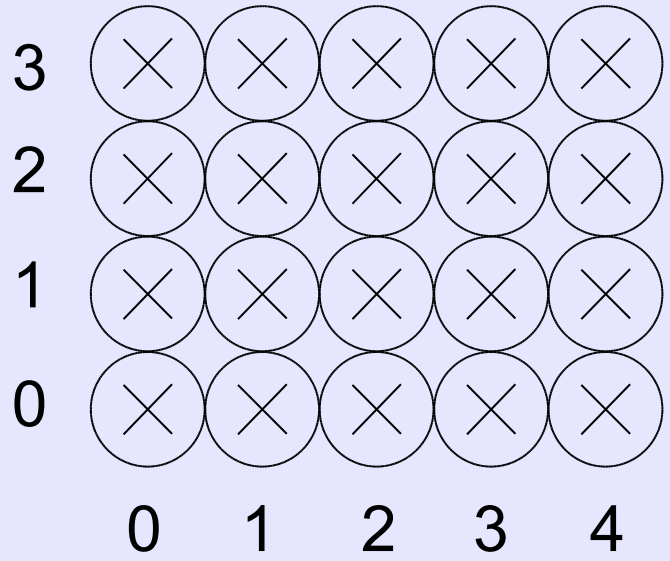
Lecture 5

Rasterizing lines, polygons

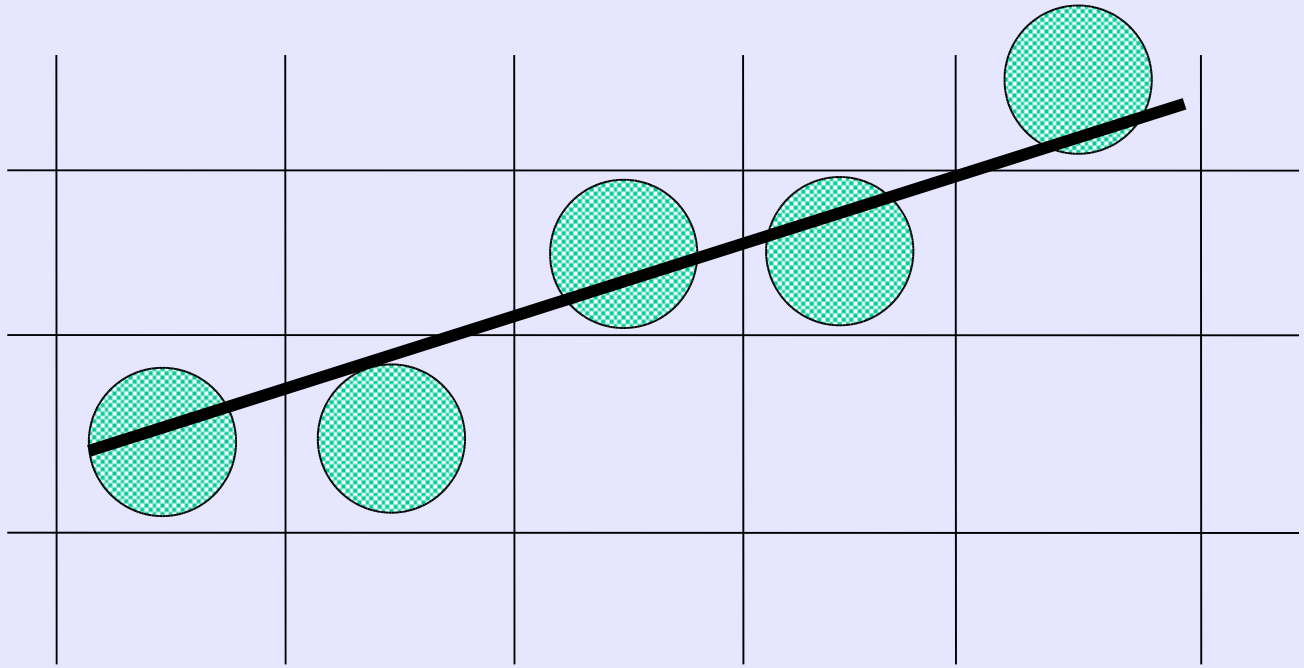
Rasterization

Array of pixels



Rasterizing Lines

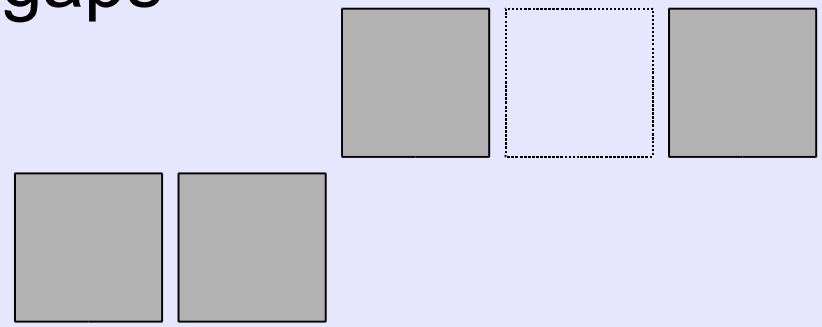
Given two endpoints, $(x_0, y_0), (x_1, y_1)$
find the pixels that make up the line.



