FGR (Fractal Generation)

1. Introduction to Fractals

A fractal, in a simple definition, is a mathematical system whose properties suggest self-similarity, self-symmetry, and a measurably high degree of roughness. Mathematically, a fractal is a geometric figure whose surface area or perimeter is infinite yet it has a finite area. For many fractals, it is useful to think of them in terms of sets and set theory.

Fractals can be defined in terms of arithmetic of complex numbers along the real-imaginary system. Points z within the fractal set are denoted by the following: z within the fractal set are denoted by the following:

$$z_n \in F \text{ iff } \limsup_{n \to \infty} |z_{n+1}| \leq 5$$

Where z_n is a point, F is the fractal set and z_{n+1} is an iteration of z_n. The above equation suggests points are described whether further iterations of some arithmetic on that point remain bounded. In this example, where any iteration is less than or equal to 5. A very generalized form of how fractals will be expressed are in the form of the below:

$$z_{n+1} = z_n + c$$

That is, provided the above arithmetic does not unbound z_{n+1} as per the fractal definition given enough iterations, that point resides in the fractal set. The arithmetic does not necessarily need to be this rudimentary: fractals can be defined in many ways in terms of a relationship between z_n and z_{n+1}.

Consider some examples below:

$$z_{n+1} = z_n^2 + c$$
$$z_{n+1} = |z_n^2| + c$$
$$z_{n+1} = -z^2 + c^{-1}$$

And many more. This document will explore some of these fractal sets.

2. Application Architecture

FGR is a visualization tool meant to describe the geometric properties of fractals based on the aforementioned fractal set definition. It primarily relies on the arithmetic of complex numbers and iteration of an arithmetic function to determine set membership.

From a selection of various fractal set definitions, the user can choose which to visualize as well as zoom in or out to see the geometry of magnified fractals. Each fractal is also colored in a way to be visually aesthetic versus typical representations in monochrome.

FGR is a CLI and window based application with mouse and keyboard controls. The application is programmed in C++ in Microsoft Visual Studio and is multiplatform for both Windows and Linux. Standard C++ coding conventions are utilized as well as an object-oriented methodology to programming.

The application is split into several parts: there are four classes to consider:

- `Set`
  - the arithmetic expression which defines the fractal set
  - contains strictly logic related to determining points in the set
- `Fractal`
  - shared behavior among all fractal sets
  - concerns coloring and appearance of fractal visuals
- `ComplexNumber`
  - a definition of a complex number in a machine readable format
  - defines arithmetic based on complex numbers
- `Pixel`
  - container for pixel values retrieved from Set and provided to Fractal

Each class contains a `.h` header file declaring class member methods and attributes as well as a `.cpp` file defining the class. Additionally there is a main driver (`Source.cpp`) which is the entry point of the application and concerns how OpenGL interfaces with the system.

2.1. Class Structure

The primary class of FGR is `Set`, a C++ class used to define fractal behavior, notably whichever arithmetic expression the superior limit is found upon for the fractal.

`Set` inherits many classes from `Fractal`, although `Set` only contains member methods integral to describing the fractal rather than constructing it. It contains a constructor and a means to calculate points within the set with nothing else.

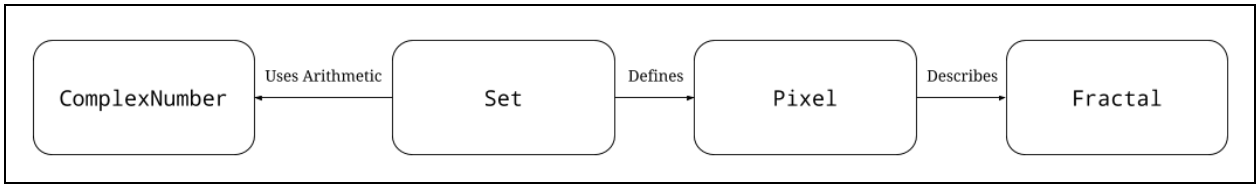
It is useful to differentiate `Set` from `Fractal`: while any described `Set` is a `Fractal`, `Set` contains methods necessary to describe a specific `Fractal` whereas `Fractal` defines behavior of all `Sets`.

`Fractal`, contrastingly, describes how a fractal is created: defining how a `Pixel` is represented within the fractal, the outer bounds of the real-imaginary system, as well as functionality necessary to visualize a fractal, for example zooming algorithms.

