COSC 4370 Homework 1

Name: Jordan Foster | **PSID**: 1669900
February, 2022

1. Problem

This assignment involves rasterizing an ellipse represented by the equation:

$$(\frac{x}{12})^2 + (\frac{y}{6})^2 = 64^2 \text{ where } y >= 0$$

Given the addition of $y >= 0$, the problem goes from rasterizing a full eclipse to *only rasterizing the top half of the ellipse,* since *y* would no longer be a factor.

2. Method

After calculating the standard form of the ellipse via calculator, the vertices *-384, 384, -768, and 768* were found. Realizing the pure mathematical nature of ellipses, the approach used was modeled after graphing problems, using the bitmap canvas as a standard graph on which the ellipse would be drawn.

The general idea behind the method was to calculate the curvature of the ellipse's perimeter and from there utilize a function to iterate across the x-axis of the canvas and plot the appropriate points based on the output of that function.

Since we are only dealing with the top half of the ellipse, we are only concerned with two sections of the shape: the top left section, and the top right section; therefore, we should only have to use two iterations to draw based on our function.

3. Implementation

For a half-ellipse, the curvature of its perimeter can be found with the expression

$$y = \pm \frac{b}{a} \sqrt{a^2 - t^2}$$

with **b** and **a** representing the semi-minor and semi-major axes of the ellipse, respectively and **t** representing the input of ***f(t)***. This will be the basis of our drawing implementation, with our midpoint being 768 based on the value of **a** (with **b** being 384)**.**

3.1. Issues with bitmap canvas

Since it's not a standard graph, the bitmap canvas offered a slight inconvenience due to the fact that it does not have negative values to correspond with **x** to the left of the midpoint.

To handle this, I simulated the decreasing negative values to the left of the midpoint with decreasing positive values which generated the desired effect. By using a simple for loop with iteration variable ***i*,** subtracting *i* from the midpoint ***a*** allowed the left side of the half-ellipse to be rendered. This (*a* - *i*) value served as our negative x values, while the inverse (*a* + *i*) served as our positive x values moving from the midpoint. With this solved, now it was time to add the curvature formula.

3.2. Issues with set_pixel function

With the expression for curvature (illustrated above) in mind, it was rather easy to plug it into the given **set_pixel** function as *(1.0*b/a)*(sqrt((a*a)-(i*i)), with i* representing *x*/the iteration variable, so no new "function" had to be created.

Though, the given set_pixel function only deals with integers, while our expression produces floats. While the set_pixel function rounded the values automatically, the approximations led to rather wide gaps when drawing the half-ellipse; the resulting image was mostly correct, but the tail ends showed missing pixels.

To work around this limitation of only being able to use integers, the fix used was to store the values from our expression into a vector and, after the initialize drawing, use another for loop to iterate across the vector searching for gaps in values. From there, another for loop would fill in the missing pixels from one *x* value to the next. This successfully filled in the missing pixels and solved the issue. Given we are still only iterate on one half, *768* values total are stored in the array, with each action performed also having its inverse performed to complete the other half, much like with the previous drawing function.

4. Result

With all issues resolved, the program was able to success produce a .bmp file that displayed a smoother half-ellipse, depicted in white on a black background.