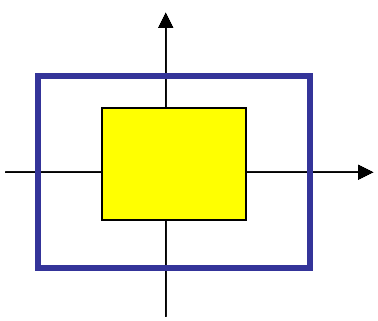
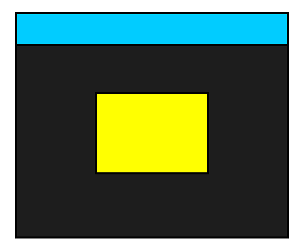
Okay, you have learned ...

- OpenGL drawing
- Viewport and World Window setup

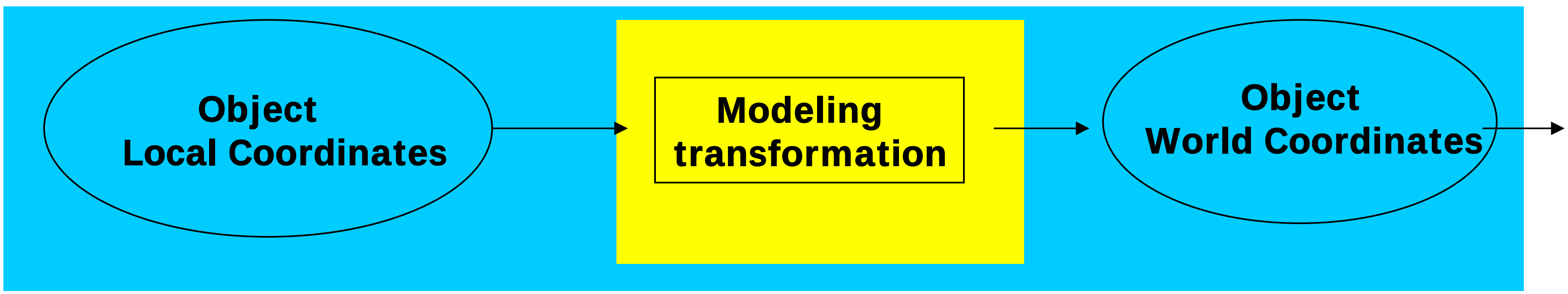


```
main()
{
   glViewport(0,0,300,200);
   glMatrixMode(GL_PROJECTION);
   glLoadIndentity();
   gluOrtho2D(-1,1,-1,1);
   glBegin(GL_QUADS);
   glColor3f(1,1,0);
   glVertex2i(-0.5,-0.5);
   glVertex2i(+0.5,0);
   glVertex2i(+0.5,+0.5);
   glVertex2i(-0.5,+0.5);
   glEnd();
}
```



