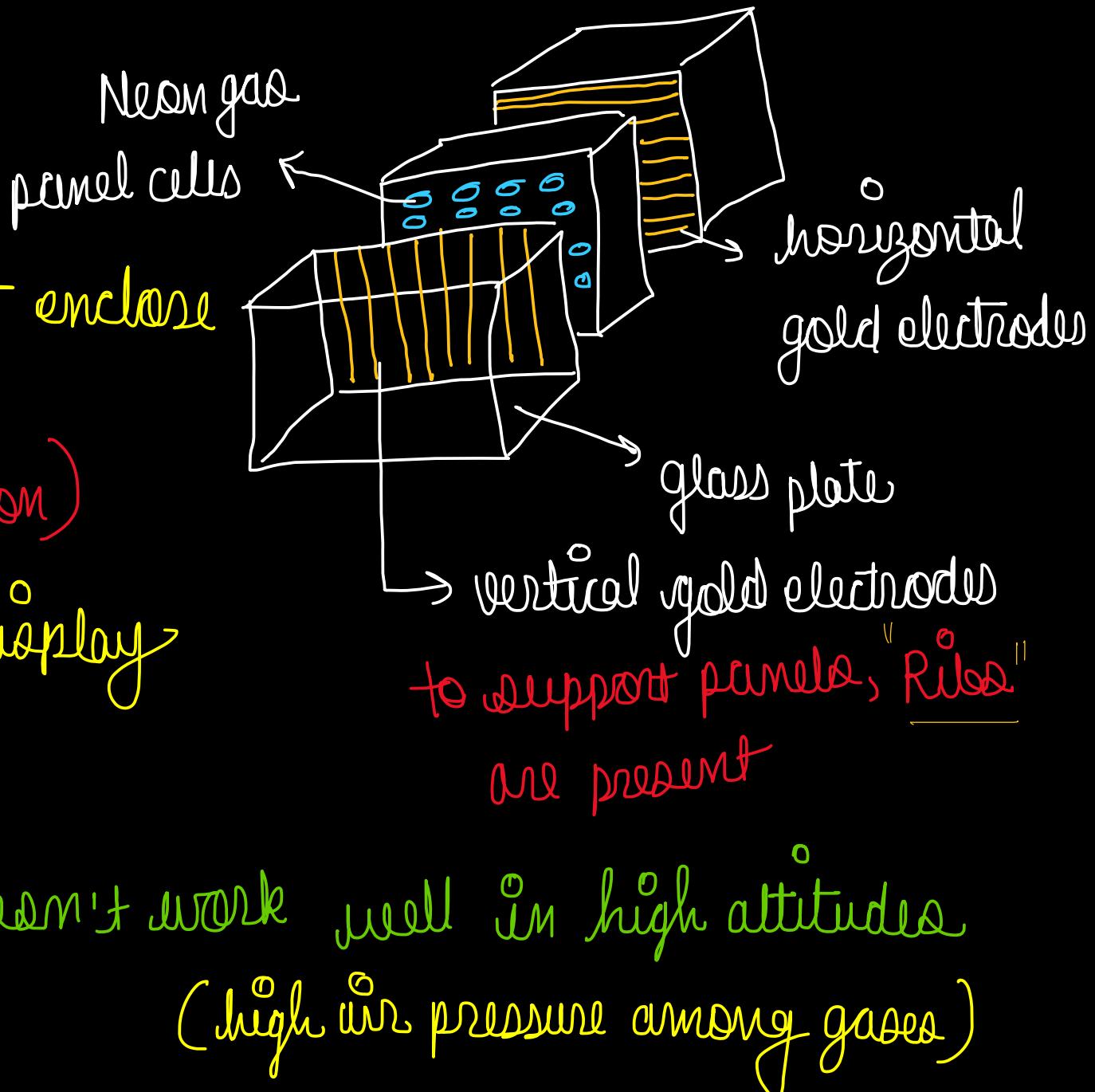
The devices which convert electrical energy into light. (Plasma, LED)

② Non Emissive Display → The devices which use optical effects to convert light into graphics pattern.
(LCD)

Gas discharge display

Plasma Display

- 2 parallel sheets of glass that enclose a mixture of discharged gases composed of (helium, xenon, neon)
- comes in both B&W & color display
- thinner than CRT
- brighter than LCD, also cheaper
- consumes more power
- generates more heat



LED (Light Emitting Diode)

- a matrix of diodes is organized to form pixel positions in the display.
- Picture definition is stored in a refresh buffer.
- Data is read from refresh buffer & converted into voltage levels that are applied to the diodes to produce the light pattern in the display.

