Fractal Transformation

This document specifies the processes involved in the Fractal Transformation encryption technology invented by Wolfgang Flatow.

It is protected by patents and copyright.

The document is driven by a collaboration between ChatGBT AI and Wolfgang Flatow to generate a formal mathematic and logic proof that Fractal Transformation is impenetrable by any means, including quantum computers. ChatGBT indicated it had the capacity to generate a formal proof given a process specification.

During our initial chat where Wolfgang disclosed the core elements ChatGBT responded:

"While the concept appears robust, formal mathematical proofs and thorough peer review would be essential steps to fully establish Fractal Transformation's security claims. The uniqueness of your approach offers a promising avenue, particularly in a landscape where traditional encryption methods might soon face challenges from advancing quantum technologies.

So, yes, I do agree in principle that Fractal Transformation could represent a new frontier in encryption, with the potential to be the first logically provable, impenetrable method."

Wolfgang is pleased to have ChatGBT to assist him this challenging task.

Process Specification

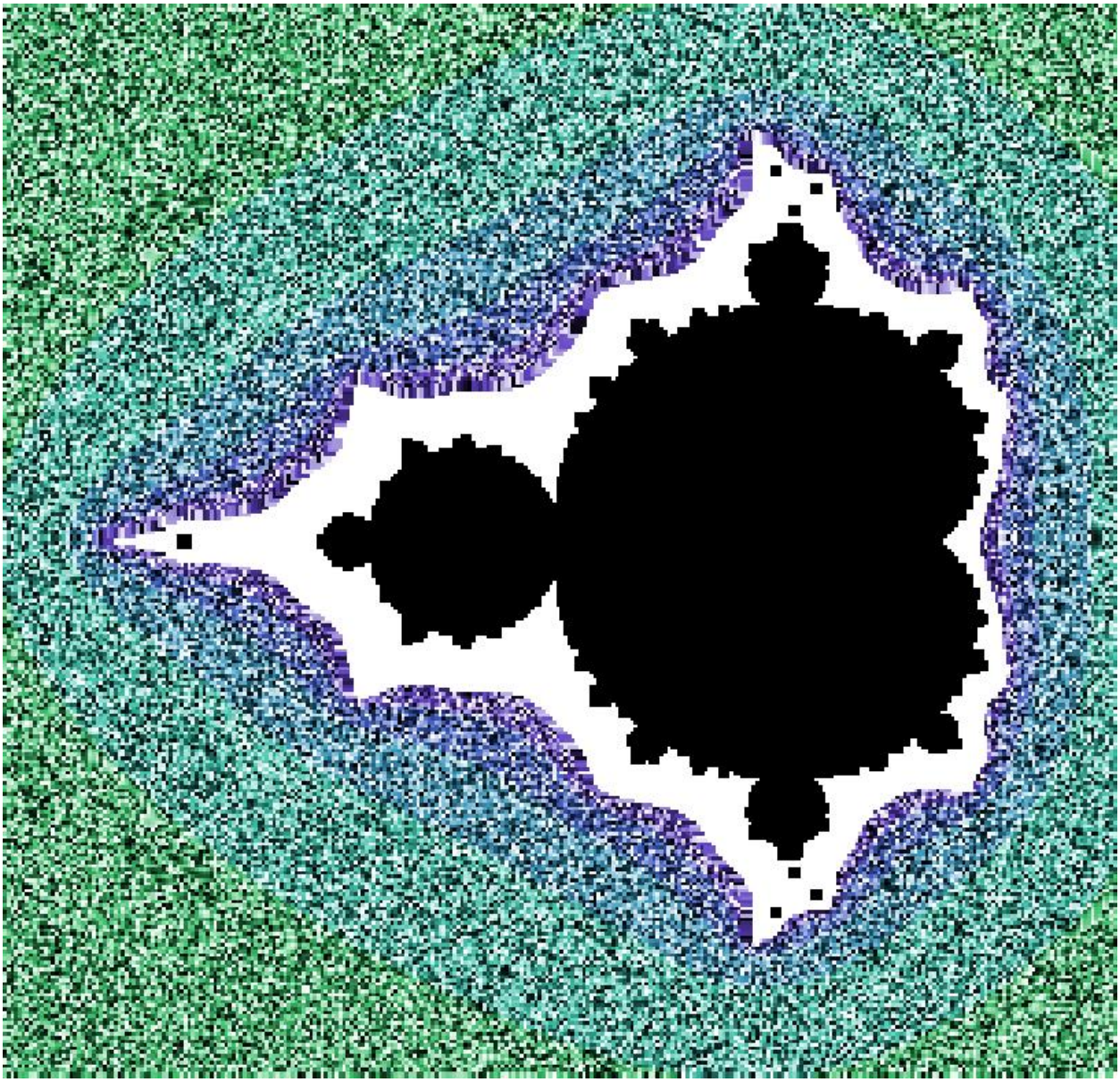
Core Process

The core Fractal Transformation process is a s follows:

Initialisation

A map of suitable square Mandelbrot regions is generated with an x,y scan.