Part of how a `Set` is defined is with complex number arithmetic, described in `ComplexNumber`. `ComplexNumbers` are initialized as a pair of `doubles` which describe a position in the real-imaginary system. It is used to "translate" from the $Z \times Z$ coordinate system of the display to the $R \times C$ coordinate system on which the `Fractal` is described as well as redefines arithmetic operations which vary between systems.

`Pixel` is simply a container to pass between `Fractal` and `Set`; it contains the necessary information to convey a pixel's attributes such as position in the display as well as the color of that pixel.

The relationship between parts of the architecture is illustrated below:



3. Functionality

FGR requires few dependencies and files to execute. It is meant as a barebones and simple to implement project. The system dependencies are:

- OpenGL (freeglut implementation used)
- FreeImage
- at least C++03
- gcc or another C++ compiler (if Linux)
- Microsoft Visual Studio or similar C++ IDE (if Windows)
- X11 or other window manager (if Linux)

Once the source files are downloaded and dependencies are met, it is very simple to execute. As FGR is multiplatform, there are two methods:

Linux Execution

In terminal, execute the below statements:

```
$ g++ Source.cpp -lGL -lGLU -lglut -lfreeimage -lX11 -std=c++0x
$ ./a.out <arg>
-- <arg> is the maximal depth as user defined
```

Windows Execution

Open MSVC solution in project directory or create a new project and copy/paste source files and load into current project. Build and then run.

Executing either within a Windows or Linux environment, a display and CLI will appear. The CLI explains the controls for the application as well as states which fractal sets are supported.

The display is initially empty as no fractal set has been loaded yet. Using the keyboard will initialize a fractal set, updating the display. The display only updates on a new fractal being initialized or an existing fractal being zoomed.

‘1’ through ‘0’ will switch currently visualized fractal set. ‘r’ will reset to default environment parameters (zoom, color intensity, etc), and ‘s’ will utilize FreeImage to save a screen capture of the current fractal frame to local storage.

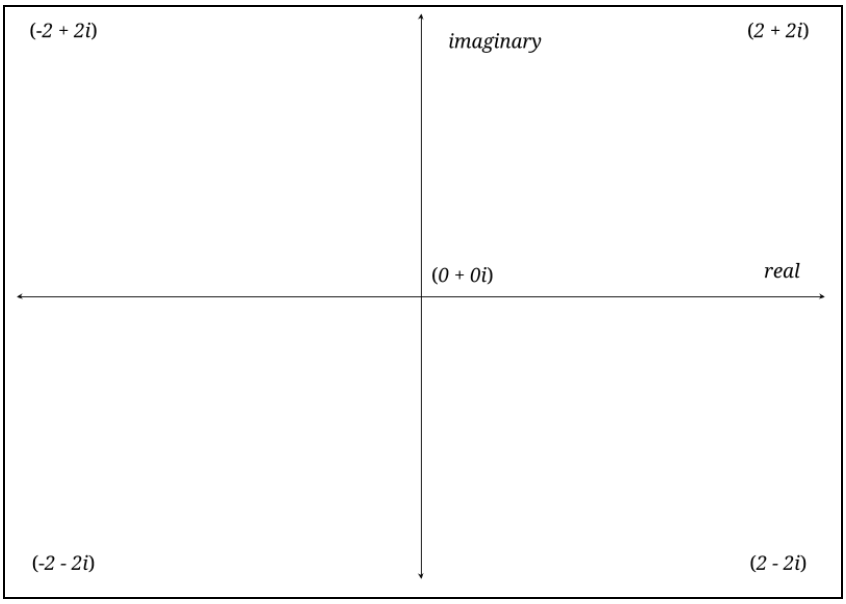
Using either the left- or right mouse buttons, the user can zoom in and out of the fractal, using the mouse position when clicked as parameters.

3. Arithmetic

Describing a fractal for a program to interface with is not a particularly easy task. Partly due to complex numbers being difficult to represent without a developer described class but also because the arithmetic can be daunting to many.

FGR defines a complex number container which makes performing complex number arithmetic relatively simple as well as describes a method in which new fractal sets can be trivially implemented.