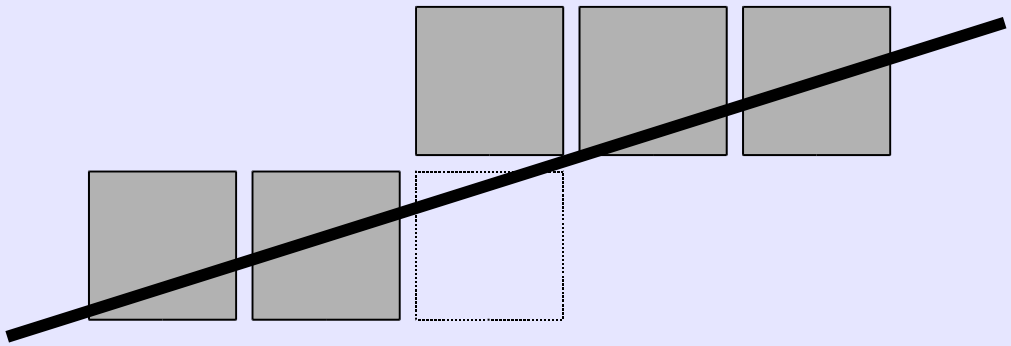
Rasterizing Lines

Requirements

* No gaps



* Minimize error (distance to line)



Rasterizing Lines

Equation of a Line:

$$y = mx + b = f(x)$$

Taylor Expansion:

$$y(x + \Delta x) = y + f'(x)\,\Delta x$$

So if we have an x,y on the line,
we can find the next point incrementally.

Rasterizing Lines

Assume –1 < m < 1, x0 < x1

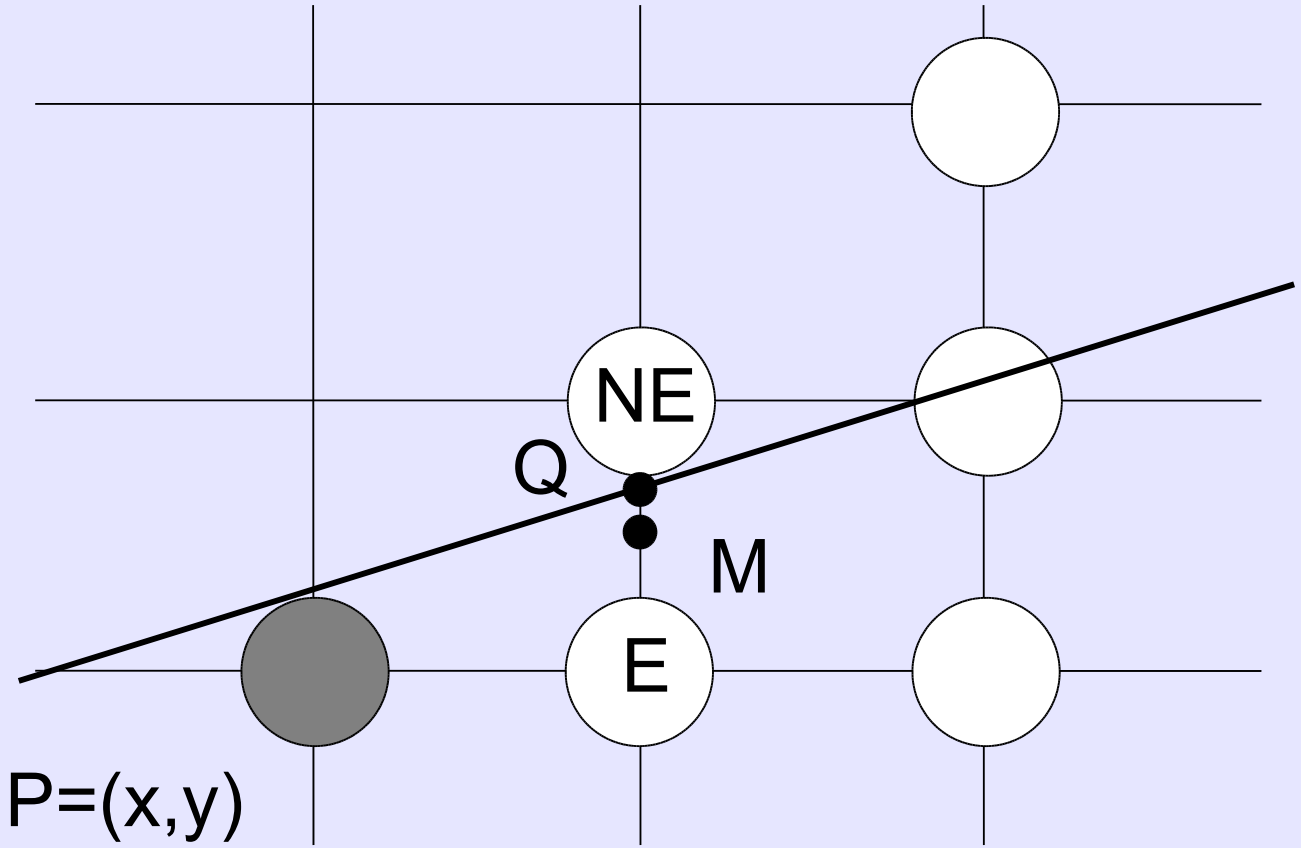
```
Line(int x0, int y0, int x1, int y1)
  float dx = x1 – x0;
  float dy = y1 – y0;
  float m  = dy/dx;
  float x = x0, y= y0;

  for(x = x0; x <= x1; x++)
     setPixel(x,round(y));
     y = y+m;
```

