An introduction to the Mandelbrot set

Bastian Fredriksson

January 2015

1 Purpose and content

The purpose of this paper is to introduce the reader to the very useful subject of fractals. We will focus on the Mandelbrot set and the related Julia sets. I will show some ways of visualising these sets and how to make a program that renders them. Finally, I will explain a key exchange algorithm based on what we have learnt.

2 Introduction

The Mandelbrot set and the Julia sets are sets of points in the complex plane. Julia sets were first studied by the French mathematicians Pierre Fatou and Gaston Julia in the early 20th century. However, at this point in time there were no computers, and this made it practically impossible to study the structure of the set more closely, since large amount of computational power was needed.

Their discoveries was left in the dark until 1961, when a Jewish-Polish mathematician named Benoit Mandelbrot began his research at IBM. His task was to reduce the white noise that disturbed the transmission on telephony lines [3]. It seemed like the noise came in bursts, sometimes there were a lot of disturbance, and sometimes there was no disturbance at all. Also, if you examined a period of time with a lot of noise-problems, you could still find periods without noise [4]. Could it be possible to come up with a model that explains when there is noise or not?

Mandelbrot took a quite radical approach to the problem at hand, and chose to visualise the data. The results showed a structure with self-similarity at all scales. This is called a fractal. There are many sorts of fractals, but what they all have in common are that they show signs of self-similarity. That means, when you zoom into the fractal, you will notice that some patterns repeat themselves. The fractal first studied by Mandelbrot was indeed the fractal generated by white noise from the telephony lines, also known as the Cantor dust fractal [3,6].

All fractals can be generated using an Iterated Function System (IFS). An IFS consists of a function f which is executed in a feedback loop, also known to computer students as a recursive function. That means, the output of the function is given as input when the function calls itself. The first time the

function is called, you input some initial value $f(0)$ [2]. The Cantor dust fractal can be created using a black square as initial value. Each time you run the IFS in a feedback loop you split each square into four smaller squares and put them beside each other as shown in fig 1. If you repeat this process many times,

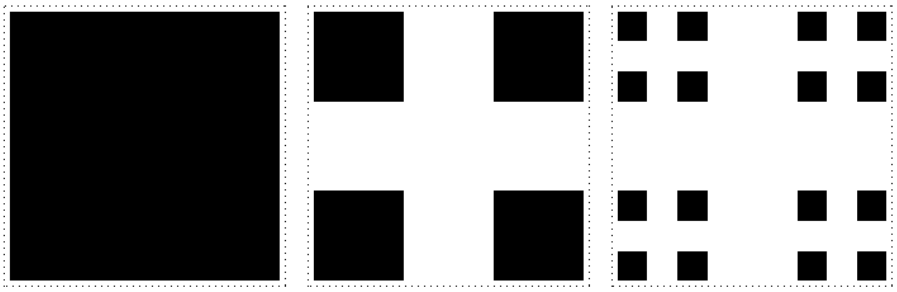

Figure 1: The Cantor dust fractal is generated by starting with a black square. In each iteration you split the square into four parts, thus creating clusters of "dust" which can be used to model white noise on telephony lines.

you will eventually end up with a large amount of uniformly distributed dots, or "Cantor dust". The dots represents short periods of time where there is disturbance on the line. The dust appears, as you can see, in clusters. These clusters represents "a burst" of noise, periods of time where there is a lot of disturbance. The challenge for the engineers became to deal with these noise clusters and correct any errors as fast as possible [4]. The example with Cantor dust is just one example among many others, of phenomena in nature that can be modeled using fractals. Many other things, like a the helix of a shell or the surface of a broccoli prove to have fractal properties. There is even a theory called *Fractal Cosmology* which states that the distribution of matter in the universe can be modeled as a fractal.

