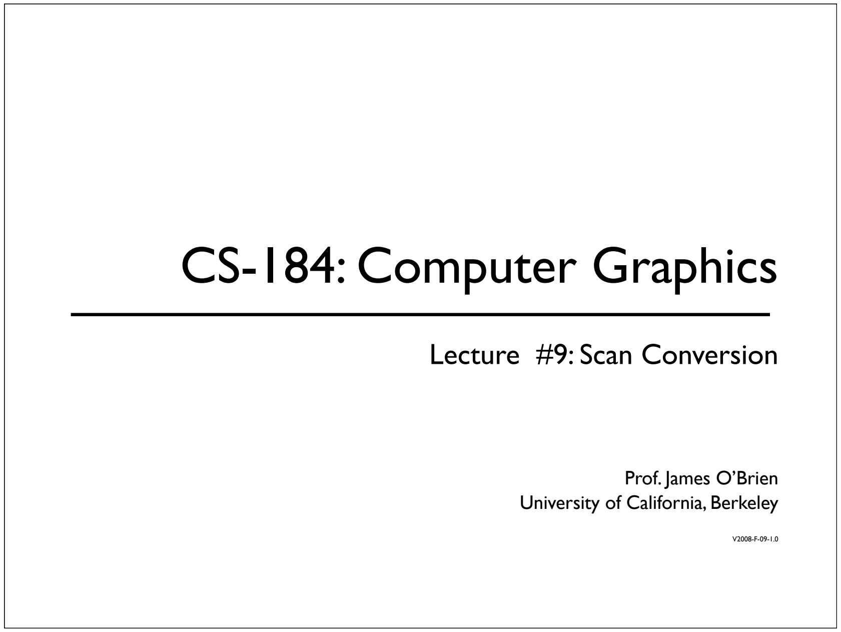


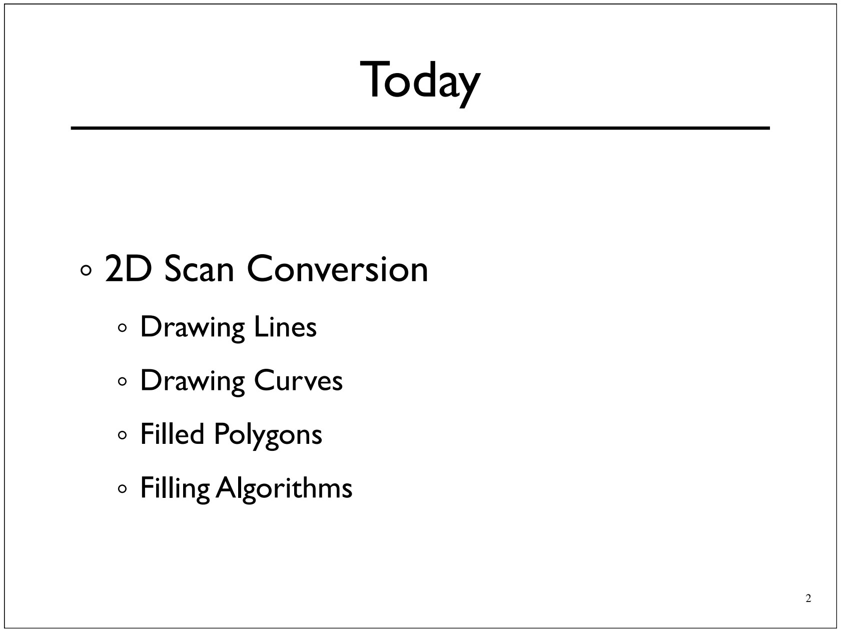
CS-184: Computer Graphics

Lecture #9: Scan Conversion

Prof. James O'Brien
University of California, Berkeley

V2008-F-09-1.0

1

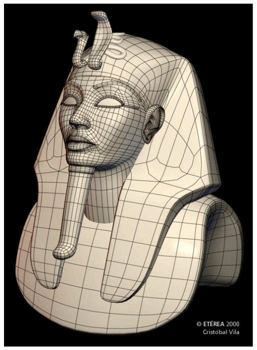


Today

- 2D Scan Conversion
  - Drawing Lines
  - Drawing Curves
  - Filled Polygons
  - Filling Algorithms

Drawing a Line

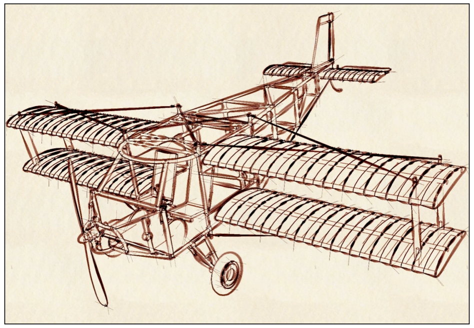
- Basically, its easy... but for the details
- Lines are a basic primitive that needs to be done well...



© ETÉREA 2000
Cristóbal Vila

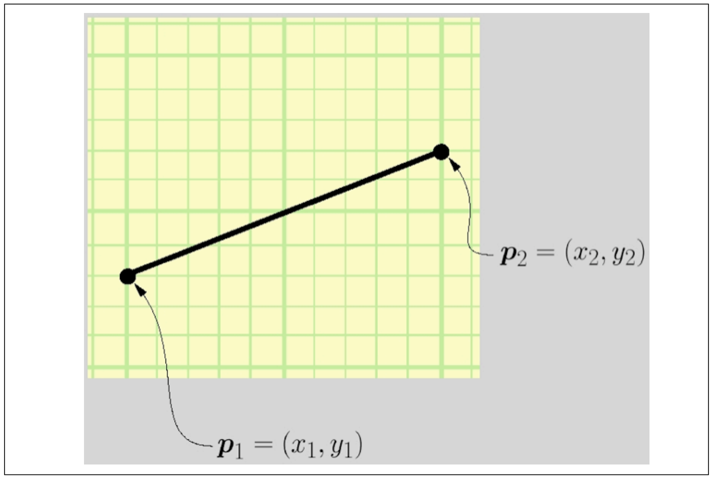
Drawing a Line

- Basically, its easy... but for the details
- Lines are a basic primitive that needs to be done well...

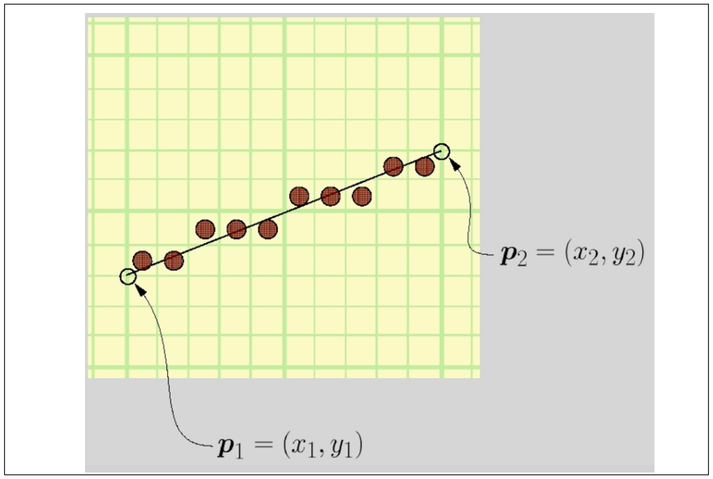


From "A Procedural Approach to Style for NPR Line Drawing from 3D models,"
by Grabli, Durand, Turquin, Sillion

Drawing a Line

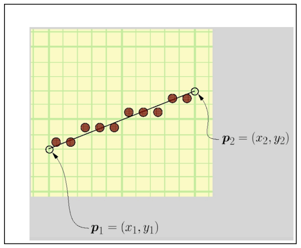

$\boldsymbol{p}_2 = (x_2, y_2)$
$\boldsymbol{p}_1 = (x_1, y_1)$

Drawing a Line


$\boldsymbol{p}_2 = (x_2, y_2)$
$\boldsymbol{p}_1 = (x_1, y_1)$

Drawing a Line

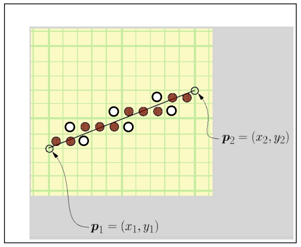
- Some things to consider
  - How thick are lines?
  - How should they join up?
  - Which pixels are the right ones?

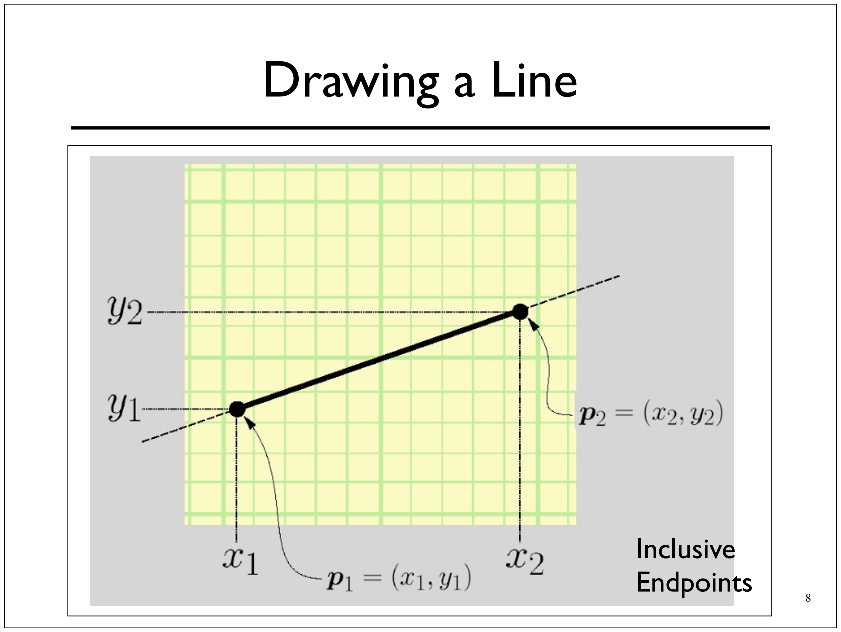


Drawing a Line

- Some things to consider
  - How thick are lines?
  - How should they join up?
  - Which pixels are the right ones?

