CS 3005: Programming in C++

Julia Set

Introduction

The Julia set is a mathematical set defined from a function. In addition to its merits in complex dynamics, it can be used to generate interesting images. Every point in a 2 dimensional plane can be categorized as inside or outside of the Julia set for a particular function choice.

Finding Points in the Julia Set

For our purposes, this is a good enough definition of the Julia set for a function (x', y') = f(x, y), where (x, y) and (x', y') are the coordinates of points in the 2 dimensional plane.

Take a point (x0, y0). Using x0, and y0 as input to f(x, y), we receive a new point from the function.

```
x1, y1 = f(x0, y0)
```

Repeat that process (iterate) by using the output of the previous call of the function as the input to the current call. We could repeat up to `n` times:

```
x2, y2 = f(x1, y1)
x3, y3 = f(x2, y2)
x4, y4 = f(x3, y3)
...
xn-1, yn-1 = f(xn-2, yn-2)
xn, yn = f(xn-1, yn-1)
```

For large enough values of n, the resulting point at xn, yn will fall into one of two categories:

1- The point will still be close to the origin (0,0) of the plane. We define close to mean a distance *less than or equal to* 2.

2- The point will be far from the origin. We define far to mean a distance greater than 2.

If the resulting point is close to the origin, it is part of the Julia set for the function f(x,y). Otherwise it is not part of the Julia set for the function f(x,y). A point that is not part of the Julia set "escapes" at the first iteration where the new (x,y) point is a distance of more than 2 from the origin. For points that escape, we will want to remember the "escape count", or on which iteration it escaped.

From Functions to Images

So, where do the interesting pictures come from?

We choose an actual function for `f(x,y)`, then color each point in the plane based on how far it is from being in the Julia set for the selected function. "How far" is the "escape count".

Actually, there are infinitely many points in the plane, so we can't do every point. Instead, we select a regular grid of points each representing the points near it in the plane. For each point in the grid:

- Calculate its `x0,y0` value, from its position in the grid.
- Iterate the application of `f(x,y)` up to `n` times.
- If the result becomes far away, remember the iteration number when it first escaped (became far away from the origin). We call this the "escape value". Points with lower escape values are further from the Julia set than those with higher escape values.
- If the result doesn't become far away (escape) within `n` iterations, assume it did not escape and remember `n` as the escape value. These points are assumed to be part of the Julia set for `f(x,y)`.

Now, we have a set of integers (the escape values) that represent how close each of the points in the grid are to the Julia set for `f(x,y)`. Larger numbers mean they are closer to the Julia set.

Since the points selected are in a regular grid, we can pair each grid point with a pixel in a rectangular image. We assign the color of pixel based on the escape value of the corresponding grid point. All pixels associated with the same escape value will have the same color.

Defining a Regular Grid

We will use the `ComplexFractal` class to calculate (x,y) values from (row,column) numbers.

Our Function f(x,y)

For our function `f(x,y)` we will use this definition:

```
x' = x*x - y*y + a
y' = 2*x*y + b
```

where, `a` and `b` are parameters used to configure a particular version of the family of functions described by these equations.

Assignment

In this assignment you will create a class to create and store a Julia set's escape values. It will inherit from `ComplexFractal`, which inherits from the `NumberGrid` class. We will use the `NumberGrid` height and width to store the grid size. The `NumberGrid` maximum number will be used as the maximum escape value, and the `NumberGrid` numbers will be the `JuliaSet` escape values. The `ComplexFractal` class will keep track of mapping between row,column and x,y coordinates. The `JuliaSet` class will add the ability to calculate escape counts for points in the plane. It will also track the `a` and `b` parameters necessary for our chosen function.

You will also extend the `ppm_menu` program to add a new command.

The new commands required are:

- `julia-parameters`: Set the parameters of the Julia Set function.
- `complex-fractal`: Choose to make a complex plane.
- `julia`: Choose to make a Julia set.

Programming Requirements

The following files must be updated or created and stored in the `src` directory of your repository.

Create `JuliaSet.{h,cpp}`

This class inherits publicly from `ComplexFractal`.

The `NumberGrid`'s maximum number will be used for the `JuliaSet`'s maximum escape count.