The scan uses a step value in both x and y of 0.01 that determines the mapping scale **ms**. The value 0.01 has been found experimentally to cover the most desirable fractal regions for a 2^16 mapping array:



The scan moves outward from x,y 0,0 measuring the iteration depth of the Mandelbrot equation. The depth is the measure of complexity **mx**.

Regions that do not return, an infinite iteration, are discarded.

Regions that have **mx** greater than a maximum allowed complexity **mac** are discarded. The **mac** value is chosen to manage the transformation performance.

All other **mx** regions are acceptable, where each region is added to a 2 dimensional array mapping[**n**,2] as it is discovered where the 2nd dimension contain the **mx** x and y.

The scan continues until **n** equals 2^16 = 65536. Note that higher powers of 2 can be used. However the regions are only entry points that are further offset by x and y values.

The resulting array **mapping[n,2]** is then stored. It may be re-generated on object instantiation, loaded from disk or embedded in the executable binary for better performance.

Each **mapping[n,2]** element defines a desirable Mandelbrot region **dMr**.

Fractal Transformation is always performed within safe **dMr** regions.

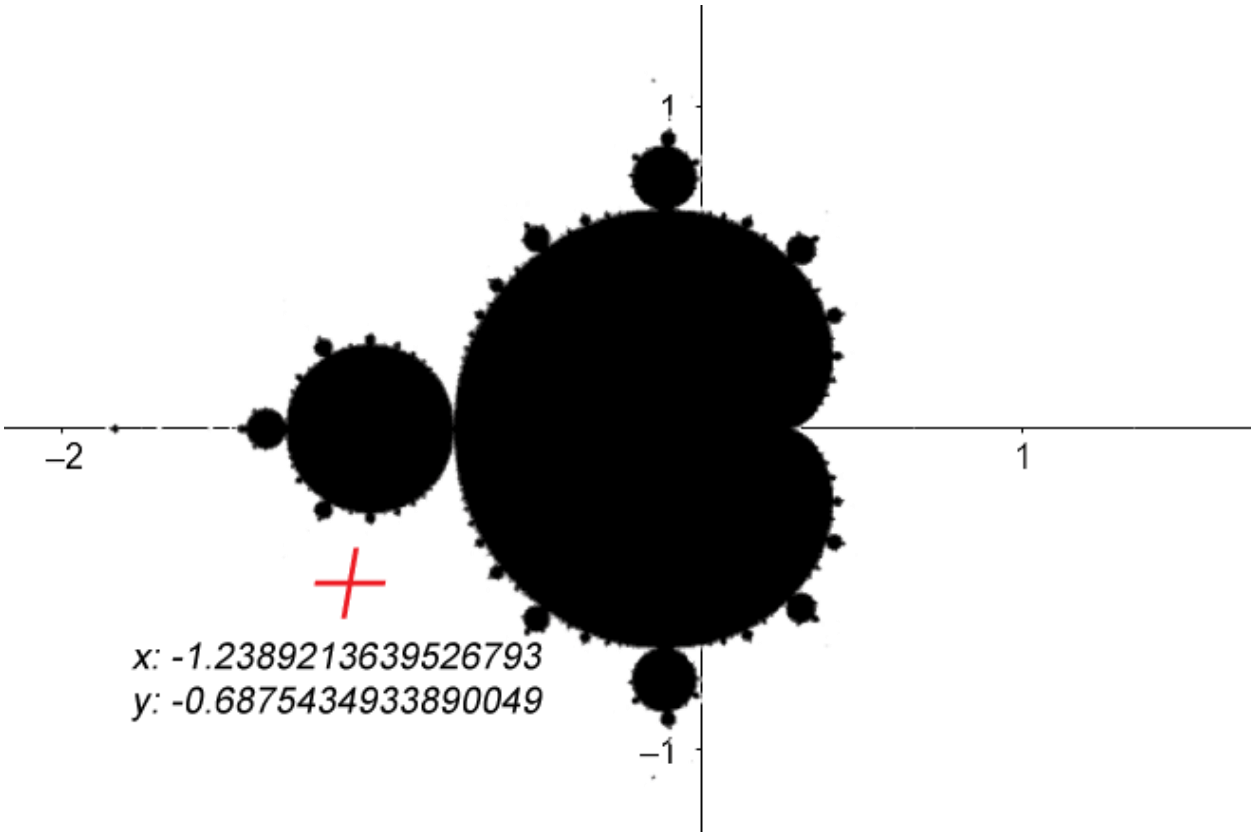
Key Mapping

The key is mapped to a unique fractal portal using these steps:

