Andrew Fryzel

U1070220

CS 4600: Assignment 2

1. **Line Rasterization (Based off Lecture 06 Pseudocode)**

The line rasterization works by first determining which axis the line is longer on. Whichever line is longer determines the major axis for plotting the pixels. If x1 > y1 we need to swap them which removes special cases / possibility of error. Have to swap first if the line is too steep and the second swap is to make sure it is from left to right.

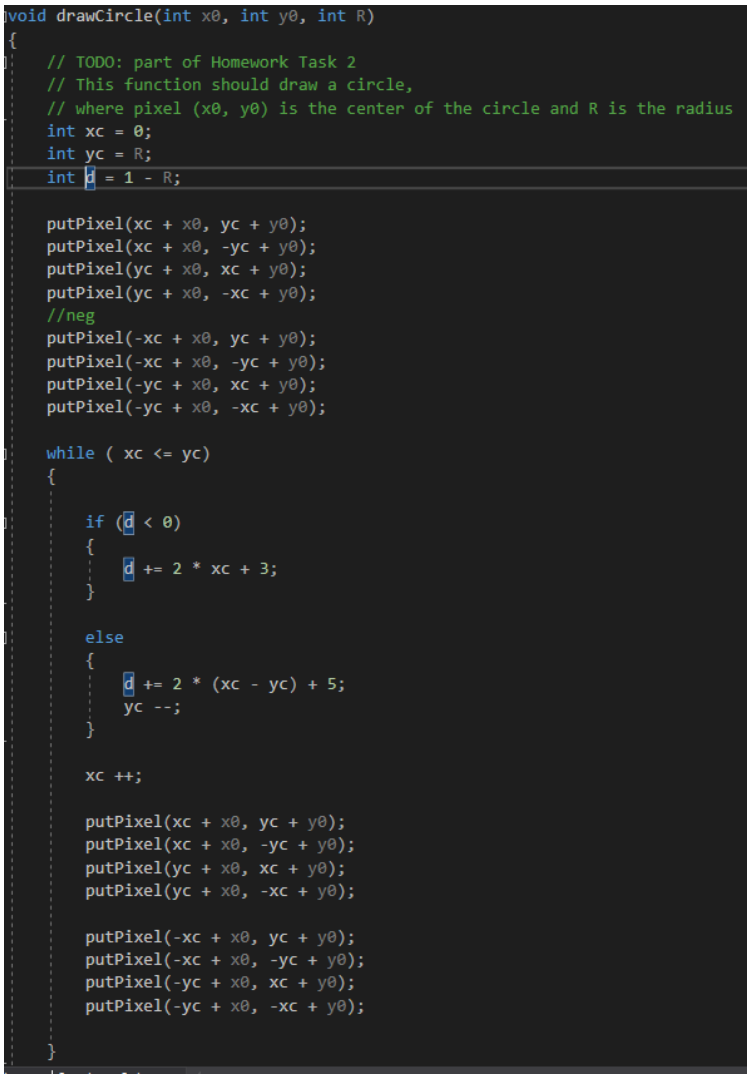
```
int dx = abs(x2 - x1);
int dy = abs(y2 - y1);

//flip to help remove error cases
bool check = dx < dy;
if (check)
{
    std::swap(x1, y1);
    std::swap(x2, y2);
}

if (x1 > x2) {
    std::swap(x1, x2);
    std::swap(y1, y2);
}
```

I then looped through the major axis and plotted all the pixels from start to end. Since sometimes the calculated point will not be accurate during sloped lines, the error value helps to account for this and ensures proper point placement. If the slope error is greater than .5 then the line has changed up/down by a pixel, so the y coordinate needs to change as well.

```
//loop through and place pixels
while (x1 < x2)
{

    if (check)
    {
        putPixel(y1, x1++);

    }
    else
    {
        putPixel(x1++, y1);
    }

    error -= dy;


    //need this for angled lines to adjust them
    if (error < 0)
    {
        //slope calc (neg or positive slope)
        if (y1 < y2)
        {
            level = 1;
        }
        else
        {
            level = -1;
        }
        //increase slope level based on level value
        y1 += level;
        error += dx;
    }
}
```

2. **<u>Circle Rasterization (Based off Lecture 06 Pseudocode)</u>**

The circle rasterization was surprisingly simpler than the line rasterization. By following the lecture pseudocode, I first placed the pixels, then looped through until xc < yc.



```
void drawCircle(int x0, int y0, int R)
{
    // TODO: part of Homework Task 2
    // This function should draw a circle,
    // where pixel (x0, y0) is the center of the circle and R is the radius
    int xc = 0;
    int yc = R;
    int d = 1 - R;

    putPixel(xc + x0, yc + y0);
    putPixel(xc + x0, -yc + y0);
    putPixel(yc + x0, xc + y0);
    putPixel(yc + x0, -xc + y0);
    //neg
    putPixel(-xc + x0, yc + y0);
    putPixel(-xc + x0, -yc + y0);
    putPixel(-yc + x0, xc + y0);
    putPixel(-yc + x0, -xc + y0);

    while ( xc <= yc)
    {

        if (d < 0)
        {
            d += 2 * xc + 3;
        }

        else
        {
            d += 2 * (xc - yc) + 5;
            yc --;
        }

        xc ++;

        putPixel(xc + x0, yc + y0);
        putPixel(xc + x0, -yc + y0);
        putPixel(yc + x0, xc + y0);
        putPixel(yc + x0, -xc + y0);

        putPixel(-xc + x0, yc + y0);
        putPixel(-xc + x0, -yc + y0);
        putPixel(-yc + x0, xc + y0);
        putPixel(-yc + x0, -xc + y0);

    }
```

3. **Interactive Drawing**

To do part 3, I used a series of global variables to store pixel locations and a Boolean flag to keep track of which mode the user is in. Although not the most elegant solution, it got the job done.