Data Members:

- `double`: the `a` parameter for the Julia set function.
- `double`: the `b` parameter for the Julia set function.

Methods:

- `JuliaSet( );` Uses `ComplexFractal`'s default constructor. Additionally sets `a` to -0.650492, `b` to -0.478235, and maximum number count to 255.
- `JuliaSet( const int& height, const int& width, const double& min_x, const double& max_x, const double& min_y, const double& max_y, const double& a, const double& b );` Constructor. Sets up the `ComplexFractal` and `JuliaSet` data members from parameters.
- `virtual ~JuliaSet( );` Must exist, but has empty code block.
- `double getA( ) const;` Return the a parameter for the Julia set.
- `double getB( ) const;` Return the b parameter for the Julia set.
- `void setParameters( const double& a, const double& b );` Sets a and b parameters. Only allows values in the range -2.0 to 2.0 for each. If either is out of range, change nothing.
- `virtual void calculateNextPoint( const double x0, const double y0, double& x1, double &y1 ) const;` Calculate the next escape point after x0, y0 and store in x1, y1. Note that x1 and y1 are return by reference. This is the Julia set function `x',y' = f(x,y)`.
- `virtual int calculatePlaneEscapeCount( const double& x0, const double& y0 ) const;` Calculate the number of iterations required for x0, y0 to escape. The return value should be in the range 0 to maximum escape count, inclusive. 0 means immediately escaped, before any applications of the function. Maximum escape count means never escaped, or escaped on the last step. Escape means the distance from the origin is *more than 2*.
- `virtual int calculateNumber( const int& row, const int& column ) const;` Calculate the number of

iterations required for row, column to escape. The return value should be in the range 0 to maximum escape count, inclusive. 0 means immediately escaped. Maximum escape count means never escaped, or escaped on the last step. If row or column is out of range, return -1. Should use `calculatePlaneCoordinatesFromPixelCoordinates()` and `calculatePlaneEscapeCount()`.

Add New Functions to `image_menu.h` and `image_drawing.cpp`

- `void setJuliaParameters(ActionData& action_data);` Asks the user for the `double`s "Parameter a? " and "Parameter b? ". Then sets the parameters. Only does this work if the `grid` is actually a `JuliaSet` object. Otherwise gives the message "Not a JuliaSet object. Can't set parameters.".

Update/Add Functions in `image_menu.h` and `controllers.cpp`

- `void setComplexFractal( ActionData& action_data );` Use `setGrid()` to set `action_data`'s grid to a `ComplexFractal` object allocated from the heap.
- `void setJuliaFractal( ActionData& action_data );` Use `setGrid()` to set `action_data`'s grid to a `JuliaSet` object allocated from the heap.
- `void configureMenu( MenuData& menu_data )` add the new actions with the names and descriptions listed below.

Table of New Commands

Command Name	Function Name	Description
julia-parameters	setJuliaParameters	Set the parameters of the Julia Set function.
complex-fractal	setComplexFractal	Choose to make a complex plane.
julia	setJuliaFractal	Choose to make a Julia set.

Update `Makefile`

The following commands should work correctly.

- `make hello` - builds the hello program
- `make questions_3` - builds the questions_3 program
- `make ascii_image` - builds the ascii_image program
- `make image_file` - builds the image_file program
- `make ppm_menu` - builds the image_file program
- `make all` - builds all programs
- `make` - builds all programs (same as `make all`)
- `make clean` - removes all .o files, and all executable programs

Additional Documentation

- C++ Reference
- Examples from class
- Julia set on Wikipedia

Sample PPM Images

- Sample Output1
- Sample Output2
- Sample Output3

Show Off Your Work

To receive credit for this assignment, you must

- use git to add, commit and push your solution to your repository for this class.
- successfully pass all unit tests and acceptance tests

Additionally, the program must build, run and give correct output.

Extra Challenges (Not Required)

- Create classes that inherit from `JuliaSet` that have different functions, `f(x,y)`, for calculating escape

values. Add the ability to use them from the program.
- Try other ways to modify the plane and parameters that would make it easier to create interesting Julia set images.