FLAME UNIVERSITY

EVERLASTING learning

FUNDAMENTALS OF COMPUTER GRAPHICS (CSIT304)

RASTER GRAPHICS AND SCAN CONVERSION

CHIRANJOY CHATTOPADHYAY

Associate Professor,

FLAME School of Computation and Data Science

RECAP

- OVERVIEW OF CG
- GRAPHICS PRIMITIVES
- CG AND RELATED FIELDS
- RELATIONSHIP AMONG VARIOUS TOPICS
  - RASTERIZATION
  - COLOR, SHADING, TEXTURE
  - CURVE, SURFACES

TODAY'S TOPIC

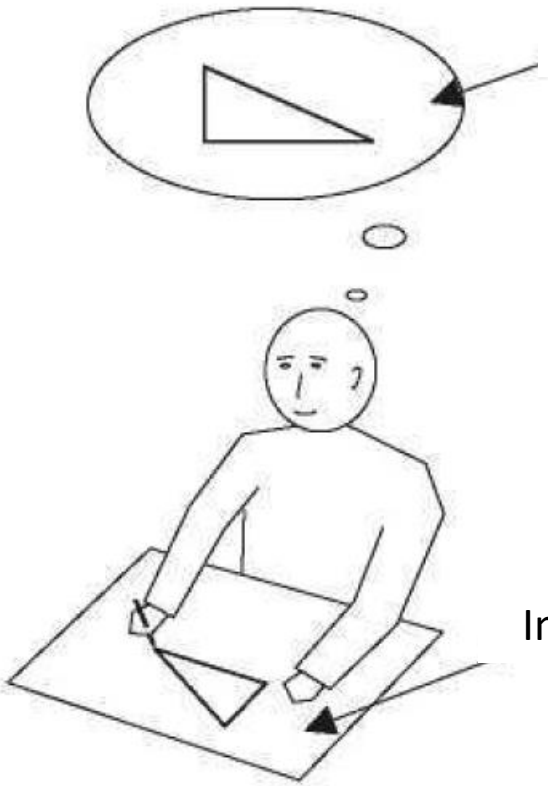
- RASTERIZATION
- CG DEVICES
- PIPELINE
- RASTERIZING PRIMITIVES
  - LINE

INTRODUCTION

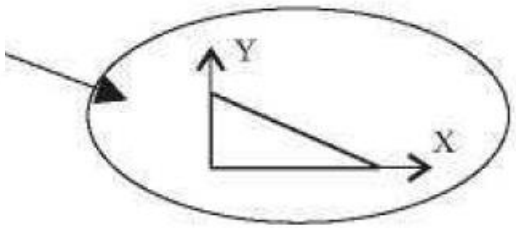
- CG deals with the theory and technology for computerized image synthesis.
- A computer-generated image can depict
  - A simple scene as the outline of a triangle on a uniform background
  - A complex scene as a real world scene of a game.
- How do these things become part of the picture?
- What makes drawing on a computer different from
  - Sketching with a pen or photographing with a camera?

AN EXAMPLE

1 Object definition



Object Space



2 Geometric Representation

3 Transformation

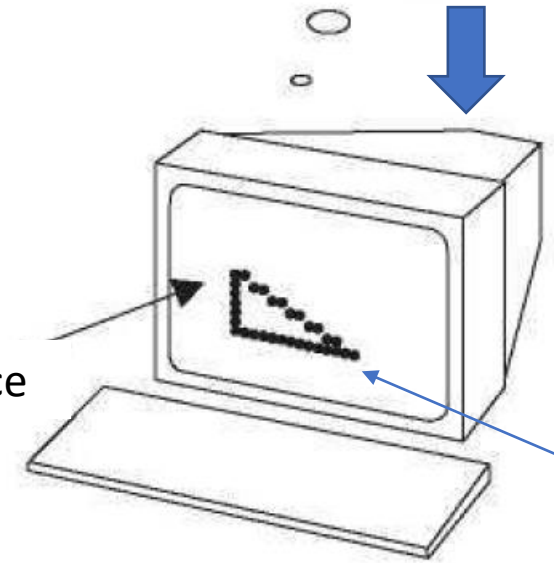
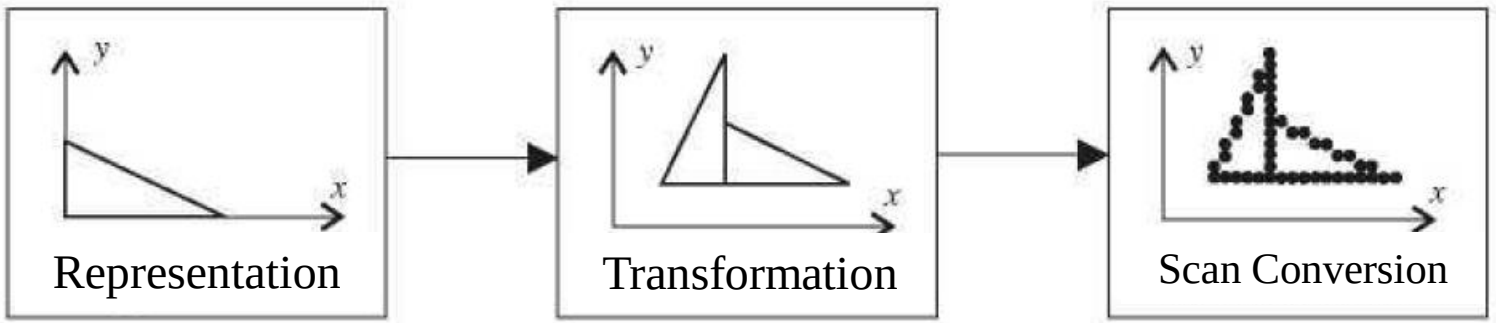