| LCD (Liquid Crystal Display) → requires very little power

- produce a picture by passing a polarized light from the surroundings or from an internal light source through a liquid crystal material that transmits the light.
- 2 glass plates and liquid crystal is filled in between.
- One glass plate → # rows of conductor in vertical direction
- Other glass plate → # rows of conductors in horizontal direction
- The pixel position is determined by intersection of horizontal & vertical conductor. Position → active part of the screen
- Temperature dependent ($0-70^{\circ}\text{C}$) operating temperature

LED vs LCD

Light Emitting Diode (LED)

- ① better response time than LCD
- ② consumes more power
- ③ more expensive
- ④ better picture quality than LCD
- ⑤ better color accuracy, contrast & black level.
- ⑥ uses gallium arsenide phosphide
- ⑦ no mercury used
- ⑧ range upto 90 inches
- ⑨ wider view angle

Liquid Crystal Display (LCD)

- ① slower response time than LED
- ② consumes less power
- ③ less expensive
- ④ good but not as great as LED
- ⑤ slight less color accuracy, contrast & black level
- ⑥ uses liquid crystals & glass electrodes
- ⑦ mercury is required
- ⑧ range 13 - 57 inches
- ⑨ wide angle less with 30°

Mathematics of Computer Graphics:

- Point representation,
- Vector representation,
- Matrices and operations related to matrices,
- Vector addition and vector multiplication,
- Scalar product of two vectors,
- Vector product of two vectors.

Parametric equations of lines and conics.

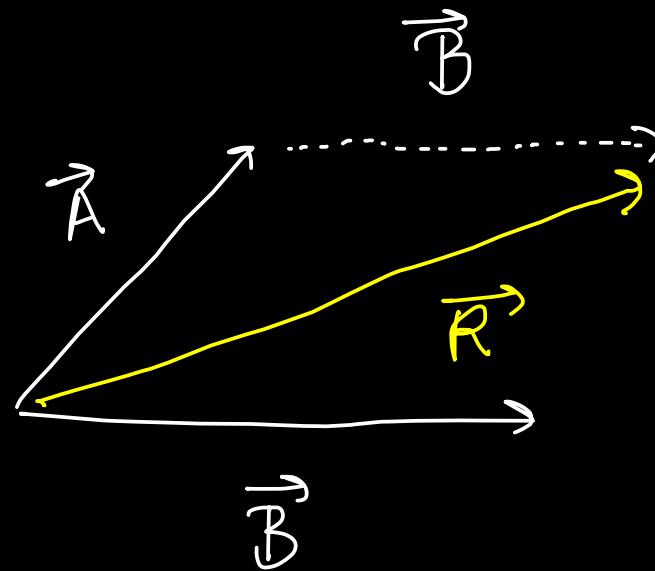
|

Vectors { Representation & Benefits }

Vector \rightarrow magnitude + direction

$$\vec{A} = |\vec{A}| \cdot \hat{A}$$

Addition of vectors





Vectors

Line Drawing Algorithms:

- ✓ DDA algorithms,
- Bresenham's Line algorithm.
- Circle and ellipse generation algorithm.

Clipping:

- Point Clipping,
- Line Clipping.
- Polygon Clipping.

Filling:

- Inside Tests,
- Flood fill algorithm,
- Boundary-Fill Algorithm and
- scan-line polygon fill algorithm.

Digital Differential Analyzer

DDA Line Drawing Algorithm

Given two points (x_1, y_1) and (x_2, y_2)
 $(1, 1)$ $(3, 3)$

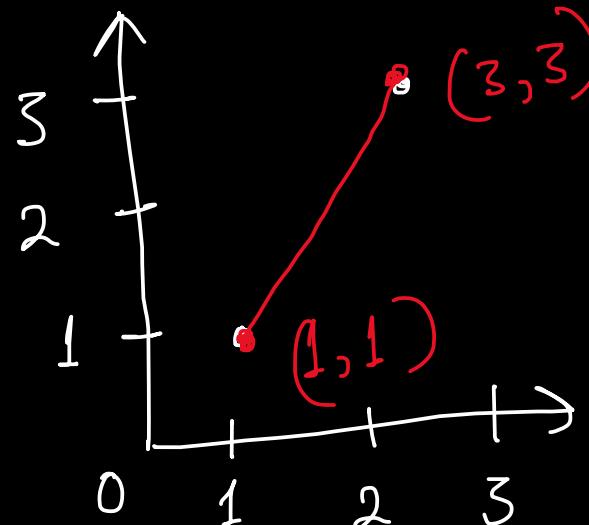
① Slope, $m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$

$$\Rightarrow \frac{3-1}{3-1} = \frac{2}{2} = 1$$

② find Δx and Δy

$$\Delta x = 2$$

$$\Delta y = 2$$



DDA Line Drawing Algorithm

$$m = \frac{\Delta y}{\Delta x}$$

$$\Delta x = \Delta y/m \quad \text{--- ②}$$

$$\Delta y = m \Delta x \quad \text{--- ①}$$

if $|\Delta x| \geq |\Delta y|$

then assign $\Delta x = 1$

$$\begin{aligned} x_{i+1}^0 &= x_i^0 + \Delta x \\ &= x_j^0 + 1 \end{aligned}$$

$$\begin{aligned} y_{i+1}^0 &= y_i^0 + \Delta y = y_i^0 + m \Delta x \\ &= y_i^0 + m(1) \\ &= y_i^0 + m \end{aligned}$$

if $|\Delta x| < |\Delta y|$

then assign $\Delta y = 1$

$$\begin{aligned} y_{i+1}^0 &= y_i^0 + \Delta y \\ &= y_i^0 + 1 \end{aligned}$$

$$\begin{aligned} x_{i+1}^0 &= x_j^0 + \Delta x \\ &= x_j^0 + \Delta y/m \\ &= x_j^0 + 1/m \end{aligned}$$

(1,1) and (3,3)

DDA

$$\Delta x = 2$$

$$\Delta y = 2$$

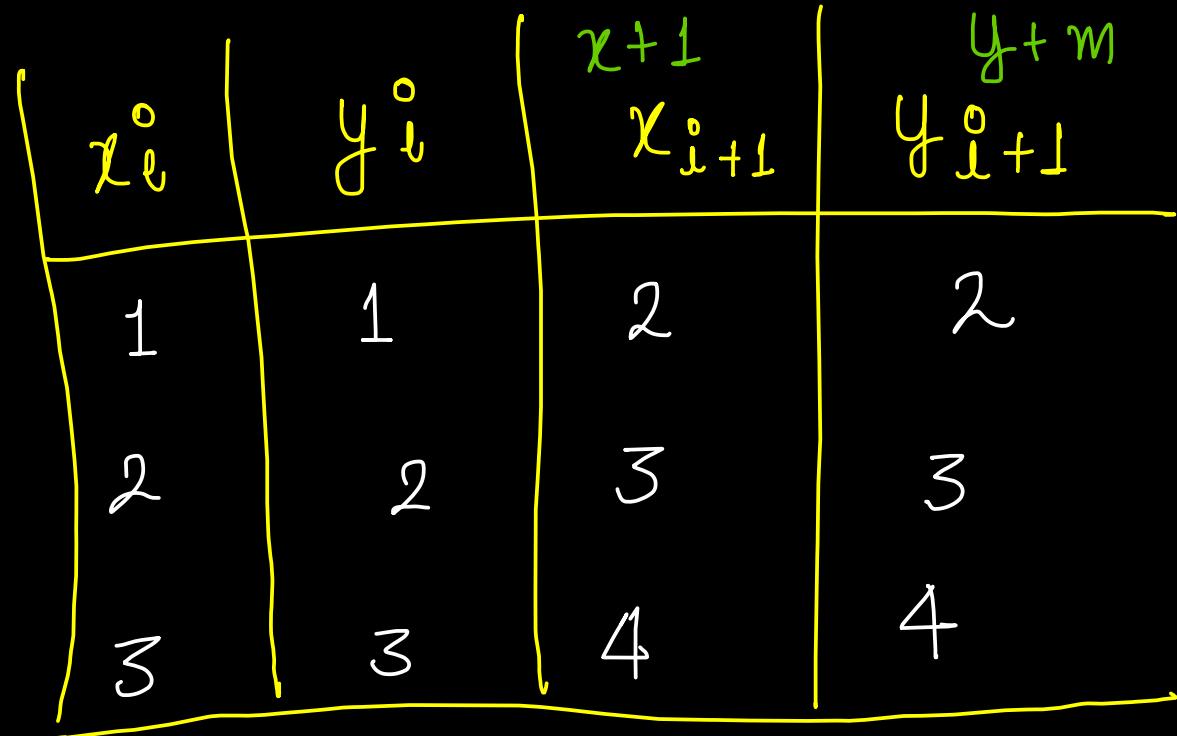
$$\therefore \Delta x \geq \Delta y$$

$$\Delta x = 1, \quad \Delta y = m \Delta x = m$$

$$x_{i+1}^o = x + \Delta x = x + 1$$

$$y_{i+1}^o = y + m \Delta x = y + m$$

$$m = \frac{\Delta y}{\Delta x} = \frac{3-2}{3-2} = 1$$



$$Ans = \{ (1,1), (2,2), (3,3) \}$$

DDA → gives coordinates in floating point values

Bresenham's Line Drawing Algorithm

① find slope, $m = \Delta y / \Delta x$ ② find Decision Parameter (P) = $2\Delta y - \Delta x$

