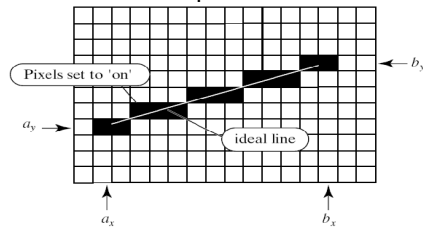


Converting to Pixels

- We want to be able to draw lines and filled polygons on a raster display, and so must approximate them with pixels



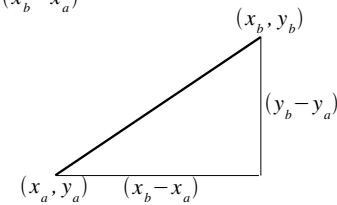
COMP3421: Computer Graphics - Pixels

Slide 1

homeofambert.org/graphics/slides/2D/pixels/COMP3421-pixels.odp

Equation of a line

- $y = mx + b$
- $m = (y_b - y_a) / (x_b - x_a)$
 $b = y_a - mx_a$



COMP3421: Computer Graphics - Pixels

Slide 2

homeofambert.org/graphics/slides/2D/pixels/COMP3421-pixels.odp

DDA Algorithm

```
void drawLine(int xa, ya, xb, yb) {  
    float m = (yb - ya) / (float) (xb - xa);  
    float b = ya - m*xa;  
    for (x = xa; x <= xb; x++) {  
        y = m*x+b;  
        drawPixel(x, Math.round(y));  
    }  
}
```

COMP3421: Computer Graphics - Pixels

Slide 3

homeofambert.org/graphics/slides/2D/pixels/COMP3421-pixels.odp

Problems with DDA

- Floating point arithmetic is slower
- Line comes apart if $m > 1$

COMP3421: Computer Graphics - Pixels

Slide 4

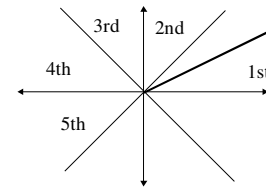
homeofambert.org/graphics/slides/2D/pixels/COMP3421-pixels.odp

Bresenham's Algorithm

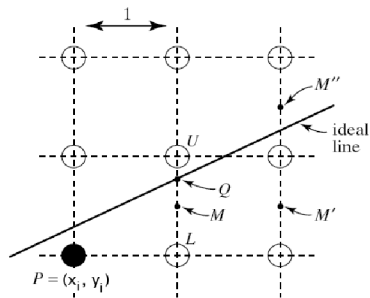
- Draws a line only using integer calculations and no multiplications
- Suitable for hardware implementation
- Key idea: do the calculations INCREMENTALLY

Bresenham's Algorithm

- Assume that line is in 1st Octant.
- When going from one pixel to next must choose between going Left and Up and left



Bresenham

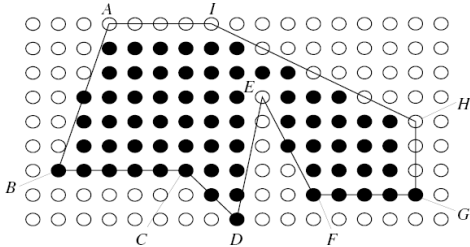


Bresenham

- Need to choose between R and U
- OHP

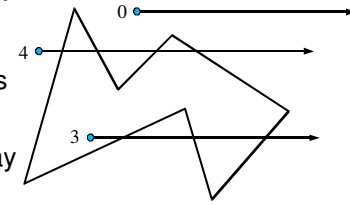
Polygon filling

- Need to calculate what pixels fit inside polygon



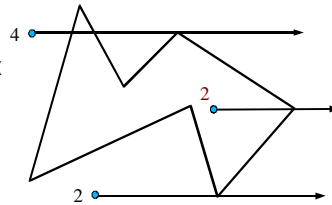
Point in polygon algorithm

- For each pixel, calculate whether it is inside polygon
- Count intersections of polygon edges with a horizontal ray
- Even \Rightarrow outside
- Odd \Rightarrow inside



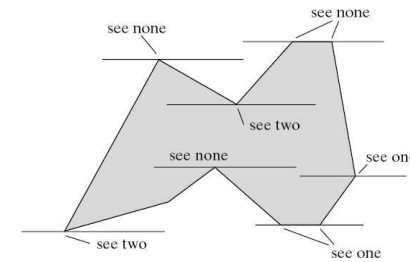
What if ray intersects a vertex?

