Basic Raster Graphics Algorithms for 
Drawing 2D Primitives (ch 3)

• Idea is to approximate mathematical “ideal” 
primitives, described in Cartesian space, by sets of 
_pixels on a raster display (bitmap in memory or 
framebuffer)
• Fundamental algorithms for scan converting 
primitives to pixels, and clipping them
• Many algorithms were initially designed for plotters
• Can be implemented in hardware, software and 
firmware
• Primitives:
  • lines
  • circles, arcs, ellipses
  • region filling
  • clipping
  • alphanumeric symbols - text
    • lines, circles, character generators 
      (bitmaps), fonts
• Want efficiency & speed (often drawing many 
primitives)
Scan Converting Lines

- Assume that the line will be 1 pixel thick and will approximate an infinitely fine line
- What properties should the line have?
  - slopes -1 to 1 = 1 pixel / column
    otherwise 1 pixel / row
  - constant brightness (irrespective of length or orientation)
  - drawn as rapidly as possible
- Other considerations:
  - pen style, line style, end point (rounded?), aliasing
- Assume (unless specified otherwise) that we'll represent pixels as disjoint circles, centered on a grid

![Diagram](image-url)
Scan Converting Lines (ctd.)

- Idea is to compute the coordinates of a pixel that lies on or near an ideal, infinitely thin line imposed on a 2D raster grid.
- Assumptions:
  - integer coordinates of endpoints
  - pixel on or off (2 states)
  - slope $|m| \leq 1$
  
  (get jaggies, but antialiasing discussed later)
Basic Incremental Algorithm:
(brute force approach)

- slope \( m = \frac{\Delta y}{\Delta x} \) or \( y = mx + b \)
- idea is to increment \( x \) by 1 (\( x_i \)) and calculate \( y_i = mx_i + b \)
- pixel \((x_i, \text{round}(y_i))\) turned on

- BUT inefficient:
  - floating point multiplication
  - addition
  - round
Can eliminate the multiplication by noting that:

\[ y_{i+1} = mx_{i+1} + b \]
\[ = m(x_i + \Delta x) + b \]
\[ = y_i + m \Delta x \quad \text{since } \Delta x = 1 \]
\[ = y_i + m \]

(if \(|m| > 1\), just do opposite: increment \(y\) and compute \(x\), \(x_{i+1} = x_i + m^{-1}\))

C code

```c
Line(int x0, int y0, int x1, int y1){
    int x;
    float m, y;
    m = (y1-y0)/(x1-x0);
    y = y0;
    for(x=x0;x<=x1;++x){
        TurnOn(x, (int)(y+0.5));
        y += m;
    }
}
```

Drawbacks:

- floating point values \((m,y)\)
- round operation
Midpoint Line Algorithm (variant of Bresenham’s) reduces to Bresenham’s for lines and circles.

- uses only integer arithmetic & no rounding
- idea is to provide the best-fit approximation to a true line by minimizing the error (distance) to the true line.
- slope $0 \leq m \leq 1$ (rest is done with reflection)
- endpoints $(x_0, y_0)$ $(x_1, y_1)$

$E$ - east pixel from $P$
$NE$ - northeast pixel
$Q$ - intersection pt of line with $x = x_p + 1$
$M$ - midpoint of $E$ & $NE$
• Idea:
  • if $Q > M$ then pick NE
  • if $Q < M$ then pick E
  • if $Q = M$ then pick either one (but be consistent)
• Error will be $\leq 1/2$
How to do Bresenham's:

- line (implicit form)
  \[ F(x,y) = ax + by + c = 0 \]
  \[ (a,b,c = ?) \]

- slope intercept form
  \[ y = \frac{dy}{dx} x + B \]
  \[ dy \cdot x - dx \cdot y + B \cdot dx = 0 \]

- therefore
  \[ a = dy \quad b = -dx \quad c = B \cdot dx \]

- \( F(x,y) = 0 \) - on line
  - \( > 0 \) - for points below line
  - \( < 0 \) - for points above line

- so, need only compute \( F(x_{p+1}, y_{p+\frac{1}{2}}) \) and test its sign.

- assume that
  \[ d = F(x_{p+1}, y_{p+\frac{1}{2}}) \]
  \[ d > 0 \] pick NE
  \[ d < 0 \] pick E
  \[ d = 0 \] pick either, say E

\[ P(x_p, y_p) \quad \text{NE} \]
\[ (x_{p+1}, y_{p+\frac{1}{2}}) \]
\[ \text{E} \]
How to do Bresenham's (ctd).

• iteratively calculate d:
  depends on pick of NE or E

• if E
  \[ d_{\text{new}} = F(x_p+2, y_p+\frac{1}{2}) \]
  \[ = a(x_p+2) + b(y_p+\frac{1}{2}) + c \]
  but
  \[ d_{\text{old}} = a(x_p+1) + b(y_p+\frac{1}{2}) + c \]
  tf.
  \[ d_{\text{new}} = d_{\text{old}} + a \]
  \[ \Delta_E = a = dy \]
  so do not have to compute F directly

• if NE
  \[ d_{\text{new}} = F(x_p+2, y_p+1+\frac{1}{2}) \]
  \[ d_{\text{new}} = d_{\text{old}} + a + b \]
  \[ \Delta_{\text{NE}} = a + b = dy - dx \]

• so at each step, pick between NE & E by sign of d, then update d by \( \Delta_{\text{NE}} \) or \( \Delta_E \).

• to begin
  \[ d = F(x_0+1, y_0+\frac{1}{2}) \]
  \[ = F(x_0, y_0) + a + b/2 \]
  \[ = a + b/2 \]
  \[ = dy - dx/2 \]

• can get rid of fraction \( dx/2 \) b replacing \( F(x,y) \) by \( 2F(x,y) \), so need only simple addition.
C Code
Line(int x0, int y0, int x1, int y1){
    int dx, dy, dE, dNE, d, x, y;
    dx = x1-x0;
    dy = y1-y0;
    d = 2 * dy - dx;
    dE = 2 * dy;
    dNE = 2 * (dy - dx);
    x = x0;
    y = y0;
    while(x<x1){
        if(d<=0){
            d+=dE;
            ++x;
        }
        else{
            d+=dNE;
            ++x;
            ++y;
        }
        TurnOn(x,y);
    }
}
Additional Issues

- Endpoint order, want $P_0$ to $P_1$ to look exactly the same as $P_1$ to $P_0$.

- Starting at edge of clip rectangle, must know error at clip point (must know error or line will be altered if assumed to be starting point)
Additional Issues (continued)

- Varying the intensity of a line as function of slope, solutions include varying intensity or antialiasing (assuming line has area)

![Diagram of lines A and B]

- Output primitives composed of lines, scan convert boundaries, beware of duplicate endpoints (if xoring especially), fill algorithms to come.
Scan Converting Circles

- equation of a circle centered at (0,0) is $x^2 + y^2 = R^2$ (R is the radius)
- explicit scan conversion $y = f(x)$ $y = \pm\sqrt{R^2 - x^2}$
- to draw $\frac{1}{4}$ of a circle: $x = 0$ to $R$, $y = \sqrt{R^2 - x^2}$

BUT it's inefficient because of $\sqrt{\&^2}$, also creates large gaps as $x$ approaches $R$ (the slope approaches infinity)
- can avoid the gaps by plotting $R\cos\theta$ or $R\sin\theta$, $\theta = 0$ to 90 degrees, but still fairly inefficient
• eight way symmetry: if you have \((x, y)\) you can (generally) compute 7 more points trivially (just 4 in total for \(x = y\)), therefore need only compute one 45 degree segment.
Midpoint Circle Algorithm (Bresenham)

• same idea as midpoint line strategy
• increment around $45^\circ$ segment, $x = 0$ to $x = y = R/\sqrt{2}$ (then use 8-way symmetry)

Let, \[ F(x,y) = x^2 + y^2 - R^2 \]
\[ = 0 \text{ when on circle} \]
\[ > 0 \text{ when outside circle} \]
\[ < 0 \text{ when inside circle} \]
• So, if midpoint $M$ is outside the circle pick SE, if inside pick E, else either (be consistent, say SE)
• $d_{\text{old}} = F(x_p + 1, y_p - \frac{1}{2}) = (x_p + 1)^2 + (y_p - \frac{1}{2})^2 - R^2$

• if $d_{\text{old}} < 0$ then chose E
  $d_{\text{new}} = F(x_p + 2, y_p - \frac{1}{2}) = (x_p + 2)^2 + (y_p - \frac{1}{2})^2 - R^2$
  $d_{\text{new}} = d_{\text{old}} + (2x_p + 3)$, therefore $\Delta E = 2x_p + 3$

• if $d_{\text{old}} \geq 0$ then chose SE
  $d_{\text{new}} = F(x_p + 2, y_p - \frac{3}{2}) = (x_p + 2)^2 + (y_p - \frac{3}{2})^2 - R^2$
  $d_{\text{new}} = d_{\text{old}} + (2x_p + 2y_p + 5)$
  $\text{tf/ } \Delta SE = 2x_p + 2y_p + 5$

