Making a Mandelbrot Movie

http://dotat.at/prog/mandelbrot/

Tony Finch <dot@dotat.at>

Monday 8 November 2010

Abstract

How I made a movie that zooms from 2^2 to 2^{-48}.

1 Intro

Benoît Mandelbrot, 1924 – 2010

When Mandelbrot died on the 14th October I thought I could celebrate his life and work by trying to do something with Mandelbrot sets that I haven't done before.

Coloured level sets

When I drew Mandelbrot sets as a teenager they looked a bit like this. The blocky colouring and noisy fine detail are not as pretty as some of the professional fractal art I had seen.

Renormalized iteration count

So the first thing I wanted to do is draw Mandelbrot sets where the points outside the set are smoothly coloured, like this.

Renormalized iteration count

```
/* You are not expected to understand this.  */
```

```
while (n++ < max &&
        x2+y2 < inf) {
    y = 2*x*y + b;
    x = x2-y2 + a;
    y2 = y*y;
    x2 = x*x;
}
nu = n - log(log(x2+y2)/2)
          / log(2);
```

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

$$\nu = n - \log_2(\log|z_n|)$$

It turns out to be quite simple to code and quick to execute. The inner loop is exactly the same as before. You just need to take a couple of logarithms when you find the point is outside the set [1].

Log normalized iteration count

You can get rid of the visual noise close to the Mandelbrot set itself by taking the log of the normalized iteration count. This is exactly the same view of the Manelbrot set as the last two I showed you, just coloured more nicely.

Tentacles!

Now you know how to make stunning pictures like this.

Demo

But this talk is supposed to be about *moving* pictures!

- Here's what I made ...

Next I wanted to make moving pictures of the Mandelbrot set, so I'm going to explain how while showing you what I made.

2 Making a movie

It turns out to be pretty easy to turn a stack of images into a movie.

`ffmpeg` will encode a directory containing serial-numbered `ppm` files into a film, though you have to Google a bit to find out how.

Make sure that `ffmpeg` uses `x264` as the codec, otherwise you will suffer from horrible compression artefacts and bloaty file sizes.

The reason for using `ppm` files is they are completely trivial to write, even if they are a bit big. (I have 22GB working files for this movie.)

I have separated the Mandelbrot generation into two stages.

The first stage just does the Mandelbrot calculation and emits an array of normalized iteration counts to a file.

The second stage converts the iteration count array into a `ppm` file.

I can change the colouring or experiment with other effects without having to re-calculate every frame from scratch.

One thing I haven't yet worked out is how to get rid of the flickering alias effects - Sorry!

The Mandelbrot calculation itself is specially designed for deep zooms.

The program re-scans the points that haven't yet escaped, going exponentially deeper and deeper until the number of pixels that escapes during a scan is below a threshold.

There is also a minimum number of pixels that must escape before it stops scanning, in case the whole image is deeper than the starting depth.

Near the end of the movie each frame requires millions of iterations per pixel, and takes many minutes to calculate.

I used GNU `parallel` to calculate multiple frames concurrently.

This zoom goes as deep as possible using double precision floating point arithmetic.

We start running out of resolution at a pixel size of 2^{-50}.

When the total depth of a zoom is 2^{53} then the startimg image has been blown up to a lightyear across, because it turns out a lightyear is about 2^{53} metres!

3 Conclusion

- `http://dotat.at/prog/mandelbrot/`
- `http://fanf.livejournal.com/`
- `http://twitter.com/fanf`

The code, slides, and notes will be available on my web site. You might also have a look at my blog and twitter feed.

References

[1] Linas Vepstas. Renormalizing the mandelbrot escape. `http://linas.org/art-gallery/escape/escape.html`.