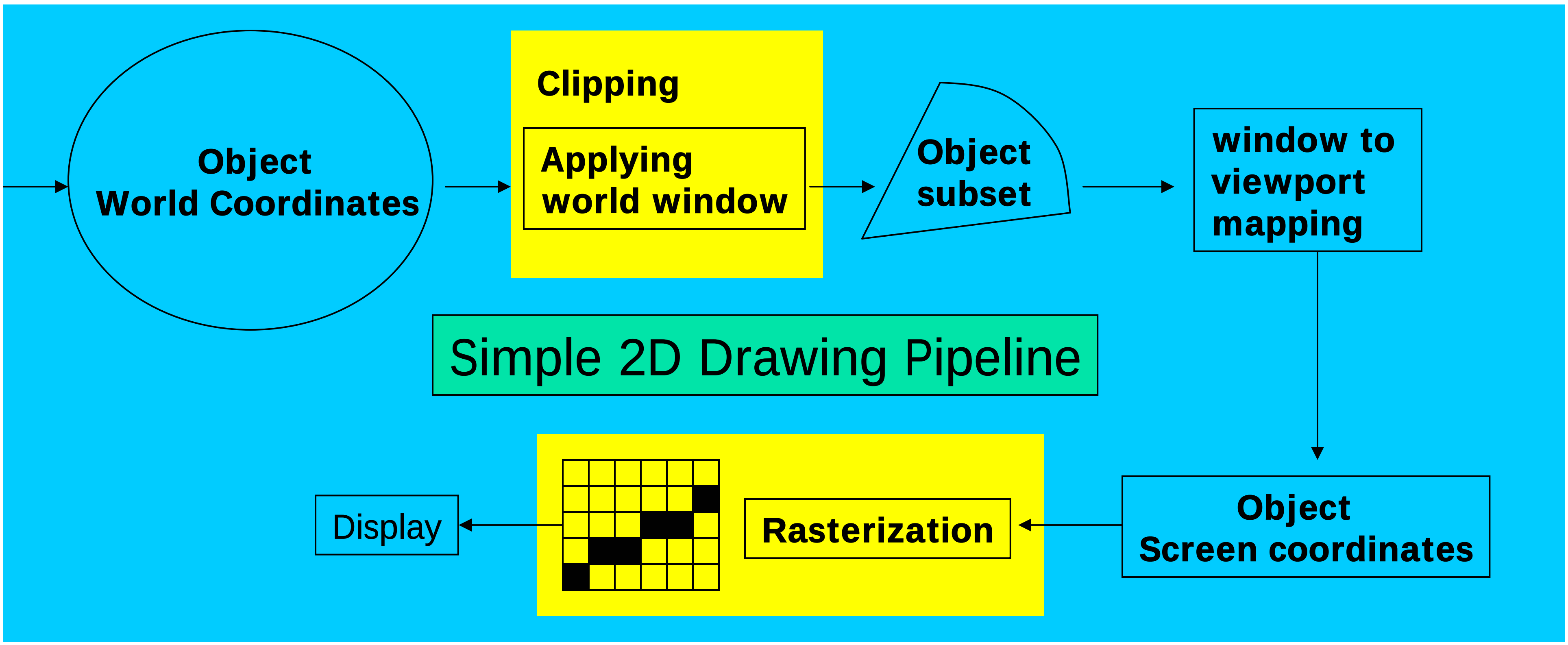
2D Graphics Pipeline

Graphics processing consists of many stages:



(next page)

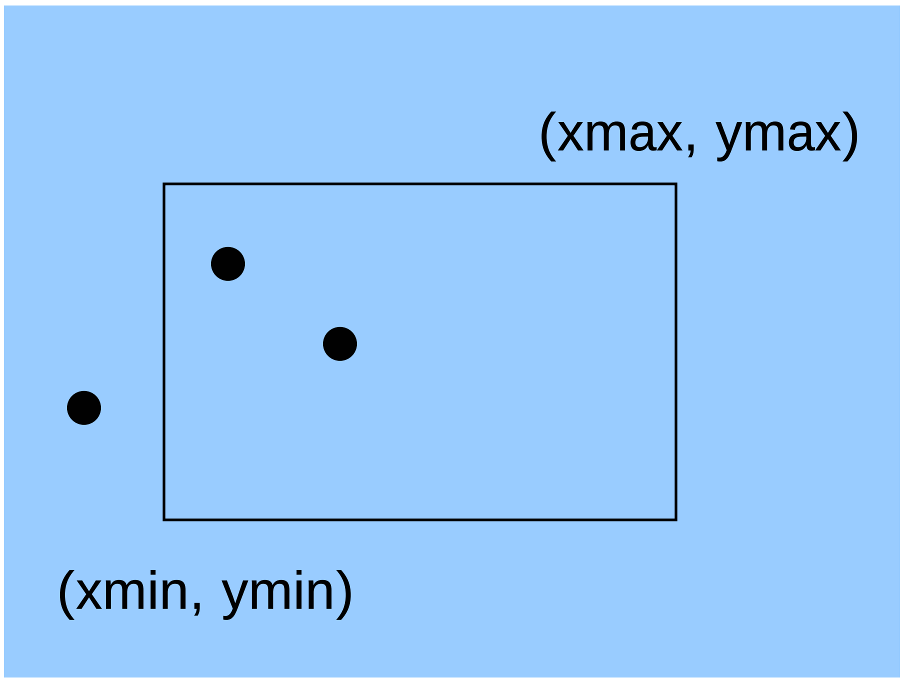
2D Graphics Pipeline (2)



Clipping and Rasterization

- OpenGL does these for you – no explicit OpenGL functions needed for doing clipping and rasterization
- Clipping – Remove objects that are outside the world window
- Rasterization (scan conversion) – Convert high level object descriptions to pixel colors in the frame buffer

