Fractal Dimension of Julia Sets

Claude Heiland-Allen
claude@mathr.co.uk

March 6, 2015

Fractal Dimension of Julia Sets

Fractal Dimension of Julia Sets

Fractal Dimension of Julia Sets

Fractal Dimension

How Long is a Coast?

How Long is a Coast?

How long is a coast?

How Long is a Coast?

It looks this long.

How Long is a Coast?

But as you look closer,

How Long is a Coast?

more details appear,

How Long is a Coast?

giving a longer length,

How Long is a Coast?

so when do you stop?

How Long is a Coast?

How long is a coast?

How Long is a Coast?

It gets longer the closer you look.

How Long is a Coast?

The concept of "length" is usually meaningless for geographical curves. They can be considered superpositions of features of widely scattered characteristic sizes; ***as even finer features are taken into account, the total measured length increases****, and there is usually no clear-cut gap or crossover, between the realm of geography and details with which geography need not be concerned.*

– B. B. Mandelbrot
"How long is the coast of Britain?"
Science: 156, 1967, 636-638

How Long is a Coast?

A better question:

How Long is a Coast?

A better question:
How much longer does a coast get the closer you look?

Box-Counting Dimension

The idea:

Box-Counting Dimension

The idea:

- Pick a box size r.

Box-Counting Dimension

The idea:

- Pick a box size r.
- Cover the boundary with boxes of size r.

Box-Counting Dimension

The idea:

- Pick a box size r.
- Cover the boundary with boxes of size r.
- Count how many boxes are needed N_r.

Box-Counting Dimension

The idea:

- Pick a box size r.
- Cover the boundary with boxes of size r.
- Count how many boxes are needed N_r.
- See how quickly N_r increases as r gets smaller.

Box-Counting Dimension

The formal definition:

$$\dim = \lim_{r \to 0} -\frac{\log N_r}{\log r}$$

Box-Counting Dimension

The formal definition:

$$\dim = \lim_{r \to 0} - \frac{\log N_r}{\log r}$$

Converges very slowly, not practical.

Box-Counting Dimension

A more practical definition:

$$\dim = \lim_{r \to 0} \log_2 \frac{N_r}{N_{2r}}$$

Box-Counting Dimension

A more practical definition:

$$\dim = \lim_{r \to 0} \log_2 \frac{N_r}{N_{2r}}$$

But finite computers have issues with infinite limits.

Box-Counting Dimension

The formula I actually use:

$$\dim = \frac{1}{2} \log_2 \frac{N_{2r_0}}{N_{8r_0}}$$

$$r_0 = \text{pixel size}$$

Box-Counting Dimension

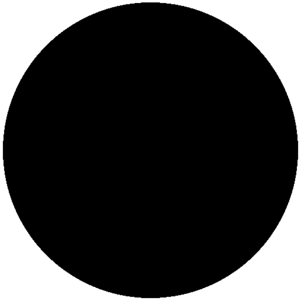
The formula I actually use:

$$\dim = \frac{1}{2} \log_2 \frac{N_{2r_0}}{N_{8r_0}}$$
$$r_0 = \text{pixel size}$$

More on the trade-offs involved later...