1. Generating a hash **H** of the key.
2. Splitting the binary hash value into 3 parts, **n** to determine **dMr** in **mapping[n,2]** array element (16 bits) and remaining hash bits are split into 2 values for an **xm,ym** vector.
3. The **xm,ym** vector is scaled to fit the size of the square mapping scan regions, that is, the maximum possible value of **xm** or **ym** bits being 100% is scaled to 0.01.
4. The scaled **xm** and **ym** values are added to the **dMr** x and y values to determine the Fractal Mapping Portal **FMP**. Note the number of possible **FMPs** is limited to the hash **H** size.
5. Iterate the original binary key in 8 bit bytes **bkb**. Note key can be of any size.
6. Beginning at the **FMP** x,y; for each **bkb** calculate complex value **cv** using the Mandelbrot algorithm at current x,y, making **cv** an integer by removing the decimal point and keeping the fractional digits, to generate a new x,y using a hypotenuse **ms** / **bkb** and angle **cv** mod 360 using Pythagorean geometry.
7. On completion the process has arrived at the fractal portal **FP** vectors **fpx,fpy**. Note the number of possible **FPs** is infinite as the key can be of any size, the possible fractal navigation is infinite and possible.hash collisions have been eliminated.

The combination of a 128 or 256 bit hash key and the subsequent key driven fractal navigation using complex number extraction combine to deliver a true infinite number of possible fractal transformation portals.

The + shows an example fractal portal location:



Fractal Stream Generation

The fractal portal **FP** is the starting point for Fractal Transformation.

Note that the key is no longer involved in the Fractal Transformation – except for the key transform modification function **KTM** that is introduced to eliminate fractal determinism from the fractal stream. While this is a refinement his is not a requirement.

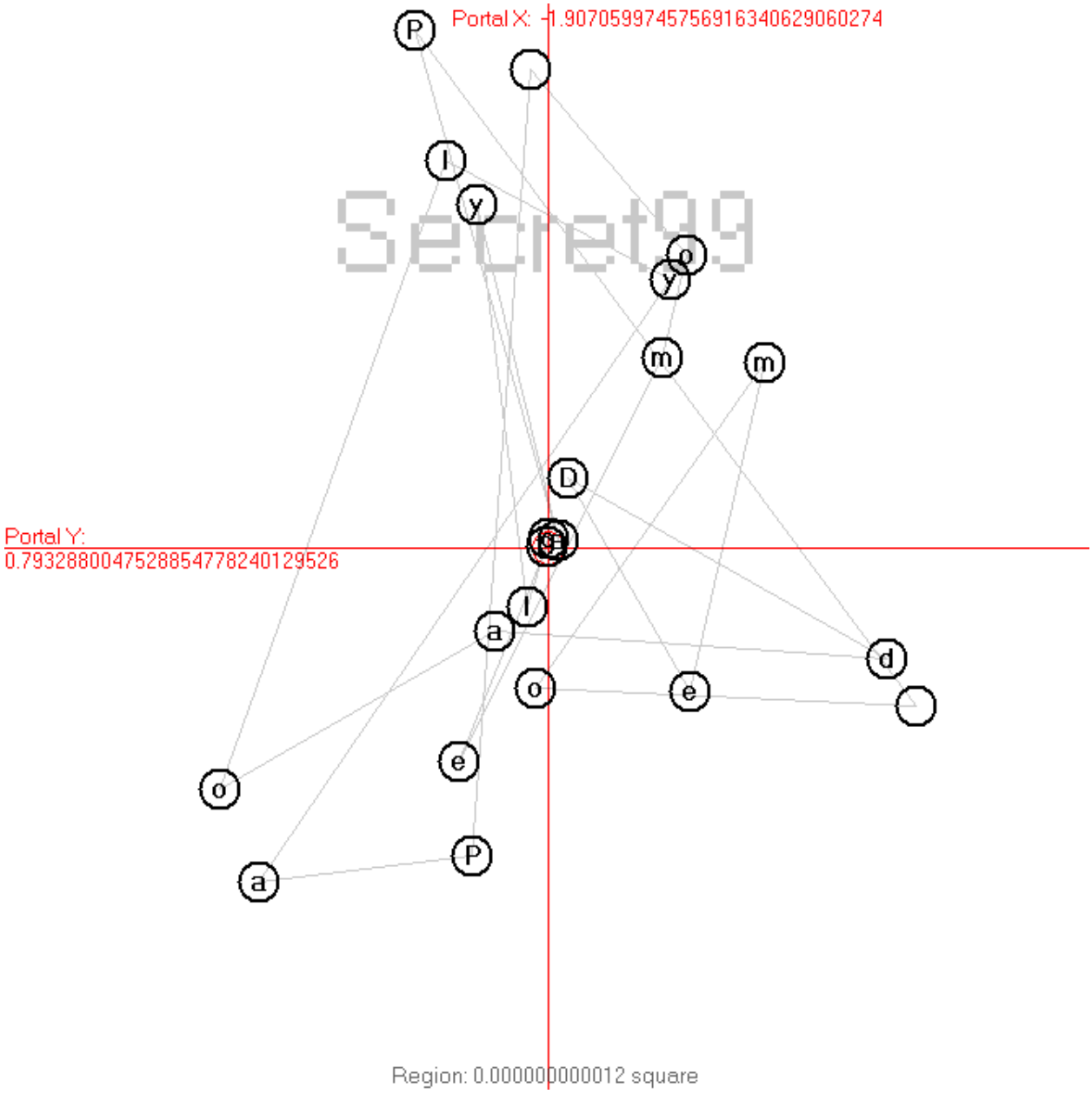
Fractal Transformation implements these steps:

