COSC 4370 - HW 1
Omar Watad, PSID: 2036321

1) Scope:

We have to render the upper section of an ellipse, which can be described by the formula (x/12)^2 + (y/6)^2 = 64^2, while ensuring that only the portion above the x-axis is included (Positive side only). Each pixel must be depicted in a uniform white color against a black background.

2) Algorithm:

The initial stage of the process involves determining the appropriate dimensions for the canvas needed for the task of rendering the ellipse. Once the size is determined, a for loop is utilized to iterate through each value of x on the canvas, and determine the corresponding value of y required to render the ellipse. To ensure that the canvas is sufficiently large to contain the ellipse, we make it larger than necessary.as we traverse for each pixel, we create an if statement that is required within the for loop to verify that the x (all pixels) value being iterated falls within the calculated boundaries of the ellipse size. Each y value is calculated by manipulating the provided equation for the ellipse ((x/12)^2 + (y/6)^2 = 64^2). Finally, the current pixel can be set using the calculated x and y values, ultimately resulting in the drawing of the ellipse.

3) Main.cpp:

To prepare the canvas for the task of rendering the ellipse, it is necessary to calculate the width of the ellipse. This requires solving the ellipse equation for y, resulting in two solutions: y = (sqrt((768 + x) * (768 - x)) / 2) and y = -(sqrt((768 + x) * (768 - x)) / 2). However, since the x and y axes utilized only encompass positive values, the negative solution can be disregarded.

To effectively render the ellipse, it is necessary to identify the midpoint at which y reaches its maximum value. This can be achieved by temporarily setting the midpoint at x = 0, and substituting this value into the equation y = (sqrt((768 + x) * (768 - x)) / 2). Solving for y in this equation results in x = 768 or x = -768. By shifting the midpoint to any x value, we can determine the appropriate placement of the pixel on the canvas by simply adding 768 or -768.

Now that we have the ellipse dimensions, we calculate the dimensions for the canvas. Since the width of the ellipse is 768, the full width of the canvas will be double that amount, giving us 1536 pixels. We also know that the height of the ellipse is 384 pixels,

which can be rounded up to 400 pixels for easier calculation. Therefore, we can set the dimensions of the BMP to 2000 pixels wide and 800 pixels high, by altering the code to state BMP bmpNew(2000, 800, false). The next step is to fill the canvas with a solid black color using the code bmpNew.fill_region(0, 0, 2000, 800, 0, 0, 0, 0).

To ensure that the ellipse is drawn for all y values greater than or equal to 0, we need to make a modification to the original for loop provided in the code. Specifically, we need to replace the inner component of the loop so that it states bmpNew.set_pixel(i, 0, 255, 255, 255, 0). By changing the value of 100 to 0, we indicate that the horizontal line will be drawn on the screen at y = 0 instead of at y = 100.

Now we create a separate loop to fill the ellipse. We need to traverse from i starting point up to i < bmpNew.bmp_info_header.width to cover the x-axis from 0 to 2000, but since the actual canvas width is 1536, we adjust the midpoint to be x = 1000 to center the ellipse on the canvas. This cause changes to the roots of the ellipse to x = 232 and x = 1768 so we only need to draw pixels on the screen from x = 232 to x = 1768, rather than the full canvas. To implement this, we can add an if statement within the for loop to check if i falls within the range of x = 232 to x = 1768. If it does, we can proceed with the drawing of pixels on the screen.

Although our traversal on the canvas is from x = 0 to x = 2001, the necessary calculations are from x = -768 to x = 768. This means we will need to have two separate x values, one is i, responsible for traversing from x = 0 to x = 2001, and the other is x, responsible for calculating from x = -768 to x = 768 depending on the current i. When i = 1000, x should equal 0. int x = i - 1000 is in the if statement to do this, which will calculate the correct x value to be used in the ellipse equation.

To determine the y-value for each pixel that needs to be drawn, we use y = (sqrt((768 + x) * (768 - x)) / 2). However, it is important to note that for the calculation, we must use the variable x and not I (as mentioned above). To draw the pixel, we add bmpNew.set_pixel(i, y, 255, 255, 255, 0).

4) Results: