Homework 1: Rasterization

Introduction to Computer Graphics and Imaging (Summer 2012), Stanford University
Due Monday, July 2, 11:59pm

In Lecture 2, we derived Bresenham’s algorithm for rasterizing lines. In this assignment, we are going to derive and implement a similar algorithm for rasterizing circles. In particular, we will rasterize the quadrant of the circle \( x^2 + y^2 = R^2 \) where \( x, y \geq 0 \) and positive integer \( R \).

One obvious way you could attempt to rasterize such a circle would be to iterate over each \( x \) corresponding to a pixel center and light up the pixel closest to \( (x, \sqrt{R^2 - x^2}) \).

**Problem 1 (10 points). Why is this approach bad? Provide at least two reasons.**

Instead, we will imitate our development of Bresenham’s algorithm for rasterizing lines. In fact, we can simplify matters somewhat: We can employ the symmetry of the circle to reduce the number of pixels we have to rasterize. Then, each time we rasterize a pixel we will also light up its symmetric counterpart.

**Problem 2 (10 points). Suppose we are given \((x, y)\) on the 45° arc of the circle between \((0, R)\) and \((R/\sqrt{2}, R/\sqrt{2})\). Find a corresponding symmetric point on the 45° arc from \((R/\sqrt{2}, R/\sqrt{2})\) to \((R, 0)\) that takes no additional operations to compute.**

For deriving methods like Bresenham’s algorithm, it is often more convenient to use implicit forms of curves. So, we define \( F(x, y) = x^2 + y^2 - R^2 \). Then, our curve is given by \( F(x, y) = 0 \). Note that \( F(x, y) < 0 \) when \((x, y)\) is inside the circle and \( F(x, y) > 0 \) when \((x, y)\) is outside the circle.

Just as in our method for rasterizing lines, we will be marching from left to right. Let’s say we are at a pixel whose center is \((x, y)\). Then, neighboring pixel centers are given by \((x \pm 1, y \pm 1)\). Our algorithm will make a local decision about which neighbor to choose next in its walk around the circle.

**Problem 3 (10 points). Show that the slope of the circle in the 45° arc we are drawing satisfies \(-1 \leq \frac{dy}{dx} \leq 0\). Use this fact to justify why if we trace pixels in increasing \( x \) along the circle, it is sufficient to consider only the neighbors \((x + 1, y)\) and \((x + 1, y - 1)\) of \((x, y)\).**
Suppose we just drew the pixel at \((x_P, y_P)\). Our method will have the property that the circle will intersect the segment between \((x_P, y_P - 1/2)\) and \((x_P, y_P + 1/2)\) (it may help if you draw this for yourself). We will define our decision variable to be \(d = F(x_P + 1, y_P - 1/2)\).

**Problem 4** (10 points). *Give a rule using \(d\) for choosing between \((x_P + 1, y_P)\) and \((x_P + 1, y_P - 1)\) using the property explained above.*

We now have a rule for choosing which pixel to visit next, but now we’ll need to update \(d\). We will call the move from \((x_P, y_P)\) to \((x_P + 1, y_P)\) a move east (E) and the move from \((x_P, y_P)\) to \((x_P + 1, y_P - 1)\) a move southeast (SE).

**Problem 5** (10 points). *Provide update rules of the form \(d_{\text{new}} = d_{\text{old}} + \Delta_E\) and \(d_{\text{new}} = d_{\text{old}} + \Delta_{SE}\) to be applied depending on which pixel you choose.*

Excitingly, you should find that if \(x_P, y_P\) are integers, so are \(\Delta_E\) and \(\Delta_{SE}\). This suggests that we can do integer arithmetic while rasterizing circles, which can buy us some time!

To complete our rasterization algorithm, we just need a way to start out. For simplicity, we’ll actually change the convention we explained in class and assume pixel centers coincide with integer coordinates. So, we’ll start at \((x, y) = (0, R)\).

**Problem 6** (10 points). *Provide an initial value for \(d\) and show that it isn’t an integer. Suggest an alternative decision variable \(h\) of the form \(h = d - c\) for some fixed constant \(c\) so that the initial value of \(h\) is an integer, and provide update and decision rules for \(h\).*

This completes our derivation of the Bresenham algorithm for rendering circles. To make sure you have all the pieces organized, you will implement it in C++.

**Problem 7** (40 points). *Implement your circle rasterization algorithm in C++. Skeleton code is provided on the CS 148 course website, including guidance on input and output. You must use integer arithmetic in your code – programs using doubles or any other type of decimal arithmetic will receive no credit.*