$m < 1$

$P < 0$

$$x_{i+1}^o = x_i^o + 1$$

$$y_{i+1}^o = y_i^o$$

$$P_{k+1} = P_k + 2\Delta y$$

$P < 0$

$m \geq 1$

$$x_{i+1}^o = x_i^o$$

$$y_{i+1}^o = y_i^o + 1$$

$$P_{k+1} = P_k + 2\Delta x$$

$P \geq 0$

$$x_{i+1}^o = x_i^o + 1$$

$$y_{i+1}^o = y_i^o + 1$$

$$P_{k+1} = P_k + 2\Delta y - 2\Delta x$$

$P \geq 0$

$$x_{i+1}^o = x_i^o + 1$$

$$y_{i+1}^o = y_i^o + 1$$

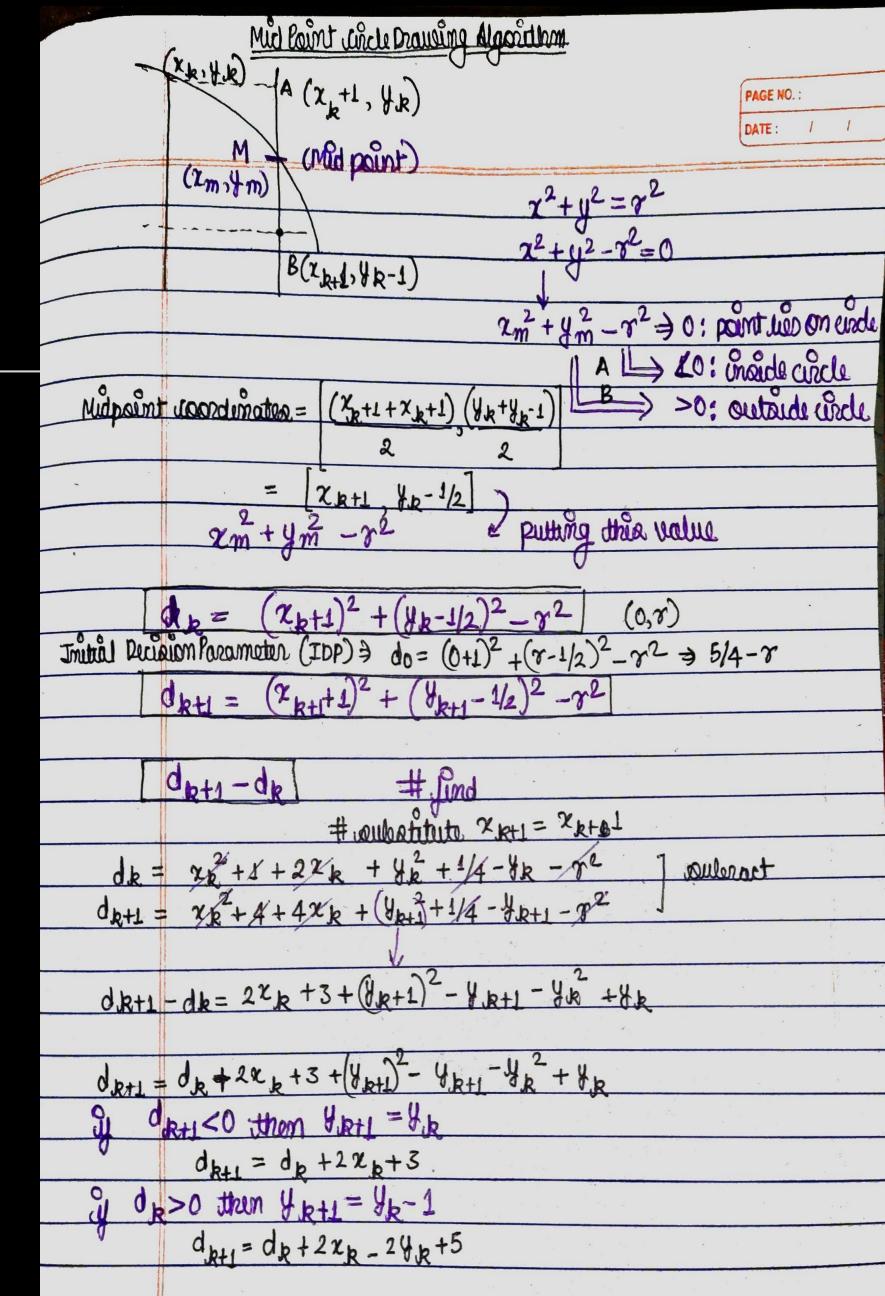
$$P_{k+1} = P_k + 2\Delta x - \Delta y$$

Mid Point Circle

Initial decision parameter = $\frac{5}{4} - \gamma$
(P)

If radius r is specified as an integer then the initial decision parameter is equal to

$$P = 1 - \gamma$$



Brensenham Circle

start point (x_0, r)

