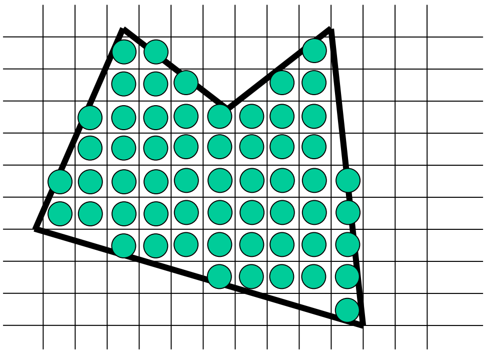
Lecture 6

Rasterizing polygons

1

Rasterizing Polygons

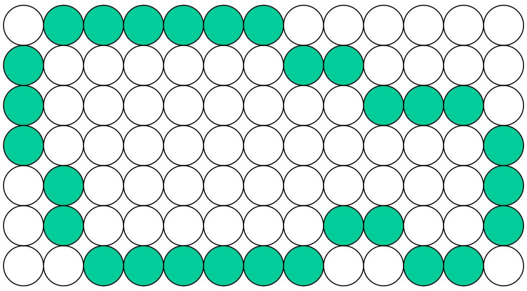
Given a set of vertices and edges,
find the pixels that fill the polygon.



2

Flood Fill

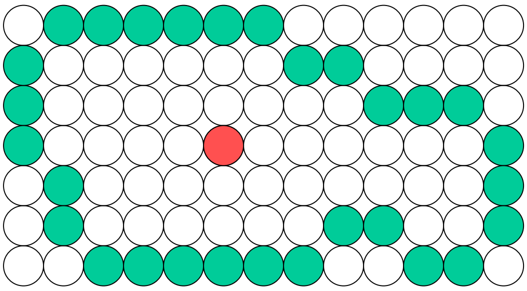
First, rasterizing its edges into the frame buffer using Bresenham’ s algorithm



How to fill polygons whose edges are already drawn?

Flood Fill

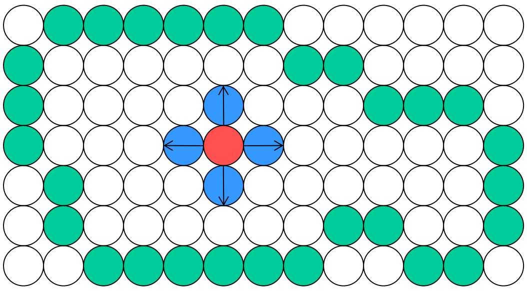
First, rasterizing its edges into the frame buffer using Bresenham’ s algorithm



Choose a point inside, and fill outwards

Flood Fill

First rasterizing its edges into the frame buffer using Bresenham’ s algorithm



Choose a point inside, and fill outwards

Flood Fill

Fill a point and recurse to all of its neighbors

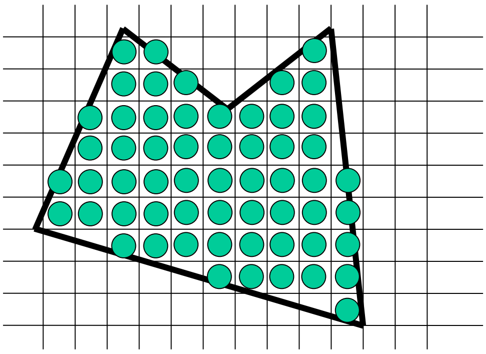
```
floodFill(int x, int y, color c)
   if(stop(x,y,c))
      return;

   setPixel(x,y,c);
   floodFill(x-1,y,c);
   floodFill(x+1,y,c);
   floodFill(x,y-1,c);
   floodFill(x,y+1,c);

int stop(int x, int y, color c)
   return colorBuffer[x][y] == c;
```

7

8

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?

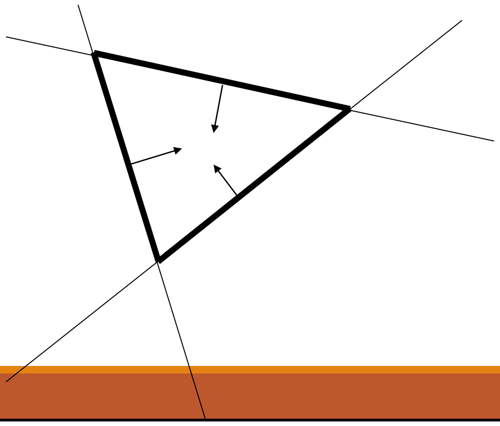
How to test if a point is inside a polygon

