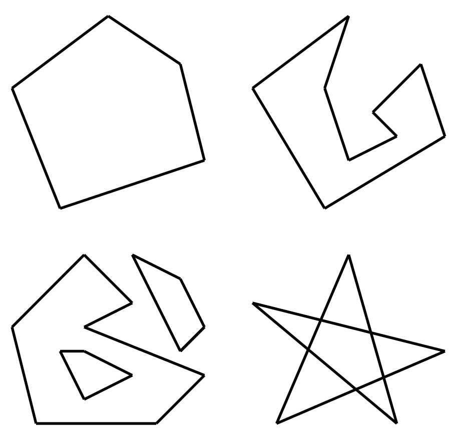
Scan Conversion: Polygons

Lines (last lecture)

Polygons

- **Types of polygons**
- **Inside tests**
- **Jordan Curve Theorem**
- **Singularities**
- **Scanline algorithms**

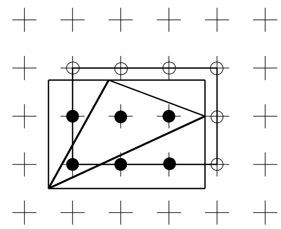
Types of Polygons



Types

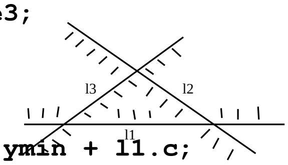
- **•Triangles**
- **•Trapezoids**
- **•Quadrilaterals**
- **•Convex**
- **•Concave**
- **•Self-intersecting**
- **•General**
  - **Multiple loops**
  - **Holes**

Triangle Tiler



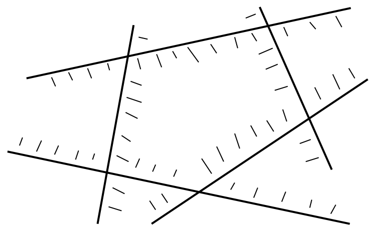
```
tile3( vert v[3] )
{
  int x, y;
  bbox b;

  bound3(v,&b);
  for( y=b.ymin; y<b.ymax, y++ )
    for( x=b.xmin; x<b.xmax, x++ )
      if( inside3(v,x,y) )
        point(x,y);
}
```


Incremental Triangle Tiler

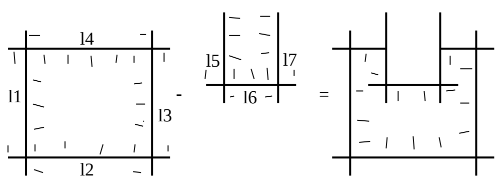



```
  edge l1, l2, l3; int e1, e2, e3;

  mkedge(&v[0],&v[1],&l2);
  mkedge(&v[1],&v[2],&l0);
  mkedge(&v[2],&v[0],&l1);
  e1 = l1.a * b.xmin + l1.b * b.ymin + l1.c;
  e2 = l2.a * b.xmin + l2.b * b.ymin + l2.c;
  e3 = l3.a * b.xmin + l3.b * b.ymin + l3.c;
  for( y=b.ymin; y<b.ymax, y++ ) {
    for( x=b.xmin; x<b.xmax, x++ ) {
      if( e1<=0 && e2<=0 && e3 <= 0 ) point(x,y);
      e1 += l1.a; e2 += l2.a; e3 += l3.a;
    }
    e1 += l1.a * (b.xmin - b.xmax) + l1.b; ...
}
```

Constructive Solid Geometry

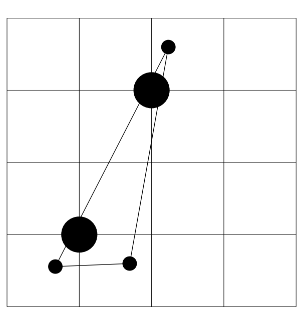


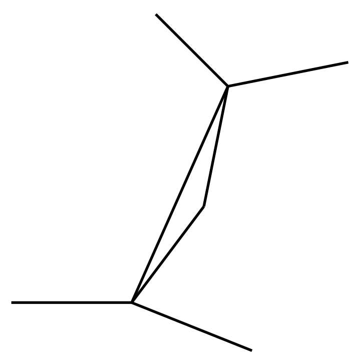
$$l_1 \cap l_2 \cap \cdots \cap l_n$$



$$\left(l_1 \cap l_2 \cap l_3 \cap l_4\right) - \left(l_5 \cap l_6 \cap l_7\right)$$

Gaps and Dinks





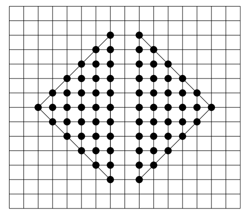
Singularities

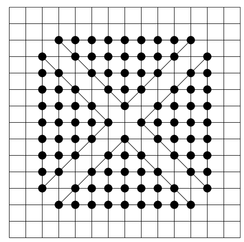
Double hits (e <= 0)

- wasted effort
- transparency
- csg

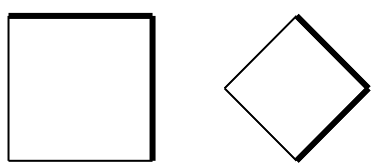
Gaps (e < 0)

