WORKSHEET B: GENERATING FRACTALS

Version 1.0
BSc Computing for Games
COMP130

Dr Ed Powley

Introduction

In this worksheet you will implement, within a provided template program, C++ code to generate and display the *Mandelbrot set* fractal. This fractal colours each pixel of the image according to an iterated mathematical formula, as described below.

1 Remapping coordinates

To generate an interesting fractal, the on-screen x and y coordinates must first be rescaled. In the skeleton project the pixel coordinates range from 0 to 800, whereas the Mandelbrot set fractal is most interesting in the region $-2 \leq x \leq 1$ and $-1.5 \leq y \leq 1.5$.

"Clouds are not spheres, mountains are not cones, coastlines are not circles, and bark is not smooth, nor does lightning travel in a straight line."

— Benoit Mandelbrot

Let p_x be the x coordinate of the pixel. This can be remapped into the range x_{min} to x_{max} using the following formula:

$$x_0 = \frac{p_x}{\text{image.width}} \times (x_{\text{max}} - x_{\text{min}}) + x_{\text{min}}$$

The y coordinate can be remapped using a similar formula.

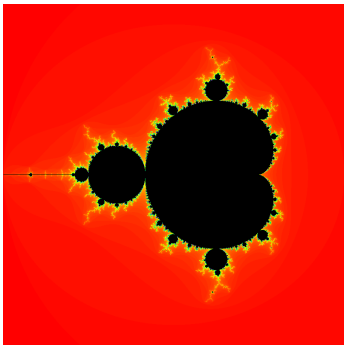
Implement the above calculations for the x and y coordinates, at the indicated parts of `Mandelbrot.cpp`.

A common pitfall in implementing this is to forget that dividing an integer by an integer in C++ results in an integer, so one or both of the division operands must be cast to a floating point type in order to obtain the correct result.

2 The Mandelbrot set

The Mandelbrot set is based on the following sequence of numbers. Let x_0 and y_0 be the coordinates of a point in the image. Then the sequence $x_1, y_1, x_2, y_2, x_3, y_3, \ldots$ is defined[1] recursively for $i = 0, 1, 2, 3, \ldots$ by:

$$x_{i+1} = (x_i)^2 - (y_i)^2 + x_0$$
$$y_{i+1} = (2 \times x_i \times y_i) + y_0$$



The Mandelbrot set fractal.

The points are coloured according to the *smallest* value of i for which $(x_i)^2 + (y_i)^2 \geq 4$. If such a value of i is not found after a large number of iterations (for example $i = 200$), the pixel is coloured black.

[1] If you are familiar with complex numbers, you may notice that this is equivalent to $z_{j+1} = z_j^2 + z_0$, where $z_j = x_j + y_j i$.

Implement an algorithm which performs the above iterative computation, determining the smallest value of i for which $(x_i)^2 + (y_i)^2 \geq 4$ and selecting a pixel colour accordingly. Implement the algorithm in `Mandelbrot.cpp` so that the program generates the Mandelbrot set fractal when it is run.

3 Colours

The Mandelbrot set image in this worksheet chooses pixel colours as follows. If the maximum number of iterations is reached, the pixel is coloured black. Otherwise, the iteration count i is used to specify a colour in HSV space, with saturation 1, value 1 and hue varying with i.

Implement the above colour scheme into your program. You will need to find an algorithm for converting colours from HSV to RGB; online sources are fine. Remember that if you use algorithms or code that are not your own, it must be attributed with comments in the source code.

4 Stretch goal

One of the appeals of the Mandelbrot set is that one can zoom into it to view an infinite level of detail and variation. Using the CImg library documentation at `http://cimg.eu`, **implement** a feature into your program that allows the user to zoom into the image using the mouse. Upon zooming, the program should regenerate the fractal with appropriately adjusted limits for the x and y coordinates.

Submission instructions

Begin by **forking** the GitHub repository at the following URL:

`https://github.com/Falmouth-Games-Academy/comp130-worksheets`

The `worksheet_B` directory contains a Visual C++ project to build upon. **Read** and **understand** the provided code. **Add** code to satisfy the requirements listed above. You may make any changes to the provided code that you see fit, and may add new files to the project, however please do **not** move or rename the project or the existing files.

Attend the timetabled COMP130 session in **week commencing 6th February 2017**, ensuring that you have uploaded all material to GitHub before this time.

Marking criteria

Remember that **it is better to submit incomplete work than to submit nothing at all**. Any attempt, even unfinished, will receive a passing grade.

Your work will be marked according to the following criteria:

- **Functional coherence**. Is your implementation correct and free of obvious bugs? Does it adhere to the specification outlined in this worksheet?
- **Sophistication**. Have you made use of appropriate code structures and data structures? Note the emphasis is on **appropriate**; extra credit will **not** be given for unnecessarily complex solutions.
- **Maintainability**. Is your code well commented? Are your identifier names appropriate and descriptive? Have you adhered to appropriate coding standards?
- **Stretch goal**. Has the zoom feature been implemented? How well does it work?