3 The Mandelbrot set

Encouraged by his findings, Benoit Mandelbrot continued his research at IBM. He began studying the work on Julia sets started by Pierre Fatou and Gaston Julia. With high-powered computers at his disposal he plotted the sets on paper. He noted that they were also fractals, with astonishing richness of detail. By making a small change to the IFS used by Fatou and Julia he came up with another fractal, which was later to be known as the Mandelbrot fractal or the Mandelbrot set. Benoit Mandelbrot wrote down his findings in the book *The Fractal Geometry of Nature* which was published in 1982 [3].

3.1 Definition

As I mentioned, the Mandelbrot set is a set of points in the complex plane. The complex plane is a two-dimensional space with the a vertical imaginary axis, and a horizontal real axis. A point in the plane can be described using a complex number $c \in \mathbb{C}$ written on the form $c = a + bi$ where $a, b \in \mathbb{R}$ and $i = \sqrt{-1}$. If you let the points belonging to the Mandelbrot set to be coloured in black, you obtain the shape depicted in figure 3.

Now we are ready to make a formal definition of the Mandelbrot set. A point $c \in \mathbb{C}$ belong to the Mandelbrot set iff

$$\lim_{n\to\infty} ||z_{n+1} = z_n^2 + c|| \nrightarrow \infty \text{ where } z_0 = 0$$

Here we have an IFS with the recursive formula $z_{n+1} = z_n + c$ and an initial value of $z_0 = 0$. In each loop, you square the previous number and seed the result with the value of c. The vertical brackets denotes the Euclidean norm, which is a measurement of how far away a point in the plane is from origo, $||z|| = \sqrt{a^2 + b^2}$. A point c belongs to the Mandelbrot set if it remains bounded when we run the formula in a feedback loop. For example, the point $c = -1 + 0.25i$ belongs to the Mandelbrot set. If we apply the formula above, we retrieve the number series $z_0 = 0, z_1 = -1+0.25i, z_2 = -0.0625-0.25i, z_3 \approx -0.059 + 0.281i, z_4 \approx 0.042 - 0.345i, z_5 \approx -1.118 + 0.221i$ and so on. It is not obvious whether the point will eventually escape towards infinity or not, and it is usually not possible to tell. This is the reason why, in practical implementations, you run the feedback loop for a maximum number of times, say n times. If the point still remains within a radius of 2 from origo you consider the point to belong to the set. Why 2? Well, it is possible to show, and this an important result, that if $||z_n|| \geq 2$, z_n will eventually escape towards infinity [7]. This is called the *bailout radius*. When you zoom into the set, you will notice that you will have to increase the size of n. This is why the rendering of fractals is such a time-consuming task.

3.2 Julia sets

Julia sets and the Mandelbrot set are intrinsically connected through the fact that they are created using the same formula. This can be utilised to perform a cryptographic key exchange, see *Application in cryptography*. While the Mandelbrot set is created using different values on c and an initial value of $z_0 = 0$, a Julia set is created using a fixed c as seed and different values on z. The complex number c can be chosen freely [1]. If the point c chosen does not belong to the Mandelbrot set, the resulting fractal will be a Cantor dust fractal. To determine whether a point z belongs to the Julia set with seed c, iterate the formula $z_{n+1} = z_n + c$ in the same manner as with the Mandelbrot set.