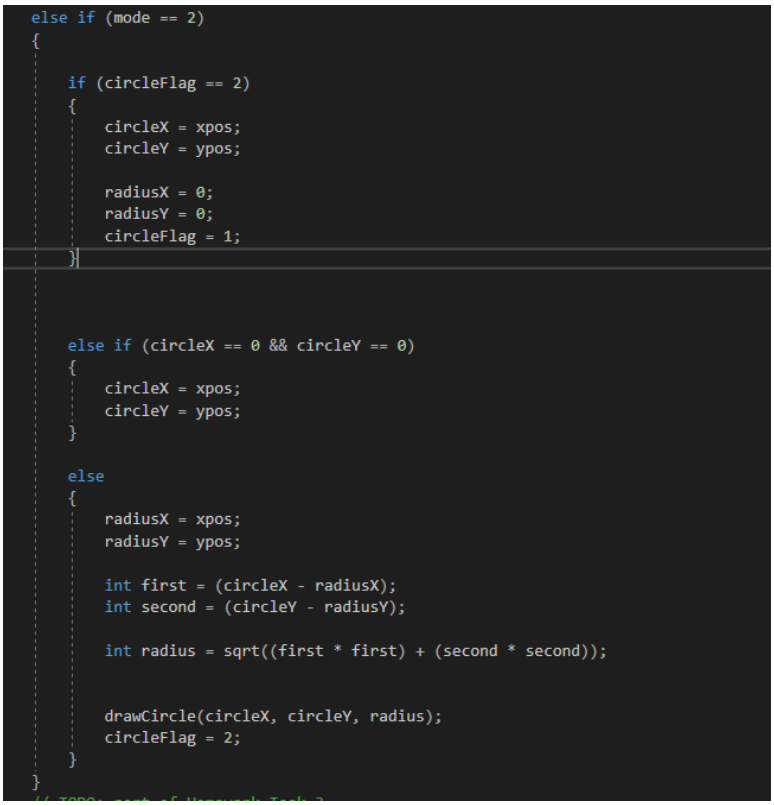
The “mode” flag controls what mode the user is in. The default setting is pixel mode and pixel mode can be made active by pressing ‘N’ for “Normal Mode”, ‘C’ for “Circle Mode”, and ‘L’ for “Line Mode”.

```
void glfwKeyCallback(GLFWwindow* p_window, int p_key, int p_scancode, int p_action, int p_mods)
{
    if (p_key == GLFW_KEY_ESCAPE && p_action == GLFW_PRESS)
    {
        glfwSetWindowShouldClose(g_window, GL_TRUE);
    }

    else if (p_key == GLFW_KEY_L && p_action == GLFW_PRESS)
    {
        std::cout << "Line mode" << std::endl;
        mode = 1;
    }

    else if (p_key == GLFW_KEY_N && p_action == GLFW_PRESS)
    {
        std::cout << "Pixel mode" << std::endl;
        mode = 0;
    }

    else if (p_key == GLFW_KEY_C && p_action == GLFW_PRESS)
    {
        std::cout << "Circle mode" << std::endl;
        mode = 2;
    }
}
```

Line mode uses if statements to keep track of the global variables and a flag called “flag” that helps to reset those variables once two points have been clicked and the line has been drawn.

```
if (p_button == GLFW_MOUSE_BUTTON_LEFT && p_action == GLFW_PRESS)
{
    std::cout << "You clicked pixel: " << xpos << ", " << ypos << std::endl;
    putPixel(xpos, ypos);

    if (mode == 1)
    {
        //  int x1, x2, y1, y2;
        if (flag == 2)
        {
            x1_ = xpos;
            y1_ = ypos;

            x2_ = 0;
            y2_ = 0;
            flag = 1;
        }



        else if (x1_ == 0 && y1_ == 0)
        {
            x1_ = xpos;
            y1_ = ypos;
        }

        else
        {
            x2_ = xpos;
            y2_ = ypos;
            drawLine(x1_, y1_, x2_, y2_);
            flag = 2;
        }
    }
```

Circle mode does something like line mode but has to also calculate the radius using the distance formula.



```
else if (mode == 2)
{

    if (circleFlag == 2)
    {
        circleX = xpos;
        circleY = ypos;

        radiusX = 0;
        radiusY = 0;
        circleFlag = 1;
    }



    else if (circleX == 0 && circleY == 0)
    {
        circleX = xpos;
        circleY = ypos;
    }

    else
    {
        radiusX = xpos;
        radiusY = ypos;

        int first = (circleX - radiusX);
        int second = (circleY - radiusY);

        int radius = sqrt((first * first) + (second * second));


        drawCircle(circleX, circleY, radius);
        circleFlag = 2;
    }
}
```

Finally, a white pixel is placed on mouse click every time the mouse is clicked, regardless of mode, by simply calling the following code in the mouse left click callback.

```
putPixel(xpos, ypos);
```

Finally, my picture using line and circle drawing: