Programming Club
Fractals

Hugh Leather

20^{th} October 2016

1 Pretty pictures

First let's have an easy way to make an image.

The simplest picture format you might use is PPM.

You can, however use any way to show the image you like – if your language lets you do graphical interfaces, then try that.

The ASCII PPM format is:

- A "magic number" for identifying the file type. A ppm image's magic number is the two characters "P3".
- Whitespace (blanks, TABs, CRs, LFs).
- A width, formatted as ASCII characters in decimal.
- Whitespace.
- A height, again in ASCII decimal.
- Whitespace.
- The maximum colour value (Maxval), again in ASCII decimal. Must be less than 65536 and more than zero. Use 255 for 8 bit.
- A single whitespace character (usually a newline).
- A raster of Height rows, in order from top to bottom. Each row consists of Width pixels, in order from left to right. Each pixel is a triplet of red, green, and blue samples, in that order. Each sample is represented by one or more ASCII characters followed by whitespace.
- A row of an image is horizontal. A column is vertical. The pixels in the image are square and contiguous.

Alternatively you can use the more compact binary PPM format, which differs in the following ways:

- The binary ppm's magic number is the two characters "P6".
- Samples (Red, Green, Blue values for each pixel) are represented in pure binary by either 1 or 2 bytes. No whitespace can appear between samples and pixels. If the Maxval is less than 256, each sample is 1 byte. Otherwise, it is 2 bytes. The most significant byte is first.
- For a Maxval of 255, each pixel requires only 3 bytes, while for the ASCII PPM it requires 6 to 12 bytes, reducing the space needed by $>$60%.

If you can't see a PPM file on your system, try converting it with Image Magick: `convert image.ppm image.jpg`

1.1 An Image Abstraction

Make a class (or other abstraction depending on your language) that encapsulates an RGB colour.

Make a class (or other abstraction depending on your language) that encapsulates a simple image with a given width and height and assumes 8 bit RGB.

Give your class operations to set and get a pixel.

Give your class an operation to write it to a file.

Test your class by making an image that is 200x100 pixels, all red and writing to a file.

Test your class by making an image that is 200x100 pixels, with black at the top left pixel, red at the top right pixel, blue at the bottom left pixel, magenta at the bottom right, and smoothly interpolating in between.

2 Mandelbrots

Build a simple Mandelbrot display in any language you like.

The Mandelbrot set is those complex points, c, for which $f_c(z) = z^2 + c$ does not diverge when iterated.

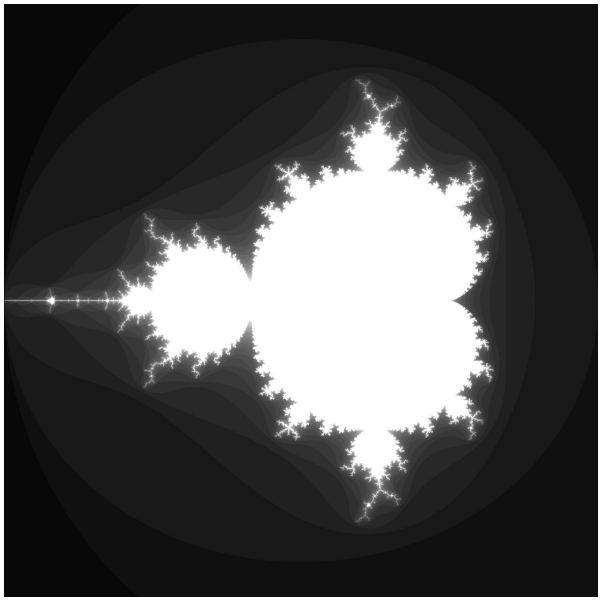
We'll take a pretty simple case, using the escape time algorithm. That is, for a given point, c, start with

$$z_0 = 0$$
$$z_n = z_{n-1}^2 + c$$

Now iterate and find out the first n for which $|z_n| > 2$ or $n > 32$ (the latter bit stops you going on forever).

This gives you number, n, in the interval, $[0, 32]$. You can convert this number into a colour, making 0 black, 32 white, and interpolating in between.