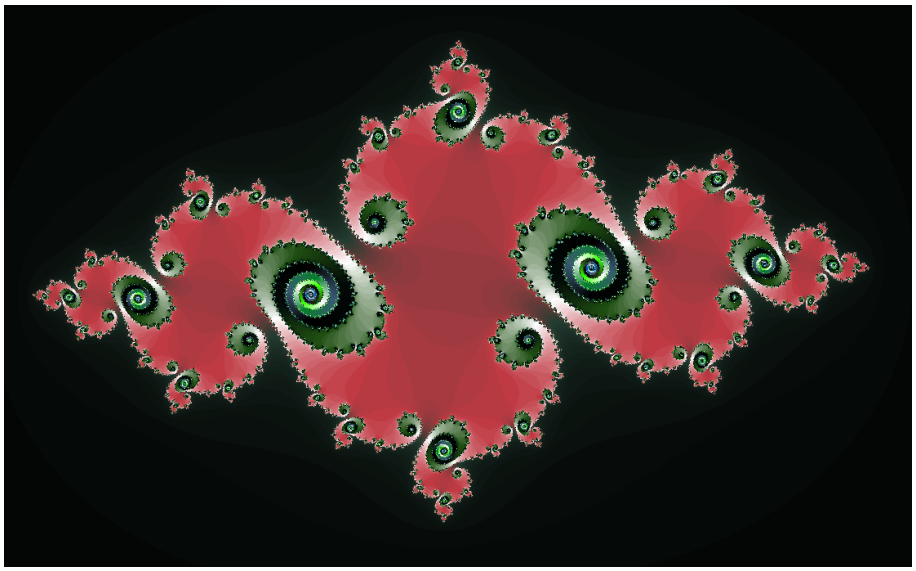

Figure 2: The Julia set generated from the seed $c = -0.755262 + 0.094211i$.

4 Exploring the set

The Cantor dust fractal is not very exciting because it looks almost the same at all scales. The Mandelbrot fractal on the other hand is quite the opposite. When you zoom into the set, you will notice that new patterns emerge. The author has spent many hours zooming into the set, exploring new fascinating structures.

The Mandelbrot set is actually a great example of how you can store an infinite amount of information on a finite medium. The prerequisite for creating an artistically appealing fractal lies in the existence of a colouring function $c(x)$. The purpose of the colouring function is often to colour the points which lies outside the set. This is called the aura of the Mandelbrot set. It is also possible to colour the set itself, although it is usually coloured in a fixed colour like black. In this section we will go through some methods of colouring and see what the result looks like.

4.1 The escape time algorithm

The most common algorithm for colouring the aura of the set is called the escape time algorithm. The escape time algorithm is based on the value of n, that means, the number of iterations before z_n end up outside the bailout radius. A simple, yet beautiful way of colouring, is to colour the actual set in black and then fade the aura from red to black. One way of defining the colouring function

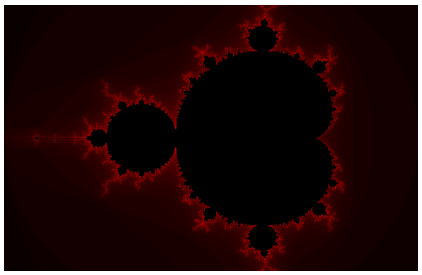

Figure 3: The aura of the set coloured using the traditional escape time algorithm. The black bulb in the middle is the actual Mandelbrot set.

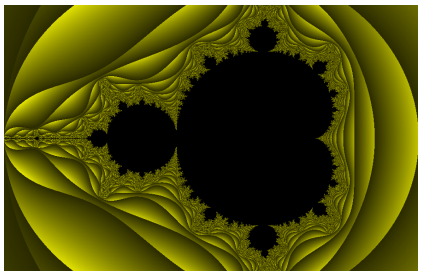

Figure 4: The aura coloured by measuring the distance from origo and scaling it with a constant.

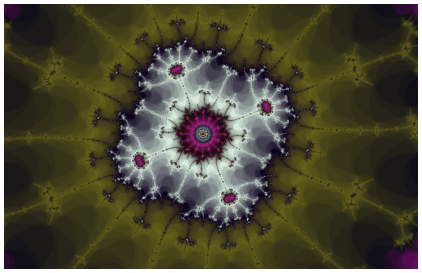

Figure 5: Part of the Mandelbrot set coloured with a palette.

