**CS559 – Lecture 6 (part b)**

**Raster Algorithms**

These are course notes (not used as slides)
Written by Mike Gleicher, Sept. 2005
With some slides adapted from the notes of Stephen Chenney

---

**Geometric Graphics**

- Mathematical descriptions of sets of points
  - Rather than sampled representations
- Ultimately, need sampled representations for display
- Rasterization

- Usually done by low-level
  - OS / Graphics Library / Hardware
  - Hardware implementations counter-intuitive
    - Modern hardware doesn’t work anything like what you’d expect

---

**Drawing Points**

- What is a point?
  - Position – without any extent
  - Can’t see it – since it has no extent, need to give it some
- Position requires co-ordinate system
  - Consider these in more depth later

- How does a point relate to a sampled world?
  - Points at samples?
  - Pick closest sample?
  - Give points finite extent and use little square model?
  - Use proper sampling

---

**Sampling a point**

- Point is a spike – need to LPF
  - Gives a circle w/roll-off
- Point sample this
- Or…
  - Samples look in circular (kernel shaped) regions around their position
- But, we can actually record a unique “splat” for any individual point

---

**Anti-Aliasing**

- Anti-Aliasing is about avoiding aliasing
  - once you’ve aliased, you’ve lost
- Draw in a way that is more precise
  - E.g. points spread out over regions
- Not always better
  - Lose contrast, might not look even if gamma is wrong, might need to go to binary display, ...

---

**Line drawing**

- Was really important, now, not so important
- Let us replace expensive vector displays with cheap raster ones

- Modern hardware does it differently
  - Actually, doesn’t draw lines, draws small, filled polygons

- Historically significant algorithms

---
Line Drawing (2)

- Consider the integer version
  - \((x_1,y_1) \rightarrow (x_2,y_2)\) are integers
  - Not anti-aliased (binary decision on pixels)
- Naïve strawman version:
  - \(Y = mx + b\)
    
    For \(x = x_1\) to \(x_2\)
    
    \[y = mx + b\]
    
    set\((x,y)\)
- Problems:
  - Too much math (floating point)
  - Gaps

Brezenham’s algorithm (and variants)

- Consider only 1 octant (get others by symmetry)
  - \(0 \geq m \geq 1\)
- Loop over \(x\) pixels
  - Guarantees 1 per column
- For each pixel, either move up 1 or not
  - If you plotted \(x,y\) then choose either \(x+1,y\) or \(x+1,y+1\)
  - Trick: how to decide which one easily
    - Same method works for circles (just need different test)
- Decision variable
  - Implicit equation for line \((d=0\) means on the line)

Midpoint method

In this method, we consider a decision variable \(d\) which is updated at each step. The decision variable helps us decide whether to move up 1 or not. Here are the steps:

1. Initialize \(d_1 = y_1 - b\) and \(d_2 = b - y_1\).
2. For \(x = x_1\) to \(x_2\):
   - If \(d_1 < d_2\), choose \(y_{k+1}\) and update \(d_1 = d_1 + 2 \Delta y - 2 \Delta x (y_k - y_{k+1})\), \(d_2 = d_2 + 2 \Delta x\).
   - If \(d_1 = d_2\), choose \(y_k\) and update \(d_1 = d_1 + 2 \Delta y\).

Derivation

\[\Delta d = d_1 - d_2\]
\[\Delta d = (y-y_k) - (y_{k+1}+1-y)\]
\[y = mx + b\]
\[\Delta d = 2(x_{k+1}+1+b) - 2y_k - 1\]
\[m = \frac{\Delta y}{\Delta x}\]

Incremental Algorithm

- Suppose we know \(p_k\) — what is \(p_{k+1}\)?
- \(p_{k+1} = p_k + 2 \Delta y - 2 \Delta x (y_{k+1} - y_k)\)
  - Since \(x_{k+1} = x_k + 1\)
- And \(y_{k+1} - y_k\) is either 1 or 0, depending on \(p_k\)

Brezenham’s Algorithm

- \(P_{k} = 2 \Delta y + x\)
- \(Y = y_1\)
- For \(X = x_1\) to \(x_2\):
  - Set \(X, Y\)
  - If \(P_k < 0\)
    - \(Y += 1\)
    - \(P_k += 2 \Delta y - 2 \Delta x\)
  - Else: \(P_k += 2 \Delta y\)
Why is this cool?

• No division!
• No floating point!
• No gaps!

• Extends to circles

• But…
  – Jaggies
  – Lines get thinner as they approach 45 degrees
  – Can’t do thick primitives