Image Space

4 Scan Conversion

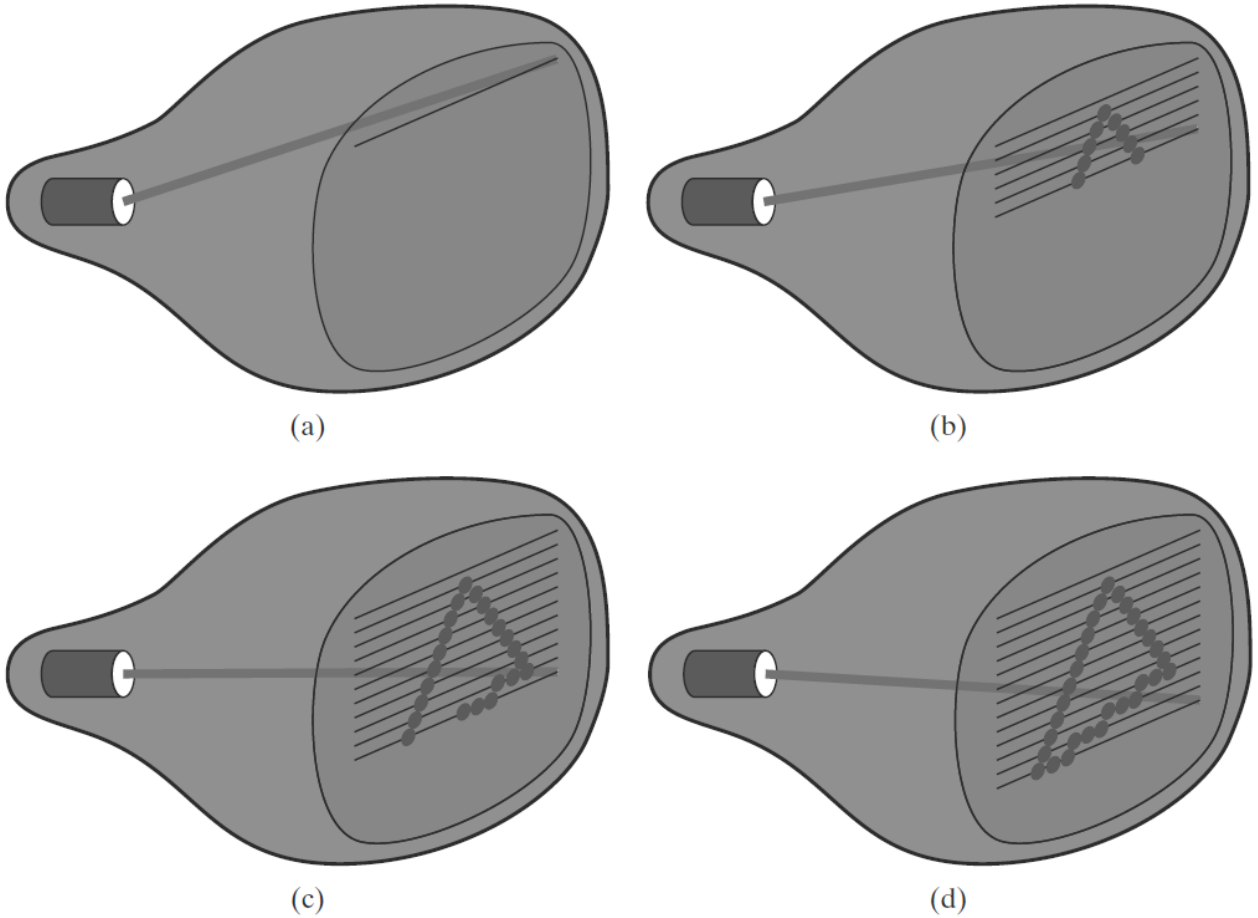
5 Anti-Aliasing

PRIMITIVE GRAPHICS PIPELINE



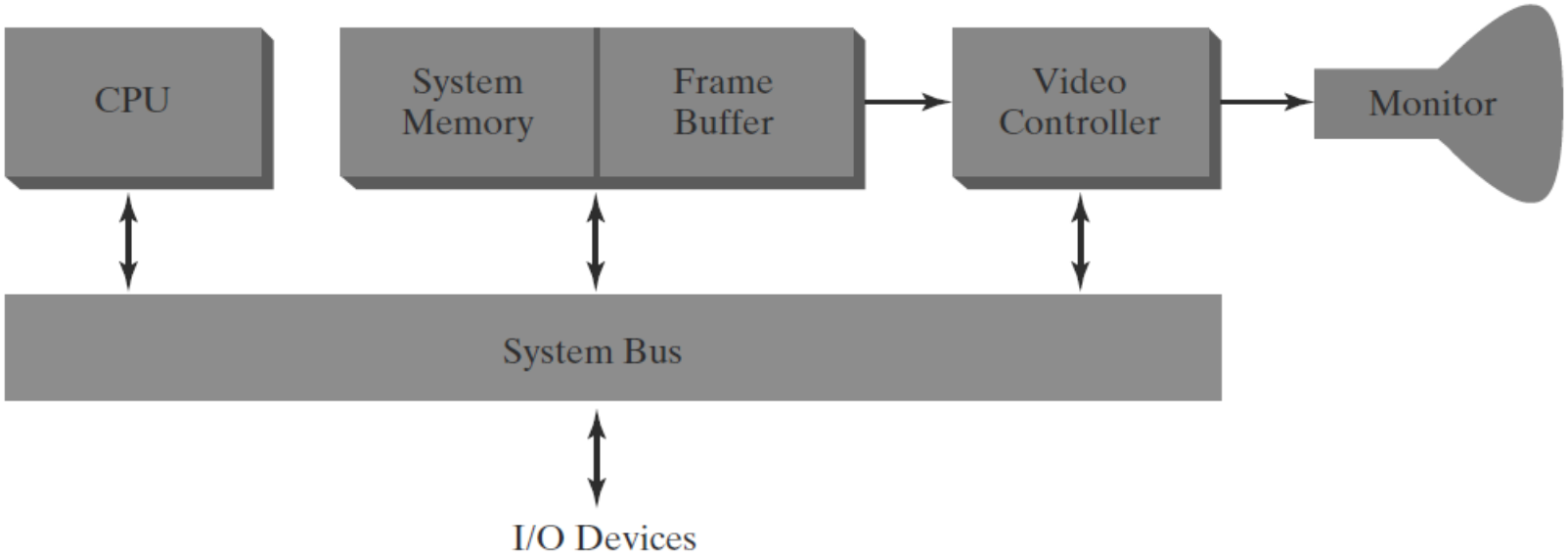
WORLD COORDINATE SYSTEM

RASTER SCAN DISPLAY

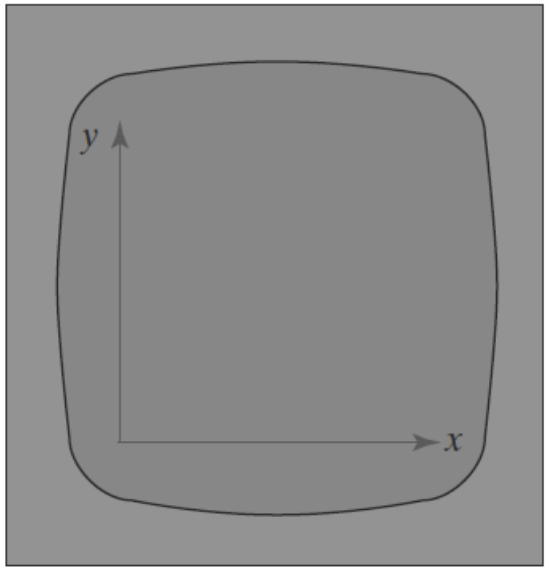


RASTER-SCAN SYSTEM

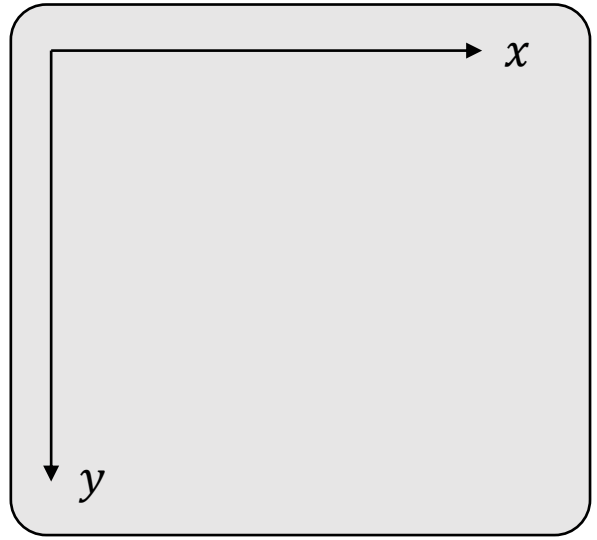
- Interactive raster-graphics systems typically employ several processing units.
- Central processing unit (CPU)
- A special-purpose processor, called the **video controller** or **display controller**
  - Used to control the operation of the display device.