- Count is sometimes incorrect if ray intersects a vertex



Ray vertex intersection

- Fix is to leave the topmost pixel of each edge.

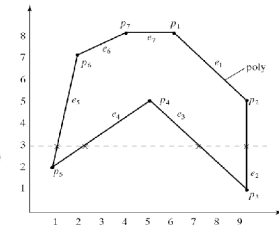


Ray Vertex intersection code

```
boolean rayIntersect(int x, int y) {  
    if (y >= y2) return false;  
    if (y < y1) return false;  
    return (y-b)/m > x;  
}
```

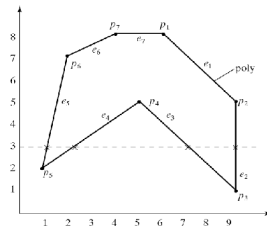
Scan line algorithm

- Checking every pixel is very inefficient
- Make it INCREMENTAL
- Result of point in pixel test only changes when we cross a polygon edge



Scan line polygon intersection

- Calculate intersections between polygon and scan line
- Sort by x coordinate
- Fill in runs of pixels between successive pairs of intersections.



Active edge list

- Also calculate intersections of polygon with scan list incrementally
- Store intersecting edges in the *Active Edge List* sorted by x intersection.
- AEL only changes when scan line crosses a vertex

Updating AEL

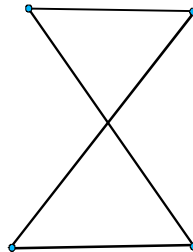
- Edges are removed when top pixel equals scan line
- Edges are added when bottom pixel equals scan line
- Keep edges to be added in an Inactive Edge List sorted by bottom pixel

Scan Line algorithm

```
for (y = iel.miny(); y <= iel.maxy(); y++){
  while ((Edge e=iel.next(y))!=null) {
    ael.add(e,y);
  }
  ael.deleteEdges(y);
  for (i = 0; i < ael.size(); i += 2) {
    xstart = ael.elementAt(i).xofi(y);
    xend = ael.elementAt(i+1).xofi(y);
    fillSpan(xstart, xend, y);
  }
}
```

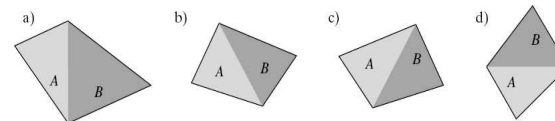
Bowtie polygons

- If polygon edges intersect AEL must be resorted at each scan line
- Bowtie polygons are evil



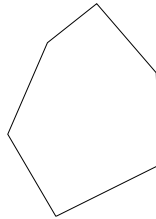
Pixels on shared edges

- Two adjacent polygons should fit together with no gaps or overlaps
- Assign pixels on shared edge to rightmost polygon (B in each example below).



Polygon filling in OpenGL

- Hardware likes fixed-sized data structures, so scan-line algorithm is not suitable for hardware
- GL_POLYGON in OpenGL must be convex
- With convex polygons AEL always has exactly two edges



GluTess

- To deal with concave and self-intersecting polygons OpenGL provides utility functions beginning with GluTess that divide a polygon up into triangles

A quadrilateral with a triangular hole in it can be described as follows:

```
tobj = gluNewTess();
gluTessBeginPolygon(tobj, NULL);
gluTessBeginContour(tobj);
gluTessVertex(tobj, v1, v1);
gluTessVertex(tobj, v2, v2);
gluTessVertex(tobj, v3, v3);
gluTessVertex(tobj, v4, v4);
gluTessEndContour(tobj);
gluTessBeginContour(tobj);
gluTessVertex(tobj, v5, v5);
gluTessVertex(tobj, v6, v6);
gluTessVertex(tobj, v7, v7);
gluTessEndContour(tobj);
gluTessEndPolygon(tobj);
```