For example:

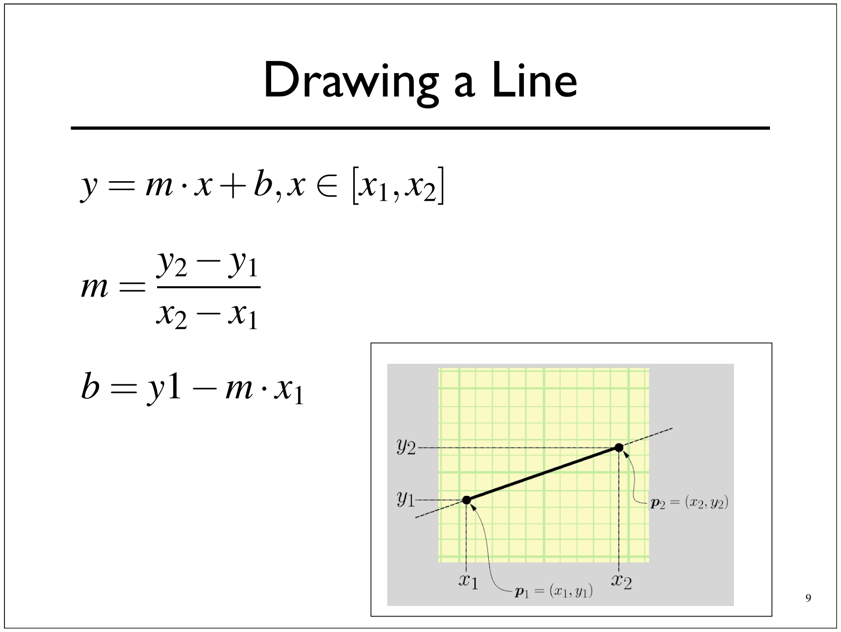




Drawing a Line

y_2
y_1
x_1
x_2
$\boldsymbol{p}_2 = (x_2, y_2)$
$\boldsymbol{p}_1 = (x_1, y_1)$

Inclusive Endpoints



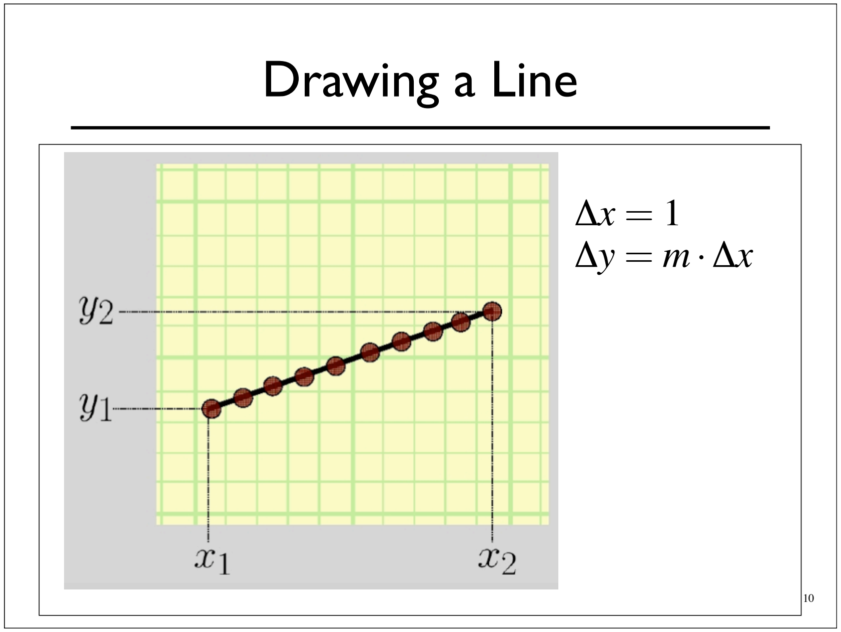
Drawing a Line

$$y = m \cdot x + b, x \in [x_1, x_2]$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

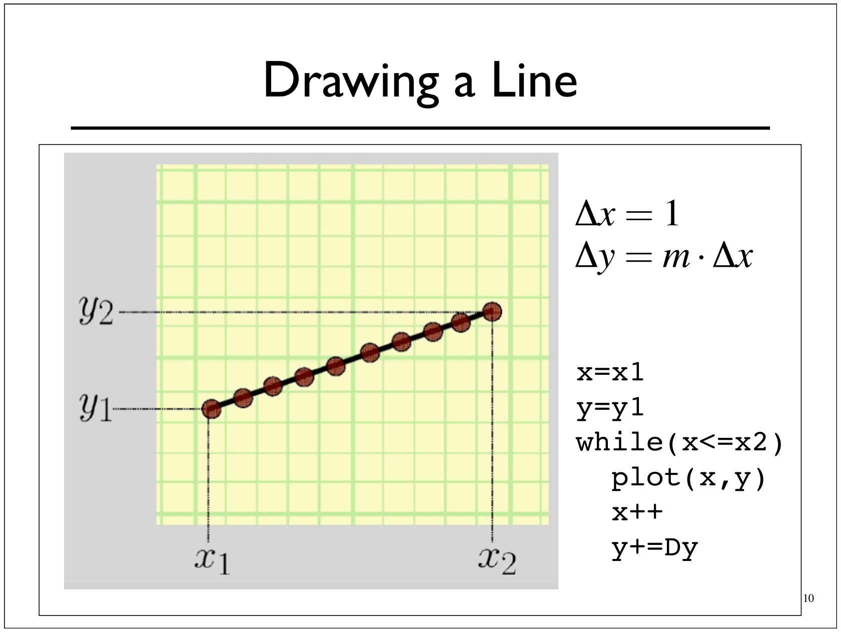
$$b = y1 - m \cdot x_1$$

y_2
y_1
x_1
x_2
$\boldsymbol{p}_2 = (x_2, y_2)$
$\boldsymbol{p}_1 = (x_1, y_1)$



Drawing a Line

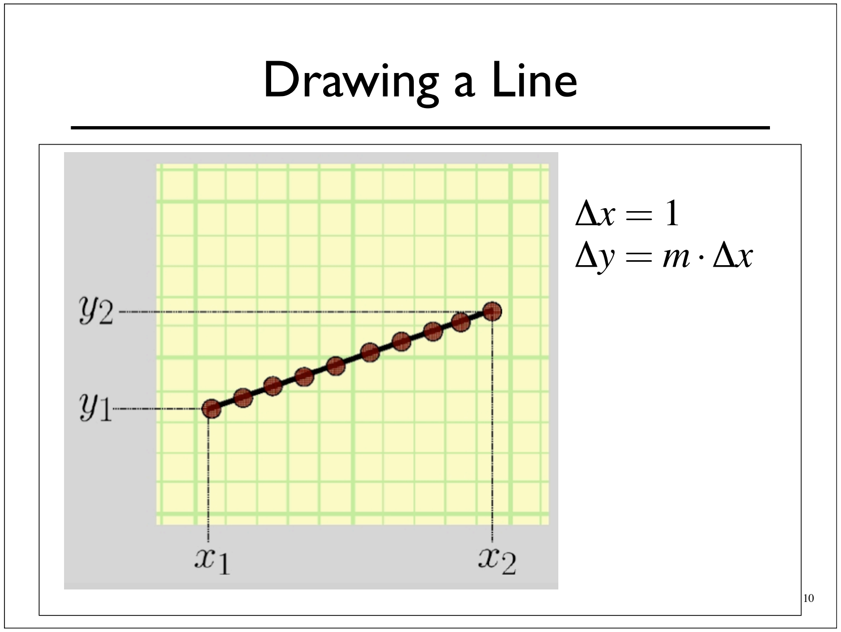
$\Delta x = 1$
$\Delta y = m \cdot \Delta x$



Drawing a Line

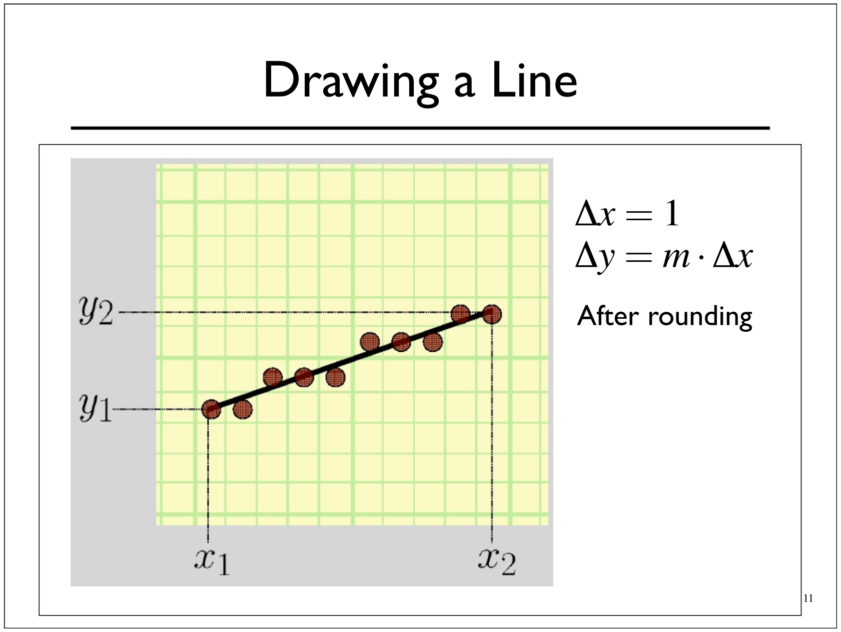
$\Delta x = 1$
$\Delta y = m \cdot \Delta x$

```
x=x1
y=y1
while(x<=x2)
   plot(x,y)
   x++
   y+=Dy
```



