Today’s Topics

• Questions? Comments?
• Let's go over the example java code for jogl programs that I provided in lab.
• Line drawing (Chapter 6)
Lines

• When drawing lines we must convert the continuous line into discrete pieces --- we need to decide which pixels should be on. We need to calculate the integer coordinates.

• The goals of line drawing are several
  – A main one is that it should be fast, because lines are drawn often
    • That is, we prefer integer arithmetic and additions and subtractions over floating point arithmetic and multiplies and divides.
  – Another is that the lines should look nice --- be straight and have constant thickness, independent of angle, length
Lines

- Recall this information about lines
- Assume we have two known points \((x_1, y_1)\) and \((x_2, y_2)\) on a plane. We can calculate the equation of a line containing those two points if we pick an unknown point \((x, y)\) on the line and compute the slope.
- Slope = \(\frac{y-y_1}{x-x_1} = \frac{y_2-y_1}{x_2-x_1}\)
- Multiply both sides by \((x-x_1)\) to get
  \[
  y-y_1 = \frac{(y_2-y_1)(x-x_1)}{(x_2-x_1)}
  \]
  \[
  y = \frac{(y_2-y_1)(x-x_1)}{(x_2-x_1)} + y_1
  \]
  \[
  y = \left(\frac{(y_2-y_1)}{(x_2-x_1)}\right) x + \left(\frac{-x_1}{(x_2-x_1)}\right) + y_1
  \]
  \[
  y = mx + b
  \]
Parametric Equation of a Line

• This information appears in section 8.7 of text but is appropriate now because it is used in the DDA algorithm (discussed next slide).

• Given 2 points on a line \((x_0, y_0)\) and \((x_{\text{end}}, y_{\text{end}})\)

\[
x = x_0 + u(x_{\text{end}} - x_0)
\]

\[
y = y_0 + u(y_{\text{end}} - y_0)
\]

• \(u\) is the parameter

• If we consider only the values \(0 \leq u \leq 1\), we get all the points on the line segment between \((x_0, y_0)\) and \((x_{\text{end}}, y_{\text{end}})\).

• When \(u = 0\), \(x = x_0\) and \(y = y_0\)

• When \(u = 1\), \(x = x_{\text{end}}\) and \(y = y_{\text{end}}\)
DDA – digital differential analyzer

• Consider a line with positive slope and $x_0 < x_{\text{end}}$
• If slope $\leq 1$ then we will sample the line at unit $x$ intervals and compute $y$.
• If slope $> 1$ then we will sample the line at unit $y$ intervals and compute $x$.
• What does that mean?

• In the DDA, the parameter $u$ (from the parametric equation of a line) is $1/\text{steps}$.
• Let's take a look at the code for this algorithm and “execute” it with actual values on the board.
```c
void lineDDA (int x0, int y0, int xEnd, int yEnd)
{
    int dx = xEnd - x0, dy = yEnd - y0, steps, k;
    float xIncrement, yIncrement, x = x0, y = y0;

    if (fabs (dx) > fabs (dy))
        steps = fabs (dx);
    else
        steps = fabs (dy);
    xIncrement = float (dx) / float (steps);
    yIncrement = float (dy) / float (steps);

    setPixel (round (x), round (y));
    for (k = 0; k < steps; k++) {
        x += xIncrement;
        y += yIncrement;
        setPixel (round (x), round (y));
    }
}
```
DDA – digital differential analyzer

- It creates good lines. The lines aren't going to differ in “thickness” based on angle, etc.
- Can you comment on any efficiency issues with this algorithm?
DDA – digital differential analyzer

• It creates good lines. The lines aren't going to differ in “thickness” based on angle, etc.

• Problems with this algorithm are
  – Rounding is expensive
  – Floating point arithmetic is expensive

• So, a goal is to do all integer arithmetic and reduce / eliminate divides.
Bresenham's Line Algorithm

• Bresenham's Algorithm is efficient (fast) and accurate.
• The key part of Bresenham's algorithm lies in the determination of which pixel to turn on based on whether the line equation would cause the line to lie more near one pixel or the other.

![Diagram](image)

Figure 3-11
Vertical distances between pixel positions and the line $y$ coordinate at sampling position $x_k + 1$.  

