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

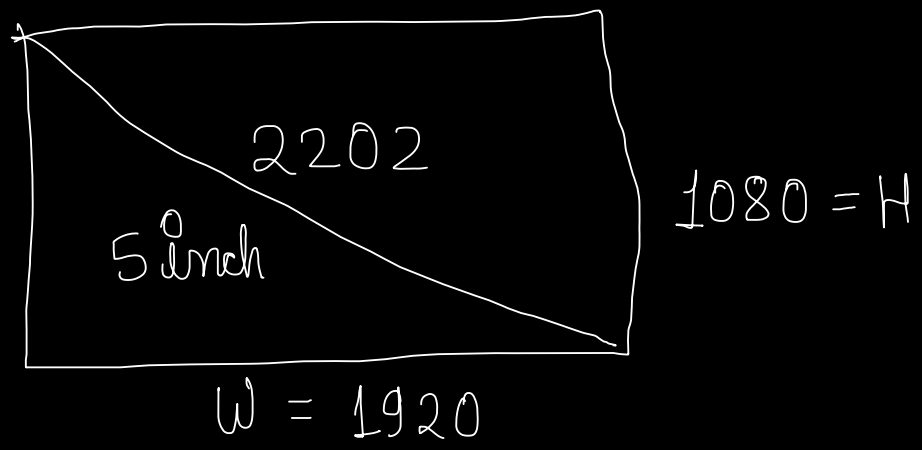
Applications

- 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
- Resolution → # pixels on entire screen → width x height ⇒ 1920 x 1080
- PPI ⇒ 2202/5 ⇒ 440 PPI
- Aspect Ratio ⇒ 16:9 (width : height)
- Frame Buffer (yaha info store hoti h)

1920 : 1080
480 : 270
16 : 9



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

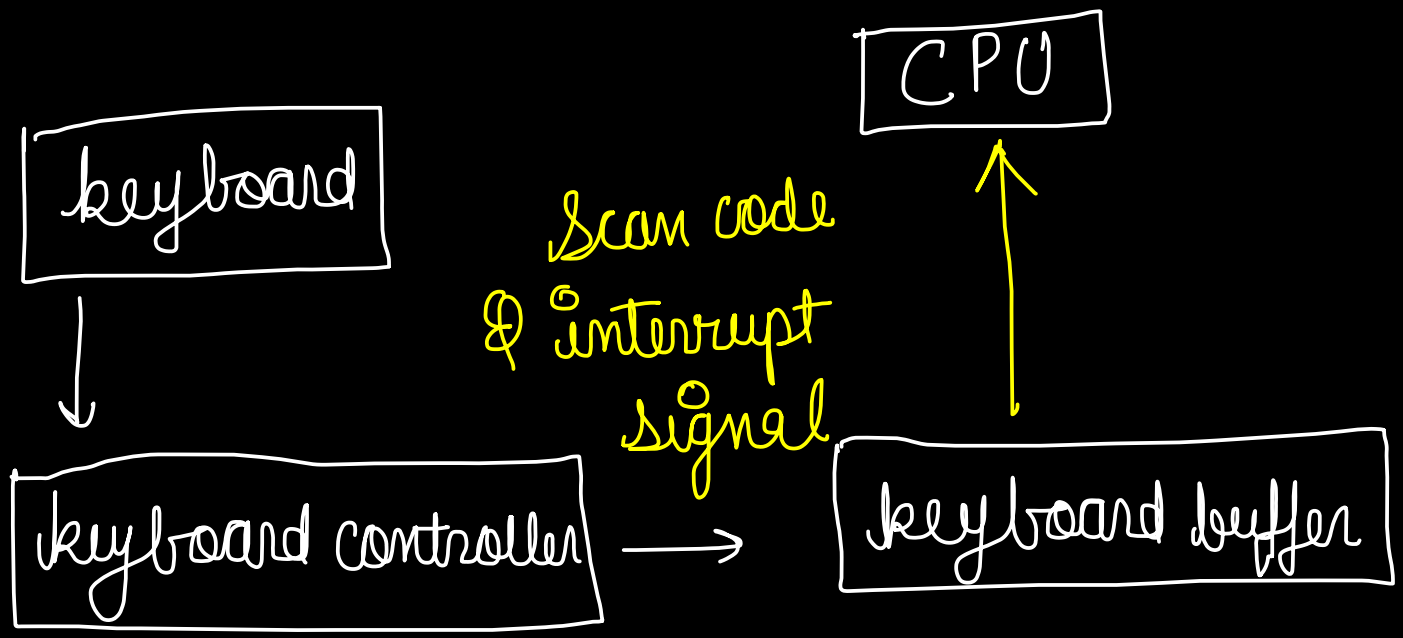
① Keyboard ② Mouse
③ Trackball/Spaceball ④ Joystick