Rasterizing Lines

Problems with previous algorithm

1. round takes time
2. uses floating point arithmetic

Midpoint Algorithm



If Q <= M, choose E. If Q > M, choose NE

Implicit Form of a Line

Implicit form

$$ax + by + c = 0$$

Explicit form

$$y = \frac{dy}{dx}x + B$$

$$dy\ x - dx\ y + B\ dx = 0$$

$$a = dy \quad b = -dx \quad c = B\ dx$$

Positive below the line
Negative above the line
Zero on the line

Decision Function

$$d = F(x, y) = a\,x + b\,y + c$$

$$d = F\left(x+1, y+\frac{1}{2}\right) = a\,(x+1) + b\left(y+\frac{1}{2}\right) + c$$

Choose NE if d > 0
Choose E if d <= 0

Incrementing d

If choosing E:

$$d_{new} = F(x+2, y+\tfrac{1}{2}) = a\,(x+2) + b\,(y+\tfrac{1}{2}) + c$$

But:

$$d_{old} = F(x+1, y+\tfrac{1}{2}) = a\,(x+1) + b\,(y+\tfrac{1}{2}) + c$$

So:

$$d_{inc} = d_{new} - d_{old} = a = \Delta E$$

Incrementing d

If choosing NE:

$$d_{new} = F(x+2, y+\tfrac{3}{2}) = a\,(x+2) + b\,(y+\tfrac{3}{2}) + c$$

But:

$$d_{old} = F(x+1, y+\tfrac{1}{2}) = a\,(x+1) + b\,(y+\tfrac{1}{2}) + c$$

So:

$$d_{inc} = d_{new} - d_{old} = a + b = \Delta NE$$

Initializing d

$$d = F(x_0 + 1, y_0 + \frac{1}{2}) = a\,(x_0 + 1) + b\,(y_0 + \frac{1}{2}) + c$$

$$= a\,x_0 + b\,y_0 + c + a + b\frac{1}{2}$$

$$= a + b\frac{1}{2}$$

Multiply everything by 2 to remove fractions
(doesn't change the sign)

Midpoint Algorithm

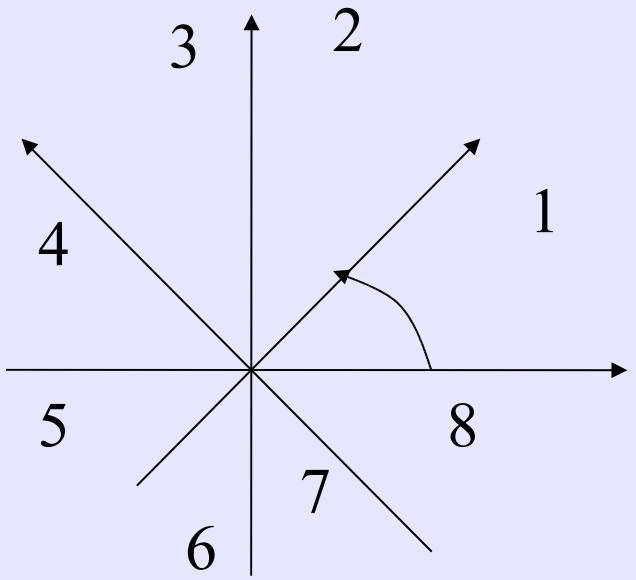
Assume 0 < m < 1, x0 < x1

```
Line(int x0, int y0, int x1, int y1)
  int dx = x1 – x0, dy = y1 – y0;
  int d = 2*dy-dx;
  int delE = 2*dy, delNE = 2*(dy-dx);
  int x = x0, y = y0;
  setPixel(x,y);

  while(x < x1)
    if(d<=0)
      d += delE; x = x+1;
    else
      d += delNE; x = x+1; y = y+1;
    setPixel(x,y);
```

See: Bresenham Algorithm

Limitations?

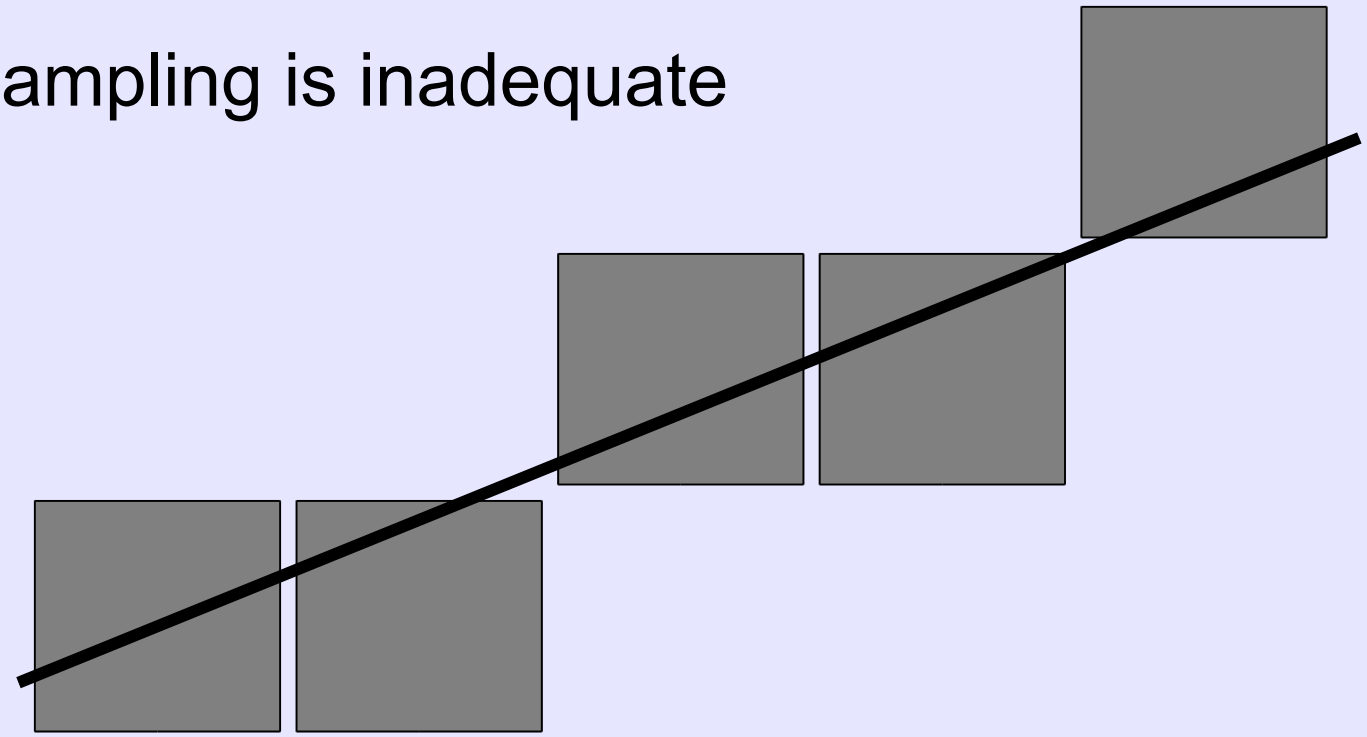
- The midpoint line algorithm assumes that the slope (m) is between 0 and 1
- This implies that this algorithm only applies to lines in region 1
- Extending to other regions left as an assignment



Anti-aliasing Lines

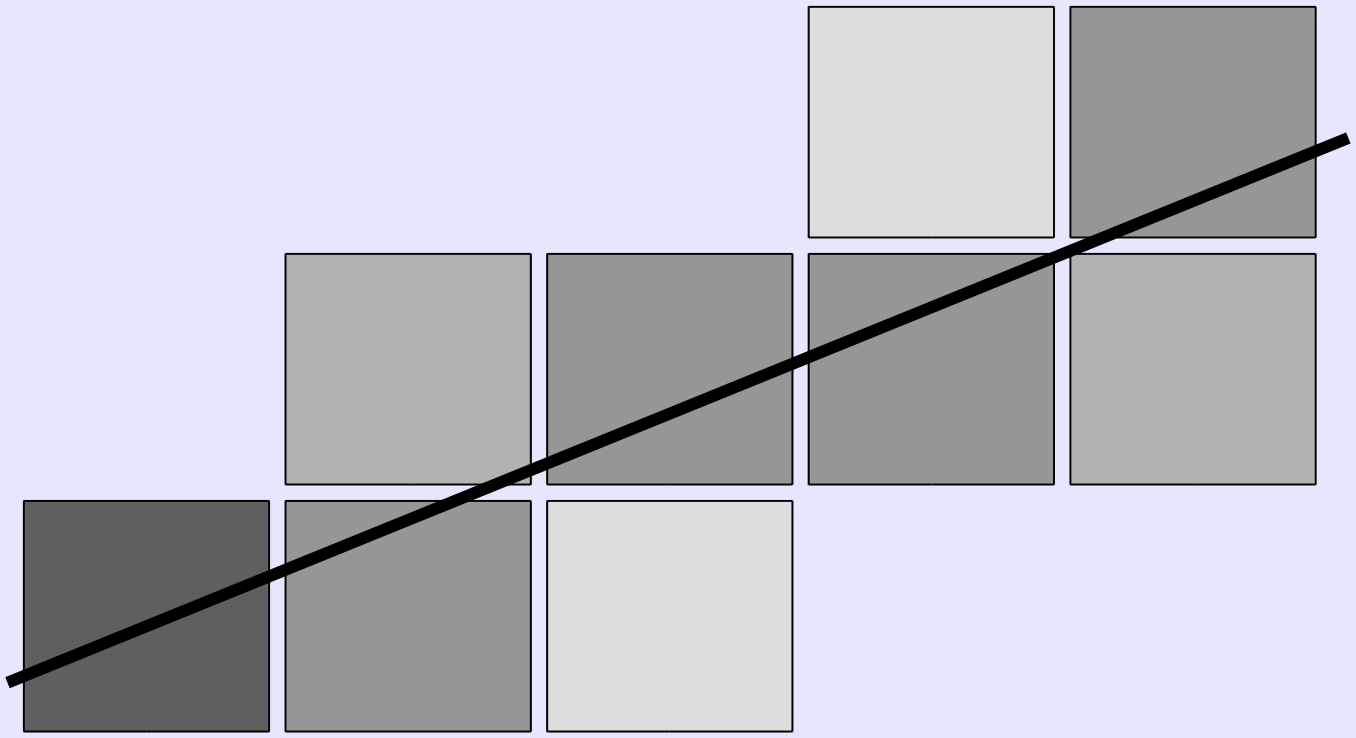
Lines appear jaggy

Sampling is inadequate



Anti-aliasing Lines

Trade intensity resolution for spatial resolution



Anti-aliasing Lines

Assume 0 < m < 1, x0 < x1

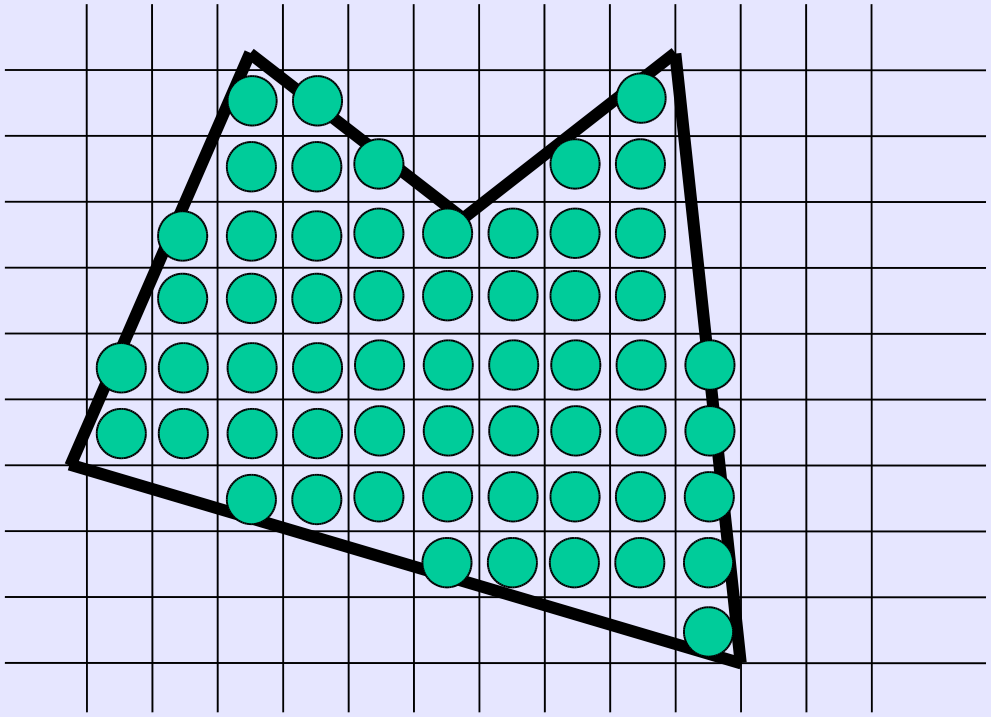
```
Line(int x0, int y0, int x1, int y1)
  float dx = x1 – x0;
  float dy = y1 – y0;
  float m  = dy/dx;
  float x = x0, y= y0;

  for(x = x0; x <= x1; x++)
     int yi = floor(y); float f = y – yi;
     setPixel(x,yi,  1-f);
     setPixel(x,yi+1, f);
     y = y+m;
```

Why line rasterization matters

- Take your representation (points) and transform it from Object Space to World Space
- Take your World Space point and transform it to Camera Space
- Perform the remapping and projection onto the image plane in Normalized Device Coordinates
- Perform this set of transformations on each point of the polygonal object
- “Connect the dots” through line rasterization

Rasterizing Polygons

Given a set of vertices and edges,
find the pixels that fill the polygon.



Rasterizing Polygons

vList is an ordered list of the polygon's vertices

```
fillPoly(vertex vList[ ])
   boundingBox b = getBounds(vList);
   int xmin = b.minX;
   int xmax = b.maxX;
   int ymin = b.minY;
   int ymax = b.maxY;

   for(int y = ymin; y <= ymax; y++)
      for(int x = xmin; x <= xmax; x++)
         if(insidePoly(x,y,vList))
            setPixel(x,y);
```