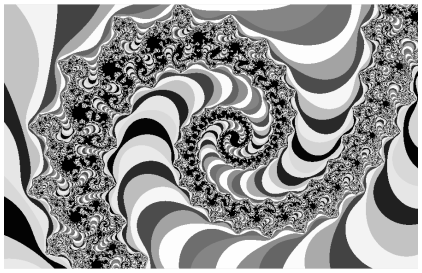

Figure 6: A Mandelbrot spiral coloured using cosine.

would then be:

$$c(n) = \begin{cases} 0 & n = max(n) \\ 255 - n/max(n) + 1 & n < max(n) \end{cases}$$

$c(n)$ returns the red component of a RGB-colour. If you want be able to colour your fractal in many different colours, it is often convenient to use three functions, $c_r(x)$, $c_g(x)$ and $c_b(x)$ where c_r returns the red component, c_g returns the green component and c_b returns the blue component. Another approach is to prepare a colour palette, and then let the function $c(x)$ point out a colour in the palette. In this way, you can create e.g. rainbow effects.

4.2 Buddhabrot

Another variant of the Mandelbrot set is the Buddhabrot. The name arises from the fact that the fractal looks like a meditating Buddha. An image of a Buddhabrot was first posted by Melinda Green on Usenet in 1993 [8]. At first glance it does not look like a Mandelbrot at all, but it is only the colouring that

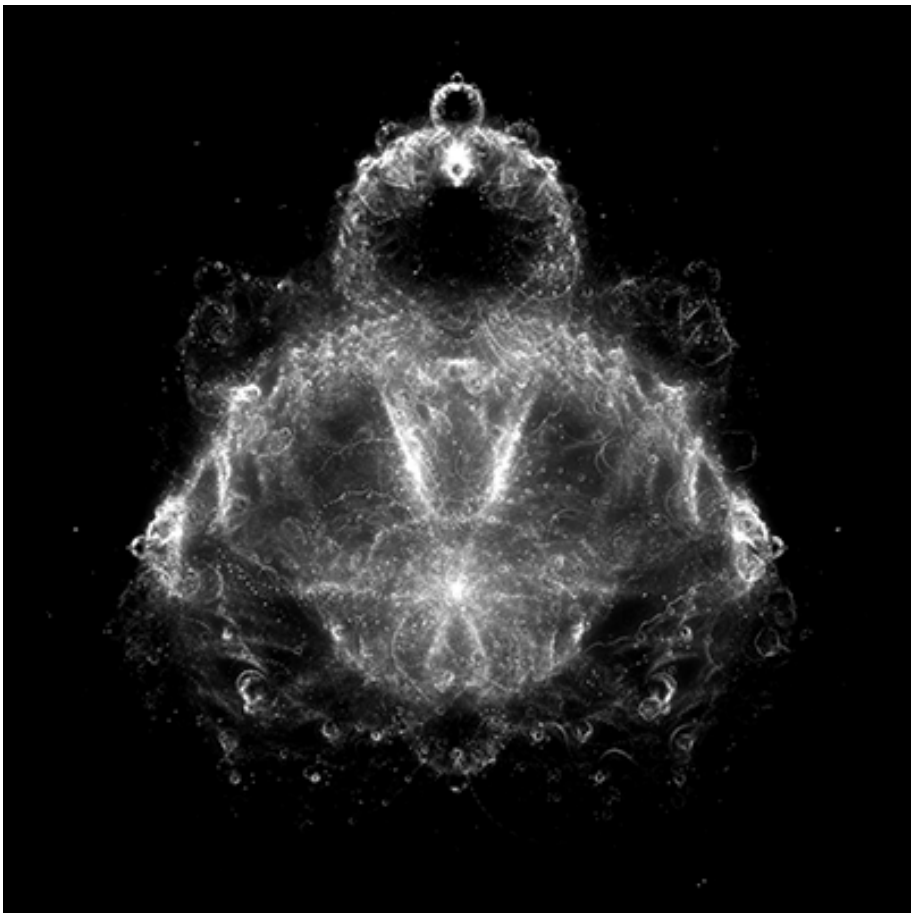

Figure 7: A Buddhabrot generated with $n = 10^6$ [5]

.

differs. Instead of counting the number of iterations required, like in the escape time algorithm, you count the number of times a specific point has been visited. The fractal is rendered by splitting the canvas into a matrix M, where each element in the matrix represents the number of times this area of the fractal has been visited. For each output from the function z_n you increment the number on the corresponding position in the matrix. Once you have executed z_n for all possible c you make a second pass over M. Here you can use a colouring function $c(x)$ similar to what we described in section 3.1 to map the elements in M to colours.