REFERENCE FRAME

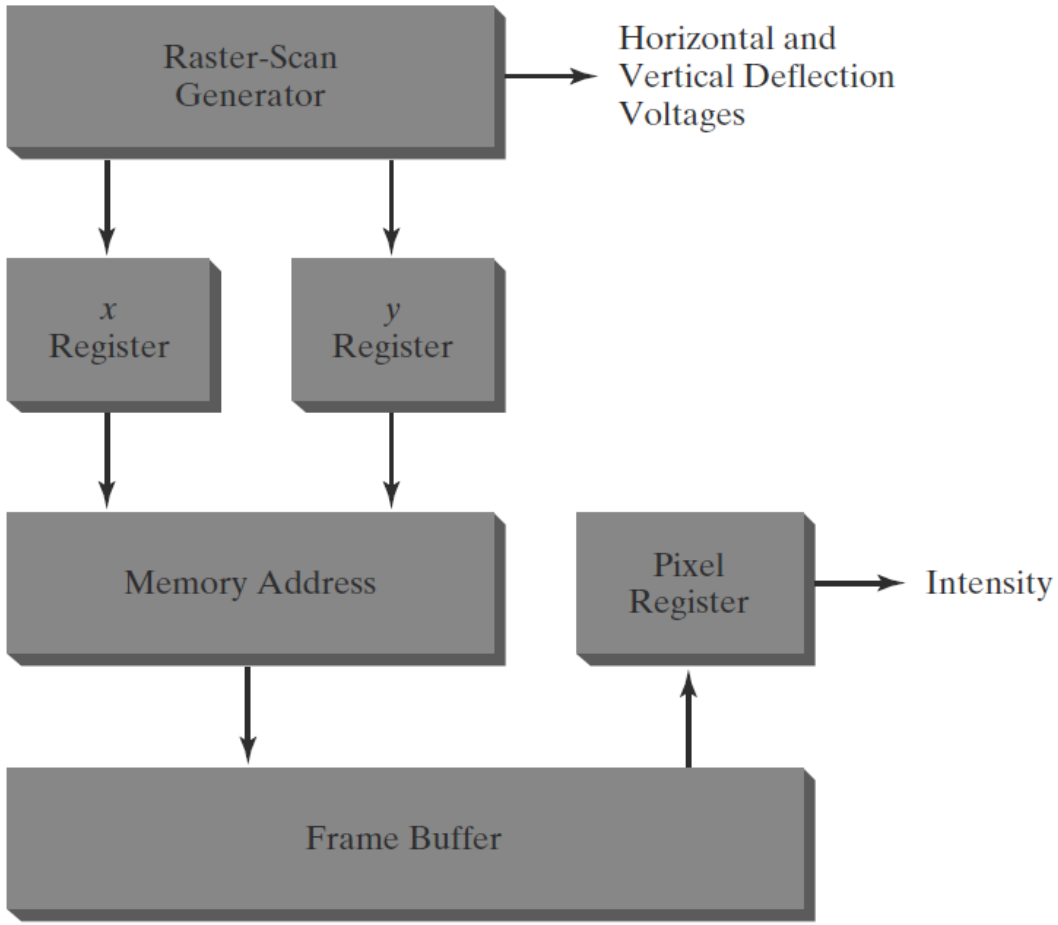



```
typedef struct tagRECT {
  LONG left;
  LONG top;
  LONG right;
  LONG bottom;
} RECT, *PRECT, *NPRECT, *LPRECT;
```

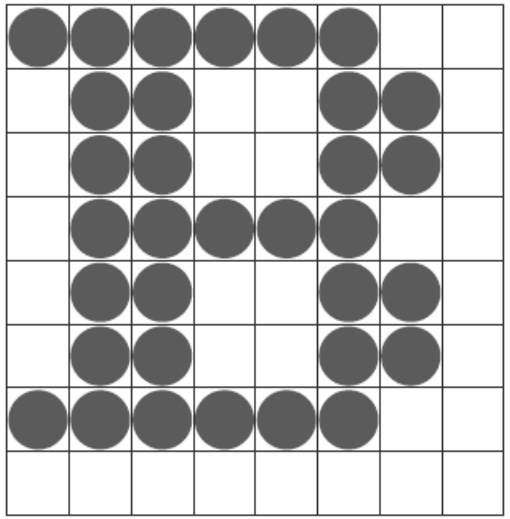


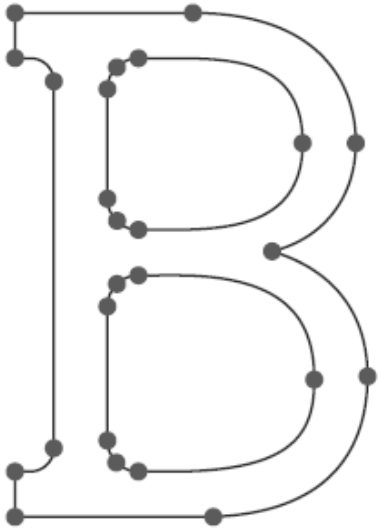
A Cartesian reference frame with origin at the lower-left corner of a video monitor.