2D Point Clipping

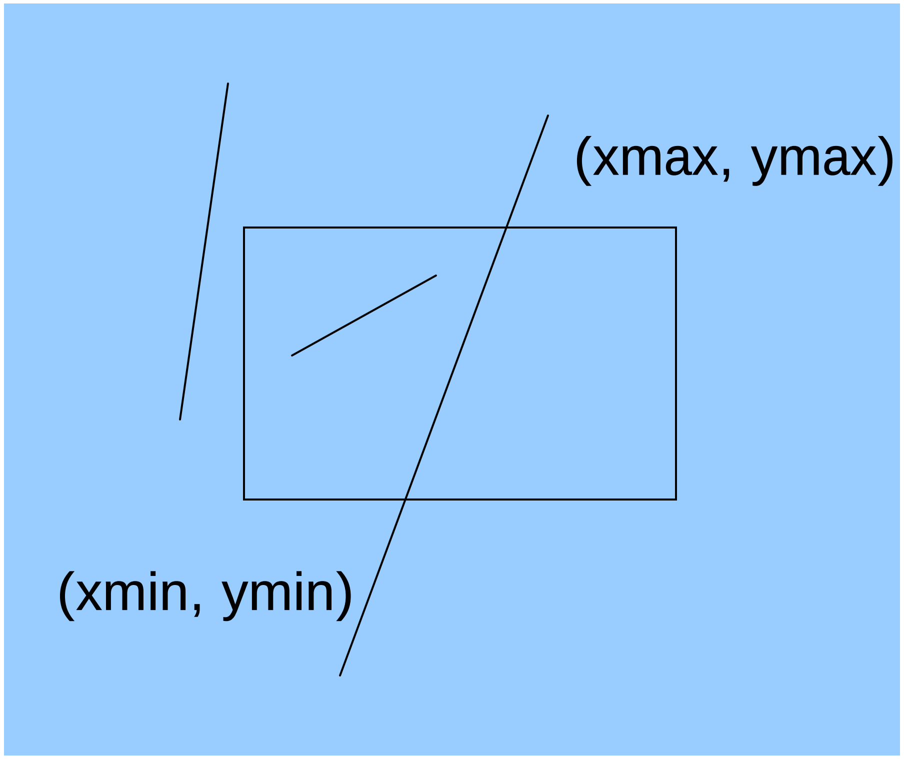


- Determine whether a point (x,y) is inside or outside of the world window?

If (xmin <= x <= xmax)
and (ymin <= y <= ymax)

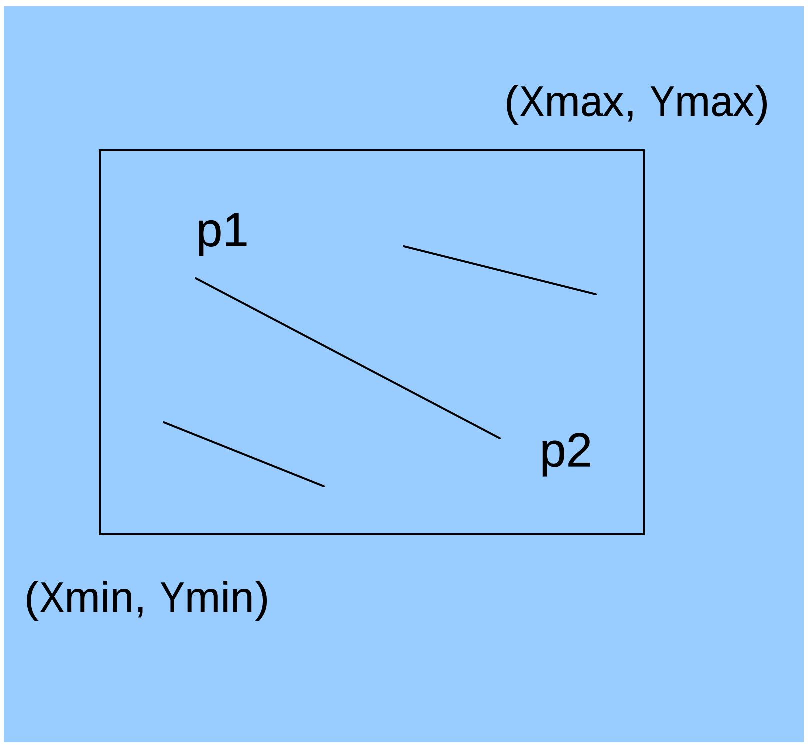
then the point (x,y) is inside
else the point is outside

2D Line Clipping



- Determine whether a line is inside, outside, or partially inside
- If a line is partially inside, we need to display the inside segment

Trivial Accept Case

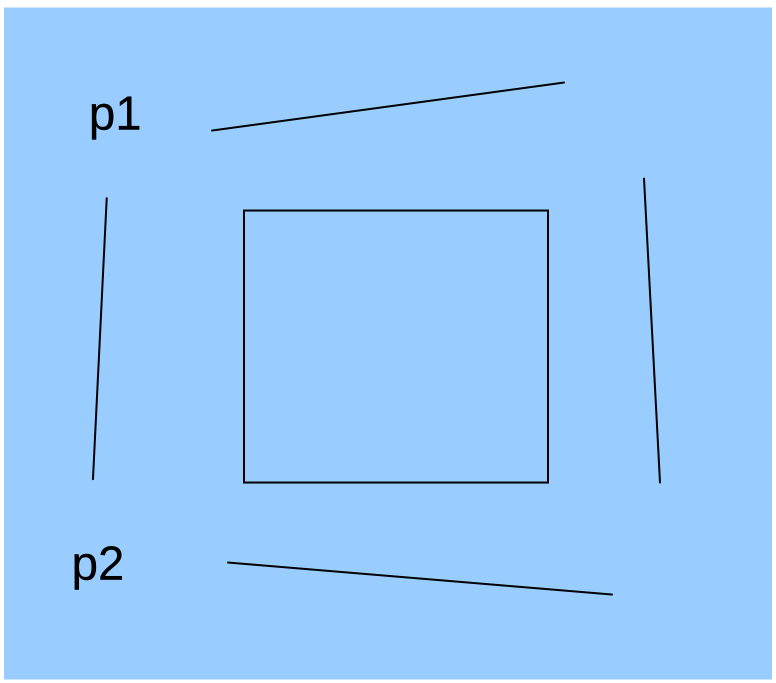


- Lines that are clearly inside the world window - what are they?

Xmin <= P1.x, P2.x <=xmax

Ymin <=P1.y, P2.y <= Ymax

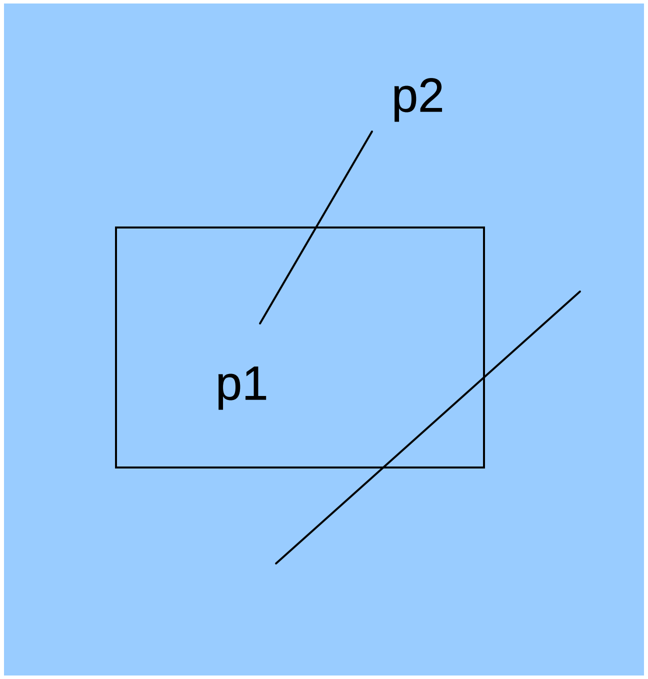
Trivial Reject Case



- Lines that are clearly outside the world window – what are they?