Drawing a Line

$\Delta x = 1$
$\Delta y = m \cdot \Delta x$

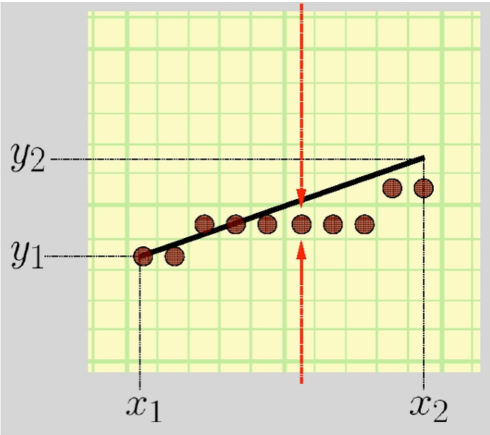


Drawing a Line

$\Delta x = 1$
$\Delta y = m \cdot \Delta x$

After rounding

Drawing a Line

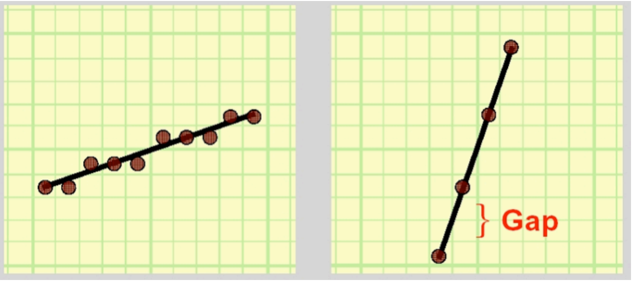


$\Delta x = 1$
$\Delta y = m \cdot \Delta x$
$y += \Delta y$

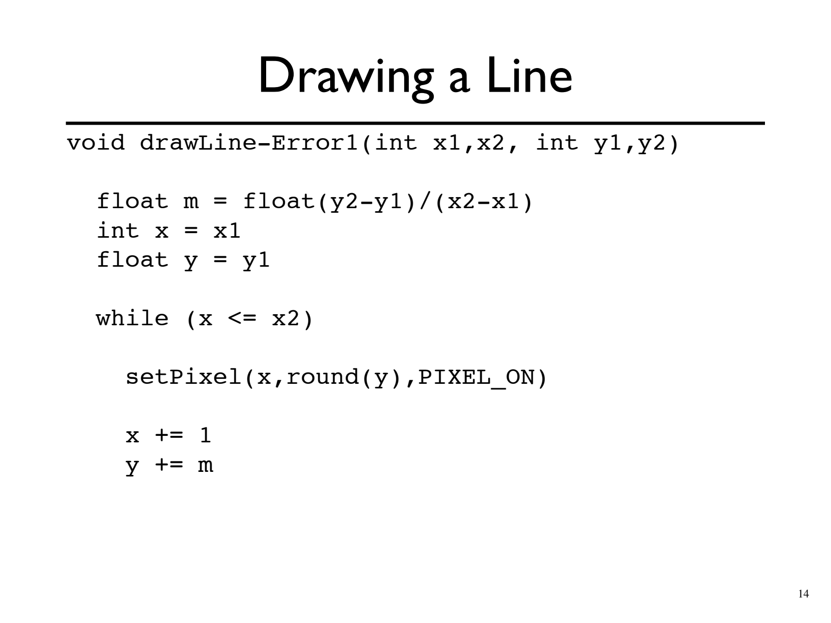
Accumulation of roundoff errors

How slow is float-to-int conversion?

Drawing a Line

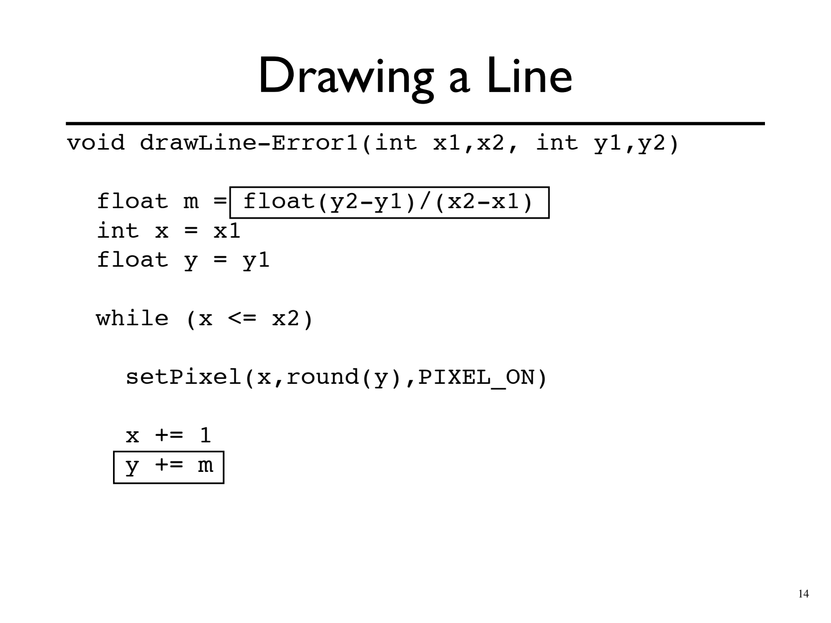


$|m| \leq 1$ $|m| > 1$



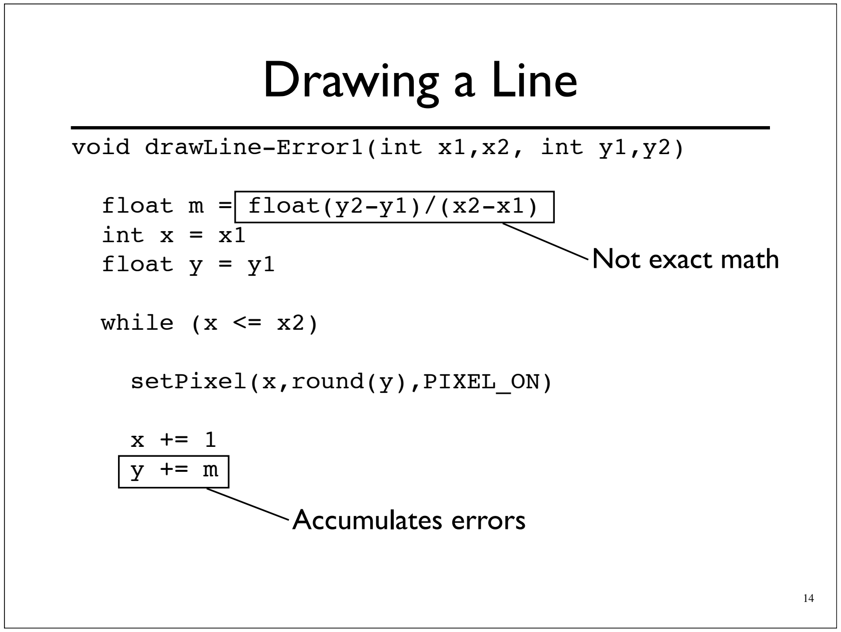
Drawing a Line

```
void drawLine-Error1(int x1,x2, int y1,y2)

  float m = float(y2-y1)/(x2-x1)
  int x = x1
  float y = y1

  while (x <= x2)

    setPixel(x,round(y),PIXEL_ON)

    x += 1
    y += m
```



Drawing a Line

```
void drawLine-Error1(int x1,x2, int y1,y2)

  float m = float(y2-y1)/(x2-x1)
  int x = x1
  float y = y1

  while (x <= x2)

    setPixel(x,round(y),PIXEL_ON)

    x += 1
    y += m
```

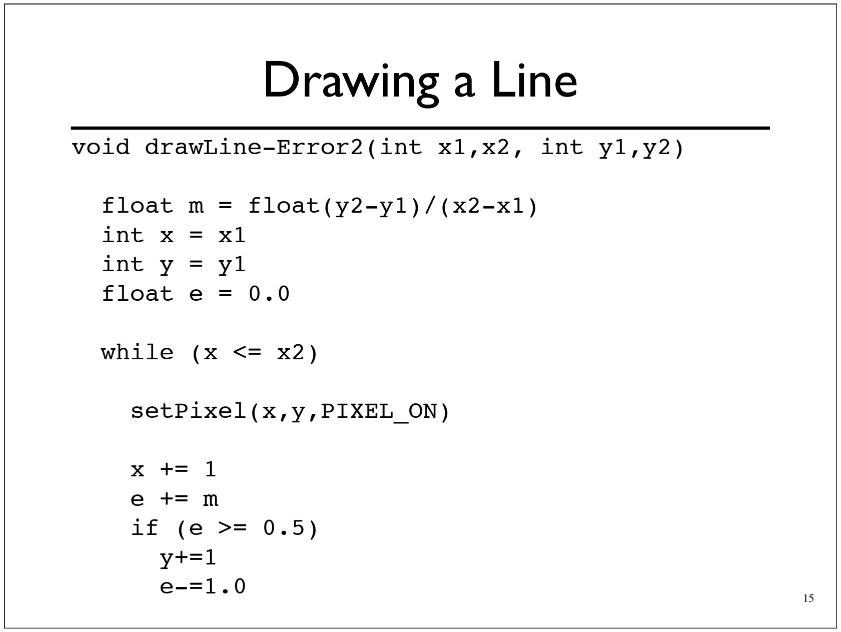


Drawing a Line

```
void drawLine-Error1(int x1,x2, int y1,y2)

  float m = float(y2-y1)/(x2-x1)
  int x = x1
  float y = y1

  while (x <= x2)

    setPixel(x,round(y),PIXEL_ON)

    x += 1
    y += m
```

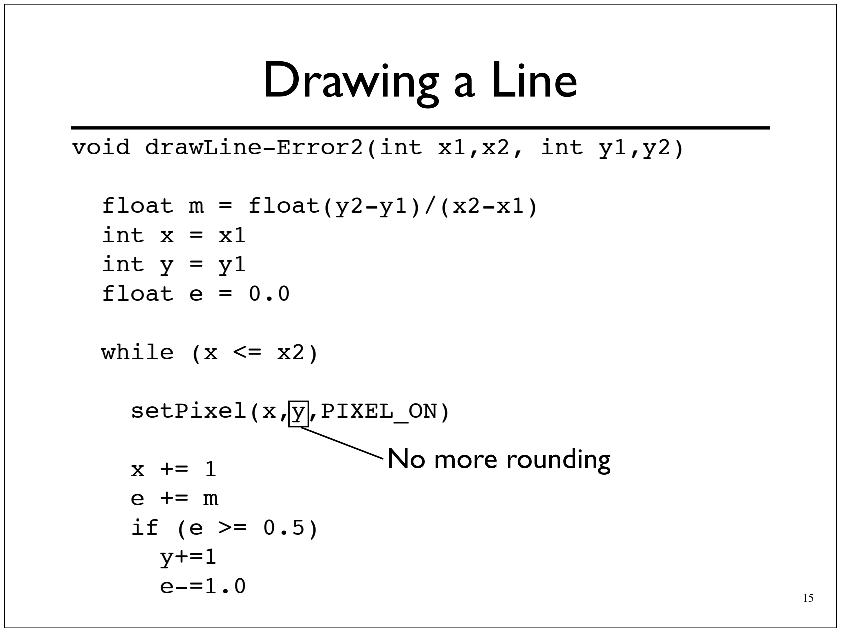
Not exact math

Accumulates errors



Drawing a Line

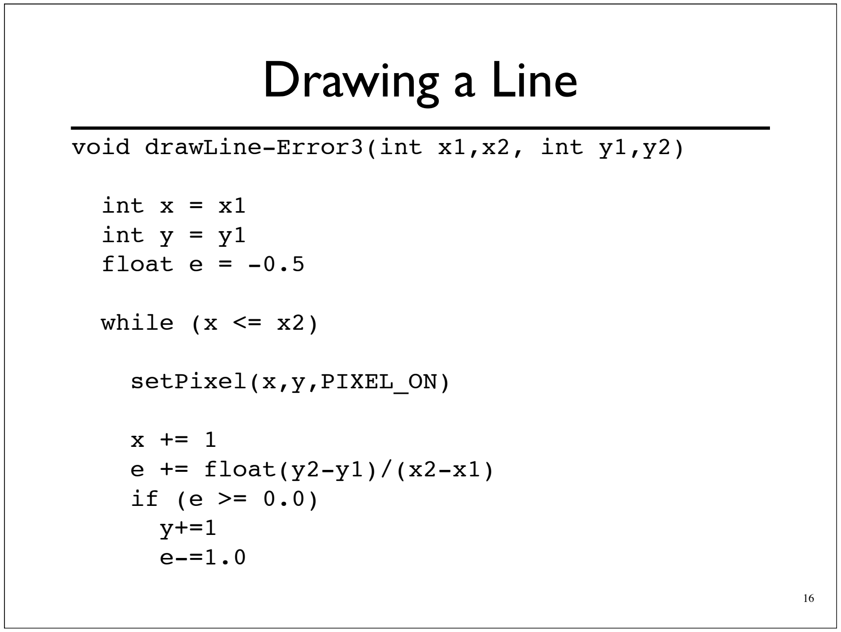
```
void drawLine-Error2(int x1,x2, int y1,y2)

  float m = float(y2-y1)/(x2-x1)
  int x = x1
  int y = y1
  float e = 0.0

  while (x <= x2)

    setPixel(x,y,PIXEL_ON)

    x += 1
    e += m
    if (e >= 0.5)
      y+=1
      e-=1.0
```



Drawing a Line

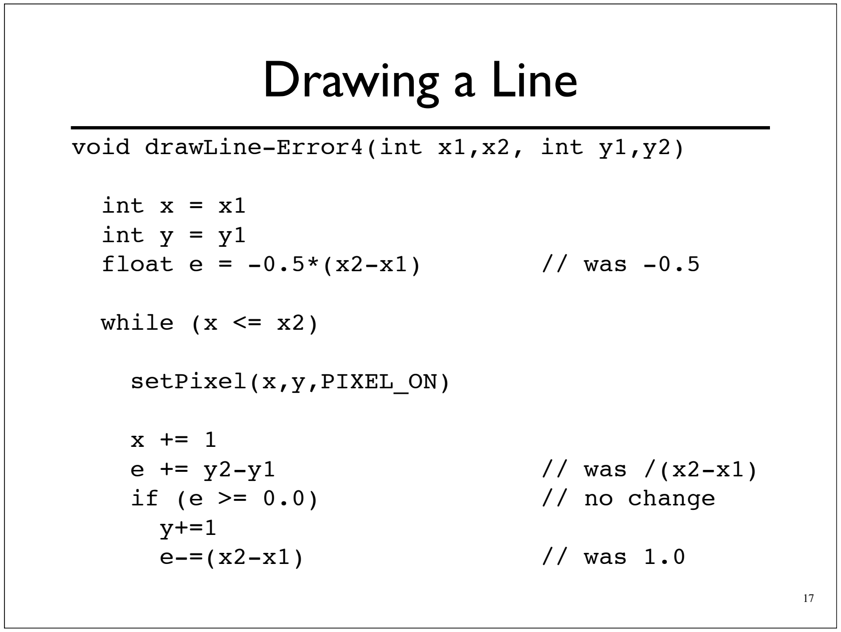
```
void drawLine-Error2(int x1,x2, int y1,y2)

  float m = float(y2-y1)/(x2-x1)
  int x = x1
  int y = y1
  float e = 0.0

  while (x <= x2)

    setPixel(x,y,PIXEL_ON)

    x += 1
    e += m
    if (e >= 0.5)
      y+=1
      e-=1.0
```

No more rounding



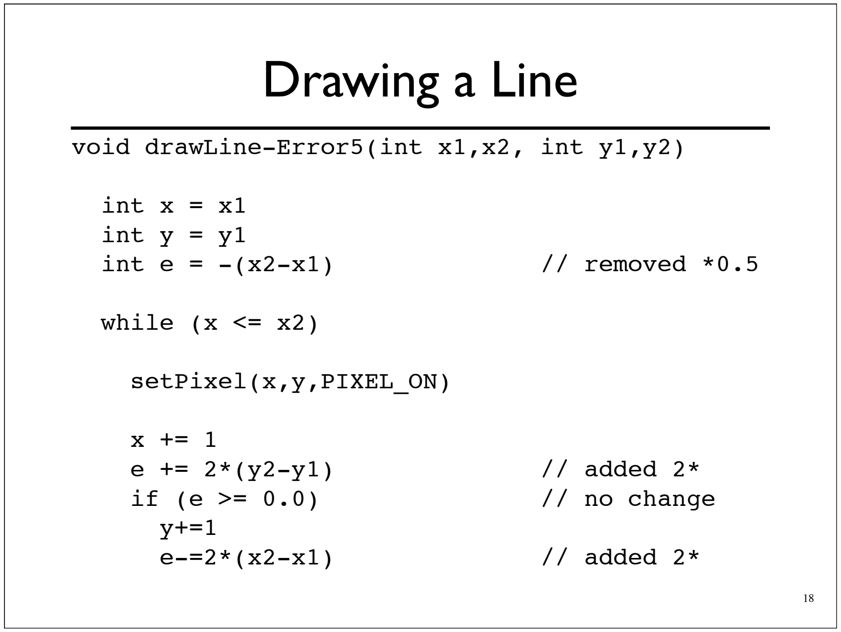
Drawing a Line

```
void drawLine-Error3(int x1,x2, int y1,y2)

  int x = x1
  int y = y1
  float e = -0.5

  while (x <= x2)

    setPixel(x,y,PIXEL_ON)

    x += 1
    e += float(y2-y1)/(x2-x1)
    if (e >= 0.0)
      y+=1
      e-=1.0
```



Drawing a Line

```
void drawLine-Error4(int x1,x2, int y1,y2)

  int x = x1
  int y = y1
  float e = -0.5*(x2-x1)        // was -0.5

  while (x <= x2)

    setPixel(x,y,PIXEL_ON)

    x += 1
    e += y2-y1                  // was /(x2-x1)
    if (e >= 0.0)               // no change
      y+=1
      e-=(x2-x1)                // was 1.0
```



Drawing a Line

```
void drawLine-Error5(int x1,x2, int y1,y2)

  int x = x1
  int y = y1
  int e = -(x2-x1)              // removed *0.5

  while (x <= x2)

    setPixel(x,y,PIXEL_ON)

    x += 1
    e += 2*(y2-y1)              // added 2*
    if (e >= 0.0)               // no change
      y+=1
      e-=2*(x2-x1)              // added 2*
```

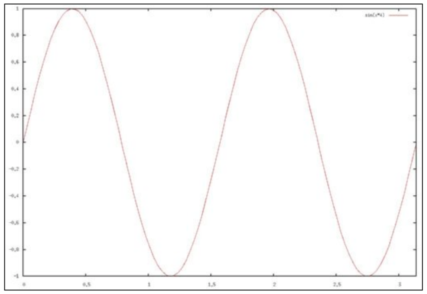
Drawing a Line

```
void drawLine-Bresenham(int x1,x2, int y1,y2)

  int x = x1
  int y = y1
  int e = -(x2-x1)

  while (x <= x2)

    setPixel(x,y,PIXEL_ON)

    x += 1
    e += 2*(y2-y1)
    if (e >= 0.0)
      y+=1
      e-=2*(x2-x1)
```

Faster
Not wrong

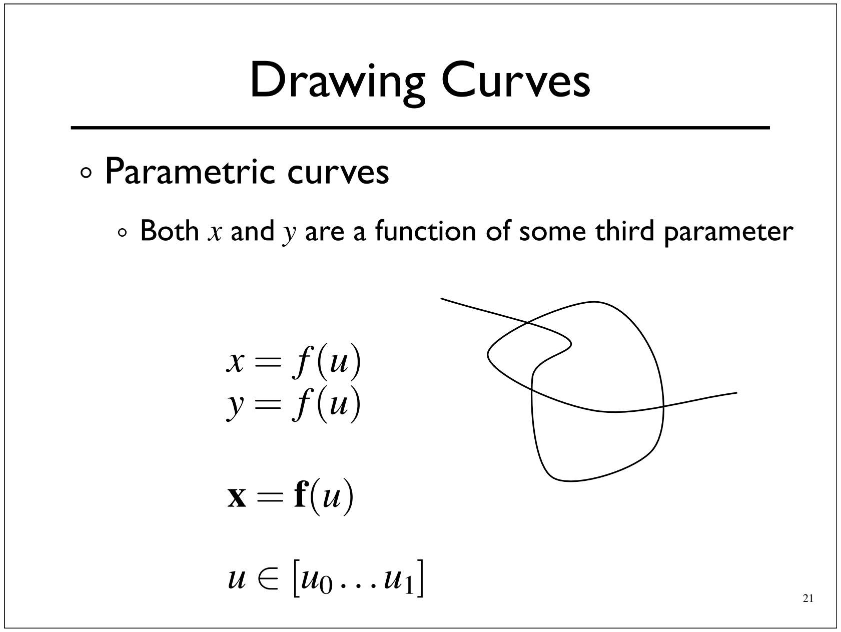
$|m| \leq 1$
$x_1 \leq x_2$

Drawing Curves



$$y = f(x)$$

Only one value of y for each value of x...



