COSC 4370 - Homework 1

Name: PSID:

February 2021

1 Problem

The assignment requires the rasterization of the arcs of two circles. The first circle is defined as $x^2 + y^2 = R^2$ where $x \geqslant 0$ and $R = 100$. The second circle is defined as $x^2 + y^2 = R^2$ where $y \geqslant 0$ and $R = 150$. The *radius* is 200, and the dimension of the image will be $radius \times radius$ (200×200).

2 Method

There are two functions that needed to be modified in the provided code: *renderPixel* and *rasterizeArc*. It was not necessary to edit any other part of the code. *renderPixel* receives integer values x and y as input and assigns a value of 1 to the pixel at that location (representing the color of the pixel). *rasterizeArc* receives an integer value r (the radius) as input and renders pixels at locations that lie on the arc of the target circles.

The general idea behind the method used was to look at every location (x and y coordinates) of a $r \times r$ array, check if it lies within the arc of either target circles and render the pixel at the location if it satisfies the previous requirement.

Moreover, a circle has eight symmetric counterparts for any point that lies on its arc. This is done by switching the x and y coordinates and applying negative values to each one, giving eight possible positions. With this observation in mind, the arc can be drawn simply by looking at one quadrant of the circle.

3 Implementation

Given $radius$, the coordinates to be analyzed will begin at 0 and end at $radius + 1$ in the x and y directions. Since the size of the image is also determined by $radius$, the placement of each pixel must be translated to fit within the image boundaries. The reason for this is the origin of the coordinate system for pixel placement is at the bottom-left while the origin of the coordinate system for determining location on an arc is at the center.

3.1 rasterizeArc

To look at each location, a nested for loop will be used, with each loop iterating from 0 to $radius + 1$ in the x and y directions. Within each iteration of i and j, the $PYTHAGOREAN\_SQUARE$ will be calculated and compared to 100^2 and 150^2. If either statements prove true, *renderPixel* will be called, inputting the i and j values.

Before the nested loops, an $ERROR\_THRESHOLD$ is assigned a arbitrary value (300 was used in the program). This value is the maximum error to determine whether a location lie on the arcs or not. The threshold is used because integer multiplication would be too exact and a clear semicircle may not be visible.

3.2 renderPixel

A check is first performed to determine if an inputted location is bounded by $0 \leqslant x \leqslant radius$ and $0 \leqslant y \leqslant radius$. If the check is passed, four values will be assigned: $PYTHAGOREAN\_SQUARE$, $ERROR\_THRESHOLD$, $SCALE$ and $OFFSET$. The $PYTHAGOREAN\_SQUARE$ and $ERROR\_THRESHOLD$ have already been explained in the previous subsection. $SCALE$ is another arbitrary value used to decrease the size of circle. $OFFSET$ is used to determine the placement of each scaled pixel. By using these two values, the arcs can fit within the predetermined dimensions.

To calculate the scaled location of a pixel, the x and y coordinates will be divided by the $SCALE$, and the $OFFSET$ will be added or subtracted.

The last part of this function sets the pixel's value at the scaled location to 1 and checks whether a location is on the either target circles or not. If the location is defined on $x^2 + y^2 = 100^2$, the pixel's symmetric counterpart, with respect to the x-axis, will be set to 1. Otherwise, if the location is defined on $x^2 + y^2 = 150^2$, the pixel's symmetric counterpart, with respect to the y-axis, will be set to 1.

4 Results

The output of the program was a .ppm file, which consists of a comment on the file, the dimensions of the image, the maximum color value of any pixel and the values for each pixel. When viewed through a image viewer, two semicircles of different radii can be seen.