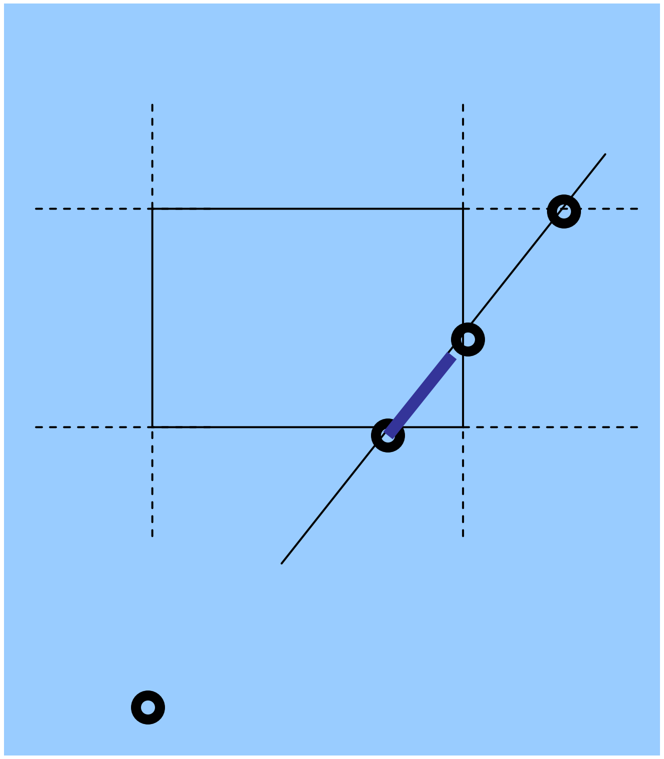
  - p1.x, p2.x <= Xmin OR
  - p1.x, p2.x >= Xmax OR
  - p1.y, p2.y <= ymin OR
  - p1.y, p2.y >= ymax

Non-Trivial Cases



- Lines that cannot be trivially rejected or accepted
  - One point inside, one point outside
  - Both points are outside, but not “trivially” outside
- Need to find the line segments that are inside

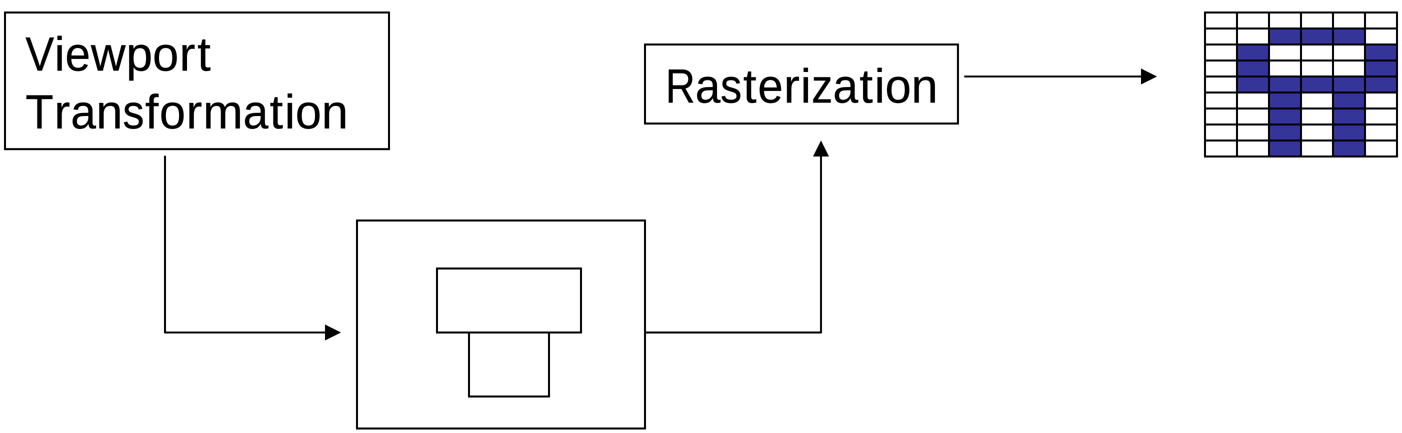
Non-trivial case clipping



- Compute the line/window boundary edges intersection
- There will be four intersections, but only one or two are on the window edges
- These two points are the end points of the desired line segment

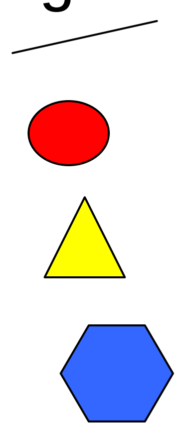
Rasterization (Scan Conversion)

- Convert high-level geometry description to pixel colors in the frame buffer



Rasterization Algorithms

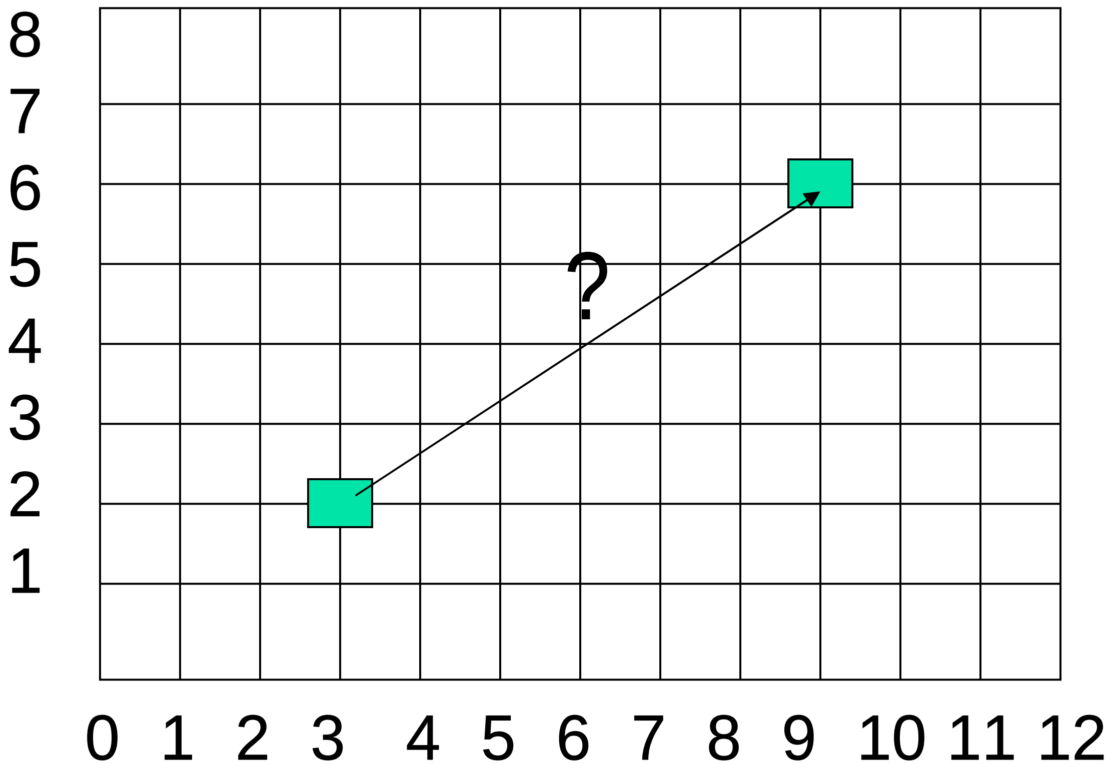
- A fundamental computer graphics function
- Determine the pixels' colors, illuminations, textures, etc.
- Implemented by graphics hardware
- Rasterization algorithms
  - Lines
  - Circles
  - Triangles
  - Polygons



Rasterize Lines

- Why learn this?
  - Understand the discrete nature of computer graphics
  - Write pure device independent graphics programs (Palm graphics)
  - Become a graphics system developer

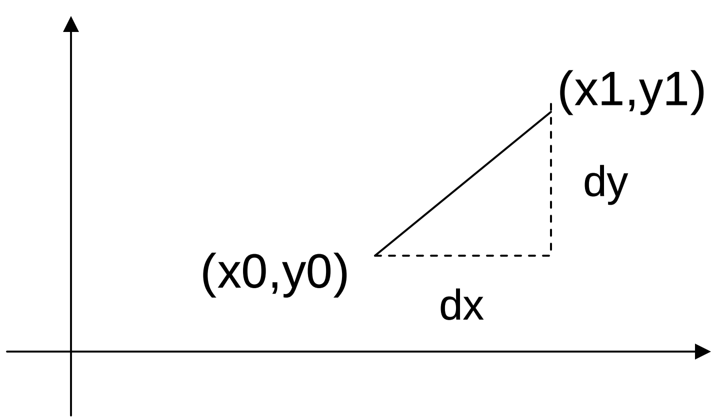
Line Drawing Algorithm (1)



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

Line Drawing Algorithm (2)