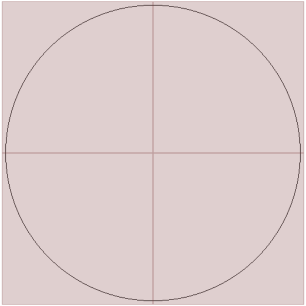
Examples

Example: circle



circle

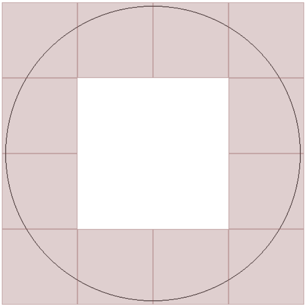
Example: circle



$r = 2^{-1}$

$N = 4$

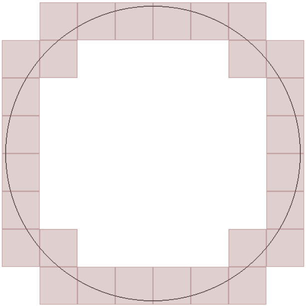
Example: circle



$r = 2^{-2}$

$N = 12$

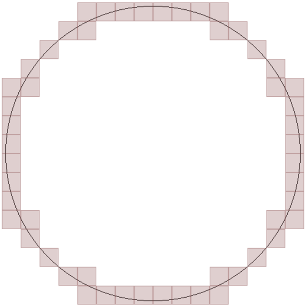
Example: circle



$r = 2^{-3}$

$N = 28$

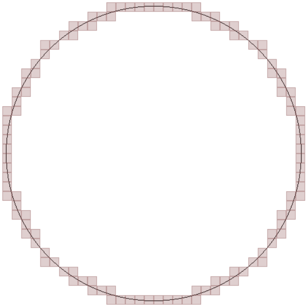
Example: circle



$r = 2^{-4}$

$N = 52$

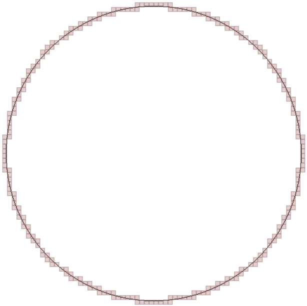
Example: circle



$r = 2^{-5}$

$N = 116$

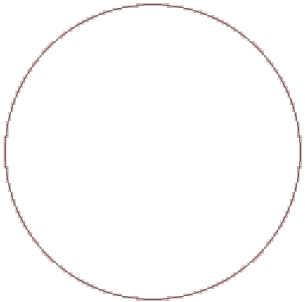
Example: circle



$r = 2^{-6}$

$N = 244$

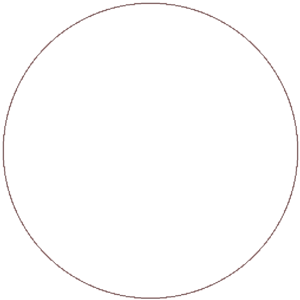
Example: circle



$r = 2^{-7}$

$N = 444$

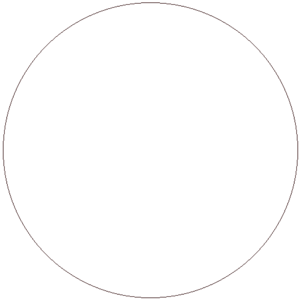
Example: circle



$r = 2^{-8}$

$N = 860$

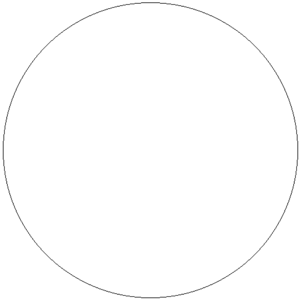
Example: circle



$r = 2^{-9}$

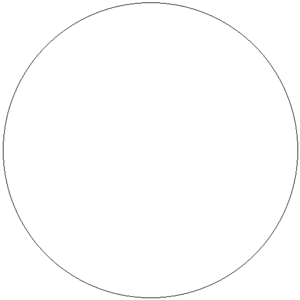
$N = 1412$

Example: circle



$$\dim \approx 0.968\ldots$$

Example: circle



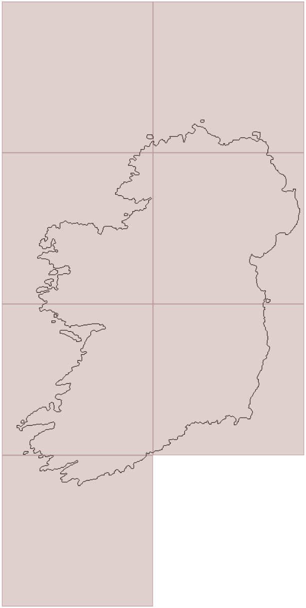
$\dim = 1$

(limit as $r \to 0$)