Now imagine your image ranges over the complex space, with the x axis ranging from -2 to 1, and the y axis ranging from -1.5i to 1.5i. You can now plot the Mandelbrot set for this part of the space. It should look like this:



Change the parameters and have a zoom about.

2.1 Other things

Here some ideas of things you might try (or move on to the next fractal).

- Build a GUI for it
- Try histogram colouring
- Try continuous colouring
- Do Multibrots - $z_n = z_{n-1}^d + c$, for some d
- Try contour or border mapping
- How do you go beyond 64 bit precision?
- Try showing Julia sets

3 Brownian Trees

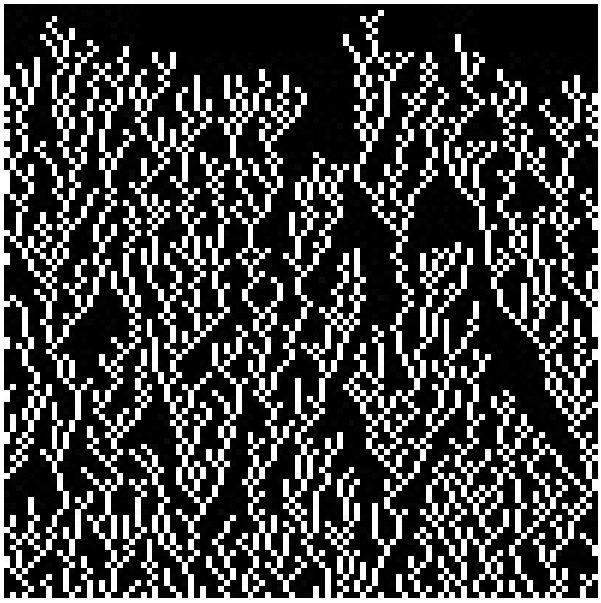
Here we're going to make very simple Brownian trees.

Choose a random x value and "drop a flake of snow" from the top of the image, straight down from the x axis, until there is a white pixel just below it, or one to left and below, or to the right and below, or until it hits the bottom. When it stops, leave it there in white. Repeat.

Rather than dropping the snow a square at time, you can make it more efficient by remembering the highest value for each x coordinate and using that directly to work out where a bit of snow should stop.

Finish when you can't drop snow more than one square.

Do this on a 100x100 image, and you should get something that looks like this:



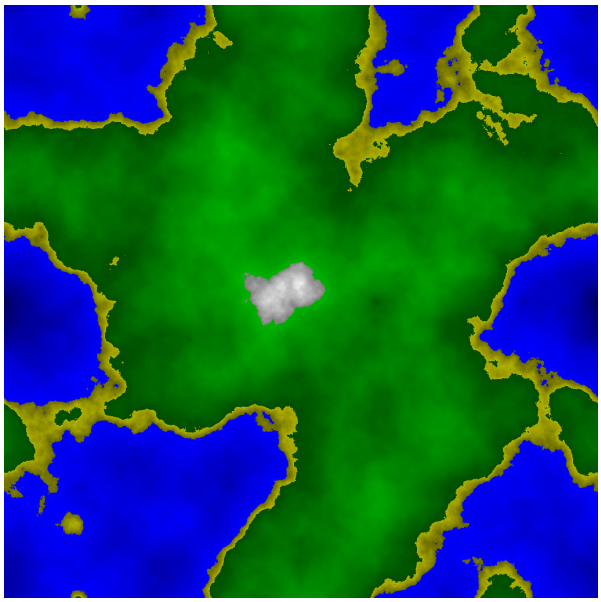
You can make the colours more interesting by changing the colour to be near the colour you land on.

4 Fractal Landscapes

Create a landscape with the diamond-square technique.

There is a very good explanation on Wikipedia, so I will just let you read that - `https://en.wikipedia.org/wiki/Diamond-square_algorithm`

Choose some good mapping for the heights to colours. Here's one I did with sea, beach, forest, and snow capped mountains (note this is with dimension 2^n using wrapping so it tiles with itself).



Find fast 3d package for your language and do a 'fly over' of the landscape in the style of Zarch `https://en.wikipedia.org/wiki/Zarch`.