Output Devices

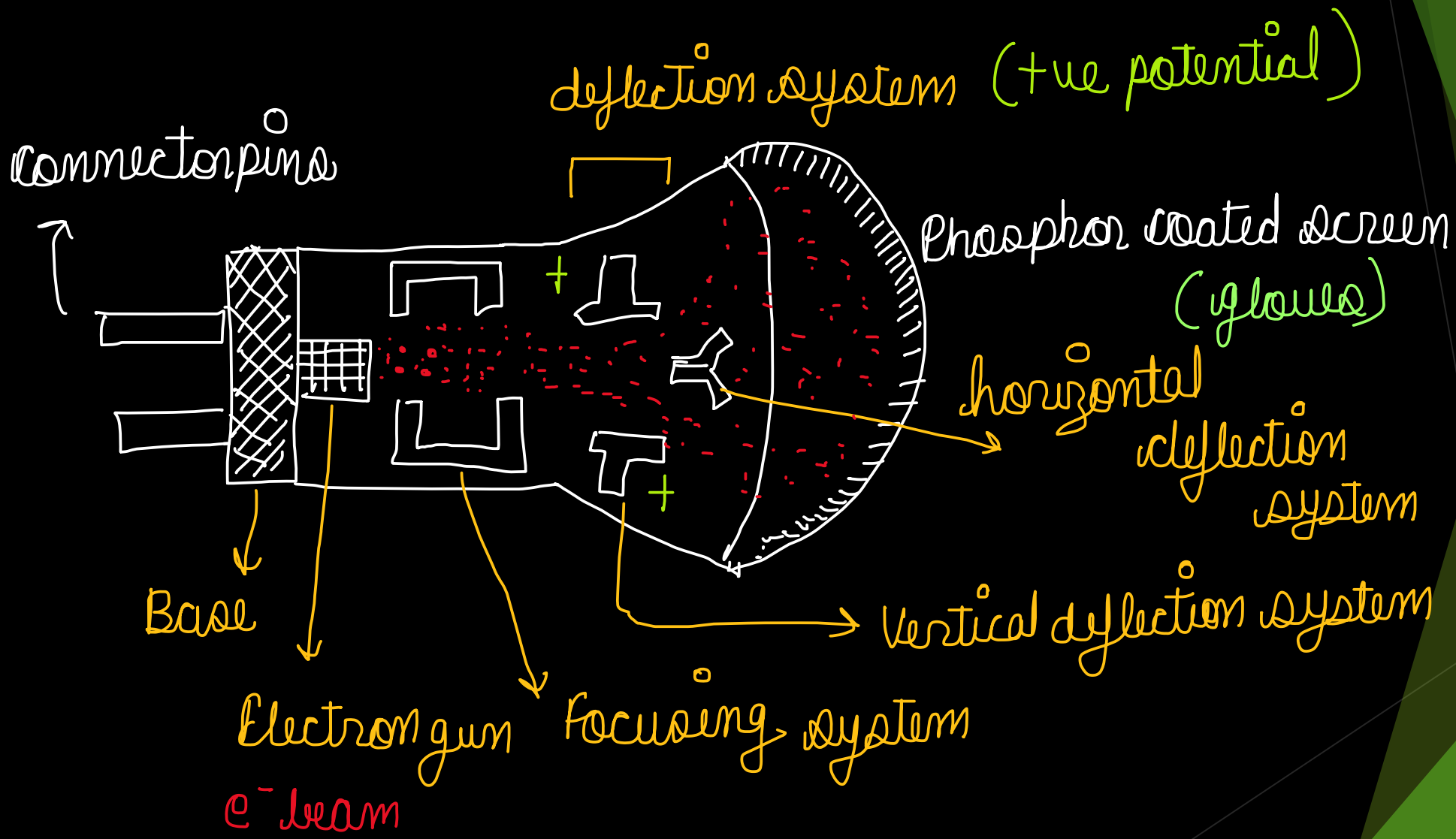
① CRT
② LED, LCD, TFT
③ Plasma
④ Colored Monitor