1. Half-space tests
2. Jordan Curve Theorem (even/odd or +1/-1)
3. Winding number test

11

Half Space Tests

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

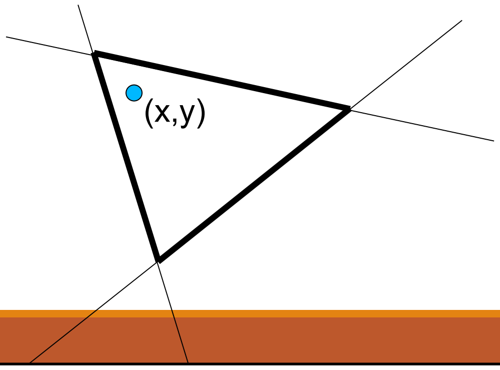


13

Half Space Tests

Easily computable:

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



Half Space Tests

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

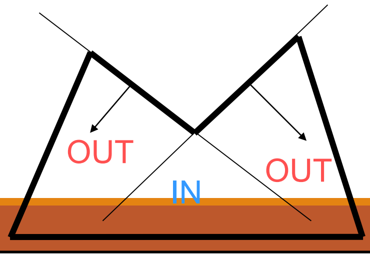
```
fillTriangle(vertex vList[3])
  //-- get the bounding box as before --//
  float e1 = lineEq(vList[0],vList[1],xmin,ymin);
  float e2 = lineEq(vList[1],vList[2],xmin,ymin);
  float e3 = lineEq(vList[2],vList[0],xmin,ymin);
  int xDim = xMax - xMin;

  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
```

Half Space Tests

Easily computable:

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

Doesn’t work on concave objects!!
→ triangulate



What does 'inside' mean?

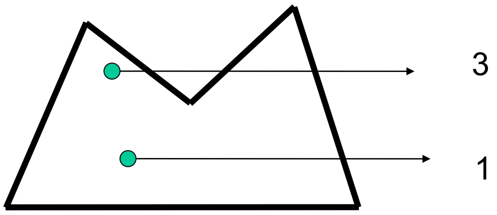
How to test if a point is inside a polygon

1. Half-space tests
2. Jordan Curve Theorem (even/odd or +1/-1)
    - Self-intersecting polygon OK
3. Winding number test

Jordan Curve Theorem

Even/odd approach

Hit test: inside or outside based on the number of intersected edges is even or odd

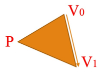


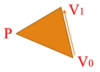
Any ray from a point inside a polygon will intersect the polygon's edges an odd number of times

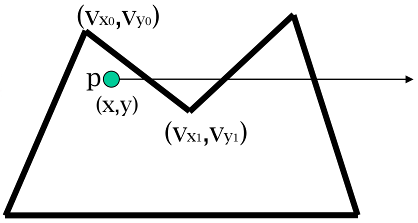
Jordan Curve Theorem

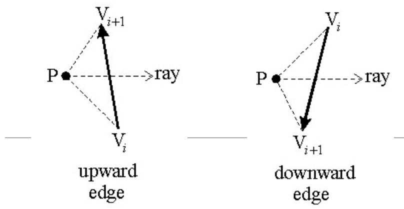
vList is an ordered list of the n polygon vertices

```
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;
```



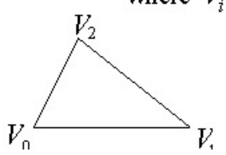






where $V_i = (x_i, y_i)$

$$2A(\Delta) = \begin{vmatrix} (x_1 - x_0) & (x_2 - x_0) \\ (y_1 - y_0) & (y_2 - y_0) \end{vmatrix} = \begin{vmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix}$$
$$= (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$$

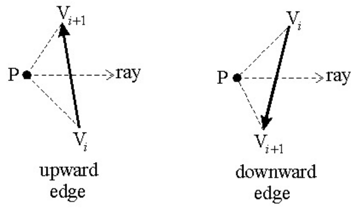


i.e., x0y1-x1y0

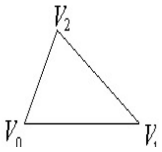
Vi=(xi,yi)

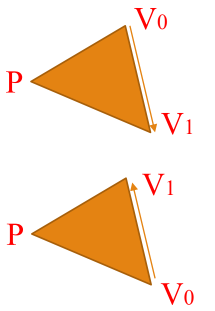
The signed area will be
-positive if the triangle *is* oriented counterclockwise
-negative if the triangle is oriented clockwise

http://geomalgorithms.com/a03-_inclusion.html
http://geomalgorithms.com/a01-_area.html



$$2A(\Delta) = \begin{vmatrix} (x_1 - x_0) & (x_2 - x_0) \\ (y_1 - y_0) & (y_2 - y_0) \end{vmatrix} = \begin{vmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix}$$
$$= (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$$





2A = x0y1-x1y0

The signed area will be
-Positive, if the triangle *is* oriented counterclockwise
- x0y1>x1y0

-negative, if the triangle is oriented clockwise
- x1y0 >x0y1

where
x0 = v0.x – p.x;
y0 = v0.y – p.y;

2Area = x0y1-x1y0

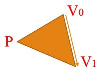
A> 0, if the triangle *is* oriented counterclockwise

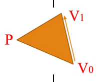
A< 0, if the triangle is oriented clockwise