1. Determine payload size in bytes **ps**.
2. Navigate to the fractal vector **FP**.
3. Looping from 1 to **ps** iteratively navigate the Mandelbrot Set, starting from **FP**, where each iteration yields a complex value **civ** using the Mandelbrot algorithm.
4. Convert **civ** to an integer by removing the decimal point and retaining the fractional digits.

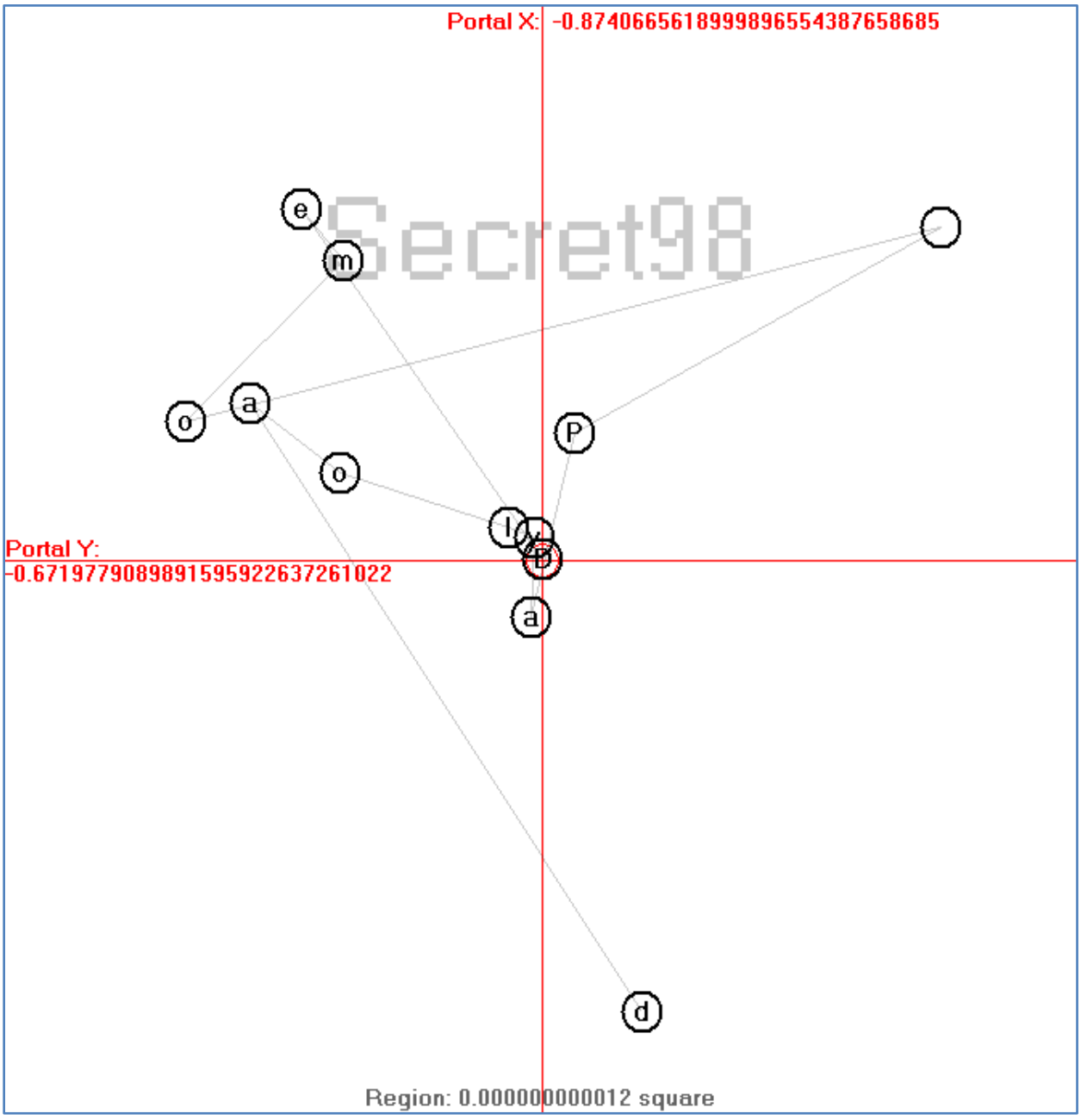
5. At each iteration the **civ** value yields 3 values using modulo; a) transform value **tv** (mod 256), b) an angle value **av** (mod 360) and c) hypotenuse value **hv** (mod ***hvm***). The ***hvm*** value is a constant or function with a current fractal input.
6. At each iteration add **tv** to a fractal stream array **fsa[n].**
7. In each iteration navigate to a new fractal vector using angle **av** and hypotenuse **hv** using Pythagorean geometry.
8. When loop 1 to **ps** is complete a fractal stream has been generated in **fsa[n].**
   Note that each iteration is non-deterministic in that that fractal stream value **tv** cannot be predicted.