Initial decision parameter = $3 - 2r$
(P)

while $(x_{i+1}^0 < y_{i+1}^0)$:

$$x_{i+1}^0 = x_i^0 + 1$$

if $P \leq 0$:

$$y_{i+1}^0 = y_i^0 ; P_{i+1}^0 = P_i^0 + 4x_{i+1}^0 + 6$$

else

$$y_{i+1}^0 = y_i^0 - 1 ; P_{i+1}^0 = P_i^0 + 4(x_{i+1}^0 - y_{i+1}^0) + 10$$

Mid Sem-Practical

- ① additive primary colors (RGB)
- ② Refresh Buffer / Refresh Memory \rightarrow Frame Buffer stores picture definition
- ③ Interlacing \rightarrow incrementally displaying visual
 - \hookrightarrow vertical retrace krke Barr Baar chikhana
- ④ Beam penetration \rightarrow 2 layers of phosphor in Random Scan Display
- ⑤ Shadow Masking \rightarrow Raster Scan Display \rightarrow 3 color dots {Shadow mask grid.}

Viva Prep

graphics controller / display controller / display processor

↳ digitizing a picture definition into a set of pixel with appropriate intensity

Q28. What is the difference between impact and non-impact printers?

Impact printers press formed character faces against an inked ribbon on to the paper.

A line printer and dot-matrix printer are examples.

Non-impact printer and plotters use Laser techniques, inkjet sprays, Xerographic process, electrostatic methods and electro thermal methods to get images onto the papers.

Examples are: Inkjet/Laser printers.



Viva Prep

DDA

$$\begin{cases} \Delta x = x_2 - x_1 \\ \Delta y = y_2 - y_1 \end{cases} \rightarrow \begin{cases} dx \\ dy \end{cases}$$

if $\Delta x \geq \Delta y$;

$$\text{steps} = \text{abs}(\Delta x)$$

else;

$$\text{steps} = \text{abs}(\Delta y)$$

$$x_{\text{inc}} = \Delta x / \text{steps}$$

$$y_{\text{inc}} = \Delta y / \text{steps}$$

$$x += x_{\text{inc}}$$

$$y += y_{\text{inc}}$$

```
void DDA(int x1, int y1, int x2, int y2){  
    int dx = x2 - x1;  
    int dy = y2 - y1;  
    int steps = dx >= dy ? abs(dx) : abs(dy);  
    float xinc = dx / (float)steps;  
    float yinc = dy / (float)steps;  
    int i = 0;  
    cout << xinc << '\t' << yinc << endl;  
    cout << "\nThe consequent coordinates are: " << endl;  
    int x = x1, y = y1;  
    cout << "x0: " << x << ", y0: " << y << endl;  
    while (i <= steps)  
        // while (i <= 10)  
    {  
        /* code */  
        putpixel(x, y, RED);  
        x += xinc;  
        y += yinc;  
        i++;  
        cout << "x_next: " << x << ", y_next: " << y << endl;  
    }  
    putpixel(x, y, RED);  
}
```

Bresenham Line Drawing

$$\Delta x = x_1 - x_0$$

$$\Delta y = y_1 - y_0$$

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

if $P \geq 0$:

$y+ = 1$

$P+ = 2\Delta y - 2\Delta x$

else :

$y = y$

$P+ = 2\Delta y$

$x+ = 1$

```
void bresenham(int x0, int y0, int x1, int y1){  
    int x=x0; int y=y0;  
    if (x0 > x1)  
    {  
        x = x1; y = y1;  
        x1 = x0; y1 = y0;  
        x0 = x; y0 = y;  
    }  
    int dx= x1-x0;  
    int dy= y1-y0;  
    int p= 2*dy-dx;  
    while(x<x1){  
        if(p>=0){  
            putpixel(x,y,WHITE);  
            y=y+1;  
            p=p+2*dy-2*dx;  
        }  
        else{  
            putpixel(x,y,WHITE);  
            p=p+2*dy;  
        }  
        x=x+1;  
    }  
}
```

Mid point circle

$$P = 1 - \gamma$$

$$x = 0, y = \gamma$$

while $x \leq y$:

draw

$$x += 1$$

if $P \leq 0$:

$$P = P + 2x + 1$$

else :

$$y = y - 1$$

$$P = P + 2(x-y) + 1 \quad \}$$

```
void drawMidPointCircle(int x0, int y0, int radius){  
    int y = radius, x = 0;  
    int decisionParam = 1 - radius;
```

(x_0, γ)

```
while (x <= y)  
{
```

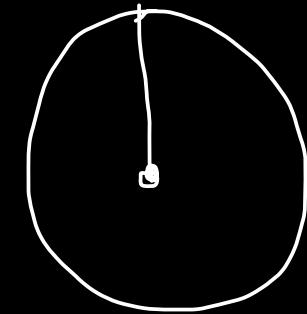
```
    putpixel(x0 + x, y0 + y, 1);  
    putpixel(x0 - x, y0 + y, 2);  
    putpixel(x0 + x, y0 - y, 3);  
    putpixel(x0 - x, y0 - y, 4);  
    putpixel(x0 + y, y0 + x, 5);  
    putpixel(x0 - y, y0 + x, 6);  
    putpixel(x0 + y, y0 - x, 7);  
    putpixel(x0 - y, y0 - x, 8);  
    delay(400);
```

```
x++;
```

```
if (decisionParam <= 0)  
    decisionParam += 2 * x + 1;
```

```
else{  
    y--;  
    decisionParam += 2 * (x-y) + 1;
```

```
}
```



Bresenham's Circle generation

$$x=0, y=r$$

$$P = 3 - 2r$$

while ($x \leq y$):

draw \longrightarrow

$$x+ = 1$$

if $P \leq 0$:

$$P = P + 4x + 6$$

else

$$y = y - 1$$

$$P = P + 4(x-y) + 10 \quad \}$$

```
void drawBresenhamCircle(int x0, int y0, int radius){
```

```
int x = 0, y = radius;
```

```
int decisionParam = 3 - 2 * radius;
```

```
while (x <= y)
```

```
{
```

```
putpixel(x0 + x, y0 + y, RED);
```

```
putpixel(x0 + y, y0 + x, RED);
```

```
putpixel(x0 - y, y0 + x, RED);
```

```
putpixel(x0 - x, y0 + y, RED);
```

```
putpixel(x0 - x, y0 - y, RED);
```

```
putpixel(x0 - y, y0 - x, RED);
```

```
putpixel(x0 + y, y0 - x, RED);
```

```
putpixel(x0 + x, y0 - y, RED);
```

```
if (decisionParam <= 0){
```

```
  x++;
```

```
  decisionParam += 4 * x + 6;
```

```
}
```

```
else{
```

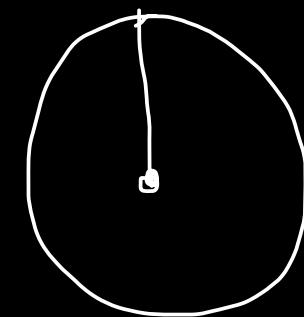
```
  x++;
```

```
  y--;
```

```
  decisionParam += 4 * (x - y) + 10;
```

```
}
```

(x_0, y)



Traffic Light

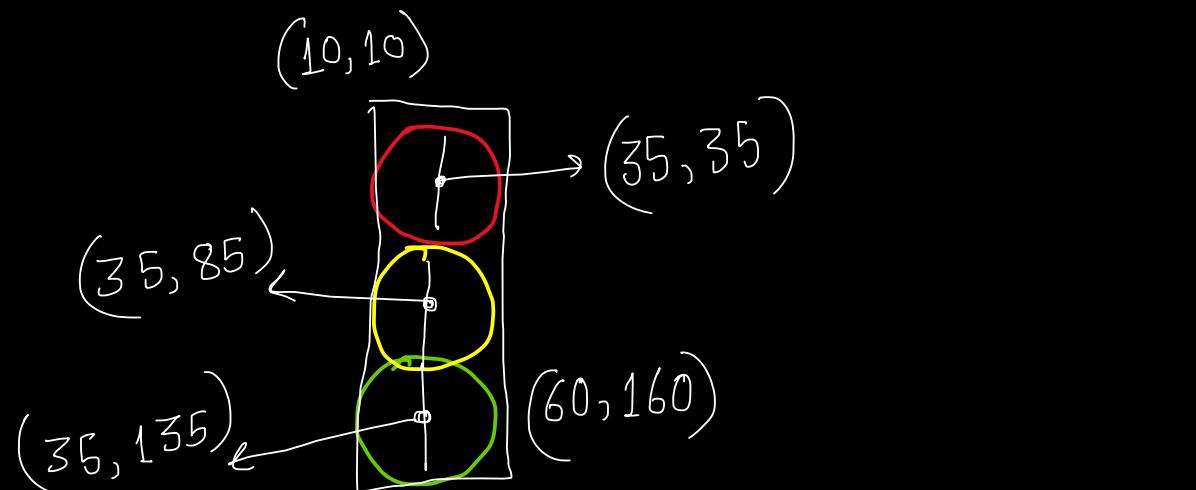
```
void drawTrafficLight(int x, int y, int lightSize, bool isRedOn, bool isYellowOn, bool isGreenOn){  
    // Draw black background  
    setfillstyle(SOLID_FILL, BLACK);  
    bar(x, y, x + lightSize, y + 3 * lightSize);  
  
    // Draw red light  
    setfillstyle(SOLID_FILL, isRedOn ? RED : DARKGRAY);  
    circle(x + lightSize / 2, y + lightSize / 2, lightSize / 2);  
    floodfill(x + lightSize / 2, y + lightSize / 2, WHITE);  
  
    // Draw yellow light  
    setfillstyle(SOLID_FILL, isYellowOn ? YELLOW : DARKGRAY);  
    circle(x + lightSize / 2, y + lightSize + lightSize / 2, lightSize / 2);  
    floodfill(x + lightSize / 2, y + lightSize + lightSize / 2, WHITE);  
  
    // Draw green light  
    setfillstyle(SOLID_FILL, isGreenOn ? GREEN : DARKGRAY);  
    circle(x + lightSize / 2, y + 2 * lightSize + lightSize / 2, lightSize / 2);  
    floodfill(x + lightSize / 2, y + 2 * lightSize + lightSize / 2, WHITE);  
}
```

