Amazing Radical Mesh Integral Calculator

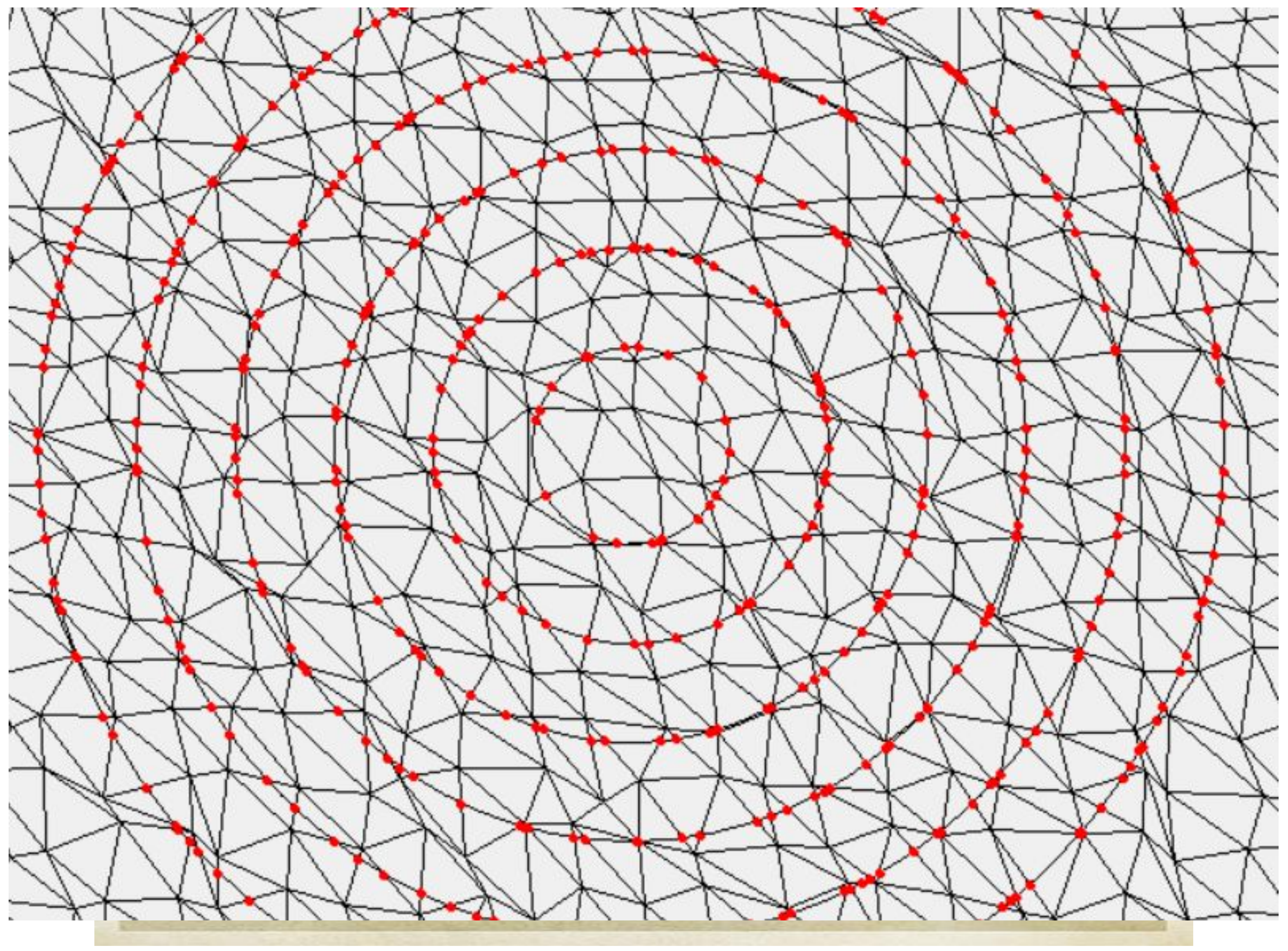
Introduction

The purpose of this project is to, given a triangulation mesh representing the earth's surface, find the average height of the surface long the edge of multiple concentric circles. This program was requested by Dr Aleksei Beltukov from the University of the Pacific Math Department faculty.