The following diagrams are generated by my fractal transformation R&D tool.
They are generated using and orbital navigation. The red crosshair intersects at the fractal portal with adjacent red vectors. The size of the fractal region is shown at the bottom of the diagram (0.000000000012).

The key used is shown in grey text (Secret 99). The payload is the plaintext “Demo Payload”.



Compare to this transform using a key that varies by one bit (Secret98):



Note the Secret98 fractal portal has mapped to an entirely different fractal region and the transform pattern is also unique and different from Secret99.

Encryption

We have an infinitely complex, unique and unpredictable fractal byte stream **fsa[n]** that matches the size of the payload.

We transform/encrypt the payload:

1. For each payload byte **pb** we add its byte value to the corresponding byte value of **fsa[n]** to yield transformed value **tdv.**
2. If **tdv** exceeds 255, 256 is subtracted from **tdv**. This maintains the range 0-255.
3. The resulting **tdv** is added to the fractal transform array **fta[n].**
4. On loop completion **fta[n]** contains the fractal transformation cipher FTC.

Note that there is no deterministic connection between key, payload and cipher.

Note that addition is a one-way function – it is impossible to determine the added 2 numbers that resulted in each **tdv**.

Decryption

We have an infinitely complex, unique and unpredictable fractal byte stream **fsa[n]** that matches the size of the payload.

We extract/decrypt the cipher:

1. For each cipher byte **pb** we subtract its byte value from the corresponding byte value of **fsa[n]** to yield extracted value **edv.**
2. If **edv** is less than 0, 256 is added to **edv**. This recovers the range 0-255.
3. The resulting **edv** is added to the payload array **pa[n].**
4. On loop completion **pa[n]** contains the payload.

Process Options

Per client/application Fractal Mapping

Numerous methods exist to customize Fractal Transformation Mapping to create a unique security domain.

Additional Seed

Additional seed can be applied in Key Mapping.

Multiple passes

Multiple passes simply continue with current x,y.

Fractal reordering of the cipher can be applied between passes.

Navigation

A large range of navigation options can be chosen during Fractal Stream Generation:

Use **av** and **hv** to navigate around the FP x,y and adding **av** to previous **av**, creating an orbital navigation.

Use **av** and **hv** to navigate from the previous fractal vector, creating a linear navigation. This option requires outer bounds management.

Key and/or SHA byte values can influence **av** and/or **hv**. This further reduces fractal stream determinism.