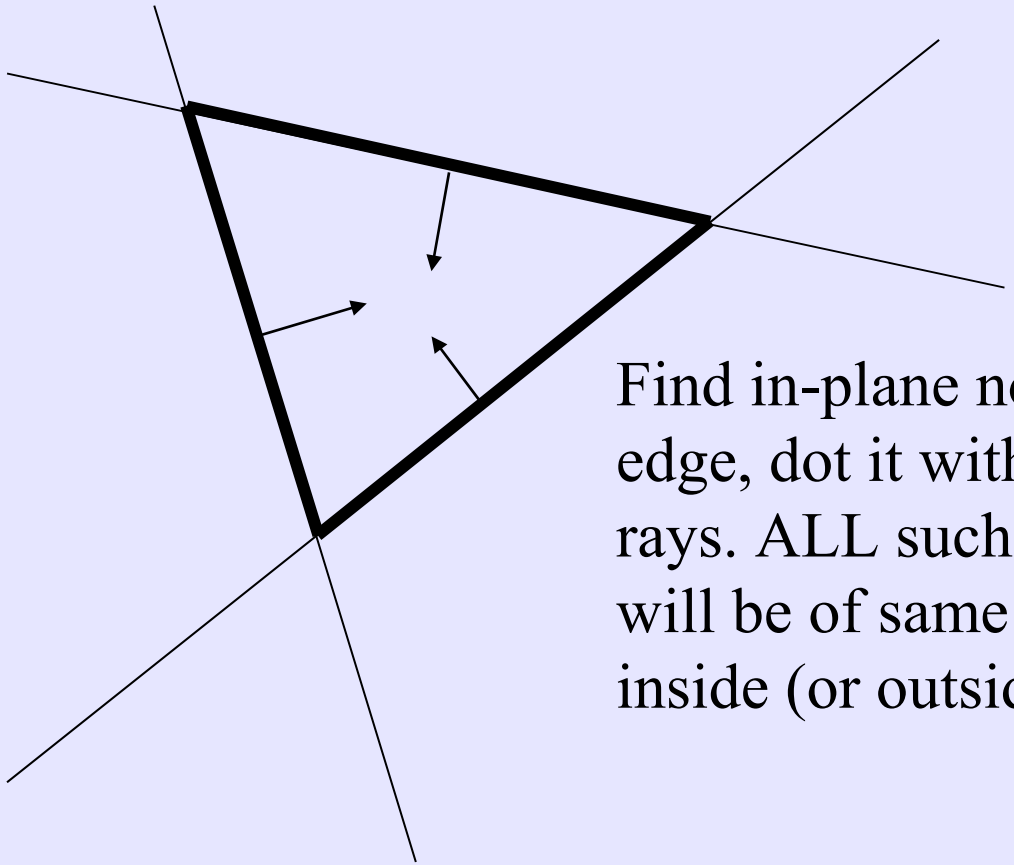
What does 'inside' mean?

How to test if a point is inside a polygon

- Half-space tests
- Jordan Curve Theorem (even/odd or +1/-1)
- 'winding number' tests

Half Space Tests

Given the edges of a triangle, the inside is the intersection of half-spaces defined by the edges



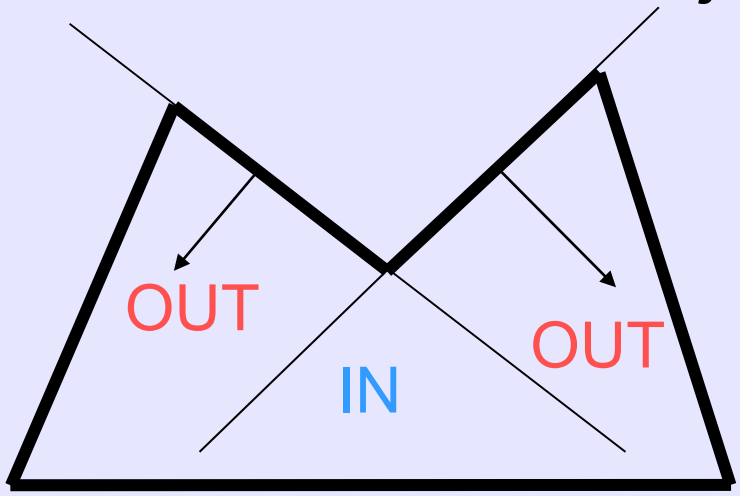
Find in-plane normal to each edge, dot it with (pt,vertex) rays. ALL such dot products will be of same sign for points inside (or outside).

Half Space Tests

Easily computable:

$$l(x, y) = ax + by + c < 0$$ Iff (x,y) is inside

Doesn't work on concave objects!! (triangulate)



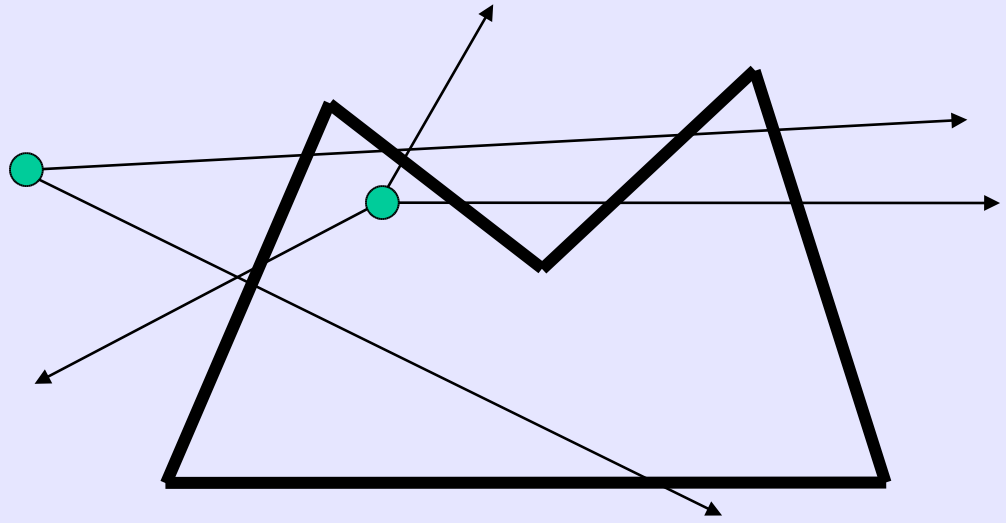
Half Space Tests

lineEq computes the implicit line value for 2 vertices & a point

```
fillTriangle(vertex vList[3])
   //-- get the bounding box as before --//
   float e1 = lineEq(vList[0],vList[1],xmin,ymin);
   float e2 = lineEq(vList[1],vList[2],xmin,ymin);
   float e3 = lineEq(vList[2],vList[0],xmin,ymin);
   int xDim = xMax - xMin;

   for(int y = ymin; y <= ymax; y++)
      for(int x = xmin; x <= xmax; x++)
         if(e1<0 && e2<0 && e3<0)
            setPixel(x,y);
      e1 += a1; e2 += a2; e3+= a3;
   e1 += -xDim*a1+b1; e2 = -xDim*a2+b2; e3 = -xDim*a3+b3
```