VIDEO CONTROLLER AND REFRESH



CHARACTER DEFINITION





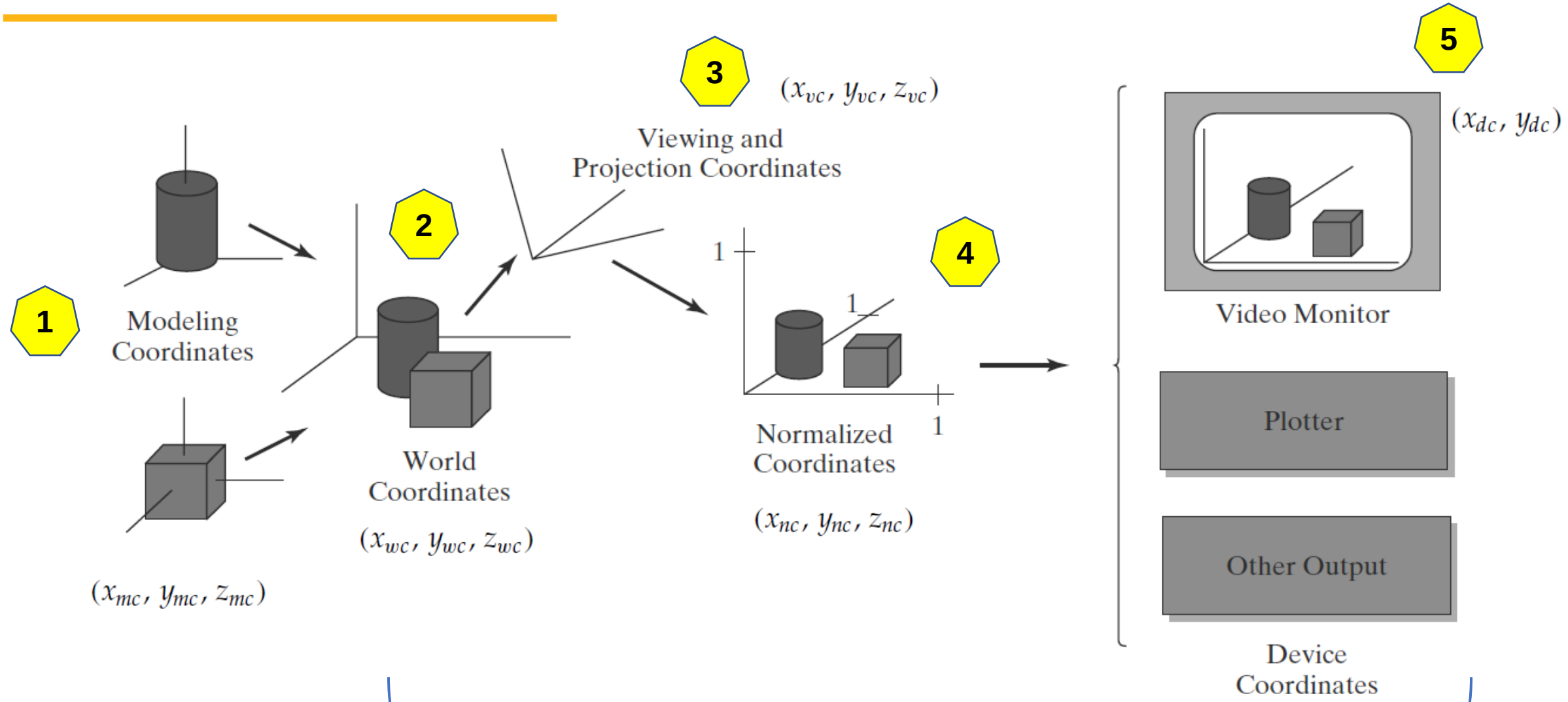
LOOKING INTO THE 3D WORLD

6 Projection Methods

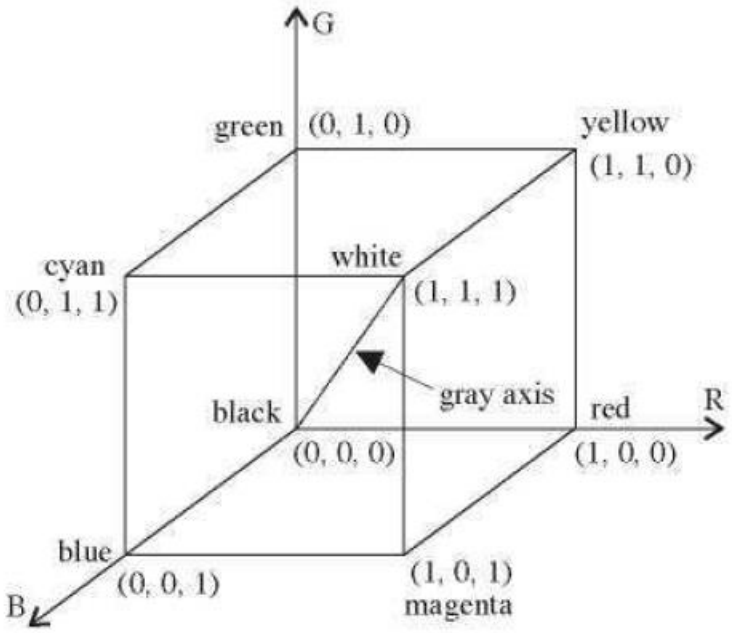
7 Hidden Surface Removal

8 Illumination and Shading

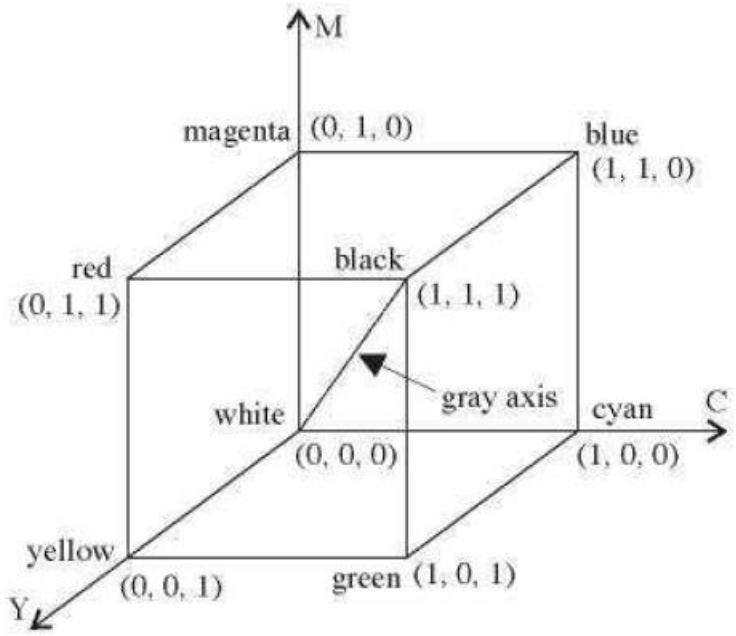
COORDINATE REPRESENTATION



COLOR REPRESENTATION



RGB- Color Space



CMY Color Space

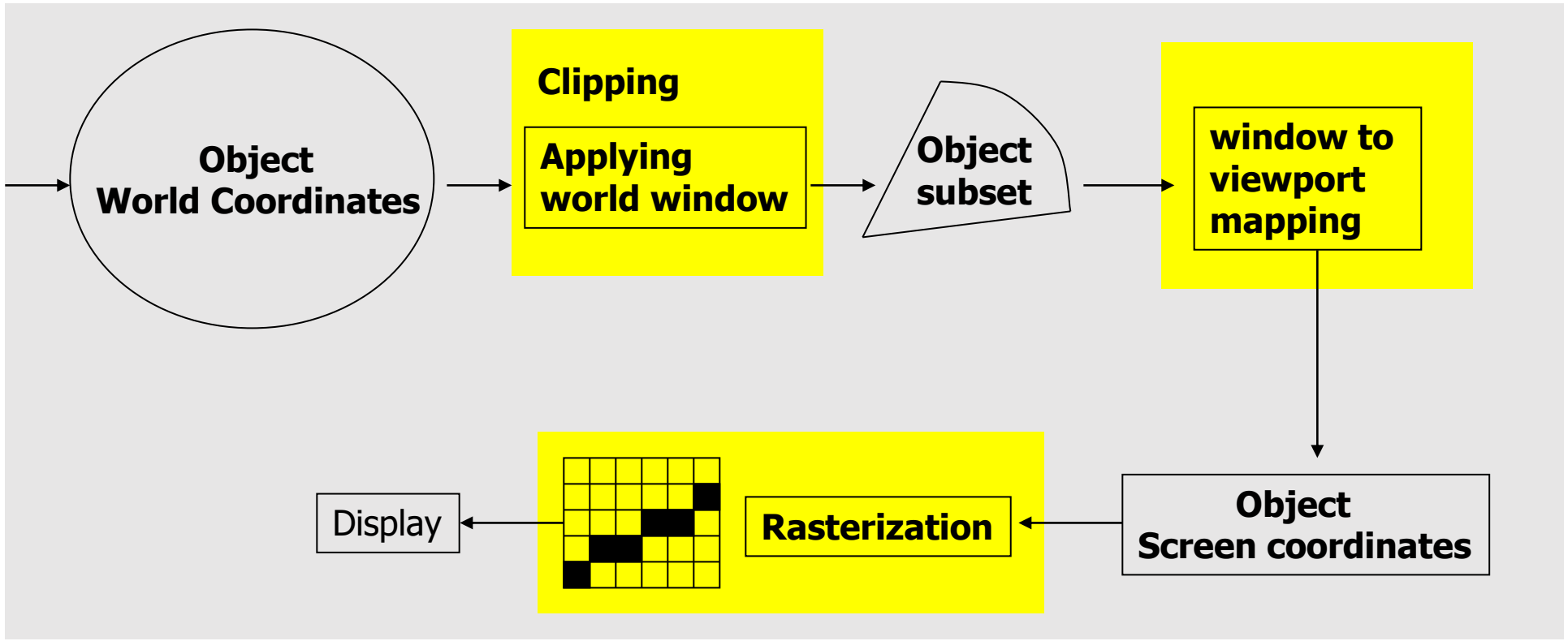
- Color coding using bits
  - 3 bit per color
  - 8 bit per color
- Black and White vs. Gray Scale
- Color map (Lookup)