Michael Eckmann - Skidmore College - CS 325 - Spring 2012
Bresenham's Line Algorithm

• Think of a line whose slope is >0, but < 1. These are first octant lines (the area of the coordinate plane between the positive x-axis and the line y=x.)
• Assume we have the pixel at \((x_k, y_k)\) turned on and we're trying to determine between the following:
  – Should \((x_{k+1}, y_k)\) or \((x_{k+1}, y_{k+1})\) be on?
  – That is, the choice is between either the pixel one to the right of \((x_k, y_k)\) or the pixel one to the right and one up.
• The decision can be made based on whether \(d_{\text{lower}} - d_{\text{upper}}\) is > 0 or not.
• The goal is to come up with an inexpensive way to determine the sign of \((d_{\text{lower}} - d_{\text{upper}})\)
Bresenham's Line Algorithm

- The y coordinate on the line is calculated by: \( y = m(x_k + 1) + b \) and
- \( d_{\text{lower}} = y - y_k \)
- \( d_{\text{upper}} = y_{k+1} - y \)
- \( d_{\text{lower}} - d_{\text{upper}} = 2m(x_k + 1) - 2y_k + 2b - 1 \)
- Since \( m \) is \(<1\) (recall it's in the 1st octant) it is a non-integer. That's not good so, replace it with \( \frac{dy}{dx} \), where \( dy \) is the change in y endpoints and \( dx \) is the change in x endpoints – these will be integer values divided --- but we'll get rid of the division.
- \( d_{\text{lower}} - d_{\text{upper}} = 2 \left( \frac{dy}{dx} \right)(x_k + 1) - 2y_k + 2b - 1 \)
- Multiply both sides by \( dx \) and get:
  - \( dx \left( d_{\text{lower}} - d_{\text{upper}} \right) = 2 \ dy \ (x_k + 1) - 2 \ dx \ y_k + 2 \ dx \ b - dx \)
  - \( dx \left( d_{\text{lower}} - d_{\text{upper}} \right) = 2 \ dy \ x_k - 2 \ dx \ y_k + 2 \ dy + 2 \ dx \ b - dx \)
  - \( 2 \ dy + 2 \ dx \ b - dx \) is a constant which is independent of each iteration

Michael Eckmann  -  Skidmore College  -  CS 325 -  Spring 2012
Bresenham's Line Algorithm

• So we end up with
  
  \[ dx \left( d_{\text{lower}} - d_{\text{upper}} \right) = 2 \ dy \ x_k - 2 \ dx \ y_k + c \]

• We only need to determine the sign of \( (d_{\text{lower}} - d_{\text{upper}}) \)

• \( dx \) is positive so it doesn't have an affect on the sign (so now there's no need to divide) --- does that make sense?

• Let \( p_k = dx \left( d_{\text{lower}} - d_{\text{upper}} \right) \) \( p_k \) is the decision parameter

• So, we check if \( 2 \ dy \ x_k - 2 \ dx \ y_k + c < 0 \) and if it is we turn on the lower pixel, so \( (x_{k+1}, y_k) \) should be the pixel turned on

• otherwise \( (x_{k+1}, y_{k+1}) \) is turned on

• As the algorithm iterates though it needs to then calculate the next value of \( p \), which is \( p_{k+1} \)
Bresenham's Line Algorithm

• Recall that to calculate $p_k$ we had: $p_k = 2 \ dy \ x_k - 2 \ dx \ y_k + c$

• So, $p_{k+1} = 2 \ dy \ x_{k+1} - 2 \ dx \ y_{k+1} + c$

• If we subtract $p_{k+1} - p_k = 2dy \ (x_{k+1} - x_k) - 2dx \ (y_{k+1} - y_k)$

• But, $x_{k+1} = x_k + 1$ because the first octant slope always increment $x$ by 1

• So, we have $p_{k+1} - p_k = 2dy - 2dx \ (y_{k+1} - y_k)$

• Then, $p_{k+1} = p_k + 2dy - 2dx \ (y_{k+1} - y_k)$

• Where $(y_{k+1} - y_k)$ is either 1 or 0 (why?) depending on the sign of $p_k$. So, either

• $p_{k+1} = p_k + 2dy$ (when $p_k < 0$)

• $p_{k+1} = p_k + 2dy - 2dx$ (when $p_k \geq 0$)
Bresenham's Line Algorithm

/* Bresenham line-drawing procedure for |m| < 1.0. */
void lineBres (int x0, int y0, int xEnd, int yEnd)
{
    int dx = fabs(xEnd - x0), dy = fabs(yEnd - y0);
    int p = 2 * dy - dx;
    int twoDy = 2 * dy, twoDyMinusDx = 2 * (dy - dx);
    int x, y;

    /* Determine which endpoint to use as start position. */
    if (x0 > xEnd) {
        x = xEnd;
        y = yEnd;
        xEnd = x0;
    } else {
        x = x0;
        y = y0;
    }

    /* From Hearn & Baker's Computer Graphics with OpenGL, 3rd Edition */
    setPixel (x, y);
    while (x < xEnd) {
        x++;
        if (p < 0)
            p += twoDy;
        else {
            y++;
            p += twoDyMinusDx;
        }
        setPixel (x, y);
    }
}