Jordan Curve Theorem

Any ray from a point inside a polygon will intersect the polygon's edges an odd number of times

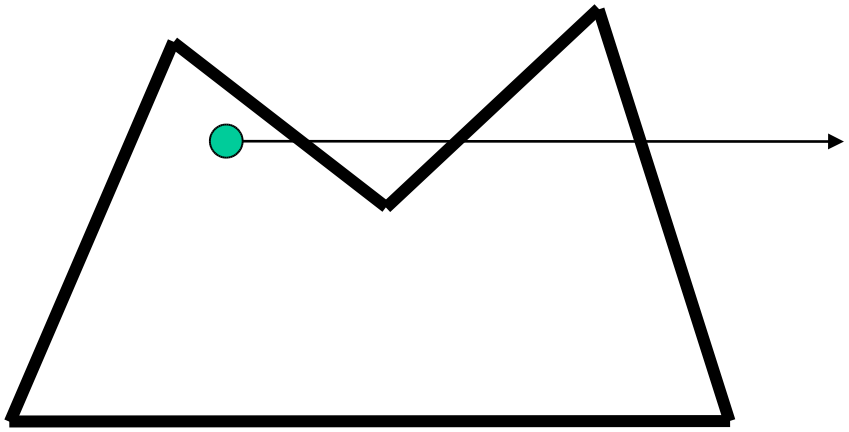


What if it goes through a vertex?

Jordan Curve Theorem

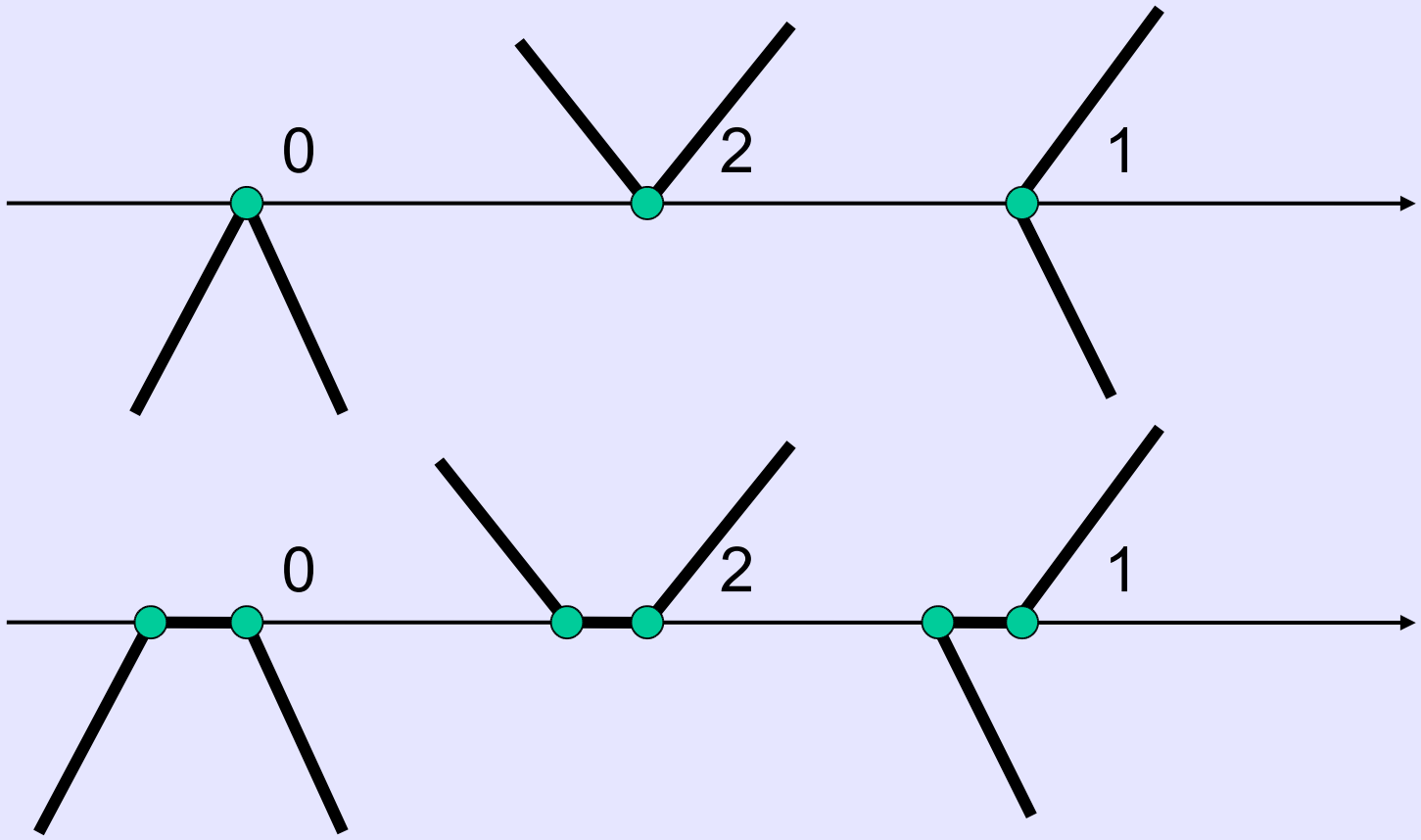
vList is an ordered list of the n polygon vertices

```
int jordanInside(vertex vList[ ], int n, float x, float y)
  int cross = 0;
  float x0, y0, x1, y1;

  x0 = vList[n-1].x – x;     y0 = vList[n-1].y – y;
  for(int i = 0; i < n; i++)
    x1 = vList[i].x – x;      y1 = vList[i].y – y;
    if(y0 > 0)
      if(y1 <= 0)
        if( x1*y0 > y1*x0)
          cross++
    else
      if(y1 > 0)
        if( x0*y1 > y0*x1)
          cross++
    x0 = x1; y0 = y1;

  return cross & 1;
```