Graphics Software

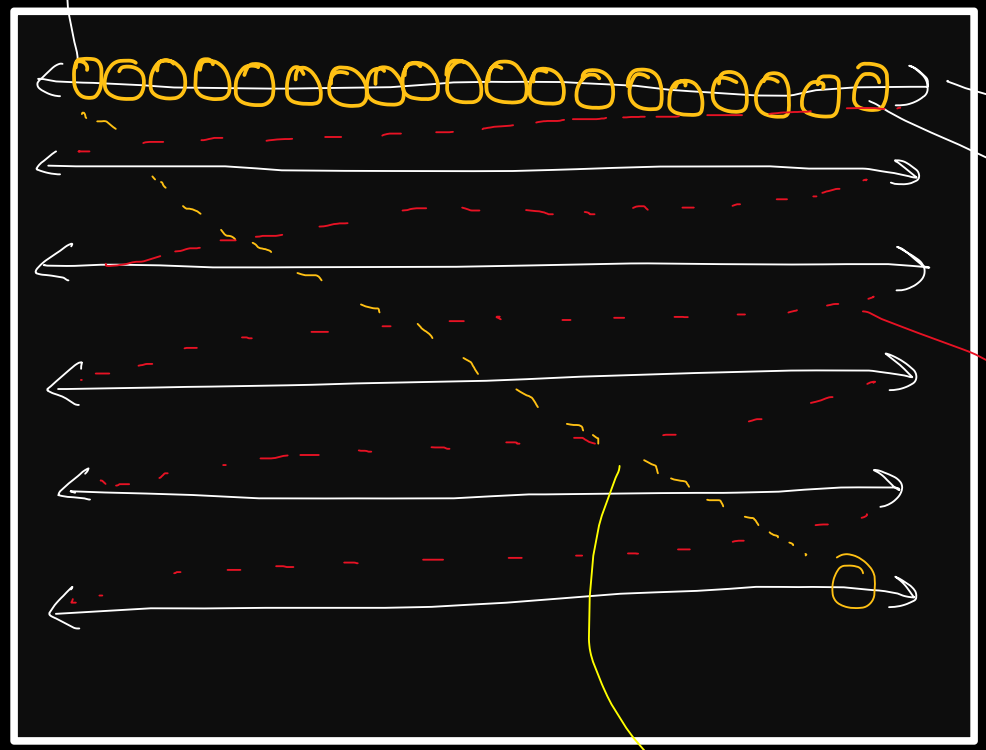
① Photoshop ③ Cad S/F
② Maya 3D ④ Corel Draw



Cathode Ray Tube(CRT)