Example: Ireland

Ireland

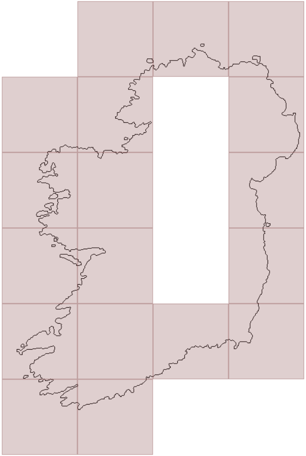
Example: Ireland



$r = 2^{-1}$

$N = 7$

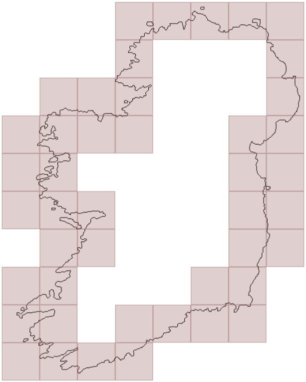
Example: Ireland



$r = 2^{-2}$

$N = 18$

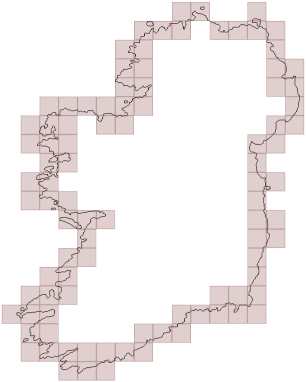
Example: Ireland



$r = 2^{-3}$

$N = 44$

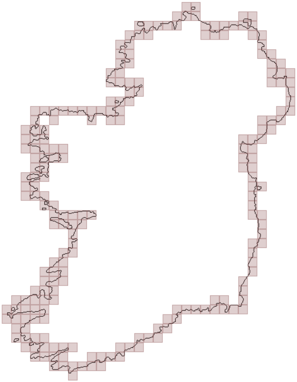
Example: Ireland



$r = 2^{-4}$

$N = 94$

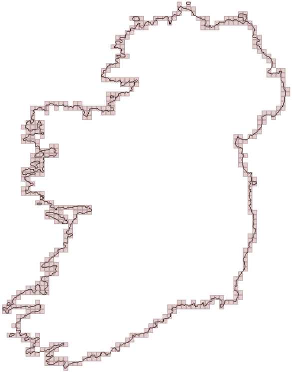
Example: Ireland



$r = 2^{-5}$

$N = 217$

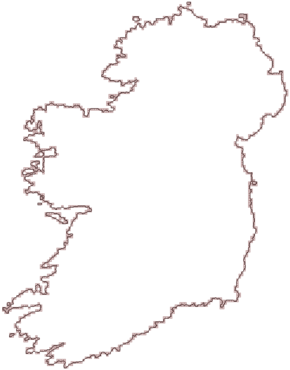
Example: Ireland



$r = 2^{-6}$

$N = 485$

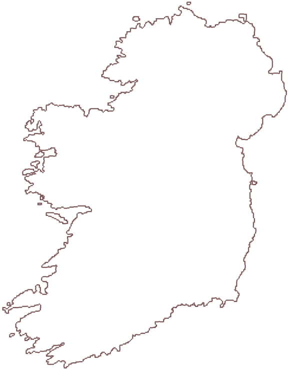
Example: Ireland



$r = 2^{-7}$

$N = 1033$

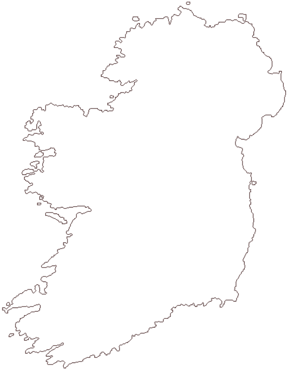
Example: Ireland



$r = 2^{-8}$

$N = 2021$

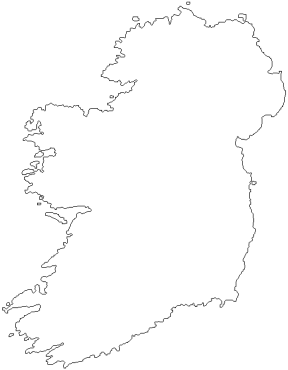
Example: Ireland



$r = 2^{-9}$

$N = 3520$

Example: Ireland

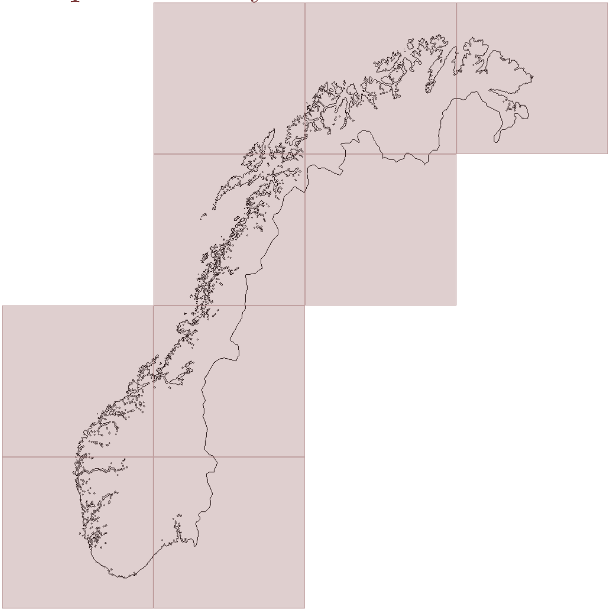


$\dim \approx 1.125\ldots$

Example: Norway

Norway

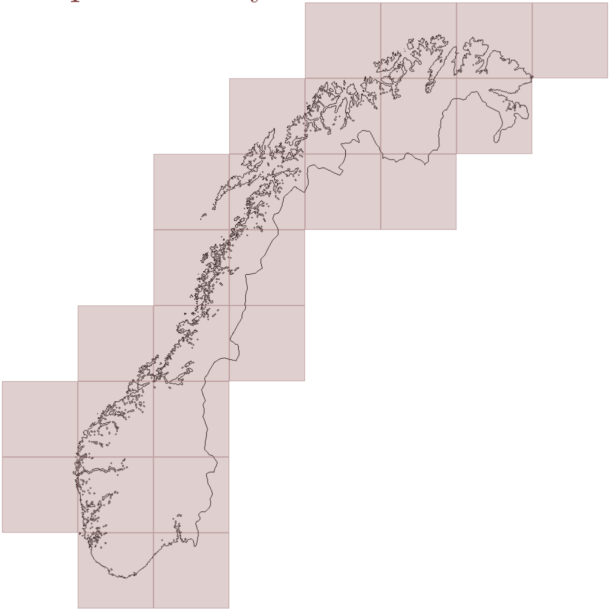
Example: Norway



$r = 2^{-1}$

$N = 9$

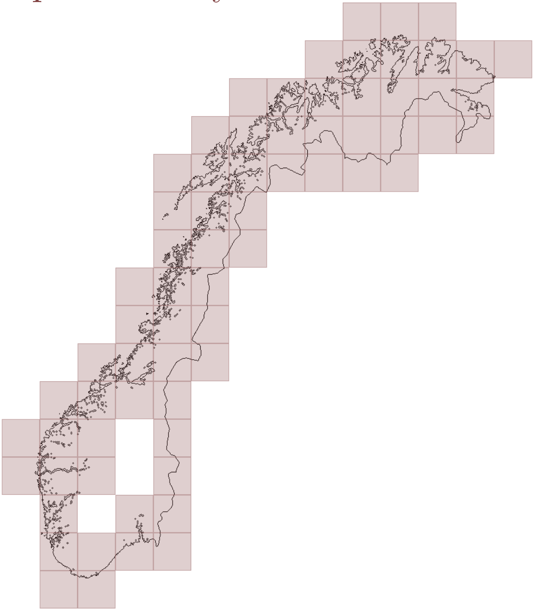
Example: Norway