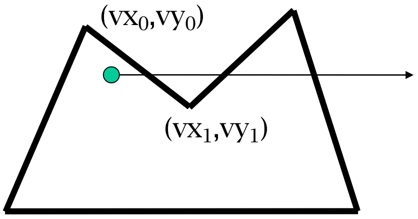
Jordan Curve Theorem

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;
```





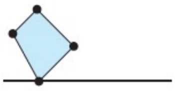


Jordan Curve Theorem

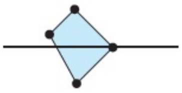
What if it goes through a vertex?

Treat these two cases differently:

- case (a): the vertex–scanline intersection is counted as either zero or two edge crossings
- case (b): the vertex–scanline intersection must be counted as one edge crossing.



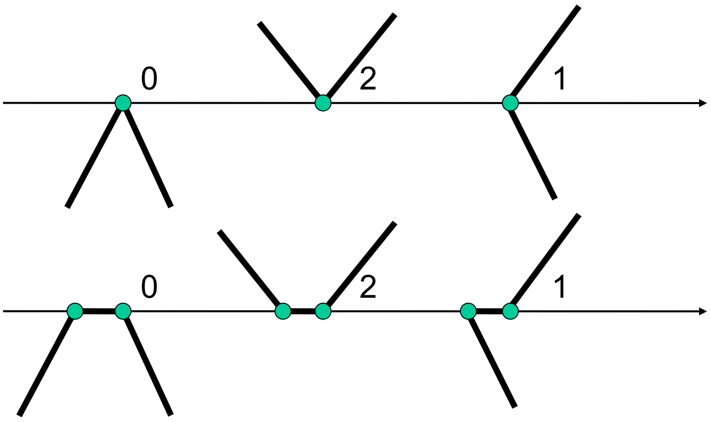
(a)



(b)

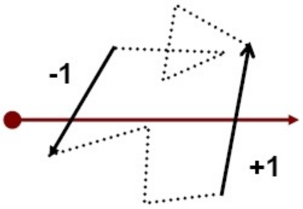
Jordan Curve Theorem

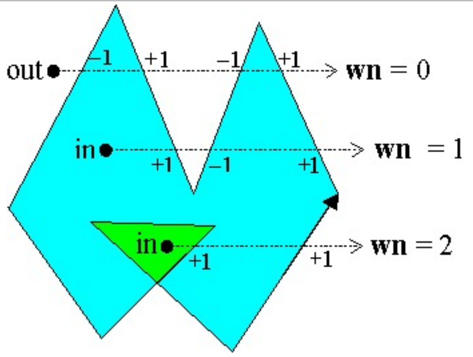
What if it goes through a vertex?

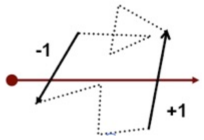


Jordan Curve Theorem

- 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 an edge in a clockwise sense







count +1, if you cross an edge in an anti-clockwise sense
count -1, if you cross an edge in an clockwise sense
Inside point : non-zero

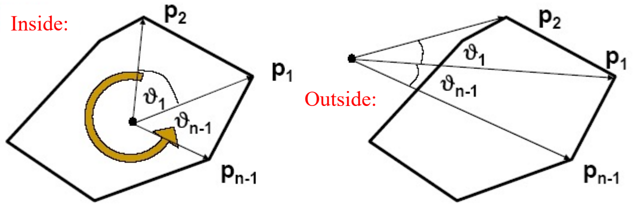
What does 'inside' mean?

How to test if a point is inside a polygon

1. Half-space tests
2. Jordan Curve Theorem (even/odd or +1/-1)
3. Winding number test

Winding Number Test-Method I

- Sum the angle subtended by the vertices

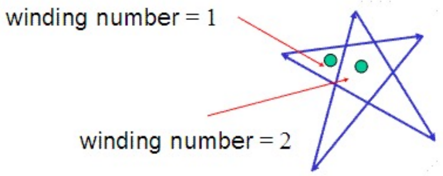


$$\sum_{i=1}^{n} \vartheta_i = 2\Pi \qquad \sum_{i=1}^{n} \vartheta_i \neq 2\Pi$$

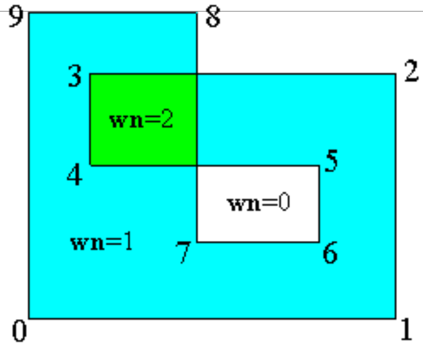
Winding Number Test – Method II

The number of times it is encircled by the edges of the polygon

- +1: if clockwise encirclements
- -1 : if counterclockwise encirclements
- a point is inside the polygon if its winding number is not zero



Winding Number Test II



+1: if clockwise encirclements
-1 : if counterclockwise encirclements
a point is inside the polygon if its winding number is not zero