Drawing Curves

- Parametric curves
  - Both x and y are a function of some third parameter

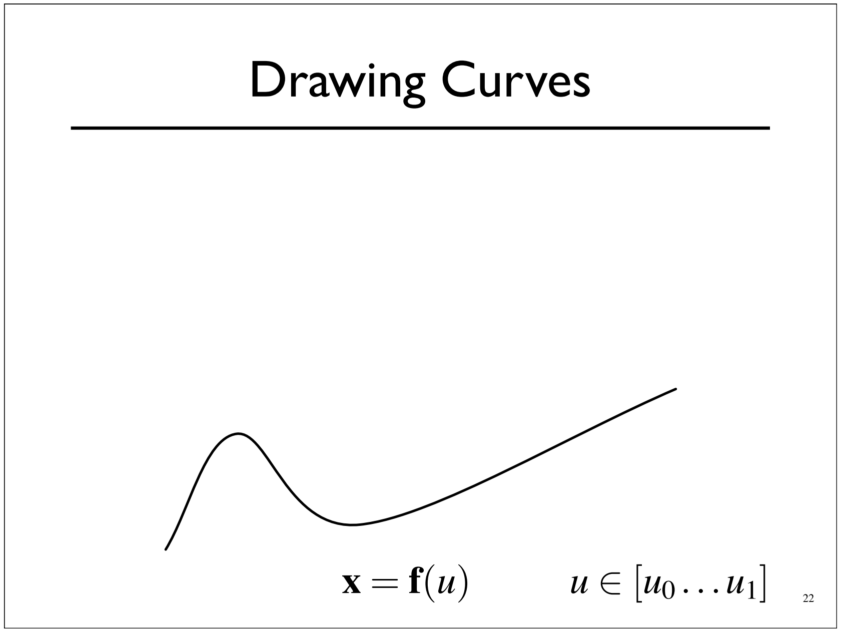
$$x = f(u)$$
$$y = f(u)$$

$$\mathbf{x} = \mathbf{f}(u)$$

$$u \in [u_0 \dots u_1]$$

21

21

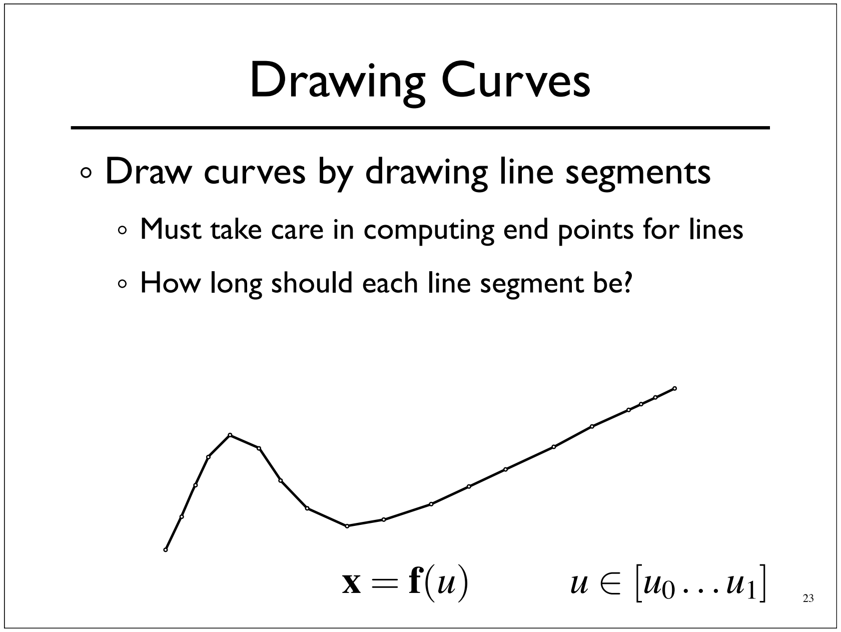


Drawing Curves

$$\mathbf{x} = \mathbf{f}(u) \qquad u \in [u_0 \dots u_1]$$

22

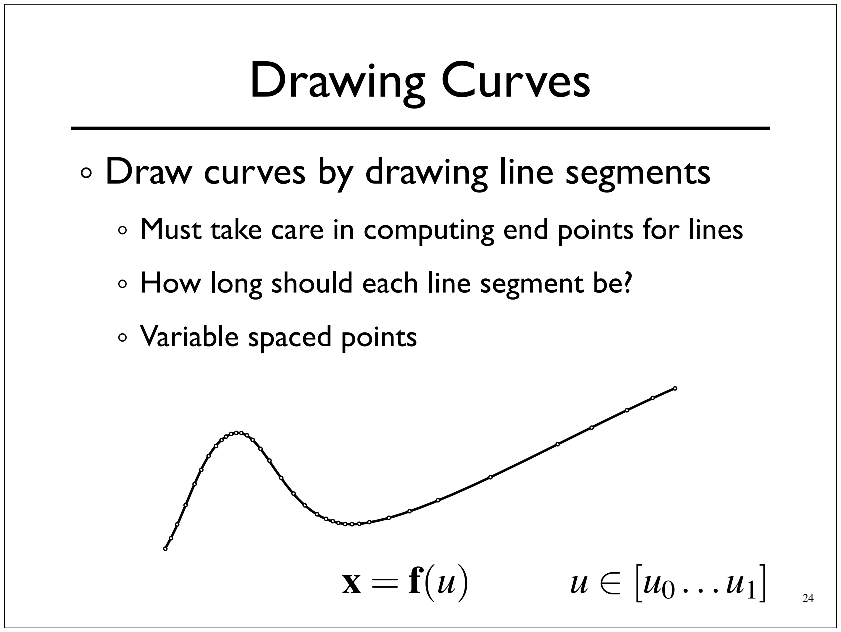
22



Drawing Curves

- Draw curves by drawing line segments
  - Must take care in computing end points for lines
  - How long should each line segment be?

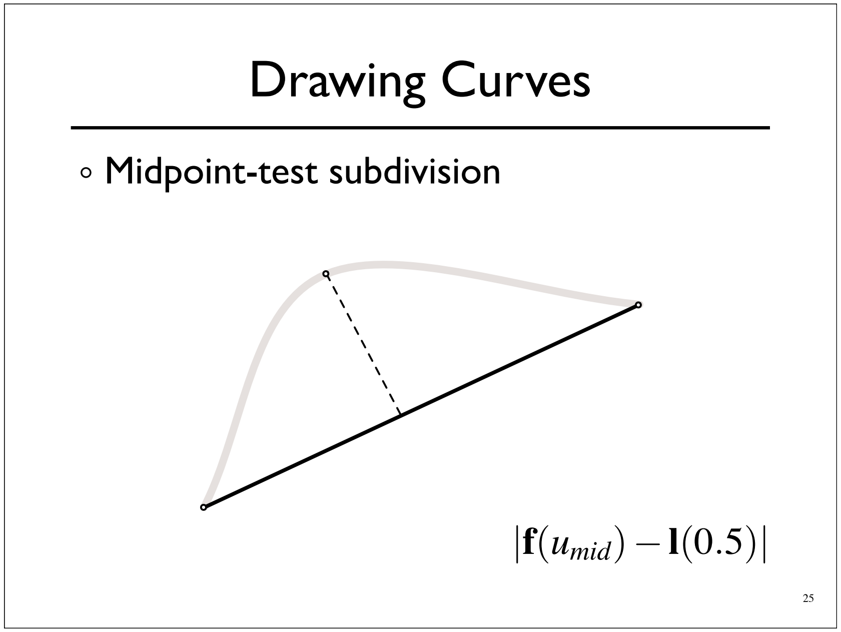
$\mathbf{x} = \mathbf{f}(u) \qquad u \in [u_0 \dots u_1]$



Drawing Curves

- Draw curves by drawing line segments
  - Must take care in computing end points for lines
  - How long should each line segment be?
  - Variable spaced points

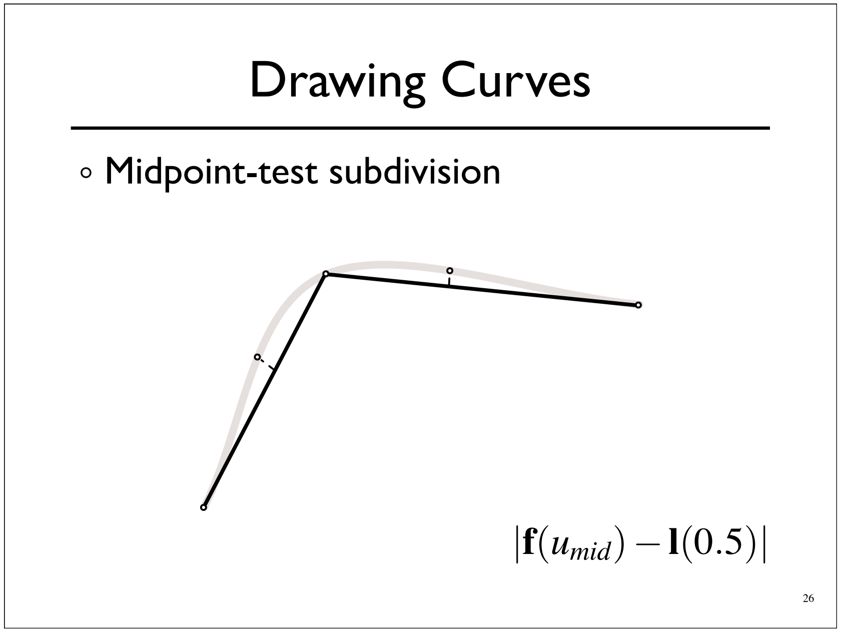
$\mathbf{x} = \mathbf{f}(u) \qquad u \in [u_0 \dots u_1]$



Drawing Curves

- Midpoint-test subdivision

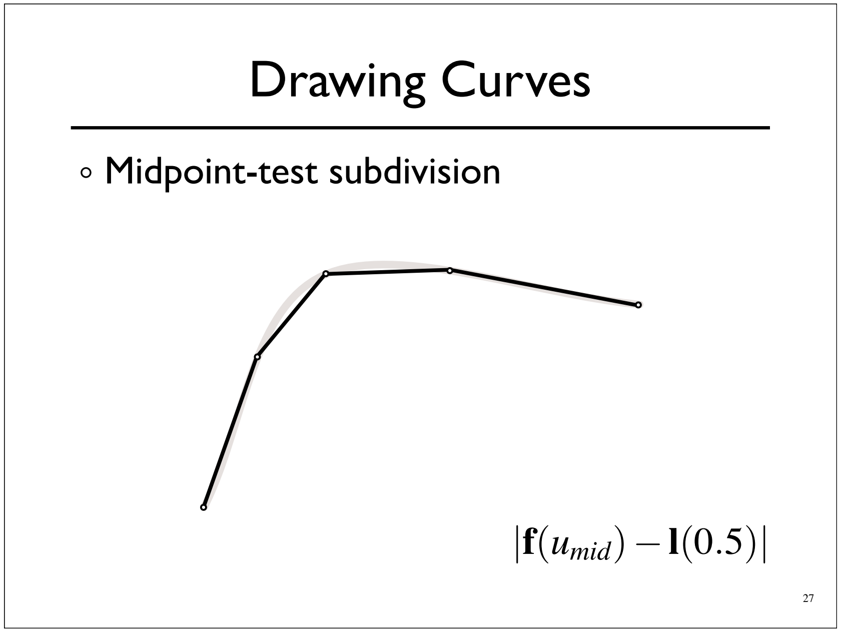
$$|\mathbf{f}(u_{mid}) - \mathbf{l}(0.5)|$$



Drawing Curves

- Midpoint-test subdivision

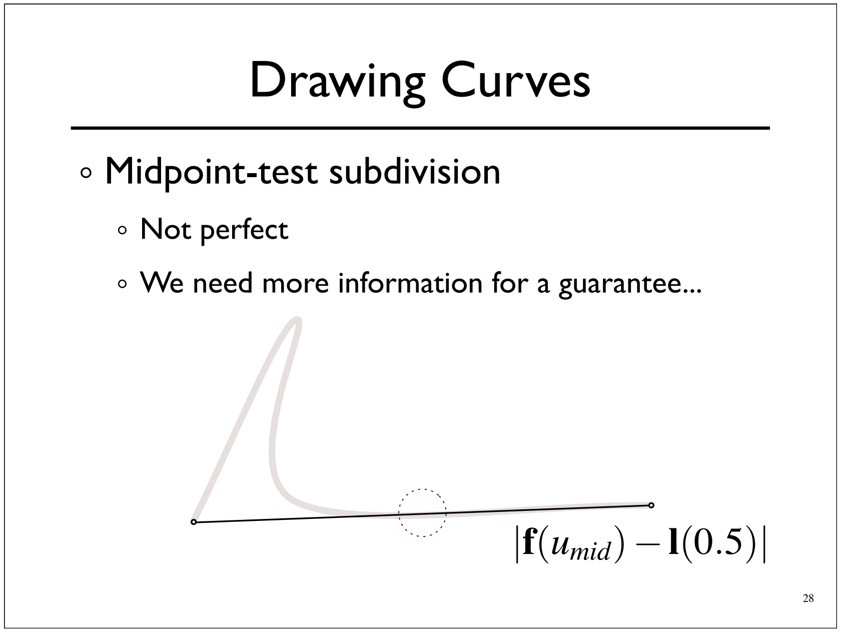
$$|\mathbf{f}(u_{mid}) - \mathbf{l}(0.5)|$$



Drawing Curves

- Midpoint-test subdivision

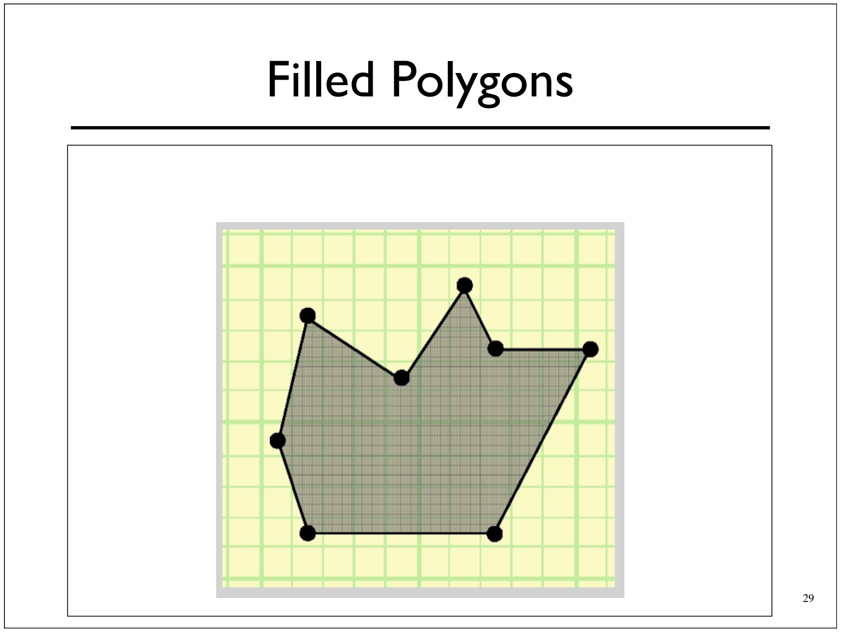
$$|\mathbf{f}(u_{mid}) - \mathbf{l}(0.5)|$$



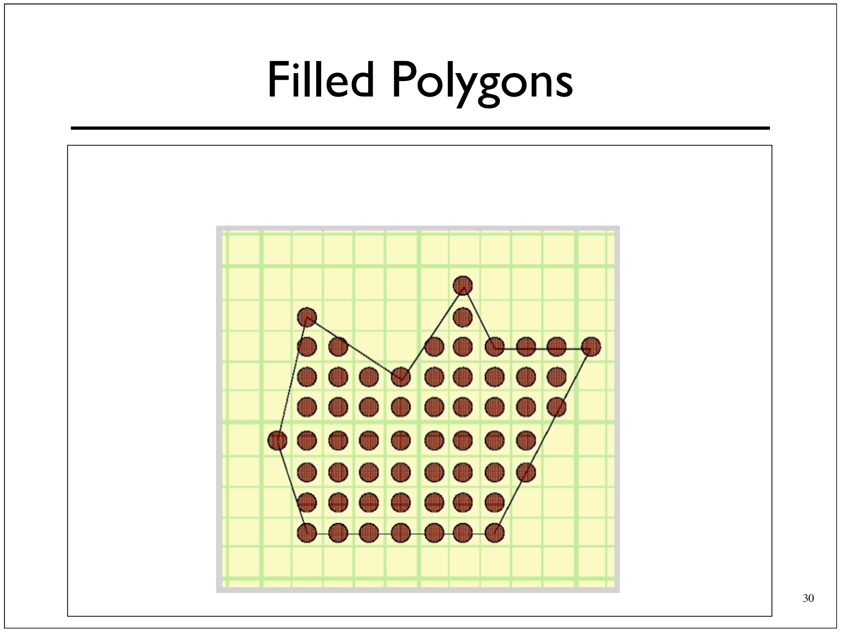
Drawing Curves

- Midpoint-test subdivision
  - Not perfect
  - We need more information for a guarantee...

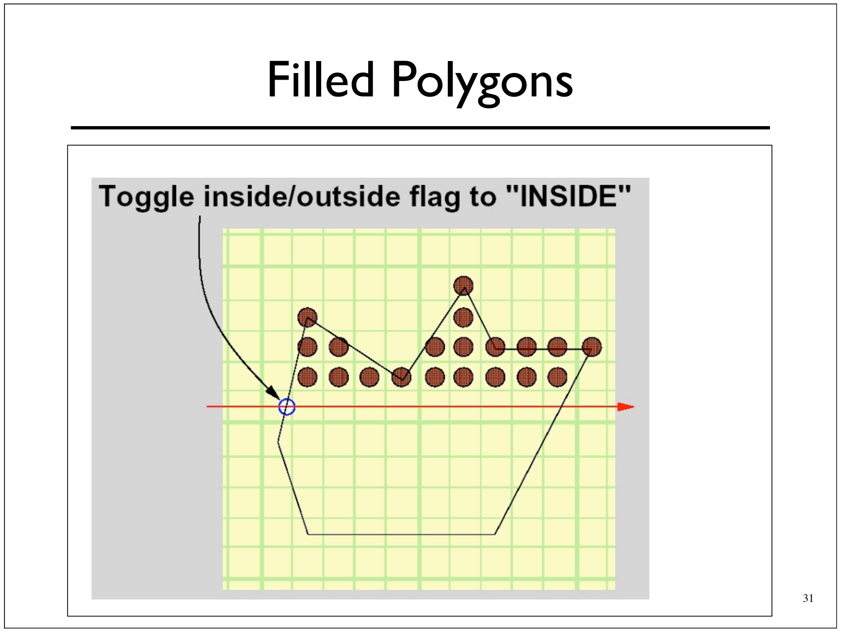
$$|\mathbf{f}(u_{mid}) - \mathbf{l}(0.5)|$$