$r = 2^{-2}$

$N = 25$

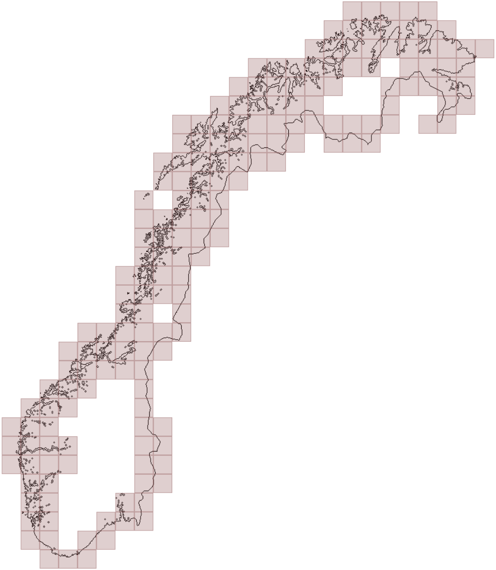
Example: Norway



$r = 2^{-3}$

$N = 68$

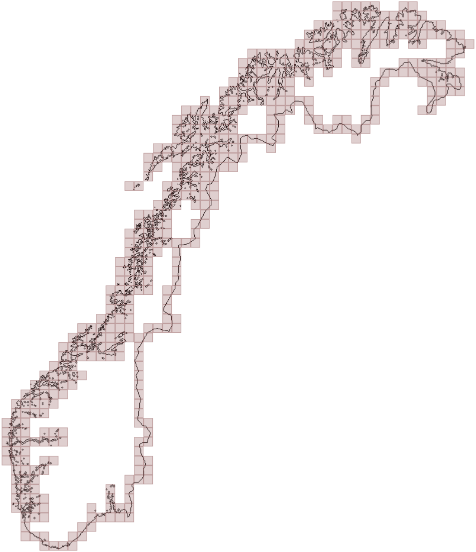
Example: Norway



$r = 2^{-4}$

$N = 180$

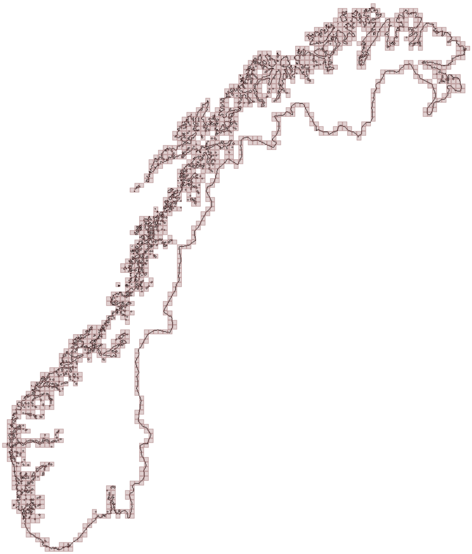
Example: Norway



$r = 2^{-5}$

$N = 488$

Example: Norway



$r = 2^{-6}$

$N = 1310$

Example: Norway

$r = 2^{-7}$

$N = 3333$

Example: Norway

$r = 2^{-8}$

$N = 7641$

Example: Norway

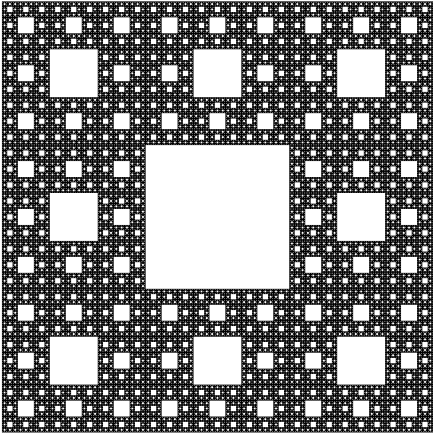
$r = 2^{-9}$

$N = 13070$

Example: Norway

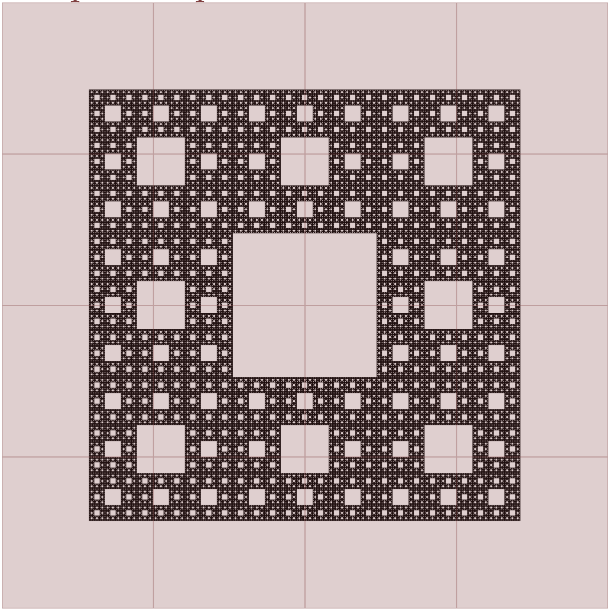
