CS134 Computer Graphics - Lab 8

Points

- Part 1 - 4 Points
- Part 2 - 6 Points

Goals

- Understand fractals
- Review complex numbers
- Implement a graphical representation for Julia Sets

Introduction

Fractals are mathematical models which have a repeating pattern that demonstrates reasons for random or chaotic behavior. In this lab, we will render one such example with a graphical visualization of Julia Sets using OpenGL.

To get started, you will need to review complex numbers because these are in Julia Sets.

Reviewing Complex Numbers

In Algebra, you may remember that imaginary numbers were used to describe solutions with negative roots.

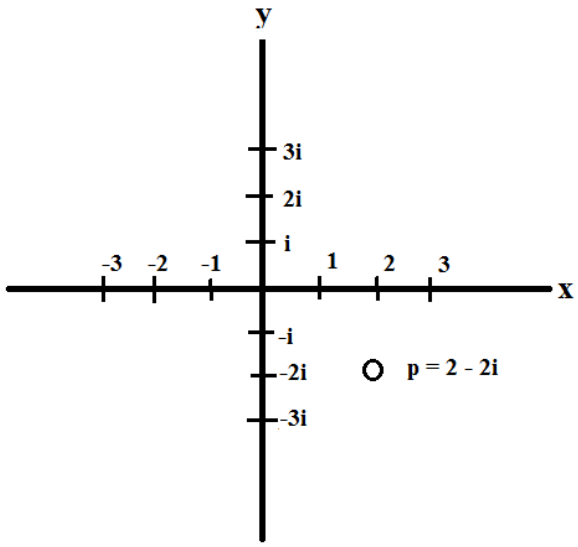
One example would be $x = \sqrt{-1}$.

Given the property of the imaginary unit, *i*: $i^2 = -1$

The solution for x above would be a complex number, $x = \sqrt{(-1)(-1)i^2} = \sqrt{i^2} = i$

A complex number can be represented as two components, a real component and an imaginary component: $a + bi$

Therefore, a complex number can be mapped onto an x-y plane as the real-imaginary plane (more formally called the complex plane).



An example of the complex plane which maps to the real and imaginary axes is shown above.

You may review complex numbers and go beyond the basics. Simplify the problems below to show that you understand the basics of complex numbers. You do not have to but may show the TA to confirm your answers.

a) $i^3 =$
b) $i^{22} =$
c) $(2 + i)(1 + 3i) =$
d) $|3 + 4i| =$

Julia Sets

In this lab. You will be using graphics to distinguish a collection of all complex numbers under a given rule. This rule is described by the Julia Set.

A formal definition of a Julia Set - the collection of all initial complex numbers (Points in the complex plane) in a polynomial that when the polynomial is iterated through, the orbit does not diverge.

Lets first understand what it means for a polynomial to iterated. What this means is that given a function f(x) and an initial condition x0, the iterated values would be
x1 = f(x0), x2 = f(x1), x3 = f(x2), … and so on.
An example:
$f(x) = x^2 + 1,\ x0 = 0.5$
$x1 = f(x0) = 1.25$
$x2 = f(x1) = 2.5625$
$x3 = f(x2) = 7.56640625$

...
NOTE: when iterating this polynomial, the result diverges to infinity with initial condition x0 = 0.5 which means the orbit for this polynomial diverges.

Now, lets do one where the function involves complex numbers.
$f(x) = x^2 + (0 + 0i),\ x0 = 0.5 + 0.5i$
$x1 = f(x0) = 0 + 0.5i$ // Do you understand why?
$x2 = f(x1) = -0.25 + 0i$
$x3 = f(x2) = 0.0625 + 0i$
...

Notice that the value will converge to 0 beyond x3. This means that if the polynomial is further iterated, it will not diverge to infinity or -infinity. Referring back to the definition of a Julia Set, we have found a complex number for the polynomial which the orbit (iterating the polynomial) does not diverge. Therefore, the point 0.5 + 0.5i is in the Julia Set for the polynomial, x^2

There are many polynomials that can be used. In this lab, we will identify complex numbers in the Julia Set for a quadratic polynomial.
$f(x) = x^2 + c$

Part 1:

We will use the standard <complex> library to use complex numbers. The complex number is templatized for float, double, long double and has many functions built in with it already.
http://www.cplusplus.com/reference/complex/

Example of allocating a complex variable that stores 3.0 + 5.0i
complex<double> val(3.0, 5.0);

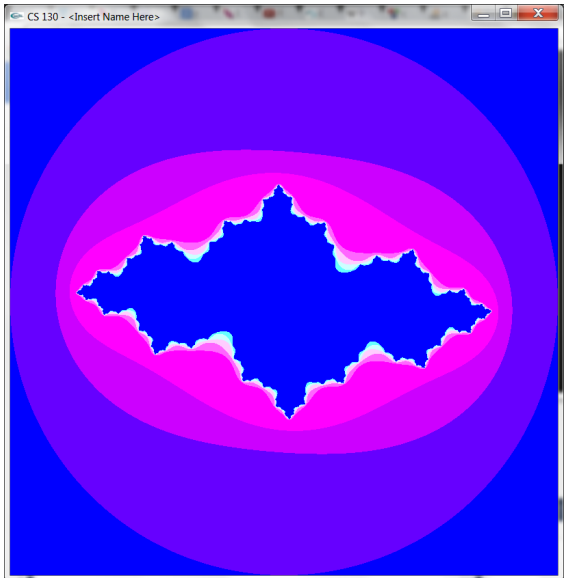
In order to visualize our Julia Set, we will be mapping colors to each point (in this case we will be testing points equal to the number of pixels in our application screen. Notice how expensive it can be to iterate through a polynomial many times to determine whether the polynomial diverges given that this is done for every pixel. We will be estimating whether a polynomial diverges up to a limit number of iterations MAX_ITERS. We should be able to guarantee the polynomial diverges if the magnitude of the orbit, x, is greater than 2.
(Magnitude of complex numbers is the absolute value)

```
iters = 0
while abs(x) < 2.0 && iters < MAX_ITERS
        x = f(x)
        ++iters
```