Filled Polygons

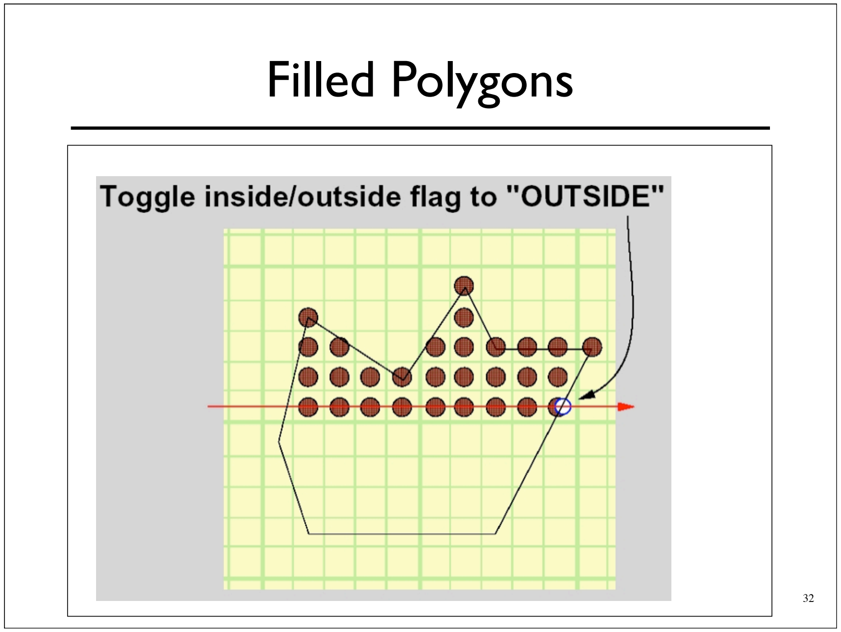


Filled Polygons



Filled Polygons

Toggle inside/outside flag to "INSIDE"

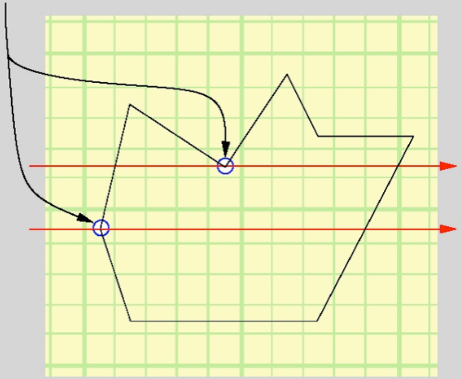


Filled Polygons

Toggle inside/outside flag to "OUTSIDE"

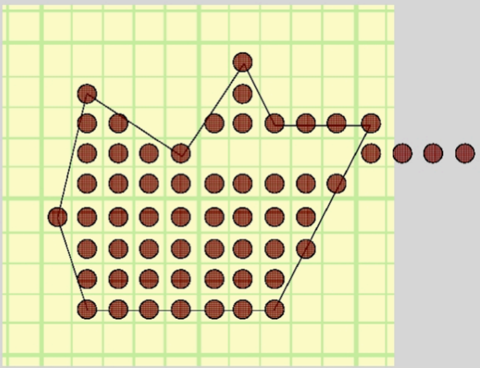
Filled Polygons

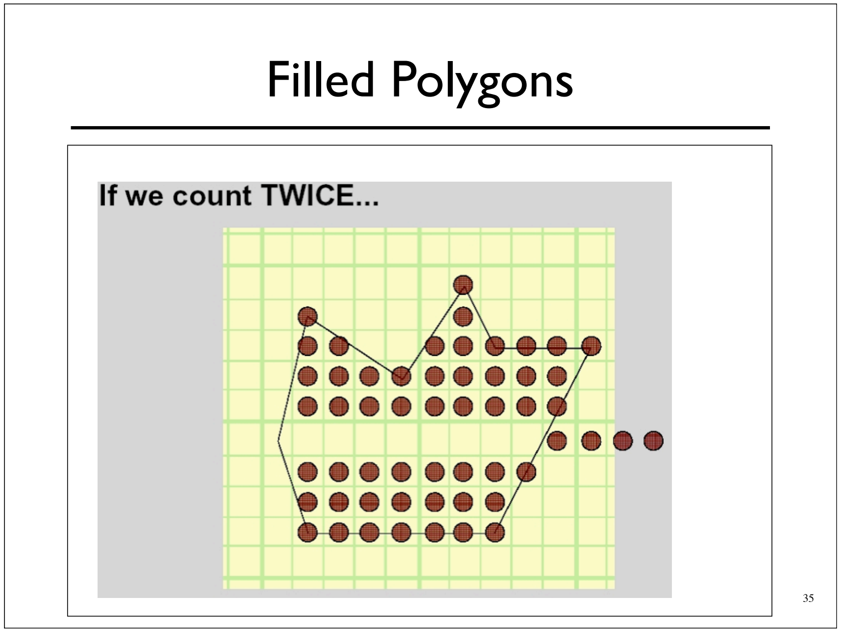
What happens at these locations?



Filled Polygons

If we count ONCE...

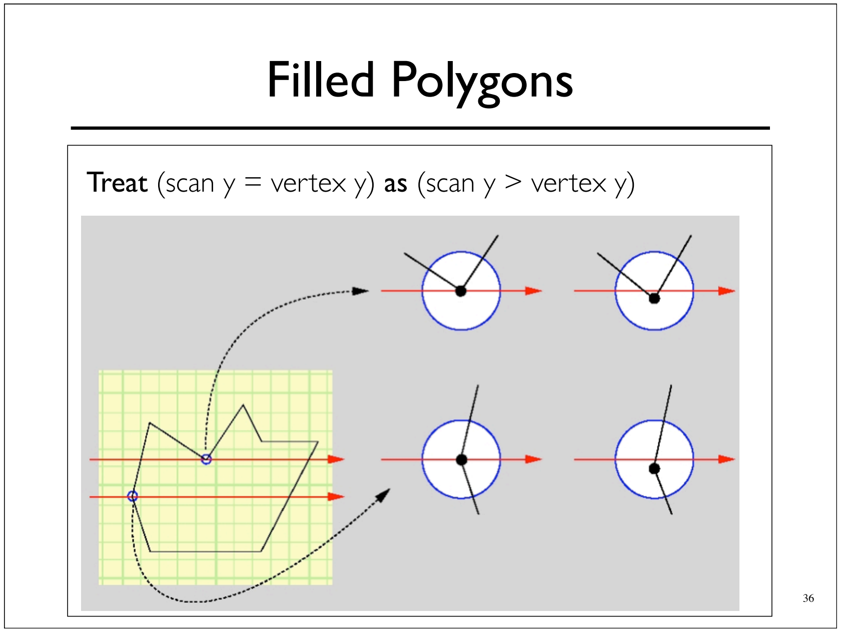




Filled Polygons

If we count TWICE...

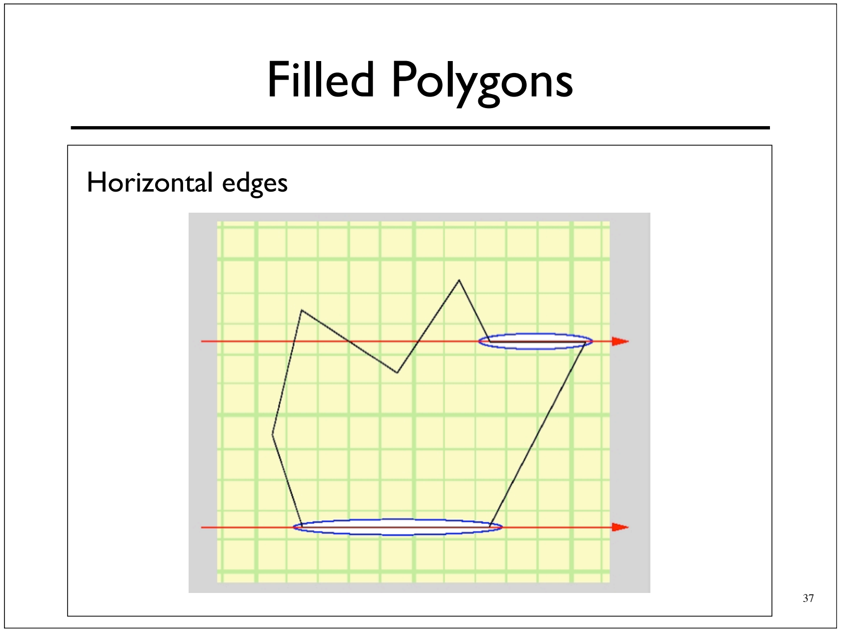
35



Filled Polygons

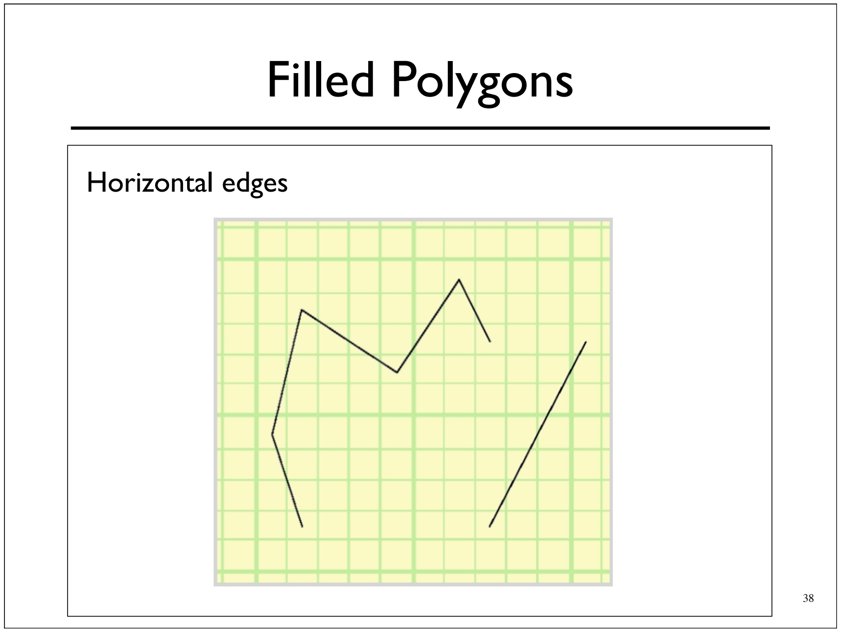
Treat (scan y = vertex y) **as** (scan y > vertex y)