AN EXAMPLE

- Mandelbrot set

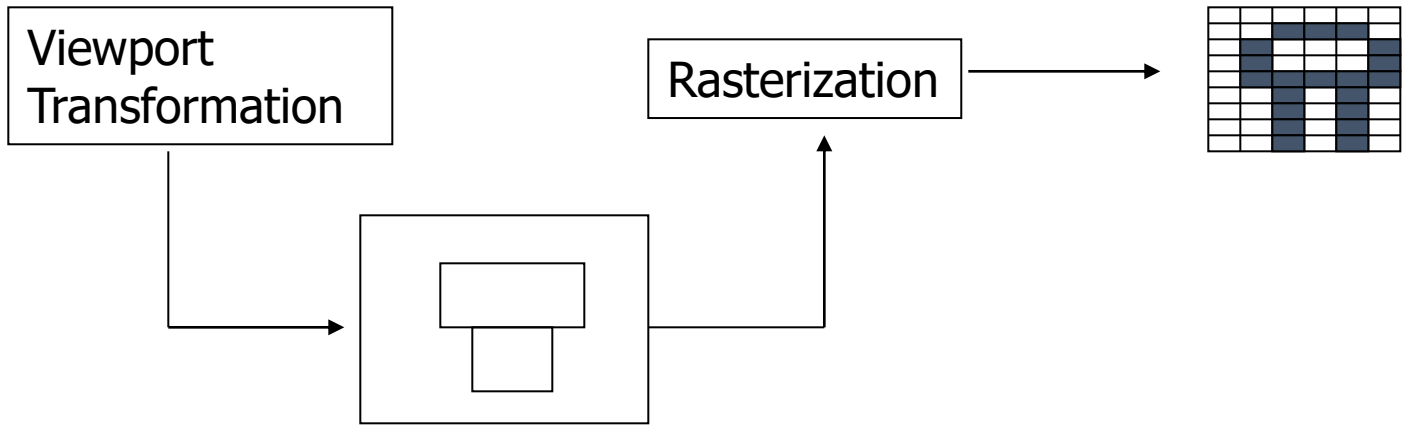
SCAN CONVERSION

2D GRAPHICS PIPELINE



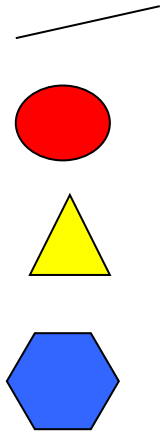
RASTERIZATION (SCAN CONVERSION)

- Convert high-level geometry description to pixel colors in the frame buffer
- Example: given vertex x, y coordinates determine pixel colors to draw line
- Two ways to create an image:
  - Scan existing photograph
  - Procedurally compute values (rendering)



RASTERIZATION

- A fundamental computer graphics function
- Determine the pixels' colors, illuminations, textures, etc.
- Implemented by graphics hardware
- Rasterization algorithms
  - Point
  - Lines
  - Circles
  - Triangles
  - Polygons



SCAN-CONVERTING A POINT

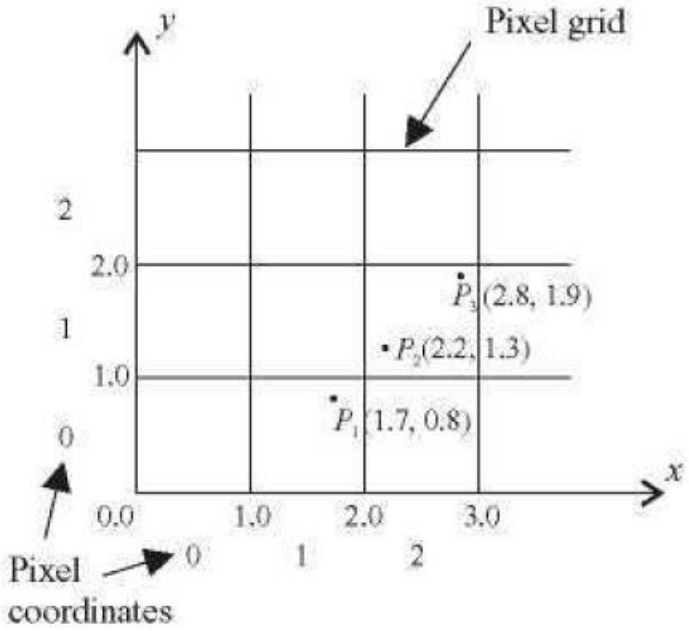
- Input: A mathematical point (x, y) where x and y are real numbers within an image area,
- Output: needs to be scan converted to a pixel at location (x',y').

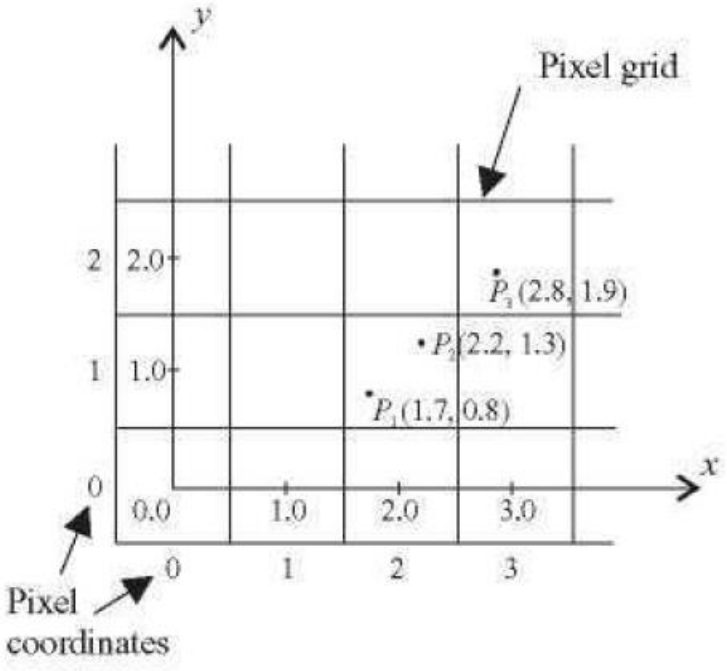
$$x' = Floor(x)$$

$$x' \leq x < x' + 1$$

$$y' = Floor(y)$$

$$y' \leq y < y' + 1$$





$$x' = Floor(x + 0.5)$$

$$x' - 0.5 \leq x < x' + 0.5$$

$$y' = Floor(y + 0.5)$$

$$y' - 0.5 \leq y < y' + 0.5$$

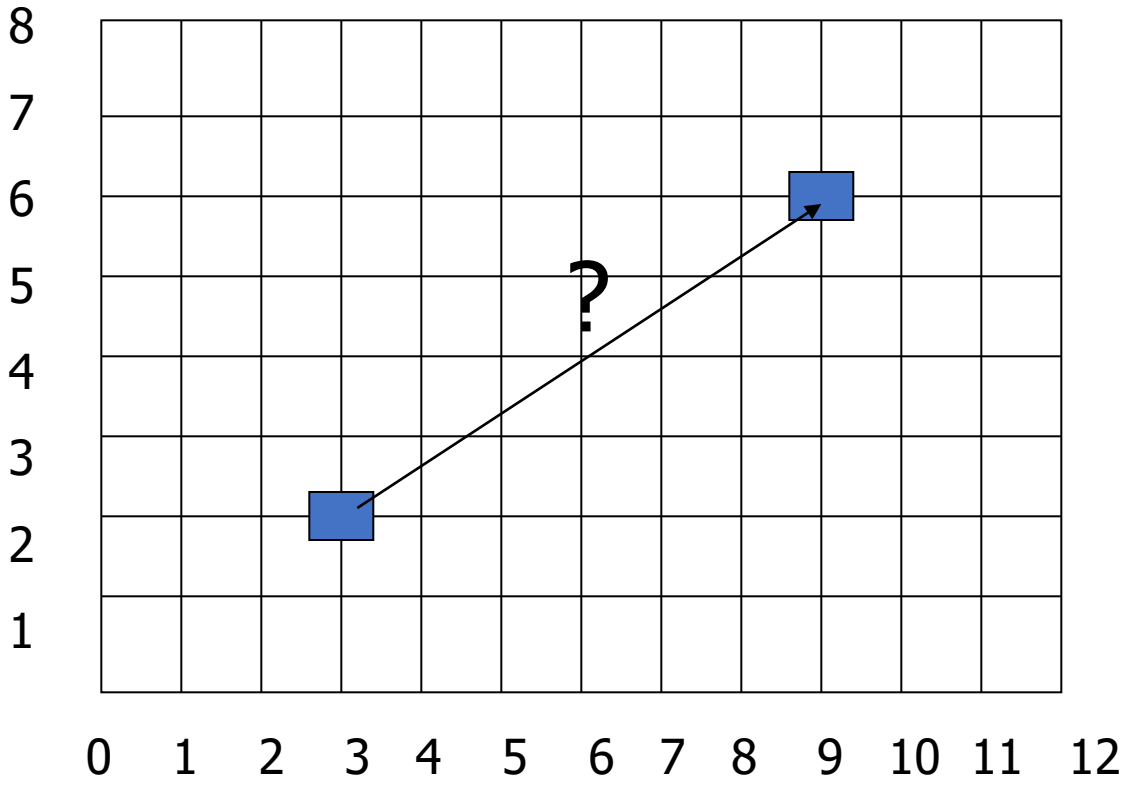
RASTERIZATION OPERATIONS

- Drawing lines on the screen
- Manipulating pixel maps (pixmaps): copying, scaling, rotating, etc.
- Compositing images, defining and modifying regions
- Drawing and filling polygons
- Aliasing and antialiasing methods

LINE DRAWING ALGORITHM

- Programmer specifies (x,y) values of end pixels
- 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. E.g. computed point
- (10.48, 20.51) rounded to (10, 21)
- Rounded pixel value is off actual line path (jaggy)

LINE DRAWING ALGORITHM

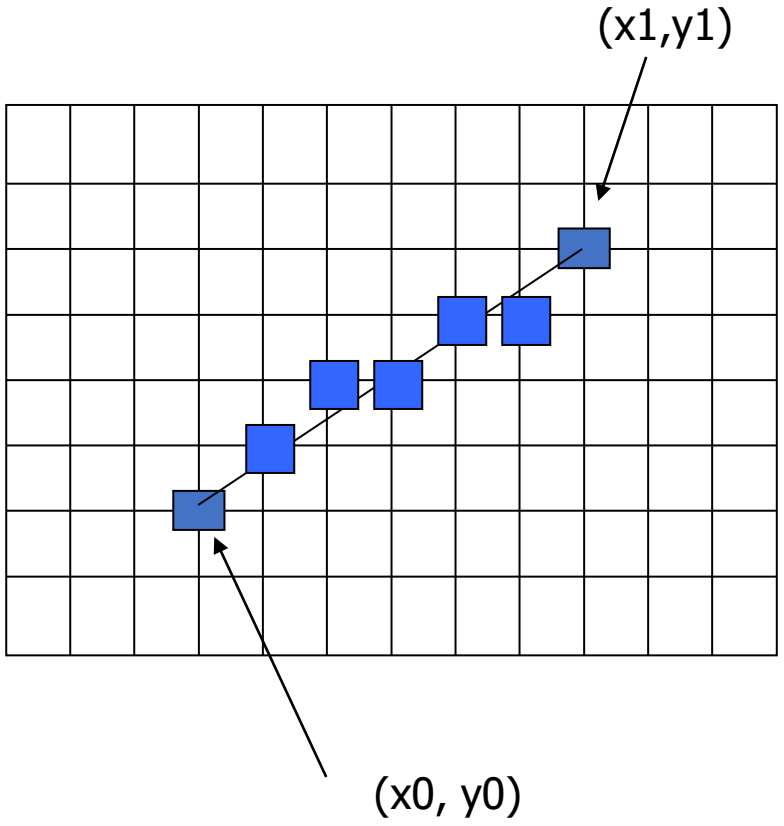


Line: (3,2) -> (9,6)

Which intermediate pixels to turn on?

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

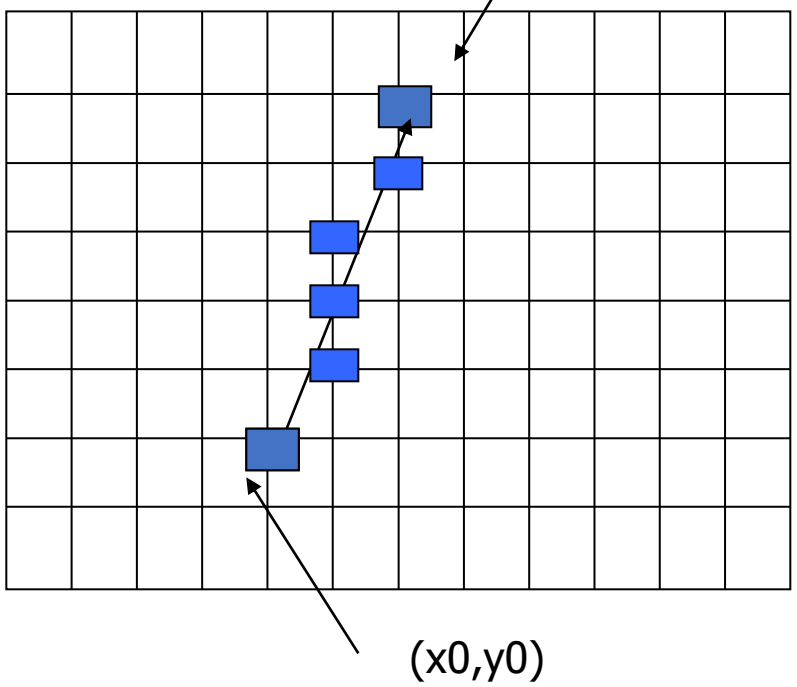
Illuminate pixel (x, round(y))

…

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)

...