$$\dim \approx 1.385\ldots$$

Example: carpet



carpet

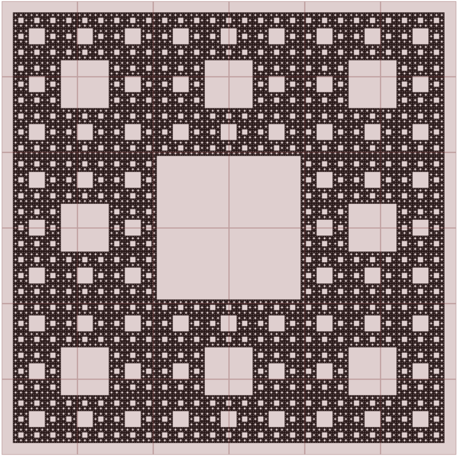
Example: carpet



$r = 2^{-1}$

$N = 16$

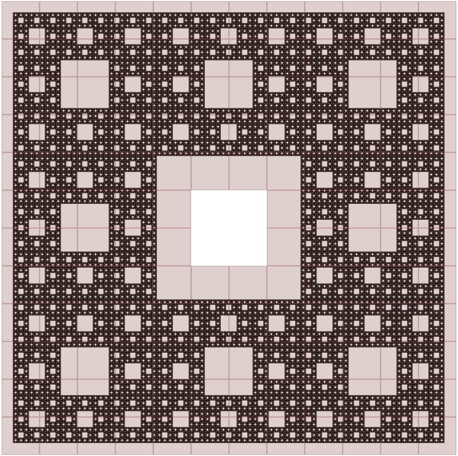
Example: carpet



$r = 2^{-2}$

$N = 36$

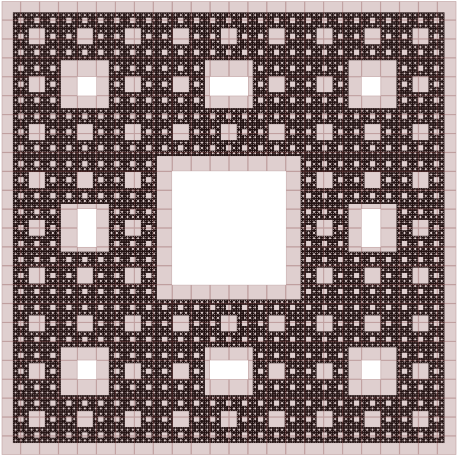
Example: carpet



$r = 2^{-3}$

$N = 140$

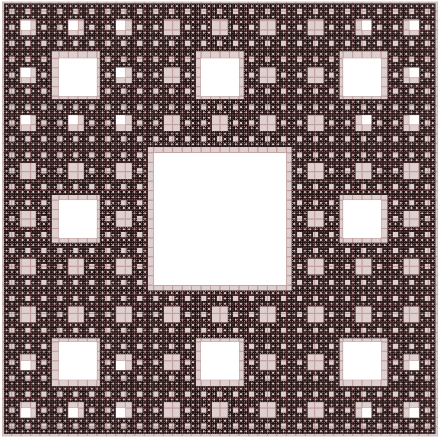
Example: carpet



$$r = 2^{-4}$$

$$N = 528$$

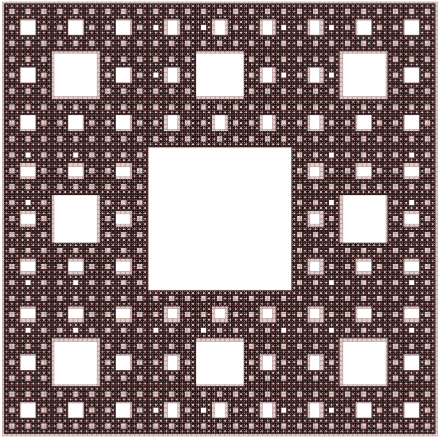
Example: carpet



$r = 2^{-5}$

$N = 1771$

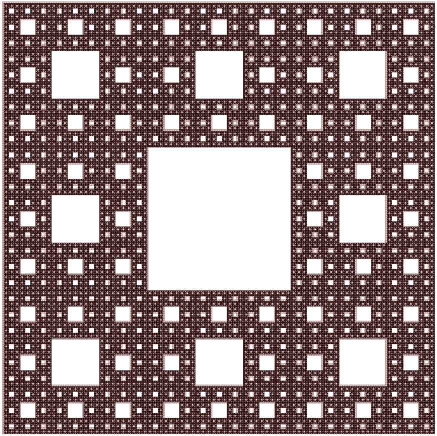