5 Implementation

This section contains sample code written in C# to get you started rendering your own fractals. Begin by creating a canvas with a size of $n \times m$ pixels. Each pixel represents a unique c. Transform the position of the pixel on screen to a complex number in the plane. Given that the coordinates of the complex number can be written as (x, y) and the plane has the following boundaries; $x_{min} \leq x \leq x_{max}$ and $y_{min} \leq y \leq y_{max}$, the transformation functions could look something like this:

```
double trans_x(double x, double x_min, double x_max) {
    return x / (m / (x_max - x_min)) + x_min;
}

double trans_y(double y, double x_min, double x_max) {
    return y_max - y / (n / (y_max - y_min));
}
```

Now, for each pixel in your canvas, iterate the formula $z_{n+1} = z_n + c$ until z_n falls outside the bailout radius or until $n = max(n)$. Depending on how you want to colour your fractal you might either store the counter n or some other information you need to remember. In the example below, a complex number is represented using two doubles, where re is the real part and im is the coefficient for the imaginary part.

```
for (int y = 0; y < n; y++) {
    for (int x = 0; x < m; x++) {
        double re = 0; // z_{n+1}
        double im = 0;
        double re_old = 0; // z_{n}
        double im_old = 0;
        double a = trans_x(x, x_min, x_max);
        double b = trans_y(y, y_min, y_max);
        int n;
        for (n = 0; Math.Pow(re, 2) +
            Math.Pow(im, 2) < 4 && n < N_MAX; n++) {
            // z_{n+1}=z_{n}^2+c
            re_old = Math.Pow(re, 2) - Math.Pow(im, 2);
            im_old = 2.0 * re * im;
            re = a + re_old;
            im = b + im_old;
        }
        colour_pixel(x, y, n);
    }
}
```

5.1 Optimisations

You might have noticed that we have done the transformation

$$||z|| = \sqrt{x^2 + y^2} \to ||z||^2 = x^2 + y^2$$

to avoid the time consuming square root function. Other optimisations you can do is to split the canvas into say, blocks of $n/10 \times m/10$ pixels, and calculate each block in its own thread. This should speed up calculations a lot, since you exploit all cores in the CPU. You can also use a technique called periodicity checking. Each number series given by calculating z_n for different n can be seen as an orbit. If there is a cycle in the orbit, then we know that z_n cannot diverge when $n \to \infty$, hence the point must be a part of the Mandelbrot set. However, cycle checking with floating point arithmetic is difficult and if you do not get it right, you might end up with a program running even slower, so I would not recommend this kind of optimisation unless you really need speed [9]. You can also decrease the bailout radius, but be careful, the results can be inaccurate. Another thing you could try is to do the calculations on your graphics card. Fractals should be suitable for graphic cards, since they are better at floating point arithmetic's, and since it is easy to split computations into smaller pieces.