Once we have found the number of iterations it took to decide whether a polynomial diverges, we will map a color for each pixel depending on the number of iterations it took. If we have more iterations and more differentiating colors, the more detailed Julia Set we get.

An overly simplified Color class is provided for you:
https://drive.google.com/file/d/0B2dqioxbKNaAUnFLcmdiX1RoeHc/edit?usp=sharing

Write a function called renderJuliaSet(double xmin, double xmax, double ymin, double ymax) that rendering each pixel mapped to a 2D visualization of a Julia Set using the complex plane (where the x-axis is the real number line and the y-axis is the imaginary number line). For this part of the lab, you will call renderJuliaSet with xmin=-2.0, xmax=2.0, ymin=-2.0, ymax=2.0. Use at least five color maps for each iteration and show the TA; the image below uses 10.
Note: You will only be calling renderPixel to the screen and no other advanced openGL calls.
Note: You may call glColor3d(r, g, b) or glColor3f(r, g, b) such that any following calls to glVertex will be assigned a color (r, g, b) until changed again.

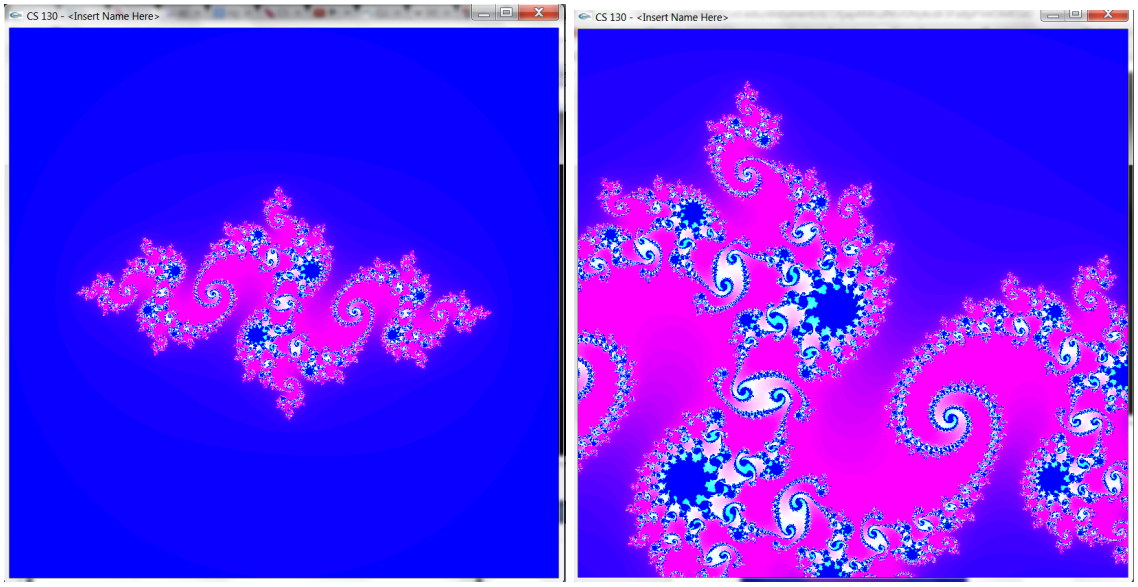


If you cannot find a decent polynomial to test: *use* $x^2 + (-0.7795 + 0.134i)$

Part 2:

Discover your own Julia Set. You may look for other polynomial constants on the web or try your own. Show a rendered scenes with bottom left (-2.0, -2.0) and top right (2.0, 2.0) and another scene zoomed in on your fractal. Use at least 100 Color mappings.
Hint: You can easily map many colors by setting a color hue so that the colors change from one gradient to another. The Color class has a function that does this for you by assuming a rotation on the color wheel: At 0 degrees you have red, at 120 degrees you have green, and at 240 degrees you have blue.

Render a zoom by calling renderJuliaSet with different xmin, xmax, ymin, ymax parameters.



You are encouraged to try out different parameters. With changes in your polynomial, the Julia Set will change. Take a look on the web at other Julia Sets!
Other neat additional functionalities to consider to your program:

- Allow program to zoom in onto certain parts of the fractal
- Enable the user to change the constants in the function or the color hues
- The Color3d function rotateHue only switches between RGB colors. However, with a slight modification, you can also switch between the grayscale for darker and lighter colors.
- Allow use of an additional degree polynomial than just a quadratic and render the Julia Set
- Mandlebrot Sets are similar to Julia Sets. Investigate the difference on the web and implement a visualization of a Mandlebrot Set.

References:
This lab is derived from:
http://www.cs.wustl.edu/~cytron/cs101/Labs/5/