Output Primitives- DDA Algorithm

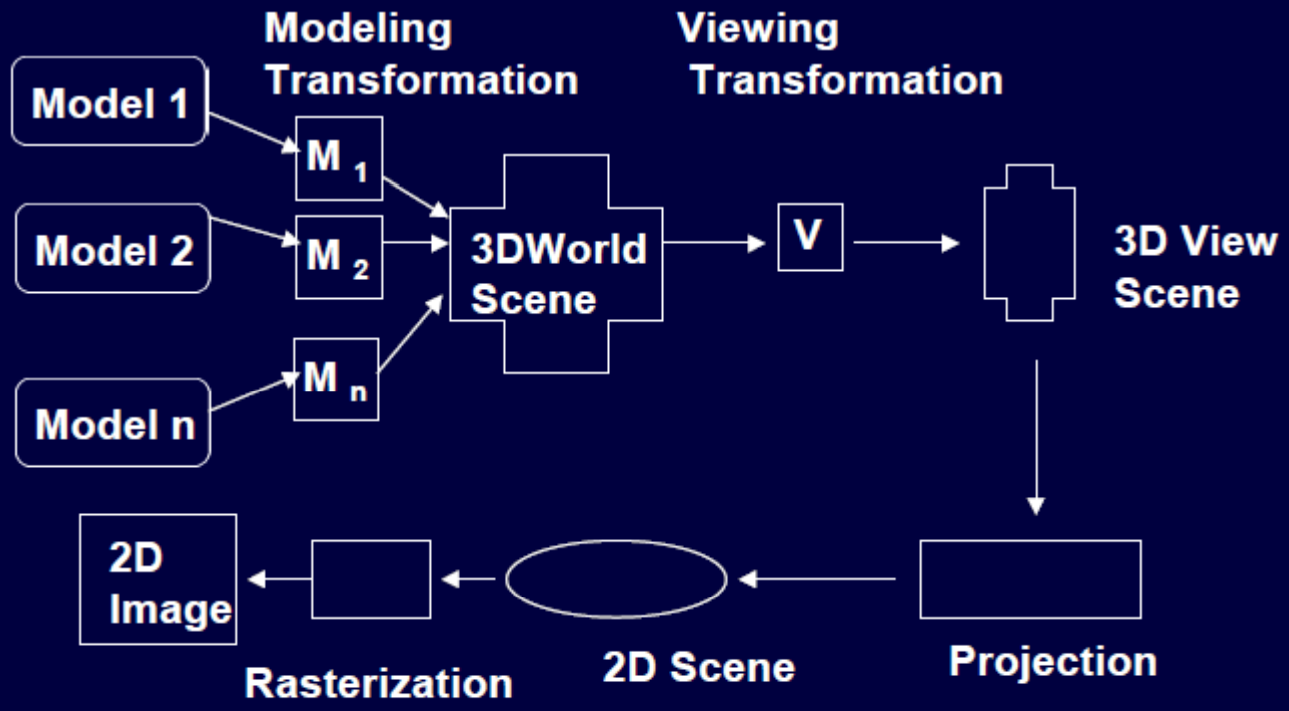
Nancy Victor
Assistant Professor
SITE, VIT,
Vellore

GRAPHICS RENDERING PIPELINE

- Rendering is the conversion of a scene into an image.

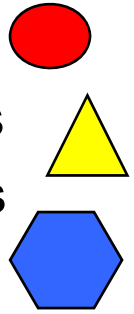
3D Scene  2D Image

Graphics Rendering Pipeline



RASTERIZATION

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



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
  - Previously glBegin(GL_POLYGON), etc
- Aliasing and antialiasing methods

OUTPUT PRIMITIVES

- Output Primitives are basic geometric structures which describes a scene or a picture.
- They are the building blocks of a picture.
  - E.g.: point, line, curves, fillcolor etc..
- Attributes:
  - Properties of output primitives.
  - An attribute describes how a particular primitive is to be displayed.
  - E.g.: intensity, line styles, text styles etc..

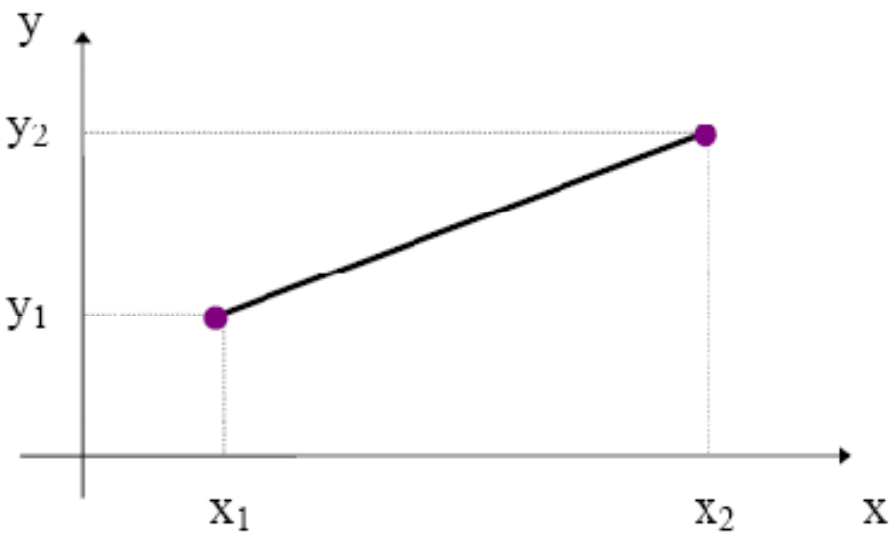
OUTPUT PRIMITIVES

- In order to draw the primitive objects, one has to first scan convert the object.
- Scan convert refers to the operation of finding out the location of pixels to the intensified and then setting the values of corresponding bits, in the graphic memory, to the desired intensity code.
- Each pixel on the display surface has a finite size depending on the screen resolution and hence a pixel cannot represent a single mathematical point.
- A point is shown by illuminating a pixel on the screen.

LINE DRAWING ALGORITHMS

- A line segment is completely defined in terms of its two endpoints.
- A line segment is thus defined as:

  Line_Seg = { $(x_1, y_1), (x_2, y_2)$ }



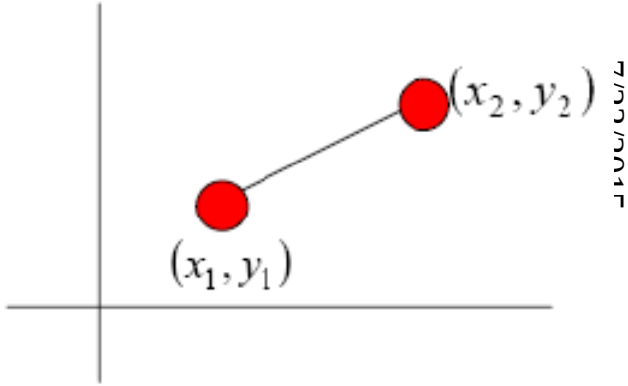
LINE DRAWING ALGORITHMS

- A line is produced by means of illuminating a set of intermediary pixels between the two endpoints.
- Lines is digitized into a set of discrete integer positions that approximate the actual line path.
- Example: A computed line position of (10.48, 20.51) is converted to pixel position (10, 21).
- The rounding of coordinate values to integer causes all but horizonatal and vertical lines to be displayed with a stair step appearance “the jaggies”.

LINE DRAWING ALGORITHMS

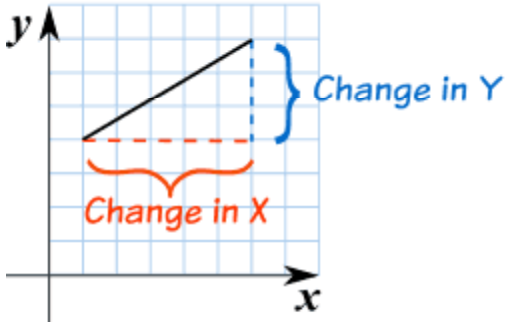
- Given Points (x_1, y_1) and (x_2, y_2)
- Slope-intercept line equation

  $y = mx + b$
- Slope, $m = y_2 - y_1 / x_2 - x_1 = \Delta y / \Delta x$
- y-intercept $b = y_1 - m.x_1$
- x-interval$\rightarrow \Delta x = \Delta y / m$
- y-interval$\rightarrow \Delta y = m\ \Delta x$



LINE DRAWING ALGORITHMS

- Slope:
- The Slope (also called Gradient) of a straight line shows **how steep** a straight line is.
- The method to calculate the slope is:
- Divide the **change in height** by the **change in horizontal distance**
- **Slope** = Change in Y /Change in X

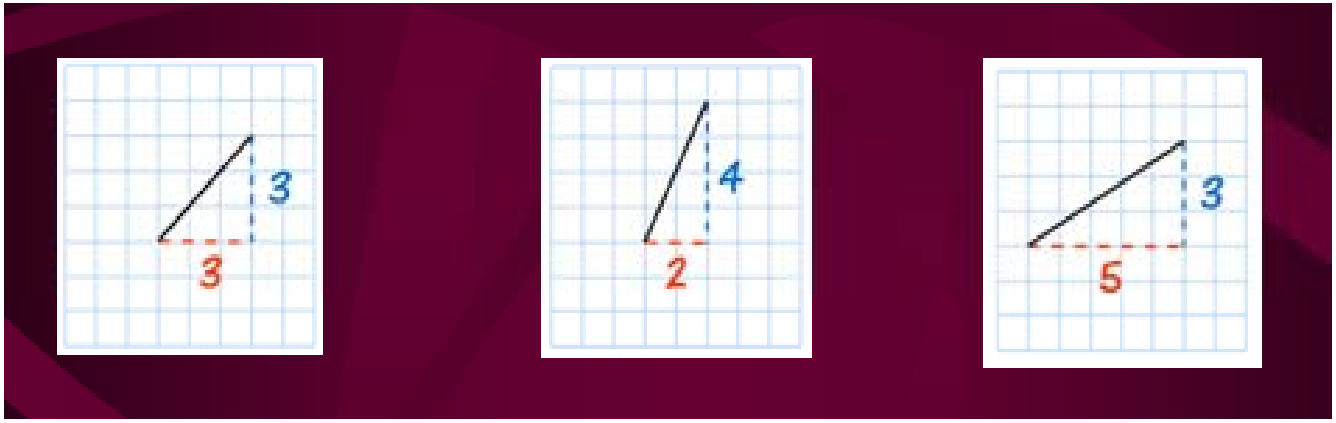


LINE DRAWING ALGORITHMS

- Slope=3/3=1
- Slope=4/2=2

  The line is steeper, and so the slope is larger
- Slope=3/5=0.6

  The line is less steep, and so the slope is smaller



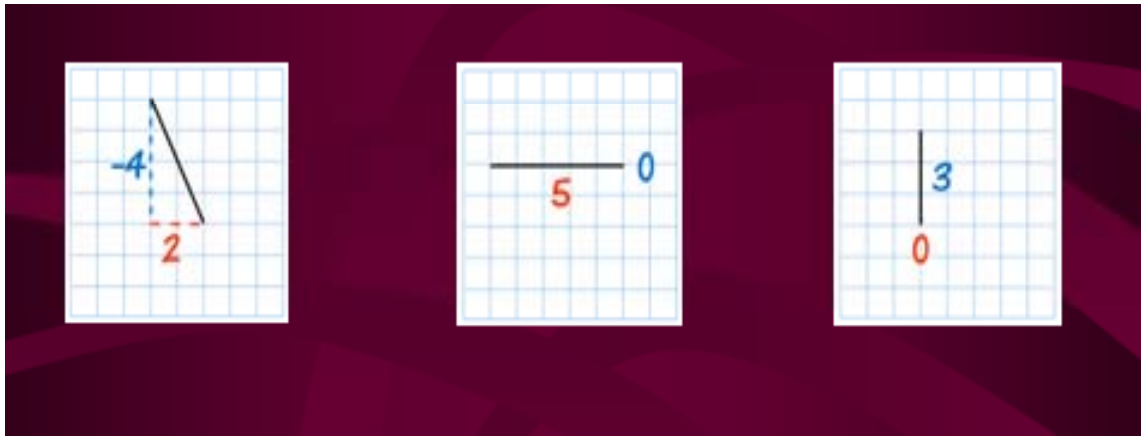
LINE DRAWING ALGORITHMS

- Slope: Positive or Negative???
- Slope=-4/2=-2

  That line goes **down** as you move along, so it has a negative slope.
- Slope=0/5=0

  A line that goes straight across has a slope of zero.
- Slope=3/0

  A "straight up and down" line's slope is "undefined".

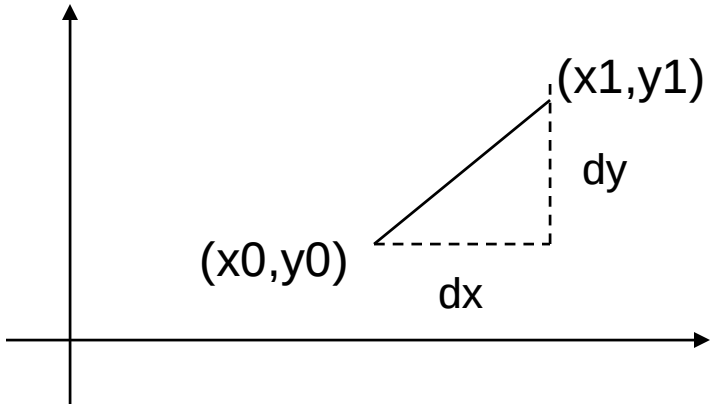


LINE DRAWING ALGORITHMS

- The direction of a line is either increasing, decreasing, horizontal or vertical. A line is **increasing** if it goes **up** towards the right. The slope is **positive**, i.e. .
- A line is **decreasing** if it goes **down** towards the right. The slope is **negative**, i.e. .
- If a line is horizontal the slope is **zero**. This is a constant function.
- If a line is vertical the slope is undefined

DDA ALGORITHM

- Digital Differential Analyzer
- 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 ALGORITHM

- A line algorithm based on calculating either Δy or Δx using the above equations.
- There are two cases:
  - Positive slope
  - Negative slope

DDA ALGORITHM (POSITIVE SLOPE)

If $m \leq 1$ then take $\Delta x = 1$

- Compute successive y by

$$y_{k+1} = y_k + m \qquad (1)$$

- Subscript k takes integer values starting from 1, for the first point, and increases by 1 until the final end point is reached.

If $m > 1$, reverse the role of x and y and take $\Delta y = 1$, calculate successive x from

$$x_{k+1} = x_k + 1/m \qquad (2)$$

- In this case, each computed x value is rounded to the nearest integer pixel position.
- The above equations are based on the assumption that lines are to be processed from left endpoint to right endpoint.

DDA ALGORITHM (NEGATIVE SLOPE)

- In case the line is processed from Right endpoint to Left endpoint, then

$$\Delta x = -1,\ y_{k+1} = y_k - m \quad \text{for} \quad m \le 1 \qquad (3)$$

or

$$\Delta y = -1,\ x_{k+1} = x_k - 1/m \quad \text{for} \quad m > 1 \qquad (4)$$

DDA ALGORITHM

- If $m < 1$,
  - use(1) [provided line is calculated from left to right] and
  - use(3) [provided line is calculated from right to left].
- If $m \geq 1$
  - use (2) or (4).

```
Procedure lineDDA(xa,ya,xb,yb:integer);
Var
   dx,dy,steps,k:integer
   xIncrement,yIncrement,x,y:real;
begin
   dx:=xb-xa;
   dy:=yb-ya;
   if abs(dx)>abs(dy) then steps:=abs(dx)
   else steps:=abs(dy);
   xIncrement:=dx/steps;
   yIncrement:=dy/steps;
   x:=xa;
   y:=ya;
   setPixel(round(x),round(y),1);
   for k:=1 to steps do
         begin
               x:=x+xIncrement;
               y:=y+yIncrement;
               setPixel(round(x),round(y),1)
         end
end; {lineDDA}
```

DDA ALGORITHM

- Suppose we want to draw a line starting at pixel (2,3) and ending at pixel (12,8).
- numsteps = 12 – 2 = 10
- xinc = 10/10 = 1.0
- yinc = 5/10 = 0.5

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
  - Round operation is expensive
- Optimized algorithms typically used.
  - Integer DDA
  - E.g.Bresenham algorithm
- Bresenham algorithm
  - Incremental algorithm: current value uses previous value
  - Integers only: avoid floating point arithmetic

Thank You…