Circle Generation

Outline
- Circle generation problem
- Circle equations
- Naive approach:
  - Based on parametric representation
  - Midpoint circle drawing algorithm
- Ellipse generation problem

Circle Generation Problem
- How to display a circle on a monitor?
  - Select pixels that are closest to the circle
  - Do it efficiently
- Simplifications:
  - Circles are 1 pixel wide
  - No need for anti-aliasing

Circle Equations
- To specify a circle:
  - Radius : r
  - Center position : \((x_0, y_0)\)
- Parametric representation:
  - \(y(\theta) = y_0 + r \cos(\theta)\)
  - \(x(\theta) = x_0 + r \sin(\theta)\)
- Implicit representation:
  - \(F(x, y) = (x - x_0)^2 + (y - y_0)^2 - r^2 = 0\)

Approach Based on Parametric Representation
- // Parametric algorithm:
  - for (float \(\theta = 0; \theta < 2\pi; \theta += d\theta\)) {
    - int \(y = \text{round}(r * \cos(\theta))\);
    - int \(x = \text{round}(r * \sin(\theta))\);
    - draw(x+x_0, y+y_0);
  }
- Computational cost depends on the parameter \(d\theta\):
  - Complexity \(O(1/d\theta)\)

Eight-Way Symmetry
- Only need to calculate 1/8 circle.
- From \((0,0)\) to \((b,b)\)
- Corresponding pixels of \((x,y)\):
  - \((y,x)\)
  - \((y,-x)\)
  - \((x,-y)\)
  - \((-x,-y)\)
  - \((-y,-x)\)
  - \((-y,x)\)
  - \((-x,y)\)
Approach using Symmetry

- Computational cost is about 1/8 of original approach
- Complexity remains the same

```c
// Algorithm using symmetry:
for (float θ = 0; θ <= π/4; θ += dθ) {
    int y = round(r * cos(θ));
    int x = round(r * sin(θ));
    draw(x, y);
    draw(x, y + r);
    draw(x - r, y);
    //... 
}
```

Limitations

- How to step through θ?
  - Small spacing → inefficiency
  - Large spacing → scatter points
- The proper value of dθ depends on the circumference of the circle
  - dθ = 1/(8πR)
- High computational cost
  - Both sine & cosine functions require Taylor expansion
  - High computational cost

Midpoint Approach

- The relative position of point (u, v) to the circle can be determined using the sign of the implicit circle function:
  - On the circle  \( F(u, v) = 0 \)
  - Outside the circle  \( F(u, v) > 0 \)
  - Inside the circle  \( F(u, v) < 0 \)

Initial Condition

- Curve starts from (0, r)
  - (0, r) should be picked.
  - M is (1, r - 1/2)
- Set \( d = F(M) \)
  - \( = 1 + (r - 1/2)^2 - r^2 \)
  - \( = 1 + (r^2 - r + 1/4) - r^2 \)
  - \( = 1/4 - r \)

Next Decision (Scenario 1)

- If \((u + 1, v)\) is picked
  - \( M' = (u + 2, v - 1/2) \)
  - \( d' = F(M') \)
    \( = (u + 2)^2 + (v - 1/2)^2 - r^2 \)
- Previous d value is known
  - \( d = F(M) \)
    \( = (u + 1)^2 + (v - 1/2)^2 - r^2 \)
- Difference:
  - \( d' - d = 2u + 3 \)
- Update:
  - \( d'' = d + 2u + 3 \)

Next Decision (Scenario 2)

- If \((u + 1, v - 1)\) is picked
  - \( M' = (u + 2, v - 1/2) \)
  - \( d' = F(M') \)
    \( = (u + 2)^2 + (v - 1/2)^2 - r^2 \)
- Previous d value:
  - \( d = F(M) \)
    \( = (u + 1)^2 + (v - 1/2)^2 - r^2 \)
- Difference:
  - \( d' - d = 2u - 2v + 5 \)
- Update:
  - \( d'' = d + 2u - 2v + 5 \)
**Overall Algorithm**

- int x, y = r;
- float d = 5.0 / 4 - r;
- while (y >= x) {
  - draw(x, y);
  - draw(x, y);
  - if (d < 0) {
    - d += x * 2 + 3;
    - x ++;
  } else {
    - d += (x - y) * 2 + 5;
    - x ++; y --;
  }
- }

**Program Transformation**

- int x = 0, y = r;
- int d = 1 - r;
- while (y >= x) {
  - draw(x, y);
  - draw(x, y);
  - if (d < 0) {
    - d += x * 2 + 3;
    - x ++;
  } else {
    - d += (x - y) * 2 + 5;
    - x ++; y --;
  }
- }

**How about Ellipse**

- Implicit representation:
  - $F(x, y) = x^2/a^2 + y^2/b^2 - 1 = 0$
- Eight-way symmetry does not exist.
- Four-way symmetry can be used.
- Need to draw 1/4 of the curve, instead of 1/8.

**Two Different Procedures Needed**

- Single procedure cannot handle the full ¼ curve
- Split curve into 2 regions at location where slope $|m| = 1$
- $|m| < 1$ in region 1:
  - Increase $x$ by 1 each time
- $|m| > 1$ in region 2:
  - Increase $y$ by 1 each time