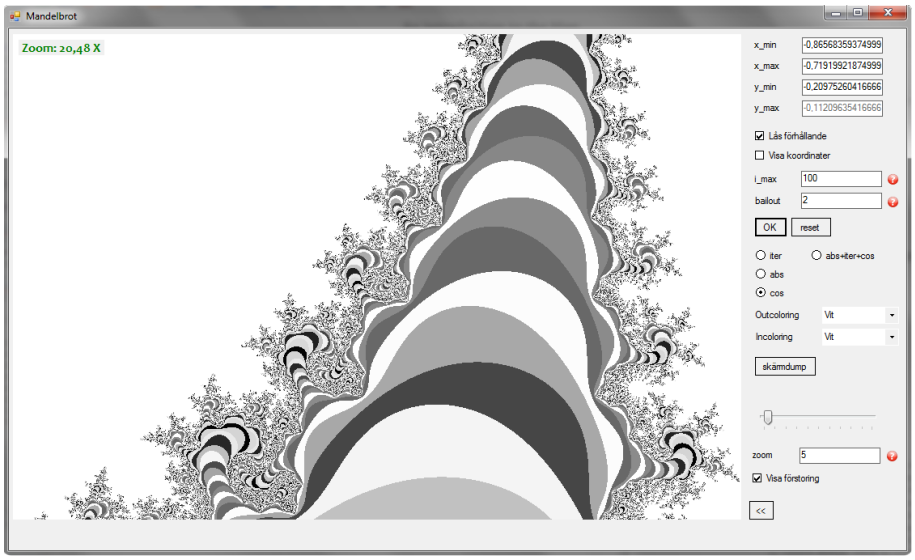

Figure 8: The Windows program *Mandelbrot* made by the author. If you do not want to make your own program, you can try *XaoS* or *Fractal eXtreme*.

6 Application in cryptography

Fractals is not only about beautiful images and computer graphics. The chaotic nature of fractals have made them suitable for cryptographic applications such as hash functions and encryption. In this section, we will go through a key exchange protocol based on the Mandelbrot set.

6.1 Fractal key exchange

In cryptography, a key exchange is the process of two parties, Alice and Bob, exchanging keys with each other allowing the use of a cryptographic algorithm. This is done through a key exchange protocol such as Diffie-Hellman. Diffie-Hellman has an important property; even if Eve is eavesdropping on the traffic sent between Alice and Bob, she cannot deduct the common secret. This is what makes Diffie-Hellman secure. Computer security researchers Mohammad Ahmad Alia and Azman Bin Samsudin at University Sains Malaysia have invented a new type of key exchange protocol which utilises floating point arithmetics [1]. The protocol is very similar to Diffie-Hellman, and relies on the following equivalence relation:

$$c^{n-x}q_n e = c^{k-x}q_k d \quad \forall x \in \mathbb{Z} \text{ where } c, d, e \in \mathbb{C} \text{ and } n, k \in \mathbb{N}$$

Where $q_i \in \mathbb{C}$ is the result from running q in a feedback loop i times. p_a and p_b as defined below are the public keys for Alice and Bob respectively.

1. The proposed protocol begins with Alice and Bob agreeing on a complex number c which belongs to the Mandelbrot set and an integer x. c and x are public and can be intercepted by a third party without compromising the protocol.

2. Alice generates her secret key consisting of the tuple (n, e) where $n > x$ and e belongs to the Mandelbrot set.

3. Bob generates his set key consisting of the tuple (k, d) where $k > x$ and d belongs to the Mandelbrot set.

4. Alice calculates z_n by iterating the formula $z_{i+1} = z_i c^2 e$, $z_0 = c$ (1) n times and sends $p_a = z_n e$ to Bob.

5. Bob calculates z_k by iterating the formula $z_{i+1} = z_i c^2 d$, $z_0 = c$ (1) k times and sends $p_b = z_k d$ to Alice.

6. Alice calculates the common secret $c^{n-x}q_n e$ by iterating the formula $q_{i+1} = q_i ce$, $q_0 = p_b$ (2) n times.

7. Bob calculates the common secret $c^{k-x}q_k d$ by iterating the formula $q_{i+1} = q_i cd$, $q_0 = p_a$ (2) k times.

The function (1) is called *Mandelfn* and the function (2) is called *Juliafn* because they bore a slight resemblance to the original Mandelbrot and Julia functions.

The keyspace, given a key with length n, is larger for the fractal key exchange algorithm than the keyspace for Diffie-Hellman, because the keyspace of Diffie-Hellman is limited to the number of primes in the field $\mathbb{Z}_p$ where p is the largest prime that can be represented by 2^n bits. However, using the fractal key exchange algorithm we can potentially end up with any key, which means that the keyspace is exactly 2^n. The fractal approach also has an interesting property. The values of n and k can be seen as a *load factor* that will slow down the calculations and increase the security of the protocol.

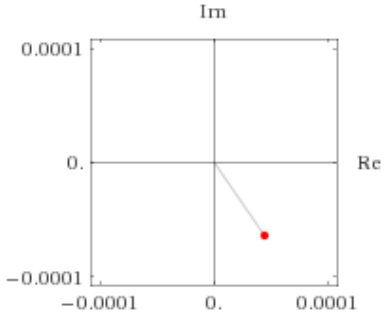

Figure 9: The common secret plotted in the complex plane.

6.2 An example

1. We choose $c = -0.6 + 0.12i$ and $x = 1$.
2. We let Alice's private key consist of $n = 2$ and $e = -0.1 + 0.72i$.
3. We let Bob's private key consist of $k = 3$ and $d = 0.21 - 0.35i$.
4. $z_1 = c^3 e$
   $z_2 = c^5 e^2$
   We send $p_a = c^5 e^3$ to Bob.
5. $z_1 = c^3 d$
   $z_2 = c^5 d^2$
   $z_3 = c^7 d^3$
   We send $p_b = c^7 d^4$ to Alice.
6. $q_1 = c^8 d^4 e$
   $q_2 = c^9 d^4 e^2$
   The common secret becomes:
   $cq_3 e = c^{10} d^4 e^3 \approx 0.0000443966 - 0.0000646526i$.

7. $q_1 = c^6 e^3 d$
   $q_2 = c^7 e^3 d^2$
   $q_3 = c^8 e^3 d^3$
   The common secret becomes:
   $c^2 q_3 d = c^{10} e^3 d^4 \approx 0.0000443966 - 0.0000646526i$.

7 Conclusion

I hope the reader has gained some insight about the Mandelbrot set. As shown in the paper, it has several practical applications, not only in computer graphics. The fractal is like pi, it shows up even when you least expect it to. This paper is, as the title suggests, only an introduction to the Mandelbrot set and fractals in general. There are many other cool things you can do with fractals. One example of a recent discovery in the field of fractals was made by Tom Lowe in 2010 when he discovered the 3-dimensional Mandelbox. Although it has never reached any commercial success, studies has shown that fractals are suitable for image compression. If you are interested in the subject, you might take a look in *References*. You can also try to enhance the key exchange algorithm described in section 5.

References

[1] Alia, M. A., and Samsudin, A. B. New Key Exchange Protocol Based on Mandelbrot and Julia Fractal Sets. *International Journal of Computer Science and Network Security, IJCSNS 7*, 2 (February 2007).

[2] Fisher, Y., Ed. *Fractal Image Compression: Theory and Application*. Springer-Verlag, London, UK, UK, 1995.

[3] IBM. Icons of progress - Fractal geometry, 2015. [Online; accessed 2015-01-21].

[4] Pritchard, J. *The Chaos Cookbook: A Practical Programming Guide*. Elsevier Science, 2014.

[5] UnreifeKirsche. Buddhabrot.

[6] Weisstein, E. W. Cantor dust. From MathWorld—A Wolfram Web Resource.

[7] Weisstein, E. W. The Mandelbrot set. From MathWorld—A Wolfram Web Resource.

[8] Wikipedia. Buddhabrot —Wikipedia, The Free Encyclopedia, 2015. [Online; accessed 2015-01-23].

[9] XaoS Development Team. Periodicity checking – XaOS v4.0 documentation, 2015. [Online; accessed 2015-01-23].

Bastian Fredriksson began his studies at The Royal University of Technology in 2012. He is currently studying his third year on the Bachelor's Programme in Computer Science.