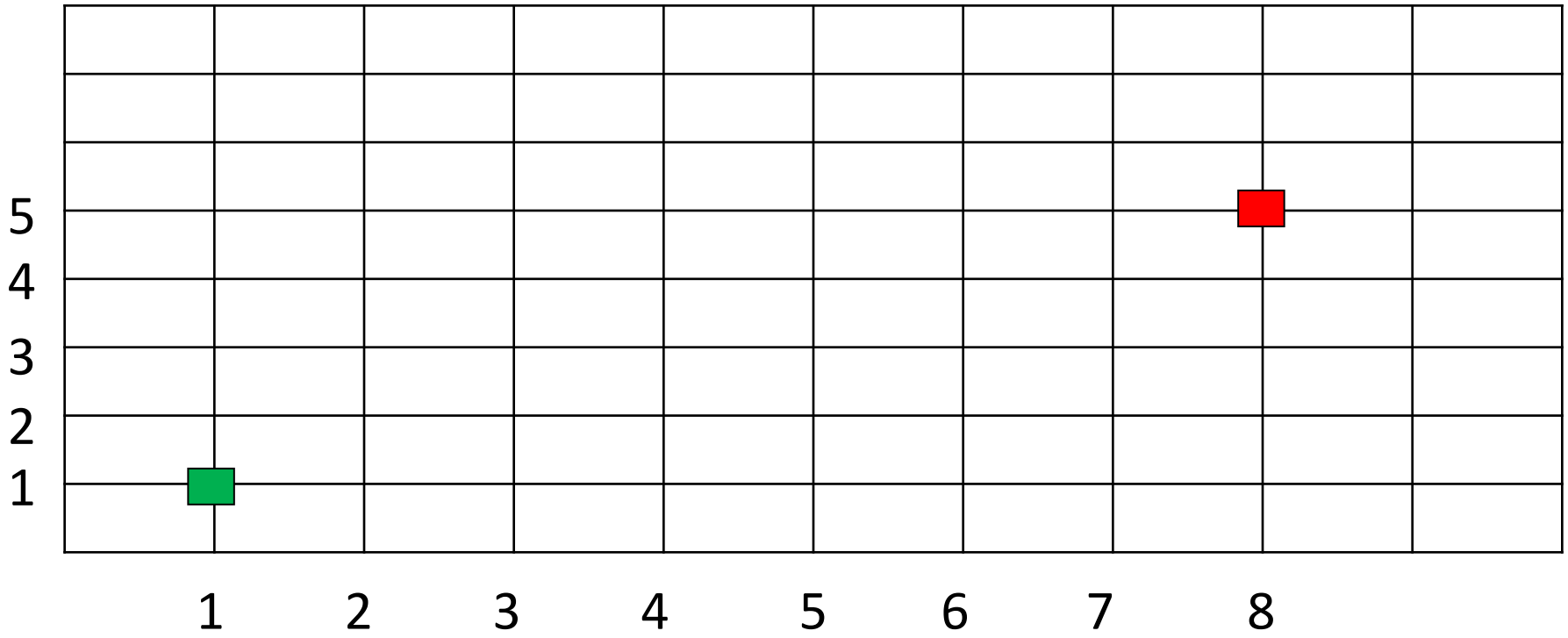
Until y == y1

DDA LINE DRAWING ALGORITHM PSEUDOCODE

```
dx = x2 – x1; dy = y2 – y1;

If (dx > dy)

   steps = dx;

Else

   steps = dy;

xi = dx/steps

yi = dy/steps

For i= 0 to steps+1

    setPixel(x1, y1)

   x1 = x1 + xi; y1 = y1 + yi;
```

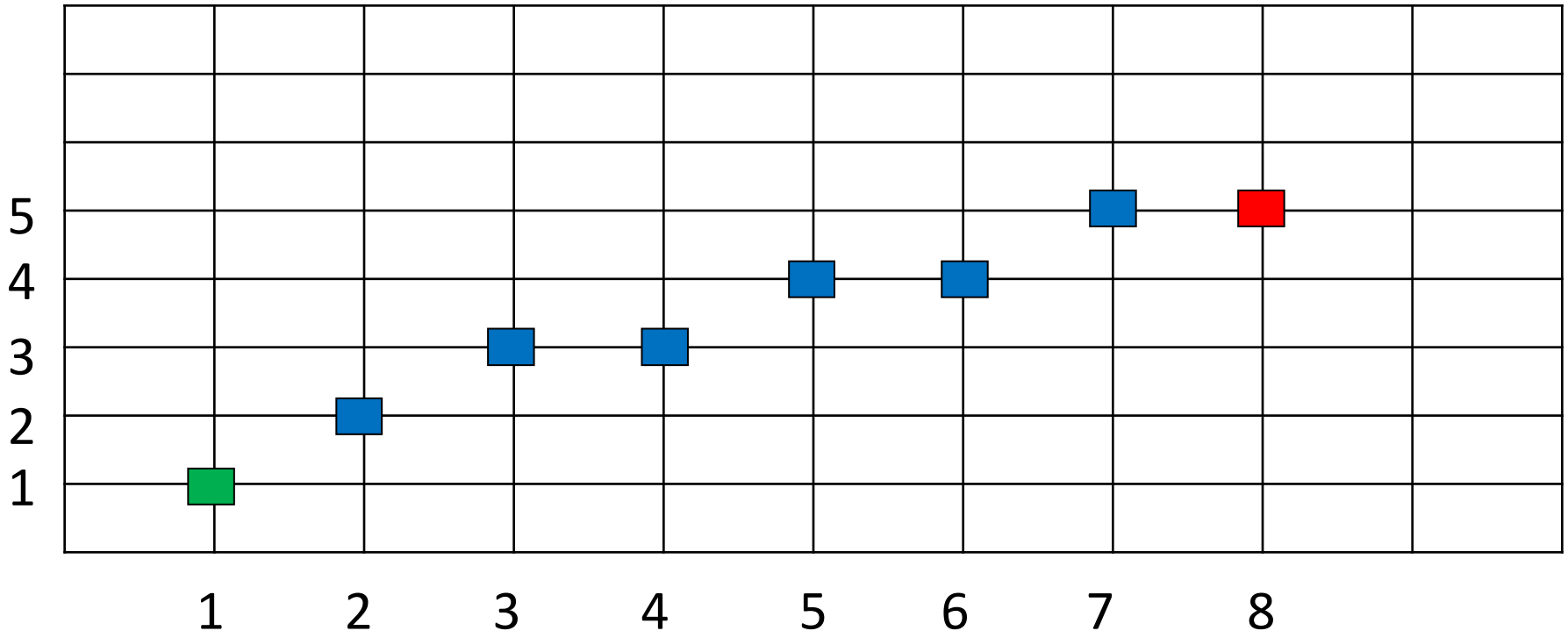
NUMERICAL EXAMPLE

- **Indicate which raster locations would be chosen by DDA algorithm when scan-converting a line from pixel coordinate (1,1) to pixel coordinate (8,5).**



NUMERICAL EXAMPLE

- **Indicate which raster locations would be chosen by DDA algorithm when scan-converting a line from pixel coordinate (1,1) to pixel coordinate (8,5).**



PYTHON CODE

- **Write a Python Code that Plots which raster locations would be chosen by DDA algorithm when scan-converting a line from pixel coordinate (1,1) to pixel coordinate (8,5).**