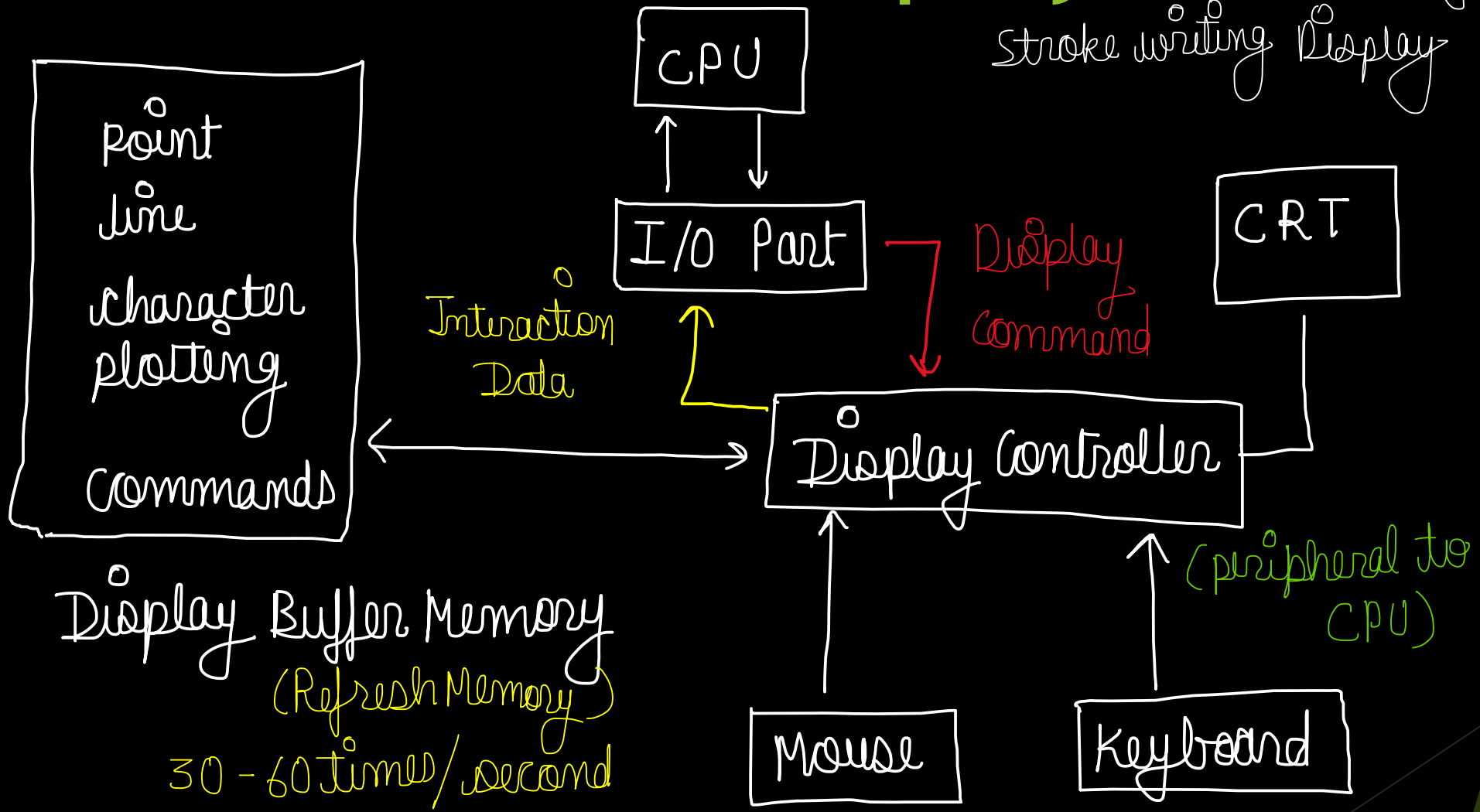
Rastor Scan Display



Intensity value ⇒ Refresh / Frame Buffer

Vector Scan Display

Random Scan Display
Calligraphic Display
Stroke writing Display



Display Controller ⇒ 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

Interlacing

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, Lyreen, 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

slow e⁻ beam → penetrates red and excites green

average e⁻ beam → excites combination of Red & green {Yellow & Orange}

Shadow Masking (Raster Scan Display)

→ gives more color than beam penetration

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

→ This type of CRT have 3 e⁻ guns for each colordot

→ A shadow mask grid is installed behind phosphor coated screen

Flat Panel Display

(volume, weight, power requirement) < CRT

→ TV, Monitor, Calculator, Laptop, etc.