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



$$m = (y1-y0) \;/\; (x1 - x0)$$
$$= dy \;/\; dx$$

$$b = y1 - m * x1$$

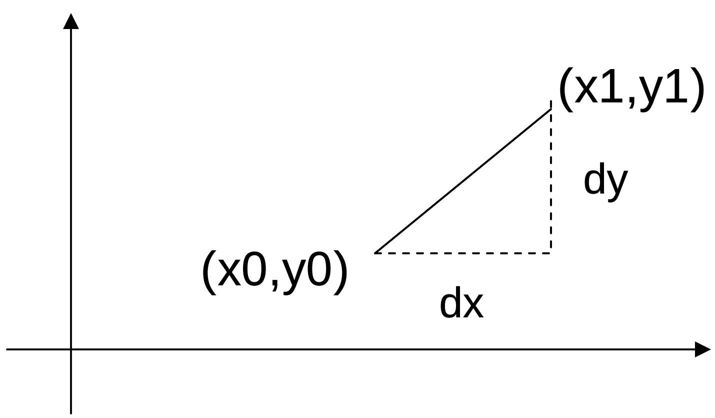
Line Drawing Algorithm (3)

Given the line equation $y = mx + b$, and end points (x0,y0) (x1, y1)

Walk through the line: starting at (x0,y0)

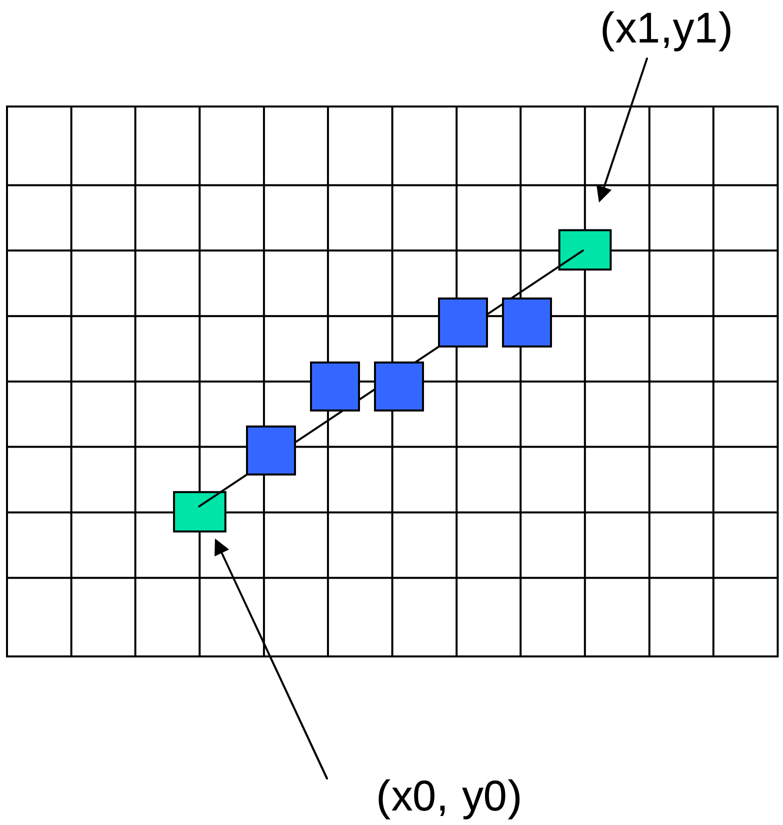
If we choose the next point in the line as $X = x0 + \Delta x$

$Y = ?$



$$
\begin{aligned}
Y &= y0 + \Delta x * m \\
&= y0 + \Delta x * (dy/dx)
\end{aligned}
$$

Line Drawing Algorithm (4)



X = x0 Y = y0

Illuminate pixel (x, int(Y))

X = x0 + 1 Y = y0 + 1 * m

Illuminate pixel (x, int(Y))