LINE DRAWING ALGORITHM DRAWBACKS

- DDA is the simplest line drawing algorithm
  - Not very efficient (time consuming)
  - Round operation is expensive

- Optimized algorithms typically used.
  - Bresenham algorithm

- Bresenham algorithm
  - Incremental algorithm: current value uses previous value
  - Integers only: avoid floating point arithmetic
  - **Midpoint version of algorithm**

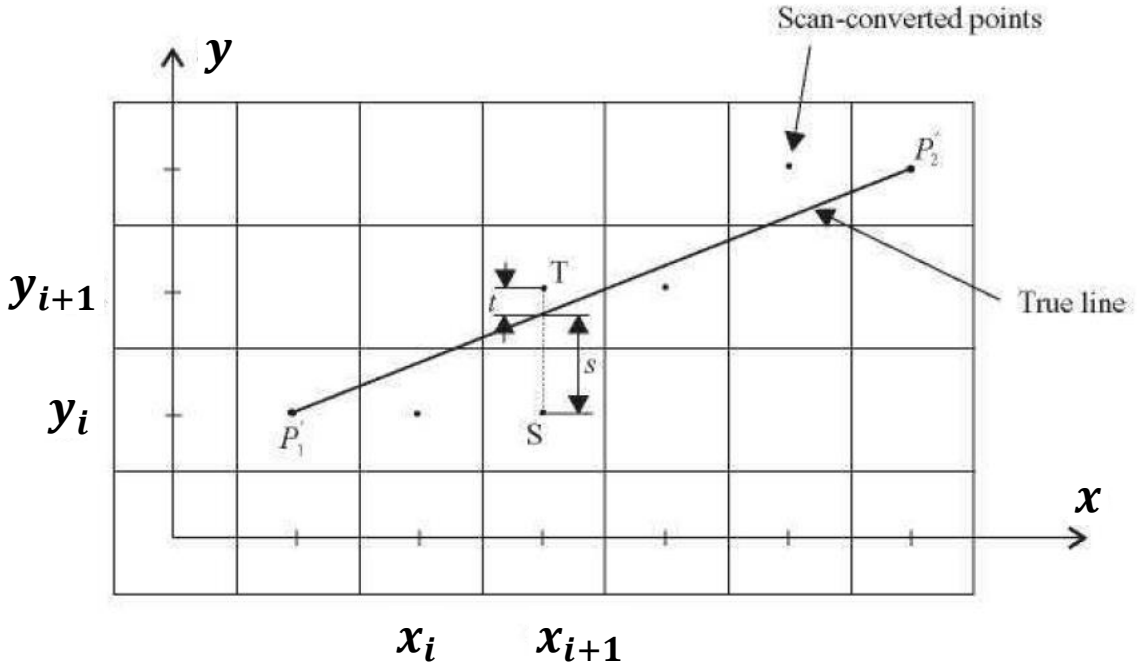
BRESENHAM'S LINE-DRAWING ALGORITHM

- Problem: Given endpoints (x_0, y_0) and (x_1, y_1) of a line

- Task is to determine **best** sequence of intervening pixels

- First make two simplifying assumptions (remove later):
  - $(0 < m < 1)$

BRESENHAM'S LINE-DRAWING ALGORITHM (DERIVATION)

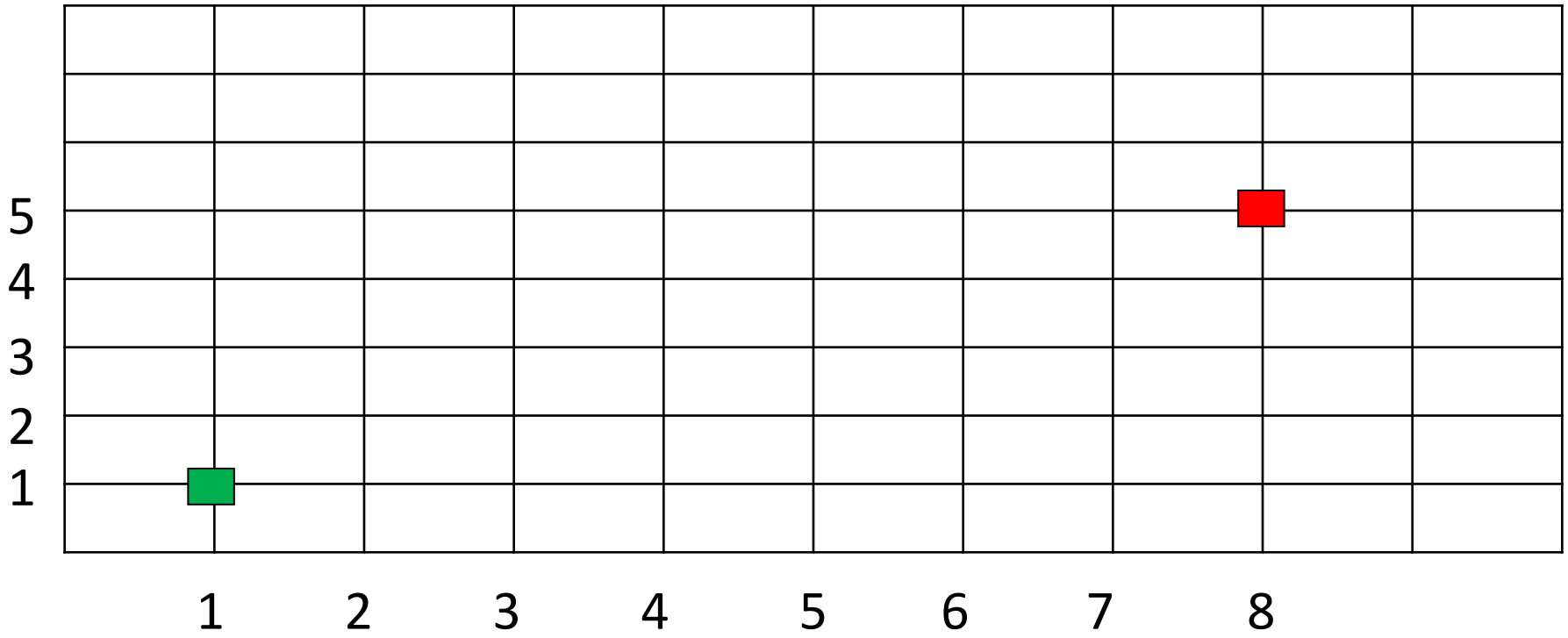


DERIVATION OF BRESENHAM LINE DRAWING ALGORITHM

```
int x = x'_1, y = y'_1;
int dx = x'_2 − x'_1, dy = y'_2 − y'_1, dT = 2(dy − dx), dS = 2dy;
int d = 2dy − dx;
setPixel(x, y);
while (x < x'_2) {
    x++;
    if (d < 0)
        d = d + dS;
    else {
        y++;
        d = d + dT;
    }
    setPixel(x, y);
}
```

NUMERICAL EXAMPLE

- **Indicate which raster locations would be chosen by the Bresenham's Line Drawing algorithm when scan-converting a line from pixel coordinate (1,1) to pixel coordinate (8,5).**



PYTHON CODE

- **Write a Python Code that Plots which raster locations would be chosen by by the Bresenham's Line Drawing algorithm when scan-converting a line from pixel coordinate (1,1) to pixel coordinate (8,5).**

FLAME UNIVERSITY | EVERLASTING learning

THANK YOU