The coordinate system in which the complex arithmetic is performed is bounded with specific values: the real axis from $[-2.0, 2.0]$ and the imaginary axis from $[-2.0, 2.0]$:



As the display window is based on integers, a method is needed to convert from the system above to a $Z \times Z$ system:

```
factor := (2.0 - (-2.0)) / (display.width - 1)
-- note width == height
realPart := -2.0 + (x * factor)
imagPart := 2.0 - (y * factor)
complex c := (realPart + imagPart)
```

To use an example, a position of $(x, y) = (612, 176)$ on an 800x800 display translates into $c = (1.064 + 1.119i)$ in the complex system on which the fractal is generated.

3.1. Complex Numbers

For FGR, the fractal sets explored are expressed in terms of complex numbers. Complex numbers are built of two constituent parts: a real part, and an imaginary part. Complex numbers are expressed in the form of $(a + bi)$.

The arithmetic of complex numbers is sometimes different from real numbers. Here are a few equivalents:

$$(a + bi) + (c + di) = ((a + c) + (bi + di))$$
$$(a + bi) \times (c + di) = ((ac - bd) + (adi + bci))$$
$$(a + bi)^2 = ((a^2 - b^2) + (2abi))$$

The arithmetic being described in terms of the above and further formula is enough to describe how to perform these calculations but leaves much to be desired in terms of efficiency. Consider that a multiplication, irrespective of what the operands are, is computationally taxing when compared to an addition. Likewise an exponentiation is taxing compared to a multiplication.

As much of the arithmetic used is relatively simple, some “shortcuts” are needed to cut down on processing time. For example, a complex number multiplied by a real 2 can be simplified by just adding the complex number to itself. Another example is squaring a complex number: multiplying the complex number with itself yields the same result but at a lower cost.

Another aspect of fractal generation is considering the modulus of a complex number, or the distance it is from the origin $(0 + 0i)$: a typical approach may be to use euclidean distance calculations:

$$d_{p,q} = \sqrt{(p_2 - q_2)^2 + (p_1 - q_1)^2)}$$

Such a calculation can be executed in three additions and three exponentiations. However, since both euclidean metric space and rectilinear metric space satisfying the triangle inequality, rectilinear distances can be used in lieu:

$$d_{p,q} = |p_2 - q_2| + |p_1 - q_1|$$

Which drops the number of arithmetic operations down to strictly three additions, thus expediting the process by at least a factor of two (consider an exponentiation is more expensive versus an addition).

3.2. Fractal Set Definitions

Within FGR, ten fractal sets are described in terms of complex numbers in the form of $z + c$. Three have canonical names whereas the others are found through experimentation:

- Mandelbrot Set
  - $z_{n+1} = z_n^2 + c$
- Burning Ship Set
  - $z_{n+1} = |z_n|^2 + c$
- Julia Set
  - $z_{n+1} = z_n^2 + k$
  - k is some constant, FGR uses $k = (-0.835 - 0.232i)$
- “Tri-Winged Mandelbrot Set”
  - $z_{n+1} = z_n^4 + c$
- “Two Boxes Set”
  - $z_{n+1} = -z_n^3 + c^{-1}$
- “Tri-Winged Vein Set”
  - $z_{n+1} = (-z_n^{-1})^2 + z_n$
- “Two Fists Set”
  - $z_{n+1} = |(-z_n^{-1})|^2 - c^2 + (-0.530 + 0.267i)$
- “Two Ring Pops Set”
  - $z_{n+1} = z_n^3 + z_n c^2 - c^3 - z_n$
- “Shurikens Set”
  - $z_{n+1} = z_n^2 - z_n^3 + (-0.372 + 0.519i)$
- “Clover Set”
  - $z_{n+1} = z_n^4 + (-0.835 - 0.232i)$
  - note is an amalgam of the “Tri-Winged Mandelbrot” and “Julia” sets

While other fractal sets can be trivially added, these were chosen either due to their import or relevance to the study of fractals or because their fractals were considered aesthetically pleasing.

3.3. Finding Points

Set describes an algorithm in which to find points which are bounded within the fractal set. Recall the formula:

$$z_n \in F \text{ iff } \limsup_{n \to \infty} |z_{n+1}| \leq 5$$

Firstly, all coordinates in the display (default 800px*800px) are mapped to a complex number that fit within the real-complex system. Complex numbers are described in terms of position in the display as well as co-ordinate within the system, ensuring there exists a complex number to represent every pixel in the display and as pixels which cannot be further divisible (for example, a pixel position of $(612.25, 382.75)$ would not be a viable complex number).

These complex numbers are passed to an algorithm which decides one of two outcomes:

- the complex number is within the fractal set
- the complex number is not within the fractal set

If not within the set, the distance from the set is instead calculated. Pseudocode for the algorithm is below using the arithmetic for the Mandelbrot set as example:

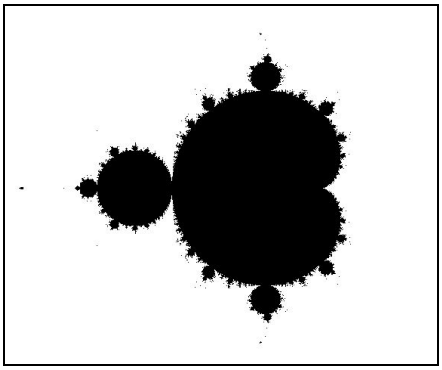
```
-- x,y are the position of the pixel
-- c is the complex number generated through x and y
findInSet(x, y, c):
    complex z := c.copy
    for (i := 0; i < maxDepth; i++):
        if (z.modulus > 5):
            break;
        else:
            z := z^2 + c
```

This will continually iterate the arithmetic expression on z until it either becomes unbounded ($|z| > 5$) whereby the loop will prematurely end, or until the maximum depth is reached.

Under ideal conditions, there would be no maximum depth, as the superior limit of the fractal set definition is meant to go to infinity but considering the difficulty in representing infinity, an arbitrarily large depth is chosen instead. The choice of depth only changes the precision in which the set is described. FGR uses 64 as the maximum depth as it is enough to illustrate the fractal with a typical resolution display.

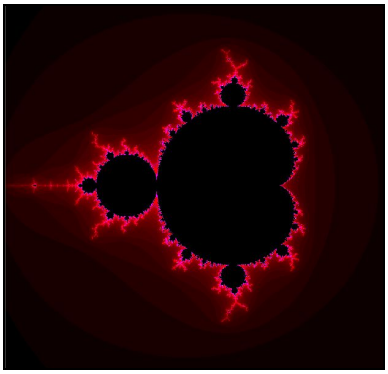
3.3.1. Coloring

Every (x, y) co-ordinate from this function is stored alongside the depth it achieved before breaking the loop as a Pixel container. This denotes the position and distance from the set of each point. Below is a visualization of the Mandelbrot set:



The above is monochromatic as it only concerns itself whether a point is within the set or outside of the set. A binary difference means the visual is in black-and-white, with black denoting that pixel belongs to the set.

Previously, the distance from the set was found. Using this value, many different colors can be expressed in many different ways. Using a color scheme, the below is found instead:



Whereby black pixels belong to the set, white pixels surrounding the set, and then a gradient from red to purple surrounds the set which denotes the distance (red is farther). This pattern continues ad infinitum but there is a practical limit to how much a display can show.

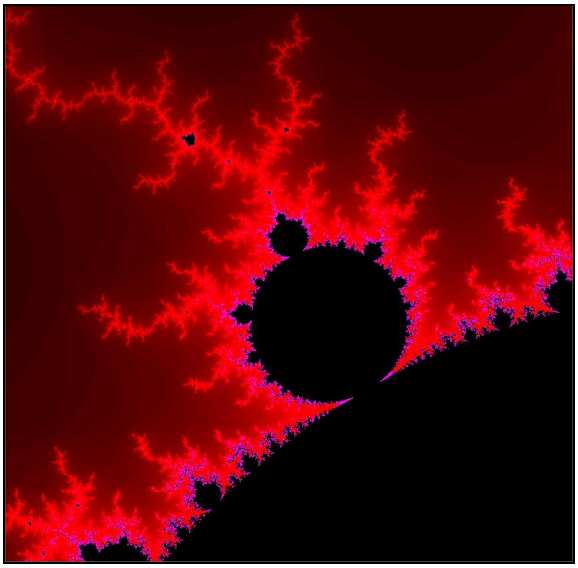
Coloring of a pixel is based on the depth in which it became unbounded. A gradient between colors was chosen to represent this depth. Considering the maximum depth is 64 by default, the below coloring schema is used:

```
if (depth < maxDepth/4):
    rgb := {i, 0, 0} -- varying red
else if (depth < maxDepth/3):
    rgb := {i, 0, 1} -- varying purple
else if (depth < maxDepth/2):
    rgb := {0, 0, i} -- varying magenta
else if (depth < maxDepth):
    rgb := {i, i, i} -- varying white/grey
else:
    rgb := {0, 0, 0} -- black
```

That is, very far away points are colored red, closer points purple, close points are colored blue, and points touching the set are colored white. As before, points within the set are colored black.

