Project 3 - Graphics Primitives

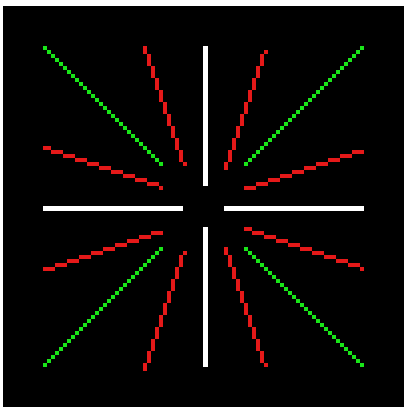
Created by Gregory Attra, last modified just a moment ago

Overview

For this project, I extended the graphics rendering engine to draw primitive graphics—points, lines, circles, ellipses, and polylines—as well as the functionality to fill circles and ellipses with a provided color. Each of these objects were comprised of some of the other objects. For example, a `Line` is a struct containing two `Point` structs, and a `Polyline` is a struct comprised of an array of `Line` structs. As such the functionality to draw one object was also used in the functionality to draw another. For example, to draw a polyline, the function `polyline_draw` iterates over each line segment and calls the `line_draw` function.

Lines

I implemented Bresenham's algorithm for line drawing. Below is the output of the test program to verify the functionality of the `line_draw` function.

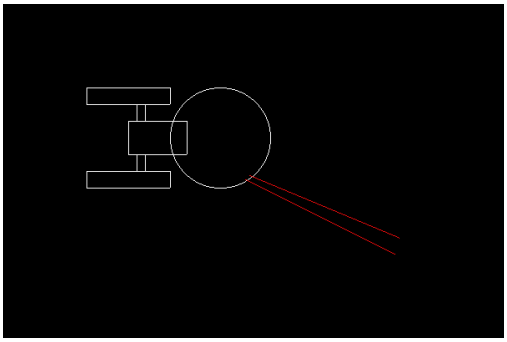


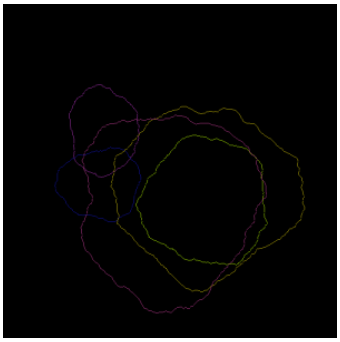
And here is the output for the performance test program `testbench.c` along with the computer specs:

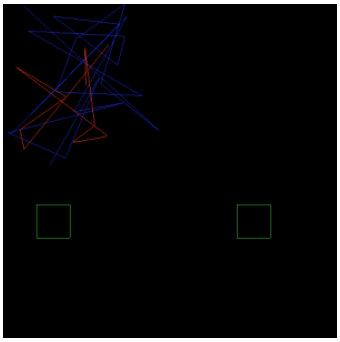
```
(base) gbattra@Gregorys-MBP images % ../bin/testbench
Average line length = 103.9
2763957.96 lines per second
```

Circles, Ellipses, Polylines

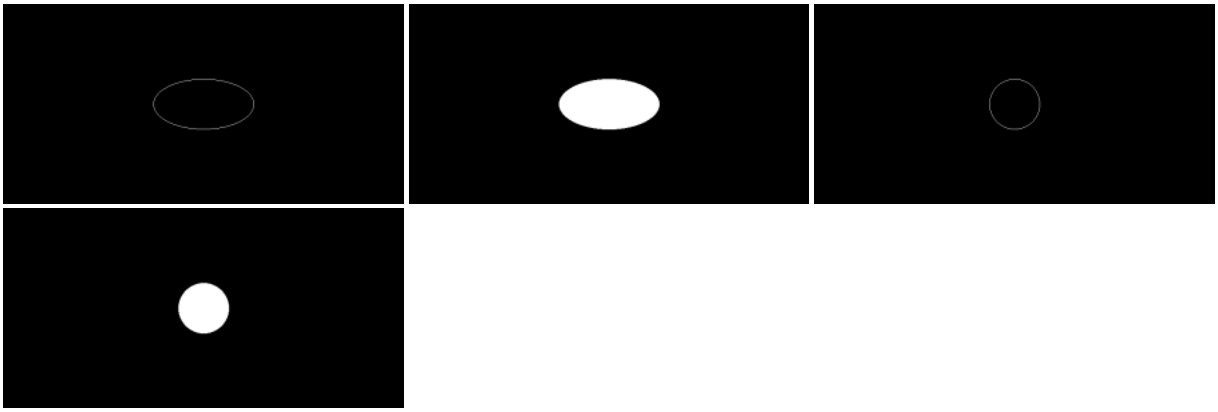
Below are the outputs for the test programs for the elipses, circle, and polyline APIs.







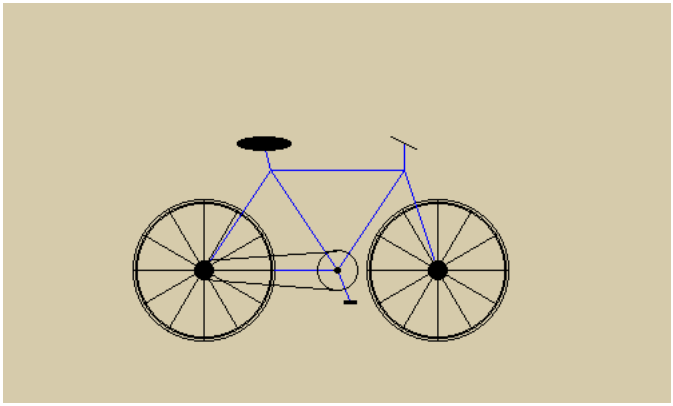
Additionally, I ran some independent test programs to output the fill functionality for ellipses and circles:



I used Bresenham's algorithm for both the circle and the ellipse functionality. For the `circle_draw` function, while the slope is less than 1, I compute the upper right point and then reflect that point seven times for each axis of symmetry. For the `draw_circleFill` function, I added a `fill` flag argument to the main draw function. Each time a new point is computed, if the fill flag is set, the program will use the `line_draw` function to draw horizontal lines between the reflected points. The same approach was taken for the `ellipse_draw` and `ellipse_drawFill` functionality with minor tweaks to account for the assymetry.

Illustration

For my custom image, I drew a simple bicycle using the `Line` , `Circle` , `Ellipse` and `Polyline` graphics primitives API built for this project:



Extensions

FLOOD FILL

As an extension, I implemented the flood fill algorithm to fill complex shapes. To do this, I implemented a stack in C. I created a struct named `Stack` which had the following parameters:

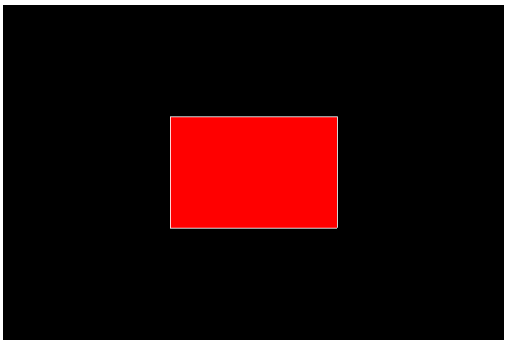
```
typedef struct
{
    int capacity;
    int item_size;
    int top;
    void *items;
} Stack;
```

By using `void` pointers, the stack object is generalizable to take any datatype. I then implemented basic stack functionality: `stack_create()` , `stack_push()` , and `stack_pop()` along with `stack_free()` .

The `floodfill()` funciton takes a `src` image, a start `Point` and a target `Color` . The function grabs the pix at the start position and pushes it to the stack. It then loops while the stack is not empty and pops the `Point` off the top of the stack. It checks if the pixel at that point has the same color as the original color at the starting pixel. If so, we assume this pixel is part of the shape to fill and we fill it, then push its neighbors to the stack. Otherwise, it skips that pixel.

Once all the pixels within the shape border have been filled, the algorithm terminates.

Here is the final result of the test program:



Reflection

I am starting to become very comfortable with C and its customs/design paradigms. Having used C++ extensively for Computer Vision, it wasn't too far of a jump to get into C. As someone with a background in OOP, I was unfamiliar with functional programming paradigms, but one main pattern I'm picking up on is passing a pointer to an object into a function which then updates the memory at that pointer. In OOP this would be considered an antipattern as a function is updating the state of an argument, as opposed to instantiating a new instance of that object and returning it to the caller. However, in C this seems to be the simplest way to mimic class methods, which are allowed to manipulate the state of the object they belong to.

I am very excited to start moving into texture and pattern rendering to make more complex graphics.

Acknowledgements

- Prof. Maxwell's notes: CS5310-F21-Lectures.pdf

No labels