Method

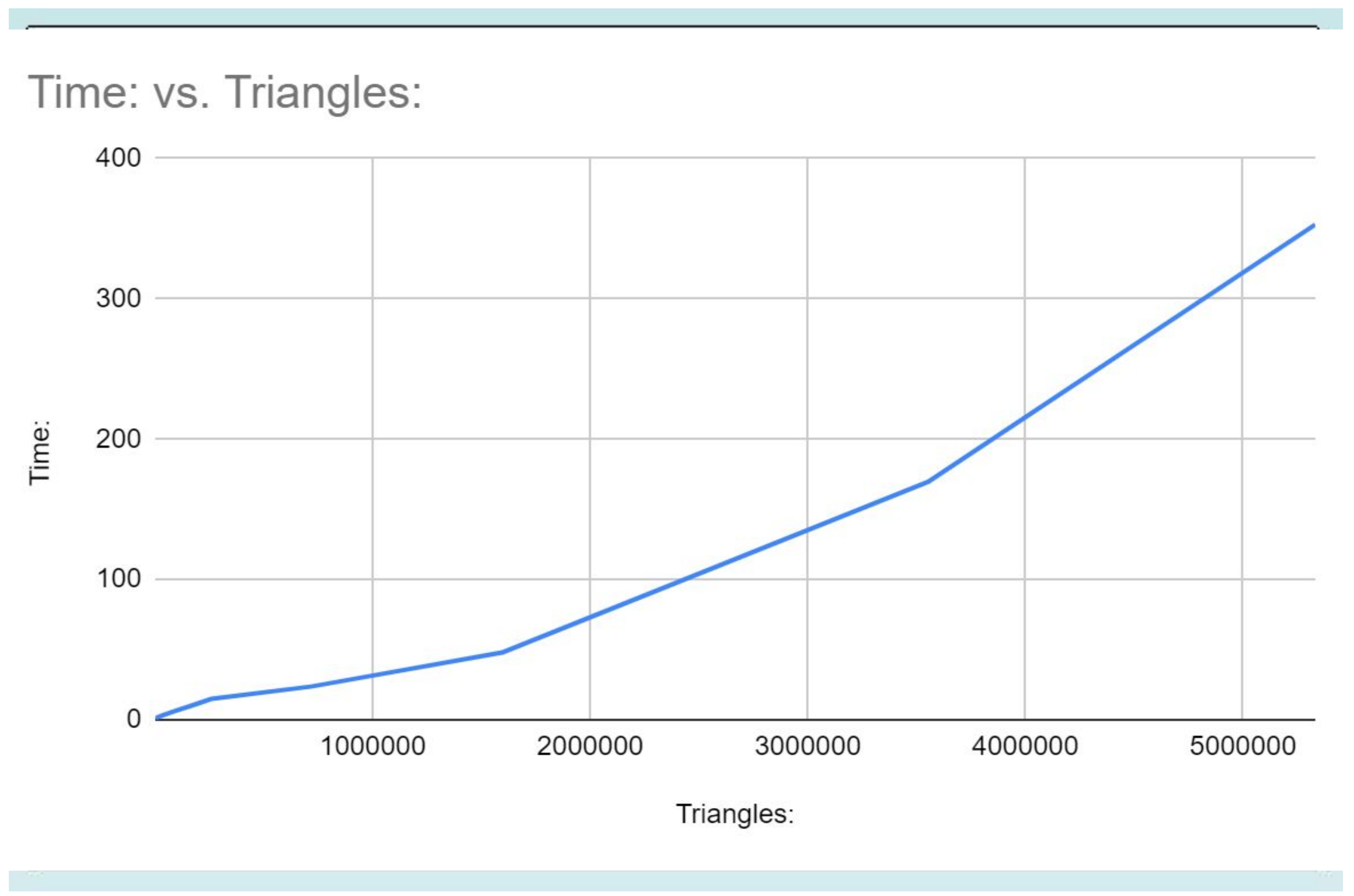
As input, we are given a set of points denoted by x position, y position, and height, which form the vertices of the triangulation. We are also given sets of three vertices to form into triangles, given by three sets of indexes, referring to points from the previous input. This completes our triangle grid. Finally, we are given a set of x and y positions, and a list of radii, denoting the sets of concentric circles for which we will calculate integrals.

Visualization of intersections, marked with red



Approach

To calculate the integral of an arc on the circle, we need the arc's beginning and ending angle, and the equation of the plane described by the corresponding triangle. Intersections for a circle will be stored as an angle from the horizontal.
The easiest approach is to consider line segments in isolation, calculating and sorting all intersections with the circle by increasing angle. For every pair of intersections, we find the triangle shared by both line segments, and use the plane it describes for the arc integral calculation.



Optimization

The majority of wasted computation time for the program lies in checking all the line segments to see if they intersect, so our optimizations will be targeted to reducing the number of line segments that need to be checked.

Triangle marching:
Logically, all triangles that intersect the circle will be connected to each other, and additionally the sides that are connected are segments that intersect the circle. This suggests that once we find the first intersection, we can change tactics and only queue line segments from the triangles connecting to intersections, until we run out of intersections. This converts our algorithm from an O(n) complexity to O(1), once we find the first intersection.

Future Plans

Triangle marching helps greatly with large circles, where we can find the first intersection relatively quickly. However, as the number of triangles increases, the time lost finding the first intersection for each circle quickly becomes a performance bottleneck.

My idea to solve this is to group all line segments into an evenly spaced grid. The segments are sorted into squares, and segments intersecting two or more squares show up in each of those squares. When we start looking for circle intersections, we know that only line segments inside squares that the circle intersects have a chance to intersect the circle themselves. Finding squares that overlap the circle is trivial, and we can queue up line segments from only those squares, thus drastically reducing the number of line segments we check that have no chance of intersecting the circle.

The issue with this approach is that the squares greatly increase the memory usage of the program. However, I hope to find a better way to represent the grid that can save on memory. Finally, my goal is to take this one step further, since the circles are concentric and only load line segments from grid squares that fall within the circle's radius from memory, which should also help with the program's intensive memory usage.

UNIVERSITY OF THE PACIFIC
Tony Tiger and Peter Rabbit