Jordan Curve Theorem

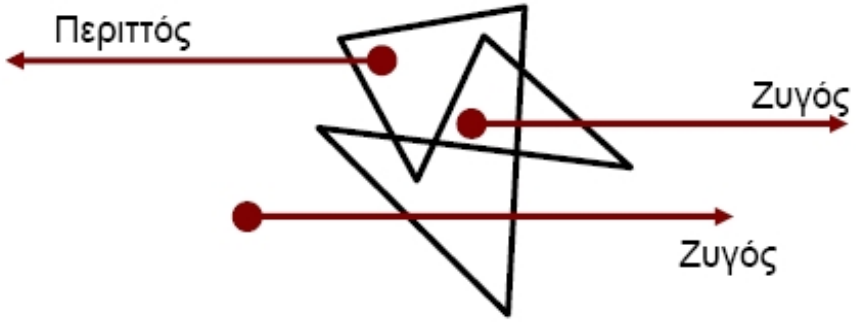
What if it goes through a vertex? (use half open intervals)



Jordan Curve Theorem

Άθροισμα πλευρών (γενικά πολύγωνα)

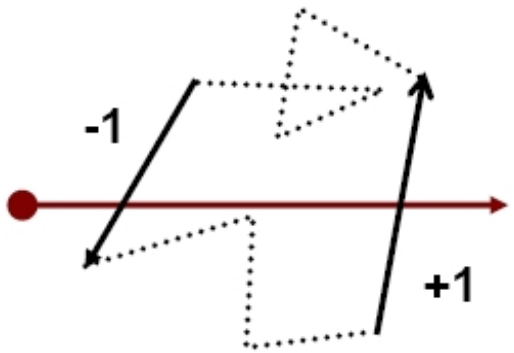
- Odd-even rule:
  - Στέλλουμε μια ακτίνα στο άπειρο προς οποιαδήποτε κατεύθυνση
  - Hit test - inside or outside based on whether number of intersected edges is even or odd



Jordan Curve Theorem

Άθροισμα πλευρών (γενικά πολύγωνα)

- Non-zero winding rule:
  - Draw a line from the test point to the outside
  - Count +1 if you cross an edge in an anti-clockwise sense
  - Count -1 if you cross and edge in a clockwise sense

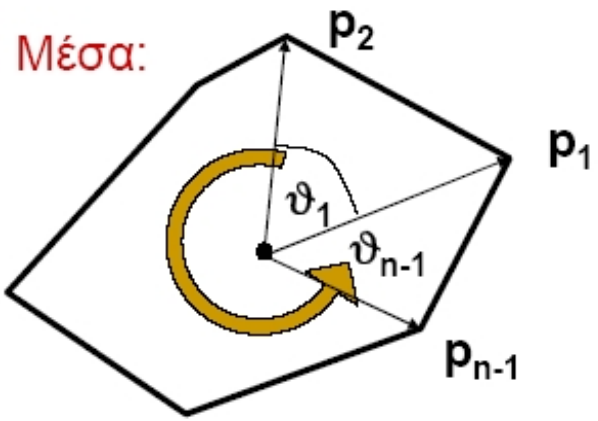


Winding number

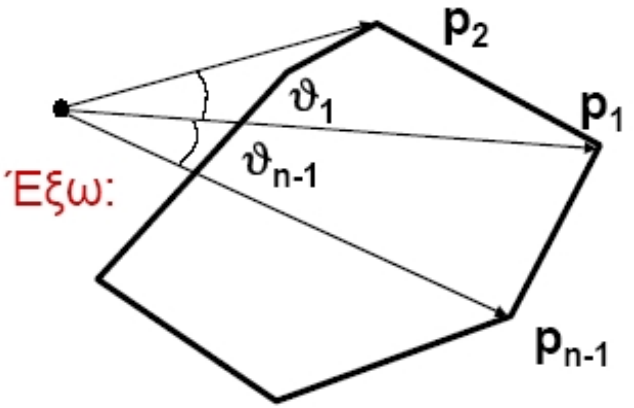
Άθροισμα γωνιών (κυρτά πολύγωνα)

- Sum the angle subtended by the vertices

Μέσα:



Έξω:



$$\sum_{i=1}^{n} \vartheta_i = 2\Pi$$

$$\sum_{i=1}^{n} \vartheta_i \neq 2\Pi$$

Next class:

* Crow's algorithm
* Flood fill