36



Filled Polygons

Horizontal edges

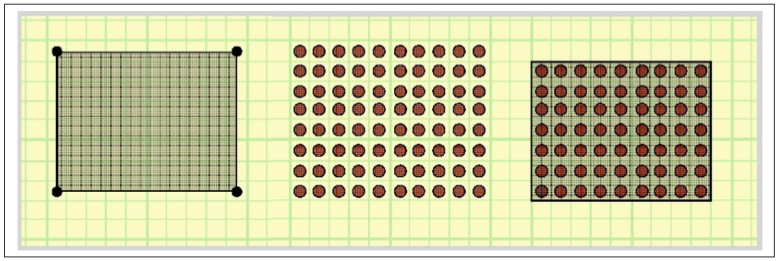


Filled Polygons

Horizontal edges

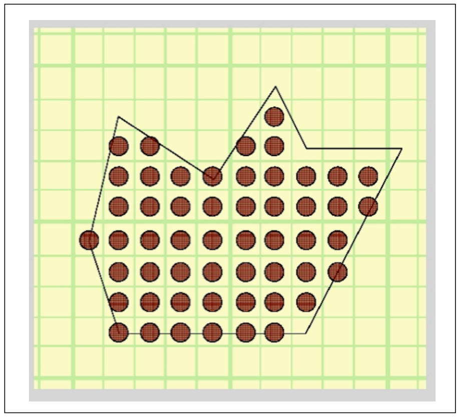
Filled Polygons

- "Equality Removal" applies to all vertices
- Both x and y coordinates



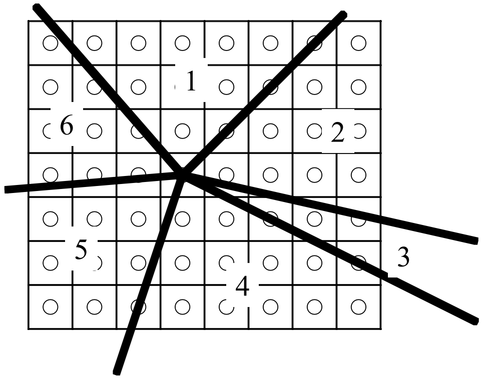
Filled Polygons

- Final result:



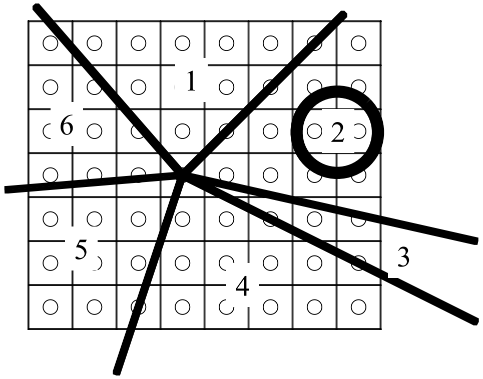
Filled Polygons

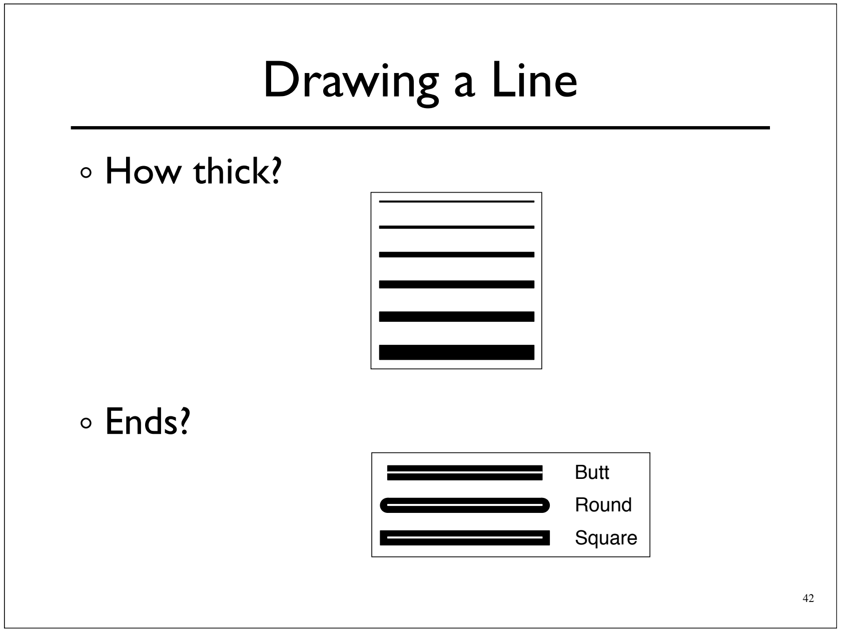
- Who does this pixel belong to?



Filled Polygons

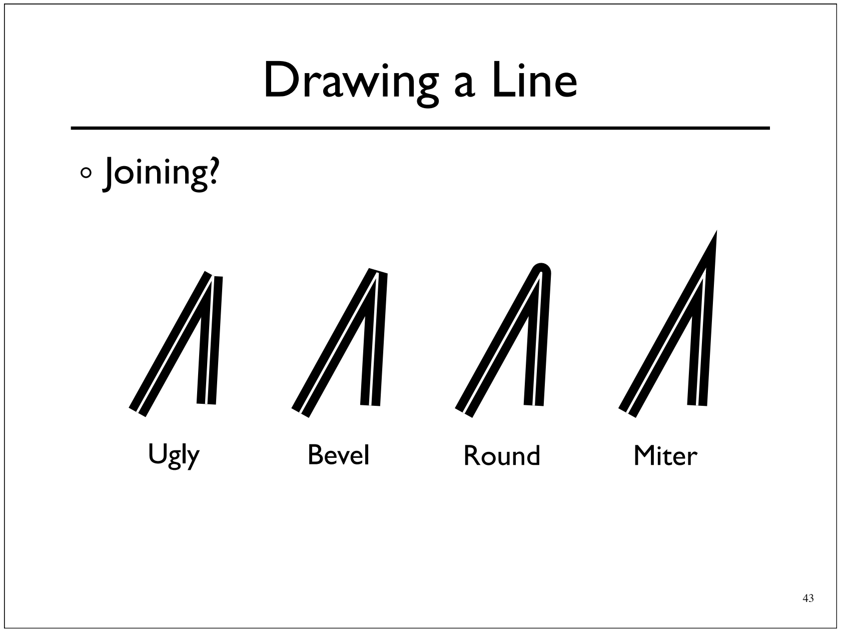
- Who does this pixel belong to?





Drawing a Line

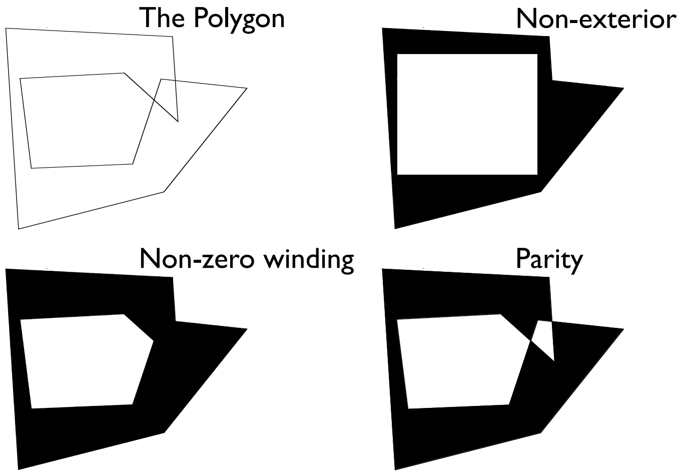
- How thick?
- Ends?



Drawing a Line

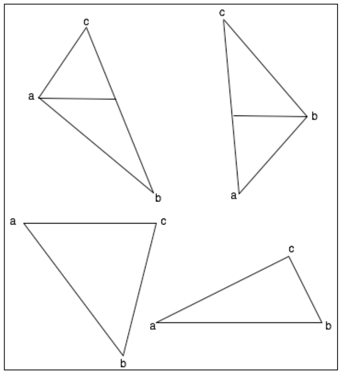
- Joining?

Inside/Outside Testing

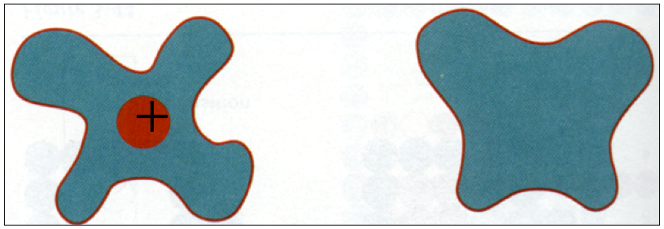


Optimize for Triangles

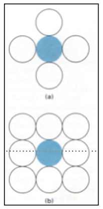
- Spilt triangle into two parts
  - Two edges per part
  - Y-span is monotonic
- For each row
  - Interpolate span
- Interpolate barycentric coordinates

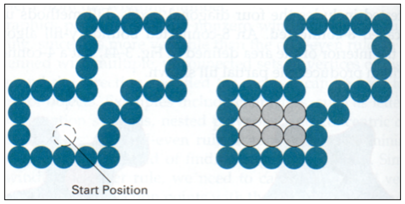


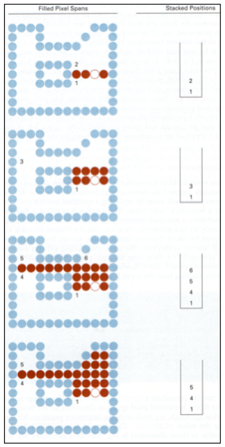
Flood Fill



Flood Fill




Start Position


Filled Pixel Spans
Stacked Positions