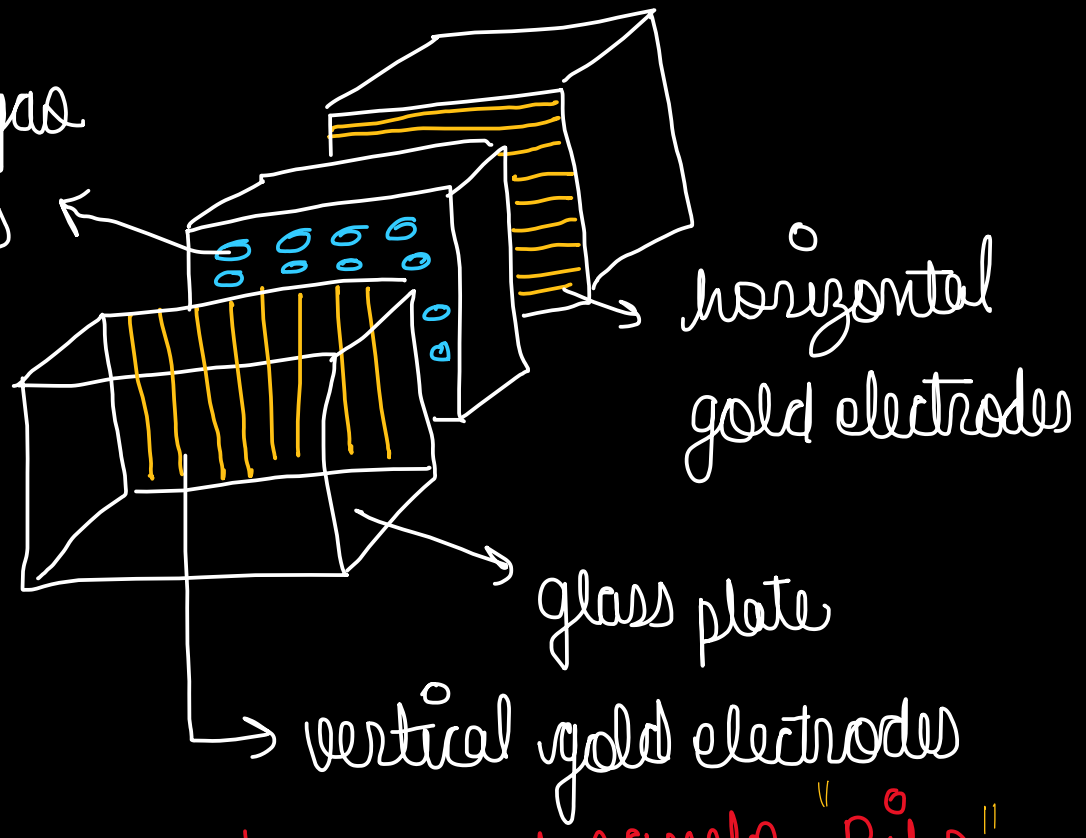
2 categories:

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, Xenon)

→ comes in both B&W & color display

→ Thinner than CRT

→ Brighter than LCD, also cheaper

→ consumes more power

→ generates more heat

→ doesn't work well in high altitudes (high in pressure among gases)

to support panels, "Ribs" are present

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°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^{0}

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.

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

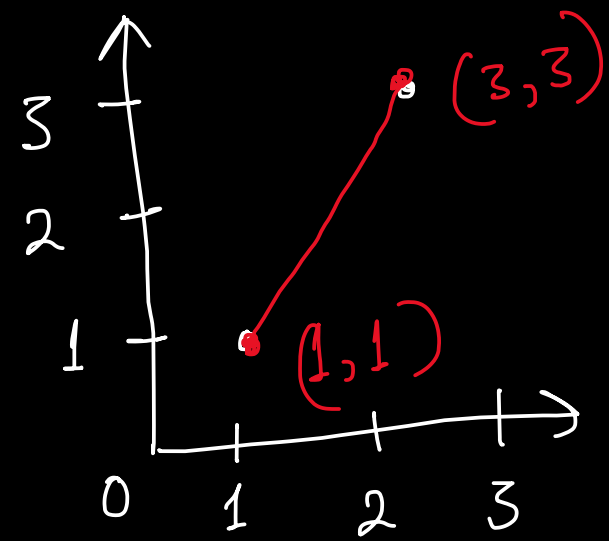
DDA Line Drawing Algorithm

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

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

$\Delta y = m \Delta x$ — (1)

$\Delta x = \Delta y / m$ — (2)

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

then assign $\Delta x = 1$

$$x_{i+1} = x_i + \Delta x$$
$$= x_i + 1$$

$$y_{i+1} = y_i + \Delta y = y_i + m \Delta x$$
$$= y_i + m(1)$$
$$= y_i + m$$

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

then assign $\Delta y = 1$

$$y_{i+1} = y_i + \Delta y$$
$$= y_i + 1$$

$$x_{i+1} = x_i + \Delta x$$
$$= x_i + \Delta y / m$$
$$= x_i + 1/m$$

(1,1) and (3,3)

DDA

$\Delta x = 2$

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

$\Delta y = 2$

$\because \Delta x \geq \Delta y$

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

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

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

x_i	y_i	$x+1$ x_{i+1}	$y+m$ y_{i+1}
1	1	2	2
2	2	3	3
3	3	4	4

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

If $m < 1$

$P < 0$

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

$y_{i+1} = y_i$

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

$P \geq 0$

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

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

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

If $m \geq 1$

$P < 0$

$x_{i+1} = x_i$

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

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

$P \geq 0$

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

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

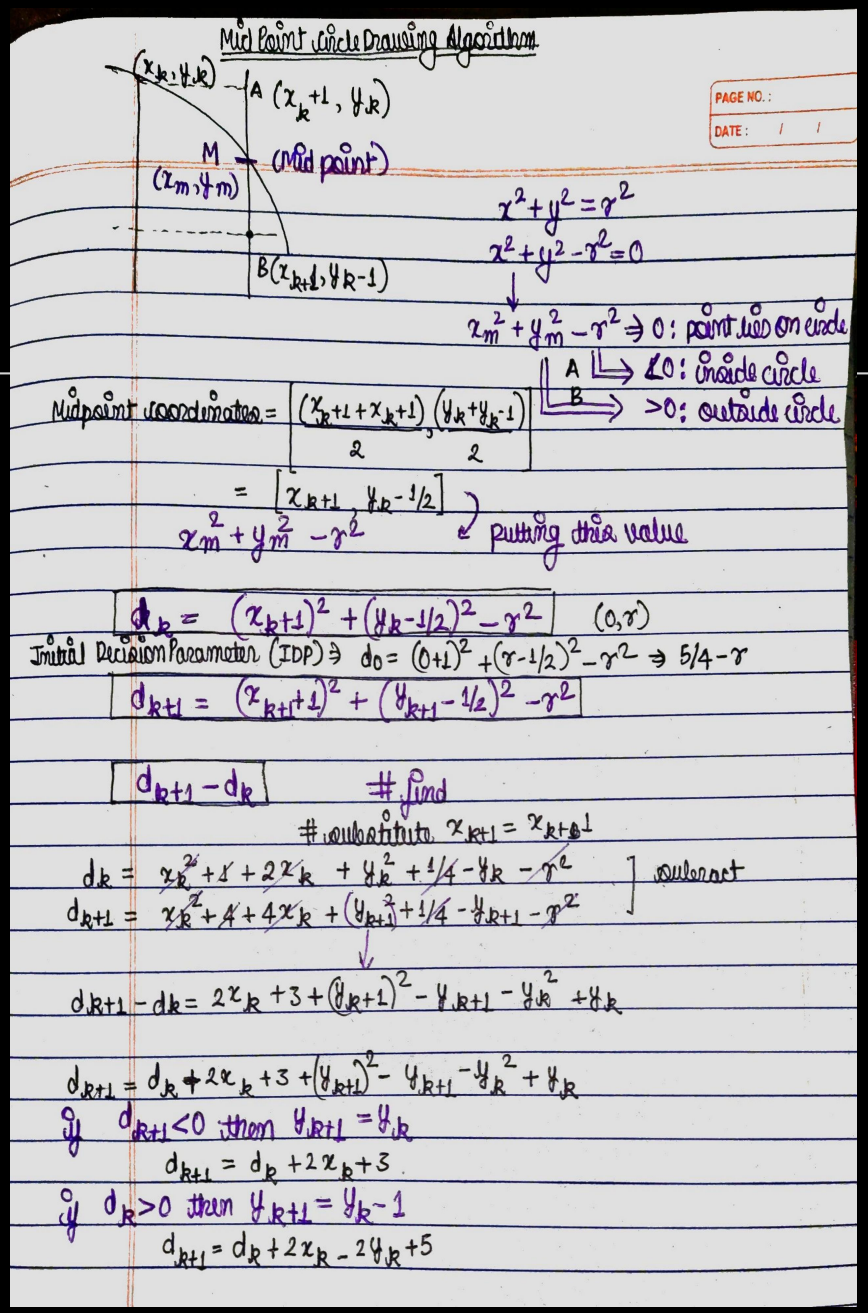
$P_{k+1} = P_k + 2\Delta x - \Delta y$

Mid Point Circle

Initial decision parameter (p) $= \frac{5}{4} - r$

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

$$P = 1 - r$$



Brensenham Circle