X = X + 1 Y = Y + 1 * m

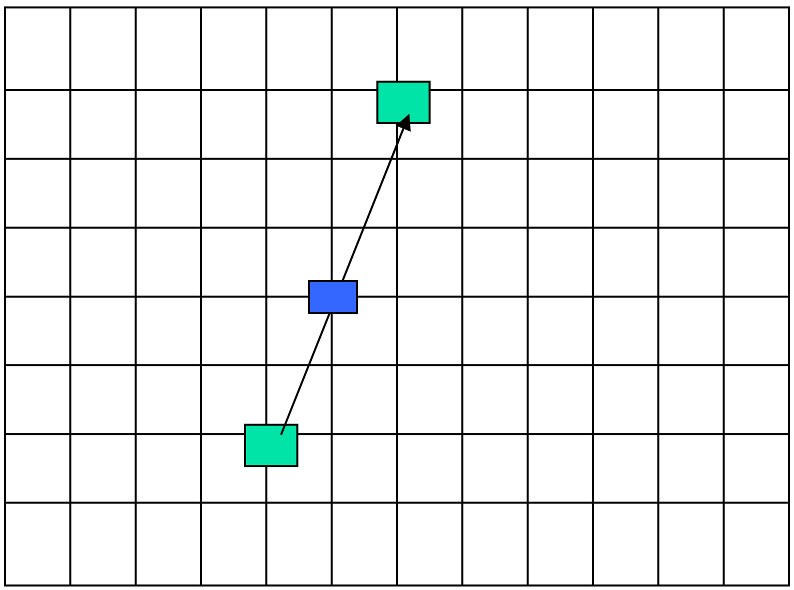
Illuminate pixel (x, int(Y))

...

Until X == x1

Line Drawing Algorithm (5)

- How about a line like this?



Can we still increment X by 1 at each Step?

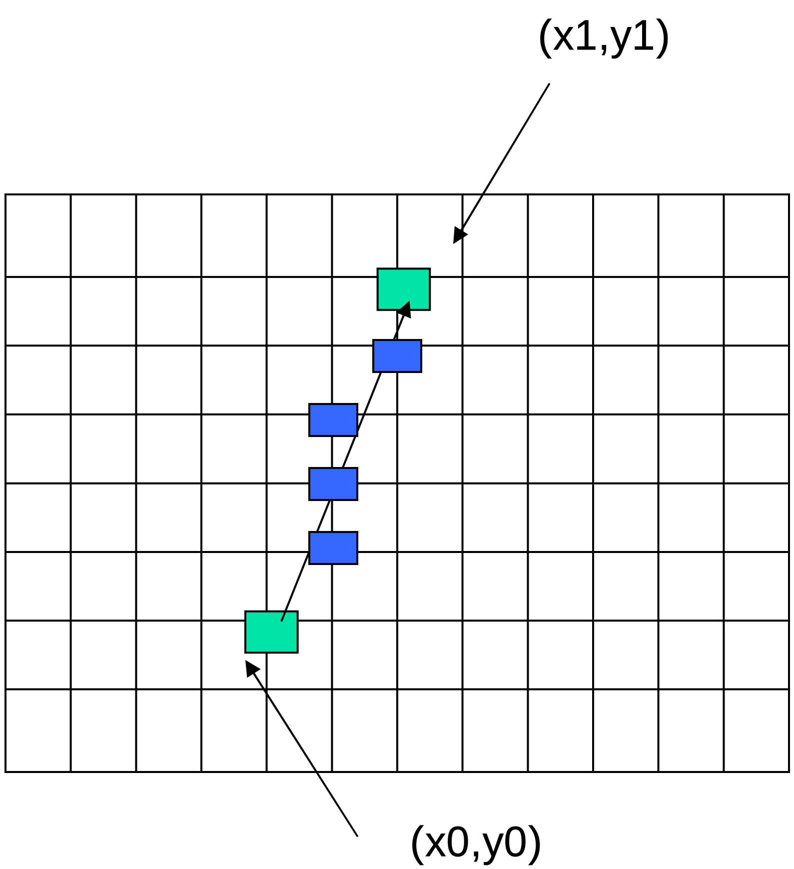
The answer is No. Why?

We don't get enough samples

How to fix it ?

Increment Y

Line Drawing Algorihtm (6)



X = x0 Y = y0

Illuminate pixel (x, int(Y))

Y = y0 + 1 X = x0 + 1 * 1/m

Illuminate pixel (x, int(Y))

Y = Y + 1 X = X + 1 /m

Illuminate pixel (x, int(Y))

...

Until Y == y1

Line Drawing Algorithm (7)

- The above is the simplest line drawing algorithm
- Not very efficient
- Optimized algorithms such integer DDA and Bresenhan algorithm (section 8.10) are typically used
- Not the focus of this course