• Note, $\Delta E$ & $\Delta SE$ are functions of $x_p$ & $y_p$, but not too expensive since linear
• So  1. Choose pixel based on $d$
  2. Update $d$ ($\Delta E$ or $\Delta SE$)

• Initial condition
  • start at $(0, R)$
  • $M = (1, R-\frac{1}{2})$
  • $F(1, R-\frac{1}{2}) = 1 + (R^2 - R + \frac{1}{4}) - R^2$
    $= \frac{5}{4} - R$

• Problem
  • floating point (\(\frac{5}{4}\))
• Solution
  \[ h = d - \frac{1}{4}, \text{ so substitute } d = h + \frac{1}{4} \]
  so
  \[ h = 1 - R \]
  \(d < 0\) means that \(h < \frac{1}{4}\), but since \(h\) treated
  as an integer, \(h < 0\)
  call \(h\) by \(d\).

\textbf{C Code (with integers only)}

```c
Circle(int R) {
    int x, y, d;
    x = 0;
    y = R;
    d = 1 - R;
    Do8Ways(x, y);
    while(y > x){
        if(d < 0){
            d += (2*x + 3);
            ++x;
        }
        else{
            d += (2*(x-y) + 5);
            ++x;
            --y;
        }
    Do8Ways(x, y);
    }
}
```
Second Order Differences

- improve performance by computing linear equations for $\Delta E$ & $\Delta SE$ incrementally, as we did for $d_{new}$.
- calculate first & second-order differences

- if choose $E (x_p y_p) \rightarrow (x_p + 1, y_p)$
  \[
  \Delta E_{\text{old}} \text{ at } (x_p, y_p) = 2x_p + 3 \\
  \Delta E_{\text{new}} \text{ at } (x_p + 1, y_p) = 2(x_p + 1) + 3 \\
  \Delta E_{\text{new}} - \Delta E_{\text{old}} = 2 \\
  \]
  &
  \[
  \Delta SE_{\text{old}} \text{ at } (x_p, y_p) = 2x_p - 2y_p + 5 \\
  \Delta SE_{\text{new}} \text{ at } (x_p + 1, y_p) = 2(x_p + 1) - 2y_p + 5 \\
  \Delta SE_{\text{new}} - \Delta SE_{\text{old}} = 2 \\
  \]
- if choose $SE (x_p y_p) \rightarrow (x_p + 1, y_p - 1)$
  \[
  \Delta E_{\text{new}} \text{ at } (x_p + 1, y_p - 1) = 2(x_p + 1) + 3 \\
  \Delta E_{\text{new}} - \Delta E_{\text{old}} = 2 \\
  \]
  &
  \[
  \Delta SE_{\text{new}} \text{ at } (x_p + 1, y_p - 1) = 2(x_p + 1) - 2(y_p -1) + 5 \\
  \Delta SE_{\text{new}} - \Delta SE_{\text{old}} = 4 \\
  \]
- So
  1. Choose pixel based on $d$
  2. Update $d$ by $\Delta E$ or $\Delta SE$
  3. Update $\Delta E$ & $\Delta SE$

- Initialize $\Delta E$ & $\Delta SE$ using $(0, R)$
C Code (with second order differences).

Circle(int R) {
    int x, y, d, dE, dSE;
    x = 0;
    y = R;
    d = 1 - R;
    dE = 3;
    dSE = -2*R + 5;
    Do8Ways(x, y);
    while(y > x) {
        if(d < 0) {
            d += dE;
            dE += 2;
            dSE += 2;
            ++x;
        }
        else {
            d += dSE;
            dE += 2;
            dSE += 4;
            ++x;
            --y;
        }
        Do8Ways(x, y);
    }
}

Scan Converting Ellipses

- standard ellipse centered at (0, 0):
  \[ F(x,y) = b^2x^2 + a^2y^2 - a^2b^2 = 0 \]

- only draw the first quadrant (rest like circle with 4 way symmetry).

- Da Silva's algorithm
  - Divide the quadrant into 2 regions at point at which curve has slope = -1.
• Reason
  • at this point slope changes such that the choice of the next pixel is altered (unlike lines and circles as previously shown).

How to find this point?