Traffic Light

```
void red_circle()
{
    setfillstyle(SOLID_FILL, RED);
    circle(35, 35, 25);
    floodfill(35, 35, WHITE);
}

void yellow_circle()
{
    setfillstyle(SOLID_FILL, YELLOW);
    circle(35, 85, 25);
    floodfill(35, 85, WHITE);
}

void green_circle()
{
    setfillstyle(SOLID_FILL, GREEN);
    circle(35, 135, 25);
    floodfill(35, 135, WHITE);
}
```



```
(10,10) → (35,35)
(35,85) ← (35,135)
(60,160)
```

```
while (1)
{
    rectangle(10, 10, 60, 160);
    red_circle();
    delay(400);
    cleardevice();
    rectangle(10, 10, 60, 160);
    yellow_circle();
    delay(400);
    cleardevice();
    rectangle(10, 10, 60, 160);
    green_circle();
    delay(400);
    cleardevice();
}
```

Smiley

```
#include <graphics.h>
int main(){
    // draw graphics using graphics.h
    int gDriver = DETECT, gMode;
    initgraph(&gDriver, &gMode, NULL);
    // draw a circle
    circle(200, 200, 100);
    // draw a smile
    arc(200, 200, 210, 330, 50);
    // draw eyes
    circle(170, 180, 5);
    circle(230, 180, 5);
    // draw a nose
    line(200, 200, 200, 230);
    getch();
    closegraph();
    return 0;
}
```

Hut

```
#include<graphics.h>
int main(){
    // draw a hut
    int gDriver = DETECT, gMode;
    initgraph(&gDriver, &gMode, NULL);
    // draw a rectangle
    rectangle(100, 100, 300, 300);
    // draw a triangle
    line(100, 100, 200, 50);
    line(200, 50, 300, 100);
    line(100, 100, 300, 100);
    // draw a door
    rectangle(150, 230, 200, 300);
    circle(150, 150, 30);
    circle(250, 150, 30);
    getch();
    closegraph();
    return 0;
}
```

Important Functions of <graphics.h>

// Initializes the graphics system

void initgraph(int *graphdriver, int *graphmode, const char *path);

(x_2, y_2)

// Closes the graphics system

void closegraph();

(x_L, y_L)

// Waits for a key press event and returns the key code

int getch();

// Draws a pixel of specified color at the given (x, y) coordinate

void putpixel(int x, int y, int color);

(x_L, y_L)

// Draws a line between two given points

void line(int x1, int y1, int x2, int y2);

(x_2, y_2)

// Draws a rectangle with given top-left and bottom-right coordinates

void rectangle(int left, int top, int right, int bottom);

x_1

y_L

x_2

y_2



Important Functions of <graphics.h>

```
// Draws a circle with given center and radius
void circle(int x, int y, int radius);
```

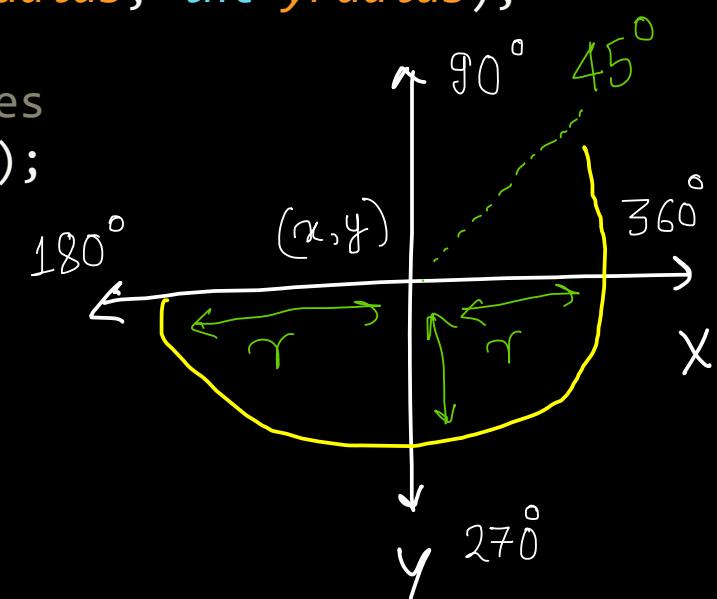
```
// Draws an ellipse with given center, horizontal and vertical radius
void ellipse(int x, int y, int start_angle, int end_angle, int xradius, int yradius);
```

```
// Draws an arc with given center, radius, and start and end angles
void arc(int x, int y, int start_angle, int end_angle, int radius);
```

```
// Sets the current drawing color to the specified color
void setcolor(int color);
```

```
// Sets the fill pattern and color for the next fill operation
void setfillstyle(int pattern, int color);
```

```
// Writes the specified text string at the given (x, y) position on the screen
void outtextxy(int x, int y, const char *text);
```