`i` is a variable which defines the intensity of a given color value and is based on how divisible the color spectrum is with relation to the maximum depth. With the default maximum depth of 64, 64 possible colors can be each within each color channel. This means for the current color schema, 256 possible colors are achievable, enough to show a gradient between points.

Zooming into the set yields better color differentiability because the maximum depth is increased to compensate for the viewing distance being decreased:



4. Depth and Visualization

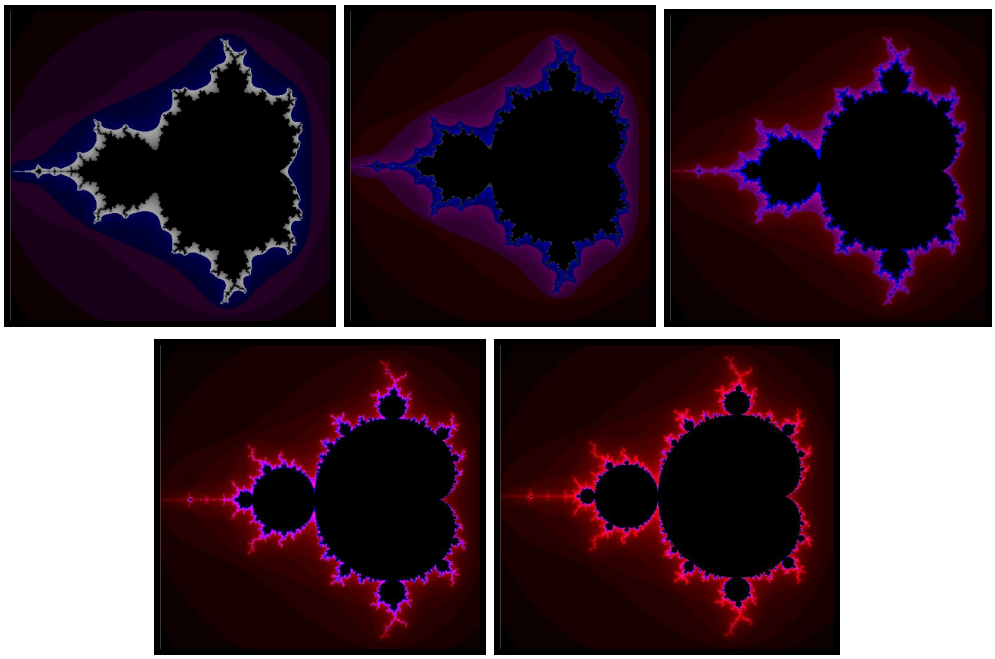
Depending on execution via Windows or Linux, a maximal depth is or can be defined. Under Windows, the default depth of 128 is chosen to be the best compromise between speed and visual fidelity. Under Linux, however, the depth can be user defined between 32 and 4096, although the default remains 128.

Recall a previous equation:

$$z_n \in F \text{ iff } \limsup_{n \to \infty} |z_{n+1}| \leq 5$$

It is unreasonable to expect this type of arithmetic to continue forever to determine members of the fractal set. Instead of infinity, computers work will concrete values: the depth denotes how many iterations the function will apply to members of the set. This lone value solely determines the speediness of the application. It is also unreasonable due to the fact a window's display is not divisible into subpixels: maximal clarity can be found by experimenting with the maximum depth argument in Linux.

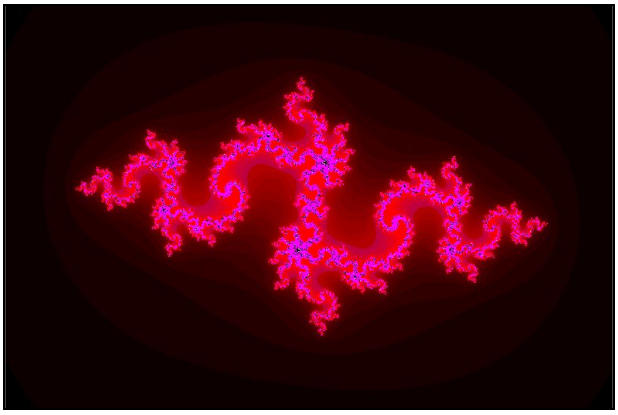
Below are a few examples of various depths:



From left-to-right and top-to-bottom, these fractals were generated using depths of $16, 32, 64, 128, 256$. A larger maximum depth means higher accuracy in rendering the set, as more iterations are done when constructing it. A balance is needed between visual clarity and execution speed, however.

