Okay, you have learned …

- OpenGL drawing
- Viewport and World Window setup

```
main()
{
    glViewport(0,0,300,200);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    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:

- Object
- Local Coordinates
- Modeling transformation
- Object
- World Coordinates

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 \((\text{xmin} \leq x \leq \text{xmax})\) and \((\text{ymin} \leq y \leq \text{ymax})\) 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?

- \(\text{Xmin} \leq P1.x, P2.x \leq \text{Xmax}\)
- \(\text{Ymin} \leq P1.y, P2.y \leq \text{Ymax}\)

Trivial Reject Case

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

- \(P1.x, P2.x \leq \text{Xmin}\) OR
- \(P1.x, P2.x \geq \text{Xmax}\) OR
- \(P1.y, P2.y \leq \text{Ymin}\) OR
- \(P1.y, P2.y \geq \text{Ymax}\)
**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

**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)**

![Graph showing a line from (3,2) to (9,6)]

**Line Drawing Algorithm (2)**

- **Slope-intercept line equation**
  
  \[ Y = mx + b \]

  - Given two end points \((x_0, y_0), (x_1, y_1)\), how to compute \(m\) and \(b\)?

  \[ m = \frac{y_1 - y_0}{x_1 - x_0} \]
  
  \[ = \frac{\Delta y}{\Delta x} \]

  \[ b = y_1 - m \cdot x_1 \]

**Line Drawing Algorithm (3)**

Given the line equation \(y = mx + b\), and end points \((x_0, y_0)\), \((x_1, y_1)\)

Walk through the line: starting at \((x_0, y_0)\)

If we choose the next point in the line as \(X = x_0 + \Delta x\)

\[ Y = ? \]

\[ Y = y_0 + \Delta x \cdot m \]

\[ = y_0 + \Delta x \cdot \frac{\Delta y}{\Delta x} \]

**Line Drawing Algorithm (4)**

\[ \begin{align*}
  X &= x_0 \\
  Y &= y_0 \\
  \text{Illuminate pixel (x, int(Y))}
\end{align*} \]

\[ \begin{align*}
  X &= x_0 + 1 \\
  Y &= y_0 + 1 \cdot m \\
  \text{Illuminate pixel (x, int(Y))}
\end{align*} \]

\[ \begin{align*}
  X &= X + 1 \\
  Y &= Y + 1 \cdot m \\
  \text{Illuminate pixel (x, int(Y))}
\end{align*} \]

... 

Until \(X == x_1\)
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

The above is the simplest line drawing algorithm

Not very efficient

Optimized algorithms such integer DDA and Bresenhan algorithm are typically used

Not the focus of this course
Triangle Rasterization Issues

- Moving Slivers

- Shared Edge Ordering

Edge Equations

- An edge equation is simply the equation of the line defining that edge:
  - Q: *What is the implicit equation of a line?*
  - A: \( Ax + By + C = 0 \)
  - Q: *Given a point \((x, y)\), what does plugging \(x\) & \(y\) into this equation tell us?*
  - A: Whether the point is:
    - On the line: \( Ax + By + C = 0 \)
    - “Above” the line: \( Ax + By + C > 0 \)
    - “Below” the line: \( Ax + By + C < 0 \)

Edge Equations

- Edge equations thus define two half-spaces:
  \[ Ax + By + C > 0 \]
  \[ Ax + By + C = 0 \]
  \[ Ax + By + C < 0 \]
And a triangle can be defined as the intersection of three positive half-spaces:

So... simply turn on those pixels for which all edge equations evaluate to > 0: