Scan Conversion

Wolfgang Heidrich

Course News

Assignment 2

- Due Monday, Feb 28

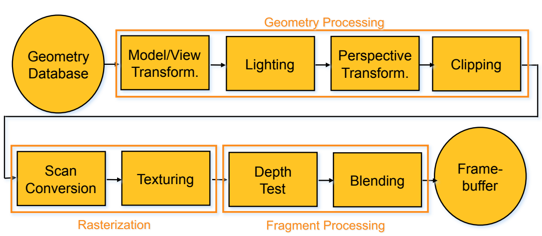
Homework 3

- Discussed in labs next wee

Reading

- Chapter 3 (this week)
- Chapter 8 (next week)

The Rendering Pipeline



Course News

Assignment 2

- Due March 2

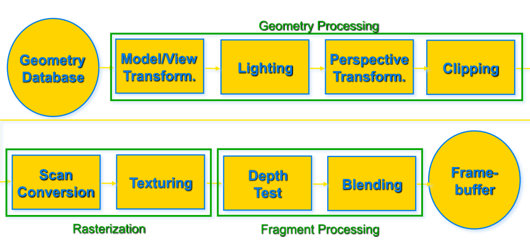
Homework 3

- Discussed in labs next week

Reading

- Chapter 3

The Rendering Pipeline

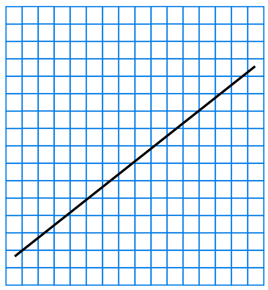


Scan Conversion - Rasterization

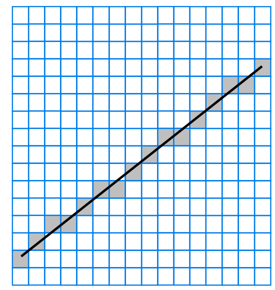
Convert continuous rendering primitives into discrete fragments/pixels

- Lines
  - *Midpoint/Bresenham*
- Triangles
  - *Flood fill*
  - *Scanline*
  - *Implicit formulation*
- Interpolation

Scan Conversion - Lines



Scan Conversion - Lines



Scan Conversion - Lines

First Attempt:

- Line (s,e) given in device coordinates
- Create the thinnest line that connects start point and end point without gap

Assumptions for now:

- Start point to the left of end point: xs< xe
- Slope of the line between 0 and 1 (I.e. elevation between 0 and 45 degrees:

$$0 \leq \frac{ye - ys}{xe - xs} \leq 1$$

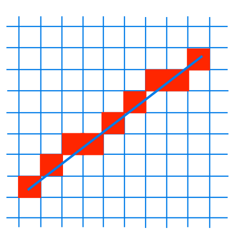
Scan Conversion of Lines - Digital Differential Analyzer

First Attempt:

```
dda( float xs, ys, xe, ye ) {
  // assume xs < xe, and slope m between 0 and 1
  float m= (ye-ys)/(xe-xs);
  float y= round( ys );
  for( int x= round( xs ) ; x<= xe ; x++ ) {
    drawPixel( x, round( y ) );
    y= y+m;
  }
}
```


Scan Conversion of Lines

DDA:



Scan Conversion of Lines Midpoint Algorithm

Moving horizontally along x direction

- Draw at current y value, or move up vertically to y+1?
  - *Check if midpoint between two possible pixel centers above or below line*

Candidates

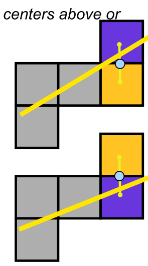
- Top pixel: (x+1,y+1)
- Bottom pixel: (x+1, y)

Midpoint: (x+1, y+.5)

Check if midpoint above or below line

- Below: top pixel
- Above: bottom pixel

Key idea behind Bresenham Alg.



Scan Conversion of Lines

Idea: decision variable

```
dda( float xs, ys, xe, ye ) {
    float d= 0.0;
    float m= (ye-ys)/(xe-xs);
    int y= round( ys );
    for( int x= round( xs ) ; x<= xe ; x++ ) {
        drawPixel( x, y );
        d= d+m;
        if( d>= 0.5 ) { d= d-1.0; y++; }
    }
}
```

Wolfgang Heidrich

Scan Conversion of Lines Bresenham Algorithm ('63)

- Use decision variable to generate purely integer algorithm
- Explicit line equation:

$$y = \frac{(y_e - y_s)}{(x_e - x_s)}(x - x_s) + y_s$$

- Implicit version:

$$L(x, y) = \frac{(y_e - y_s)}{(x_e - x_s)}(x - x_s) - (y - y_s) = 0$$

- In particular for specific x, y, we have
  - *L(x,y)>0 if (x,y) below the line, and*
  - *L(x,y)<0 if (x,y) above the line*

Wolfgang Heidrich

Scan Conversion of Lines Bresenham Algorithm

- Decision variable: after drawing point (x,y) decide whether to draw
  - $(x+1,y)$: *case E (for "east")*
  - $(x+1,y+1)$: *case NE (for "north-east")*
- Check whether $(x+1,y+1/2)$ is above or below line

$$d = L(x+1, y+\frac{1}{2})$$

- Point above line if and only if $d<0$

Wolfgang Heidrich

Scan Conversion of Lines

Bresenham Algorithm

- Problem: how to update d?
- Case E (point above line, $d<= 0$)
  - *x= x+1;*
  - $d= L(x+2, y+1/2)= d+ (y_e-y_s)/(x_e-x_s)$
- Case NE (point below line, $d> 0$)
  - *x= x+1; y= y+1;*
  - $d= L(x+2, y+3/2)= d+ (y_e-y_s)/(x_e-x_s) -1$
- Initialization:
  - $d= L(x_s+1, y_s+1/2)= (y_e-y_s)/(x_e-x_s) - 1/2$

Wolfgang Heidrich

Scan Conversion of Lines

Bresenham Algorithm

- This is still floating point
- But: only sign of d matters
- Thus: can multiply everything by $2(x_e-x_s)$

Wolfgang Heidrich

Scan Conversion of Lines

Bresenham Algorithm

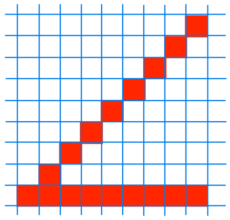
```
Bresenham( int xs, ys, xe, ye ) {
    int y= ys;
    incrE= 2(ye - ys);
    incrNE= 2((ye - ys) – (xe-xs));
    for( int x= xs ; x<= xe ; x++ ) {
        drawPixel( x, y );
        if( d<= 0 ) d+= incrE;
        else { d+= incrNE; y++; }
    }
}
```

Wolfgang Heidrich

Scan Conversion of Lines

Discussion

- Bresenham sets same pixels as DDA
- Intensity of line varies with its angle!

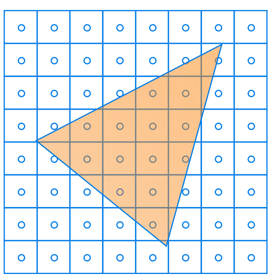


Scan Conversion of Lines

Discussion

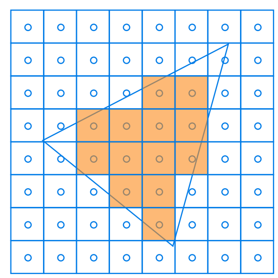
- Bresenham
  - *Good for hardware implementations (integer!)*
- DDA
  - *May be faster for software (depends on system)!*
  - *Floating point ops higher parallelized (pipelined)*
    - E.g. RISC CPUs from MIPS, SUN
  - *No if statements in inner loop*
    - More efficient use of processor pipelining

Scan Conversion of Polygons



Scan Conversion of Polygons

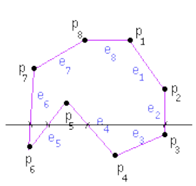
One possible scan conversion



Scan Conversion of Polygons

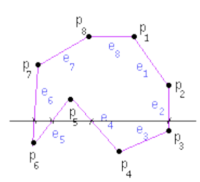
A General Algorithm

- Intersect each scanline with all edges
- Sort intersections in x
- Calculate parity to determine in/out
- Fill the ‘in’ pixels



Scan Conversion of Polygons

- Works for arbitrary polygons
- Efficiency improvement:
  - *Exploit row-to-row coherence using “edge table”*

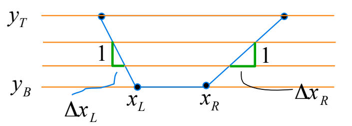


Edge Walking

Past graphics hardware

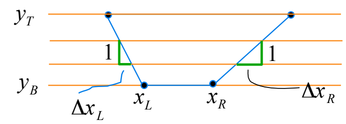
- Exploit continuous L and R edges on trapezoid

scanTrapezoid$(x_L, x_R, y_B, y_T, \Delta x_L, \Delta x_R)$



Edge Walking

```
for (y=yB; y<=yT; y++) {
 for (x=xL; x<=xR; x++)
     setPixel(x,y);
 xL += DxL;
 xR += DxR;
}
```

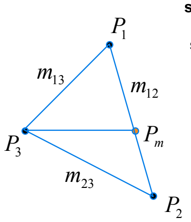


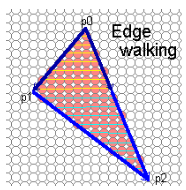
Edge Walking Triangles

- Split triangles into two regions with continuous left and right edges

scanTrapezoid$(x_3, x_m, y_3, y_1, \frac{1}{m_{13}}, \frac{1}{m_{12}})$

scanTrapezoid$(x_2, x_2, y_2, y_3, \frac{1}{m_{23}}, \frac{1}{m_{12}})$





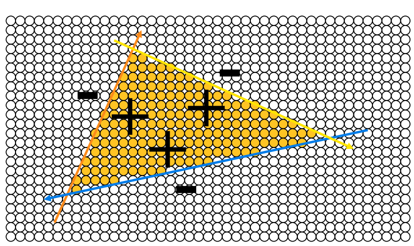
Edge Walking Triangles

Issues

- Many applications have small triangles
  - *Setup cost is non-trivial*
- Clipping triangles produces non-triangles
  - *This can be avoided through re-triangulation, as discussed*

Modern Rasterization: Edge Equations

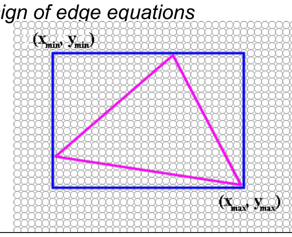
Define a triangle as follows:



Using Edge Equations

Usage:

- Go over each pixel in bounding rectangle
- Check if pixel is inside/outside of triangle
  - *Using sign of edge equations*



Computing Edge Equations

Implicit equation of a triangle edge:

$$L(x,y) = \frac{(y_e - y_s)}{(x_e - x_s)}(x - x_s) - (y - y_s) = 0$$

(see Bresenham algorithm)

- L(x,y) positive on one side of edge, negative on the other

Question:

- What happens for vertical lines?

Edge Equations

Multiply with denominator

$$L(x,y) = (y_e - y_s)(x - x_s) - (y - y_s)(x_e - x_s) = 0$$

- Avoids singularity
- Works with vertical lines

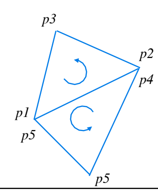
What about the sign?

- Which side is in, which is out?

Edge Equations

Determining the sign

- Which side is "in" and which is "out" depends on order of start/end vertices…
- Convention: specify vertices in counter-clockwise order



Edge Equations

Counter-Clockwise Triangles

- The equation L(x,y) as specified above is *negative inside, positive outside*
  - *Flip sign:*

$$L(x,y) = -(y_e - y_s)(x - x_s) + (y - y_s)(x_e - x_s) = 0$$

Clockwise triangles

- *Use original formula*

$$L(x,y) = (y_e - y_s)(x - x_s) - (y - y_s)(x_e - x_s) = 0$$

Discussion of Polygon Scan Conversion Algorithms

On old hardware:

- Use first scan-conversion algorithm
  - *Scan-convert edges, then fill in scanlines*
  - *Compute interpolated values by interpolating along edges, then scanlines*
- Requires clipping of polygons against viewing volume
- Faster if you have a few, large polygons
- Possibly faster in software

Discussion of Polygon Scan Conversion Algorithms

Modern GPUs:

- Use edge equations
  - *And plane equations for attribute interpolation*
  - *No clipping of primitives required*
- Faster with many small triangles

Additional advantage:

- Can control the order in which pixels are processed
- Allows for more memory-coherent traversal orders
  - *E.g. tiles or space-filling curve rather than scanlines*

Triangle Rasterization Issues (Independent of Algorithm)

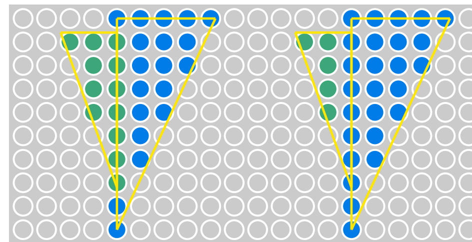
Exactly which pixels should be lit?
- A: Those pixels inside the triangle edge (of course)

But what about pixels exactly on the edge?
- Draw them: order of triangles matters (it shouldn't)
- Don't draw them: gaps possible between triangles

We need a consistent (if arbitrary) rule
- Example: draw pixels on left or top edge, but not on right or bottom edge

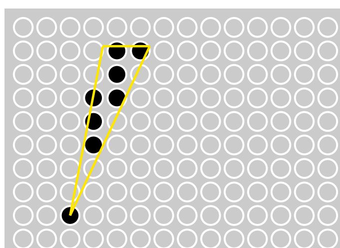
Triangle Rasterization Issues

Shared Edge Ordering



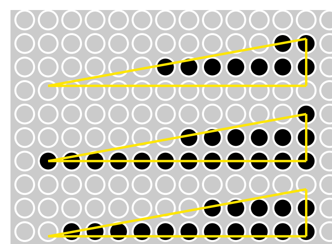
Triangle Rasterization Issues

Sliver



Triangle Rasterization Issues

Moving Slivers



Triangle Rasterization Issues

These are ALIASING Problems
- Problems associated with representing continuous functions (triangles) with finite resolution (pixels)
- More on this problem when we talk about sampling…

Coming Up:

Monday
- Scan conversion / shading

Wednesday/Friday
- Clipping, hidden surface removal