- missing pixels





Shadow Test

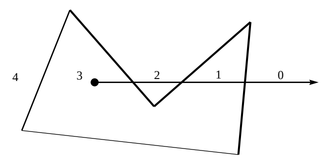


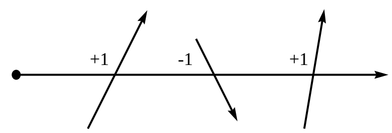
```
int shadow( int a, int b ) {
  return (a>0) || (a==0 && b > 0);
}

int inside( int e, int a, int b, int c ) {
  return (e == 0) ? !shadow(a,b) : (e < 0);
}
```

Jordan Curve Theorem

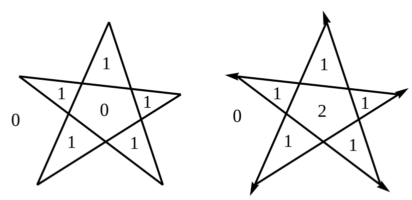
Two definitions of inside

- **Even-Odd parity**
- **Winding number**





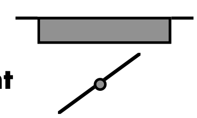
Example:

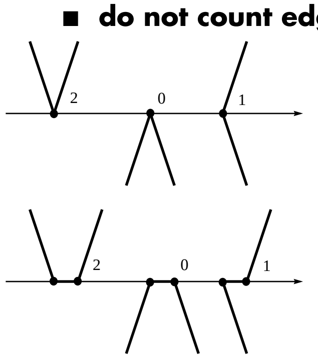


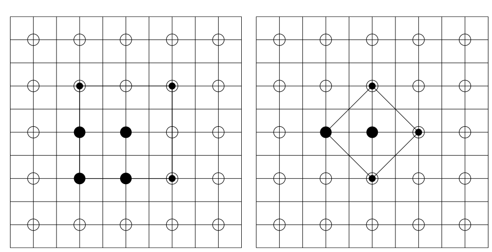
Singularities

Resolve singularities

- **horizontal line part of lower half space**
- **do not count edges passing through point**







Point in Polygon

```
int inside( vert v[], int n, int x, int y ) {
  int cross = 0; int x0, y0, x1, y1;
  x1 = v[n-1].x - x; y1 = v[n-1].y - y;
  while( n-- ) {
    x0 = x1;       y0 = y1;
    x1 = v->x - x; y1 = v->y - y;
    if( y0 > 0 ) {
      if( y1 <= 0 )
        if( x1*y0 > y1*x0 ) cross++;
    }
    else {
      if( y1 > 0 )
        if( x0*y1 > y0*x1 ) cross++;
    }
  }
  return cross & 1;
```

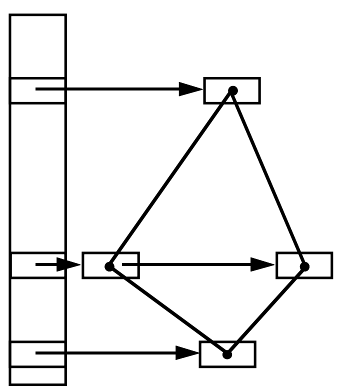

Scanline Algorithm

Incrementalize the Jordan Test

1. For each polygon vertex

A. Bucket sort by y value

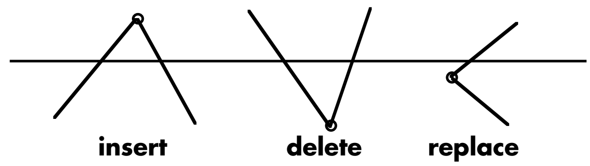
B. Sweep downward



Scanline Algorithm

2. For each scanline

A. Maintain *active edge list* (for crossings count)

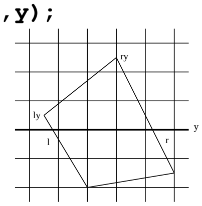


B. Sort active edge list by x

C. Sweep leftward (updating crossings count)

Crow's Algorithm for Convex Polygons

```
scany( vert v[], int n, int bot)
  li = ri = bot;
  ly = ry = y = ceil(v[bot].y);
  for( rem = n; rem > 0 ; ) {
    while( ly <= y && rem-- > 0 ) {
      i = li; li = mod(li-1,n); ly = ceil(v[li].y);
      if( ly > y )
        differencey(&v[i],&v[li],&l,&dl,y);
    }
    while( ry <= y && rem-- > 0 ) {
      i = ri; ri = mod(ri+1,n); ry = ceil(v[ri].y);
      if( ry > y )
        differencey(&v[i],&v[ri],&r,&dr,y);
    }
    for( ; y < ly && y < ry; y++ ) {
      scanx(&l,&r,y);
      increment(&l,&dl);
      increment(&r,&dr);
    }
```



Crow's Algorithm for Convex Polygons

```
scanx( vert *l, vert *r, int y)
  lx = ceil(l->x);
  rx = ceil(r->x);
  if( lx < rx ) {
    differencex( l, r, &s, &ds, lx );
    for( x=lx; x<rx; x++ ) {
      point(x,y);
      increment(&s,&ds);
    }
  }

differencex(vert *l,vert *r, vert *s,vert *ds, int x)
{
  ds->v = (r->v - l->v)/ (r->x - l->x);
   s->v = l->v + (x - l->x) * de->v;
}
differencey(vert *b,vert *t, vert *e,vert *de, int y)
{
  de->v = (t->v - b->v)/ (t->y - b->x);
   e->v = b->v + (x - b->x) * de->v;
}
```