Important Functions of <graphics.h>

```
// Sets the font and size of the text to be drawn
void settextstyle(int font, int direction, int charsize);

// Returns the maximum x coordinate of the current graphics screen
int getmaxx();

// Returns the maximum y coordinate of the current graphics screen
int getmaxy();

void floodfill(int x, int y, int boundary_color);
```

Bresenham v/s DDA line generation

Accuracy: DDA line algorithm provides high accuracy for line generation, as it generates points based on the exact slope of the line. Bresenham algorithm, on the other hand, generates points based on integer arithmetic operations, which can result in some inaccuracies in the line.

Speed: Bresenham algorithm is faster than DDA algorithm, as it uses only integer arithmetic operations, which are faster than floating-point arithmetic operations used in DDA algorithm.

Efficiency: Bresenham algorithm is more efficient than DDA algorithm in terms of memory usage, as it requires only two variables to generate the line, whereas DDA algorithm requires variables for both x and y coordinates.

Implementation: Bresenham algorithm is more complex to implement than DDA algorithm, as it involves checking for multiple decision parameters and adjusting the values of these parameters based on the value of the error term. DDA algorithm, on the other hand, is relatively simple to implement, as it involves only basic arithmetic operations.

Applications: DDA algorithm is suitable for applications that require high accuracy in line generation, such as CAD systems and architectural design. Bresenham algorithm, on the other hand, is suitable for applications that require fast and efficient line generation, such as real-time graphics and video games.

Bresenham Circle v/s Mid Point Circle

Accuracy: Bresenham's algorithm is more accurate than the Midpoint algorithm. Bresenham's algorithm uses integer arithmetic and only calculates the points that lie closest to the actual circle, resulting in fewer rounding errors. Midpoint algorithm uses floating-point arithmetic, which can lead to some inaccuracies.

Efficiency: Midpoint algorithm is more efficient than the Bresenham's algorithm. The Midpoint algorithm uses only addition and subtraction, whereas Bresenham's algorithm ~~involves division and multiplication operations~~, which can be more time-consuming.

Implementation: Bresenham's algorithm is harder to implement than Midpoint algorithm. Bresenham's algorithm is more complex, and it requires additional calculations for each octant of the circle. In contrast, the Midpoint algorithm is simpler and only requires one set of calculations.

Circle size: Bresenham's algorithm works better for small circles, while Midpoint algorithm works better for large circles. Bresenham's algorithm is limited by the accuracy of integer arithmetic, which can make it difficult to draw large circles. Midpoint algorithm, on the other hand, can handle larger circles without loss of accuracy.

Symmetry: Bresenham's algorithm has better symmetry than Midpoint algorithm. Bresenham's algorithm calculates points that are symmetric to each other along the circle, while the Midpoint algorithm doesn't.

Parametric Equations

Parametric equations are a way of representing geometric shapes such as lines and conics using parameters instead of the traditional x and y coordinates. These parameters can be either angles or distances, depending on the shape being represented.

The parametric equation of a line in two dimensions is usually written in the form:

$$x = x_1 + t(x_2 - x_1)$$

$$y = y_1 + t(y_2 - y_1)$$

where (x_1, y_1) and (x_2, y_2) are two points on the line, and t is a parameter that varies between 0 and 1. As t varies, the point (x, y) traces out the line segment connecting the two points.

For a conic section such as an ellipse, the parametric equation is written in terms of two parameters, usually denoted by u and v . For example, the parametric equation for an ellipse centered at the origin is:

$$x = a \cos u$$

$$y = b \sin u$$

where a and b are the lengths of the major and minor axes of the ellipse, respectively. As u varies from 0 to 2π , the point (x, y) traces out the entire ellipse.

Parametric Equations

Similarly, the parametric equation for a hyperbola can be written in terms of two parameters, usually denoted by u and v , as:

$$x = a \cosh u$$

$$y = b \sinh u$$

where a and b are again the lengths of the major and minor axes of the hyperbola. As u varies, the point (x, y) traces out the two branches of the hyperbola.

In general, the advantage of using parametric equations to represent geometric shapes is that they provide a more flexible and compact way of expressing complex shapes. They are also useful for animations and computer graphics, where the motion of a point along a path can be easily described using a parametric equation.



