• gradient vector at point P
  \[
  \text{grad } F(x,y) = \frac{dF}{dx} \hat{i} + \frac{dF}{dy} \hat{j}
  \]
  \[
  \frac{d}{dx} (b^2x^2 + a^2y^2 - a^2b^2) = 2b^2x
  \]
  \[
  \frac{d}{dy} (b^2x^2 + a^2y^2 - a^2b^2) = 2a^2y
  \]
  so \( F(x,y) = 2b^2x \hat{i} + 2a^2y \hat{j} \)

• a tangent slope of -1 means a grad of 1
  • true when \( i \) & \( j \) are of equal magnitude
  • if \( j > i \) then in region 1
  • if \( i > j \) then in region 2
  • therefore, if at the next midpoint
    \[ a^2(y_p - \frac{1}{2}) \leq b^2(x_p + 1) \] we switch from region 1 to region 2
  • as with previous midpoint algorithms, evaluate midpoint between 2 pixels and use the sign to chose the closer pixel
Scan Converting Ellipses (ctd.)

- region 1 (current pixel is \( P(x_p, y_p) \))
  - \( d \) is \( F(x, y) \) evaluate as the midpoint between \( E \) & \( SE \)
- to \( E \)
  \[
  d_{\text{old}} = F(x_p + 1, y_p - \frac{1}{2}) = b^2(x_p + 1)^2 + a^2(y_p - \frac{1}{2})^2 - a^2b^2 \\
  d_{\text{new}} = F(x_p + 2, y_p - \frac{1}{2}) = b^2(x_p + 2)^2 + a^2(y_p - \frac{1}{2})^2 - a^2b^2 \\
  d_{\text{new}} = d_{\text{old}} + b^2(2x_p + 3) \\
  \Delta E = b^2(2x_p + 3)
  \]

- to \( SE \)
  \[
  d_{\text{new}} = F(x_p + 2, y_p - \frac{3}{2}) = b^2(x_p + 2)^2 + a^2(y_p - \frac{3}{2})^2 - a^2b^2 \\
  d_{\text{new}} = d_{\text{old}} + b^2(2x_p + 3) + a^2(-2y_p + 2) \\
  \Delta SE = b^2(2x_p + 3) + a^2(-2y_p + 2)
  \]
• region 2 (current pixel is \(P(x_p, y_p)\))
  
  • d is \(F(x,y)\) evaluate as the midpoint between E & SE

• to S
  
  \[d_{\text{old}}=F(x_p+\frac{1}{2}, y_p-1)=b^2(x_p+\frac{1}{2})^2+a^2(y_p-1)^2-a^2b^2\]
  
  \[d_{\text{new}}=F(x_p+\frac{1}{2}, y_p-2)=b^2(x_p+\frac{1}{2})^2+a^2(y_p-2)-a^2b^2\]
  
  \[d_{\text{new}}= d_{\text{old}} + a^2(-2y_p + 3)\]
  
  \[\Delta S = a^2(-2y_p + 3)\]

• to SE
  
  \[d_{\text{new}}=F(x_p+ \frac{3}{2}, y_p-2)=b^2(x_p+ \frac{3}{2})^2+a^2(y_p-2)^2-a^2b^2\]
  
  \[d_{\text{new}}= d_{\text{old}} + b^2(2x_p + 2) + a^2(-2y_p + 3)\]
  
  \[\Delta SE = b^2(2x_p + 2) + a^2(-2y_p + 3)\]

• initial condition
  
  • start at \((0,b)\) in region 1, so midpoint is \(F(1, b-\frac{1}{2}) = b^2 + a^2(-b + \frac{1}{2})\)
  
  • if crossing into region 2 at \(P(x_p,y_p)\) E, SE choices change to S, SE and midpoint is \(F(x_p + \frac{1}{2}, y_p - 1)\)

• stop when \(y = 0\).
C Code (more or less) for Ellipse Scan Conversion

Ellipse(int a, int b) {
    int x, y, d1, d2;
    x = 0;
    y = b;
    d1 = b**2 - ((a**2)*b) + ((a**2)/4);
    Do4Ways(x, y);
    while((a**2*(y-(0.5)))>(b**2*(x+1))){
        if(d1<0){
            d1 += ((b**2)(2*x + 3));
            ++x;
        }
        else{
            d1 += ((b**2)(2*x + 3)) +
                    ((a**2)(-2*y + 2))
            ++x;
            --y;
        }
    Do4Ways(x, y);
    }

    d2 = ((b**2)((x+0.5)**2)) +
                ((a**2)((y-1)**2)) -
                (a**2 * b**2)
    while(y>0){
        if(d2<0){
            d2 += ((b**2)(2*x + 2)) +
                    ((a**2)(-2*y + 3));
            ++x;
            --y;
        }
        else{
            d2 += ((a**2)(-2*y + 3));
            --y;
        }
    Do4Ways(x, y);
    }
}