Okay, you have learned …

- OpenGL drawing
- Viewport and World Window setup

```c
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
2D Graphics Pipeline (2)

Simple 2D Drawing Pipeline

- Object World Coordinates
- Clipping
  - Applying world window
  - Object subset
- window to viewport mapping
- Display
- Rasterization
- Object Screen 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 \((xmin \leq x \leq xmax)\)
and \((ymin \leq y \leq ymax)\)
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?

\[ X_{\text{min}} \leq P1.x, \ P2.x \leq x_{\text{max}} \]

\[ Y_{\text{min}} \leq P1.y, \ P2.y \leq y_{\text{max}} \]
Trivial Reject Case

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

- $p1.x, p2.x \leq X_{\text{min}} \text{ OR}$
- $p1.x, p2.x \geq X_{\text{max}} \text{ OR}$
- $p1.y, p2.y \leq y_{\text{min}} \text{ OR}$
- $p1.y, p2.y \geq y_{\text{max}}$
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
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)
Slope-intercept line equation

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

\[
m = \frac{y_1 - y_0}{x_1 - x_0} = \frac{\text{dy}}{\text{dx}}
\]

\[
b = y_1 - m \times x_1
\]
Given the line equation $y = mx + b$, and end points $(x_0,y_0)$ and $(x_1, y_1)$

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

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

Then:

$Y = ?$

$$Y = y_0 + \Delta x \cdot m$$

$$= y_0 + \Delta x \cdot \frac{dy}{dx}$$
Line Drawing Algorithm (4)

\[(x_0, y_0)\]
\[X = x_0 \quad Y = y_0\]
Illuminate pixel \((x, \text{int}(Y))\)
\[X = x_0 + 1 \quad Y = y_0 + 1 \times m\]
Illuminate pixel \((x, \text{int}(Y))\)
\[X = X + 1 \quad Y = Y + 1 \times m\]
Illuminate pixel \((x, \text{int}(Y))\)

... 

Until \(X == x_1\)
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 Algorithm (6)

\[
X = x_0 \\
Y = y_0
\]

Illuminate pixel \((x, \text{int}(Y))\)

\[
Y = y_0 + 1 \\
X = x_0 + 1 * \frac{1}{m}
\]

Illuminate pixel \((x, \text{int}(Y))\)

\[
Y = Y + 1 \\
X = X + 1 / m
\]

Illuminate pixel \((x, \text{int}(Y))\)

... 

Until \(Y == y_1\)
The above is the simplest line drawing algorithm

Not very efficient

Optimized algorithms such as integer DDA and Bresenhan algorithm (section 8.10) are typically used

Not the focus of this course