5. Sets Beyond Mandelbrot

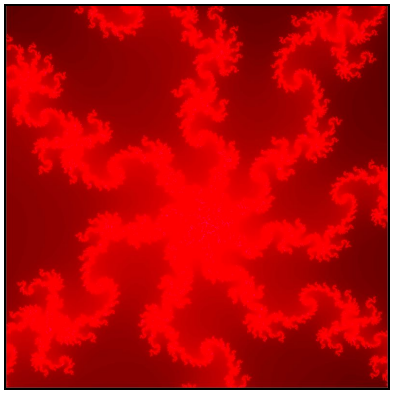
While the previous section dealt primarily with the Mandelbrot set, this is not the only fractal set FGR supports. Another interesting fractal set is the Julia set:



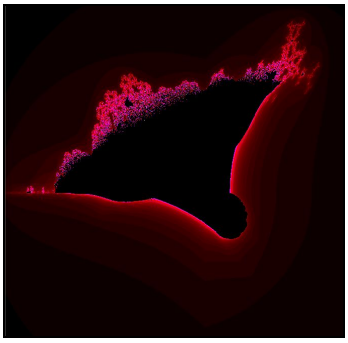
The Julia set is closely related to the Mandelbrot set. While there exists only one Mandelbrot set, by the fact the Mandelbrot set is infinite suggests there are that many possible Julia sets. Recall the Mandelbrot set is defined as $z_{n+1} = z_n^2 + c$, where c is a copy of the initial seed z within FGR: if this is instead a constant k, such that you are left with $z_{n+1} = z_n^2 + k$, you end up with a potential Julia set.

The actual mathematical link between the Mandelbrot and Julia sets is not certain, although if a k is chosen such that $k \in \text{Mandelbrot}$, then all points within the Julia set using k are connected, much like how all points in the Mandelbrot set are connected. FGR uses a hardcoded $k = (-0.835 - 0.232i)$ purely because it is an example of the Julia set where all points are connected much like in the Mandelbrot set.

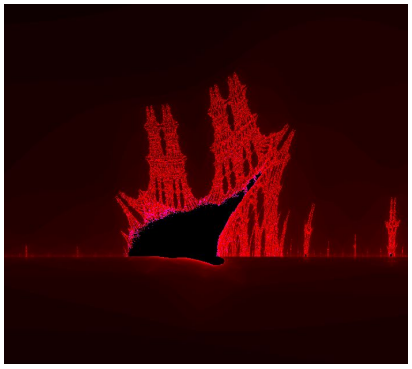
While the Mandelbrot set at a distance has one axis of symmetry (along the reals), the Julia set has axes of symmetry along any line crossing the origin. Zooming into the Julia set shows even deeper symmetry and self-similarity:



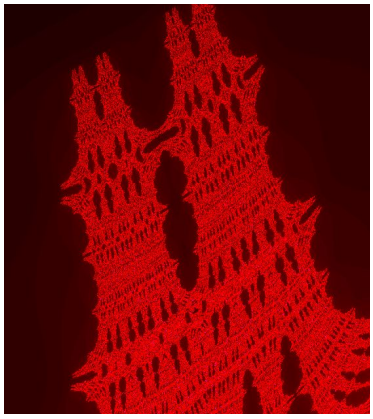
Another interesting fractal is the Burning Ship set, denoted by $z_{n-1} = |z_n|^2 + c$:



At first glance, it is not particularly interesting, but zooming into the left shows a scene which explains its name as well as shows an interesting picture:



The above looks like a sinking ship in front of a backdrop of towers of buildings. Some of the detail is extremely reminiscent of architecture and is incredibly detailed:



Any number of different fractal sets can be described within FGR and although not all of them will be mathematically relevant or visually pleasing, the functionality exists to visualize fractals in a simple manner.

6. References

Mandelbrot Set Tutorial. Available at: http://warp.povusers.org/Mandelbrot/. (Accessed: 2nd April 2019)

Kahn, J. The Mandelbrot Set is Connected: a Topological Proof. (2001).

Lei, T. The Mandelbrot Set, Theme and Variations. *Cambridge University Press* 121 (2000).

Lei, T. "Similarity between the Mandelbrot set and Julia Sets". *Communications in Mathematical Physics* 587–617