start point (x_0, r)

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

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

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

if $p \leq 0$;

$y_{i+1} = y_i$; $p_{i+1} = p_i + 4x_{i+1} + 6$

else

$y_{i+1} = y_i - 1$; $p_{i+1} = p_i + 4(x_{i+1} - y_{i+1}) + 10$

Mid Sem-Practical

DDA

$\Delta x = x_2 - x_1$ → dx

$\Delta y = y_2 - y_1$ → dy

if $\Delta x >= \Delta y$;

steps = abs (Δx)

else ;

steps = abs (Δy)

x_inc = Δx / steps

y_inc = Δy / steps

x += x_inc

y += y_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);
}
```

Brenshanham Line Drawing

$$\Delta x = x_1 - x_0$$
$$\Delta y = y_1 - y_0$$

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

if $p >= 0$:
- $y += 1$
- $p += 2\Delta y - 2\Delta x$

else :
- $y = y$
- $p += 2\Delta y$

$x += 1$

```
void brensanham(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

$P = 1 - r$

$x = 0, y = r$

while $x <= y$:

draw

$x += 1$

if $P <= 0$:

$P = P + 2x + 1$

else :

$y = y - 1$

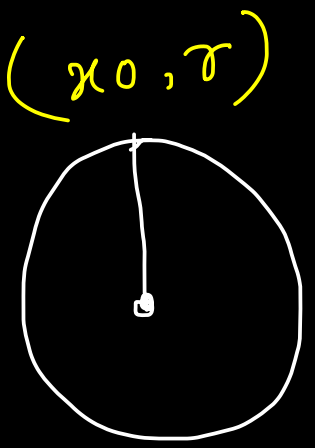
$P = P + 2(x-y) + 1$

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

    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;
        }
    }
}
```



Brensanham's Circle generation

$x = 0, y = r$

$p = 3 - 2r$

while $(x \leq y)$:

draw ⟶

$x+ = 1$

if $p <= 0$:

$p = p + 4x + 6$

else

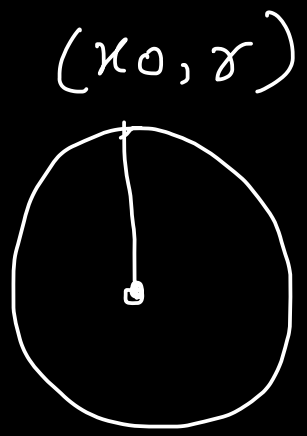
$y = y - 1$

$p = p + 4(x - y) + 10$

```
void drawBrensanhamCircle(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;
        }
    }
}
```



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);
}
```

Smiley

```c
#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;
}
```