Example: carpet



$r = 2^{-6}$

$N = 6418$

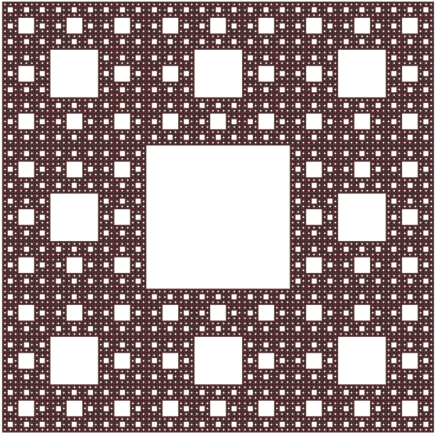
Example: carpet



$r = 2^{-7}$

$N = 23340$

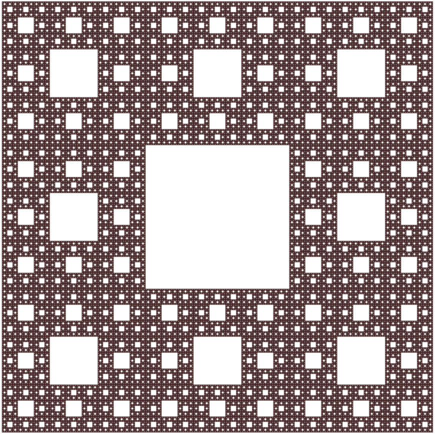
Example: carpet



$r = 2^{-8}$

$N = 82680$

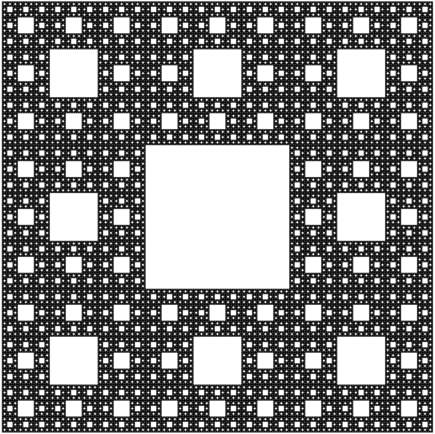
Example: carpet



$r = 2^{-9}$

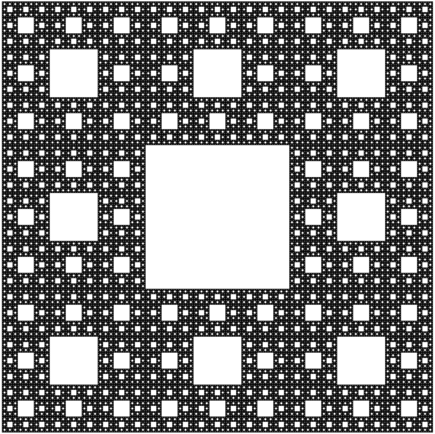
$N = 262144$

Example: carpet



$\dim \approx 1.860\ldots$

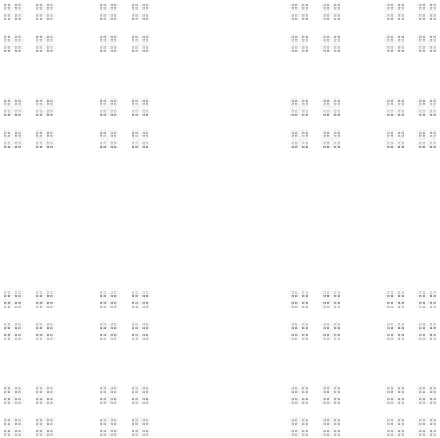
Example: carpet



$$\dim = \frac{\log 8}{\log 3}$$
$$\approx 1.893\ldots$$

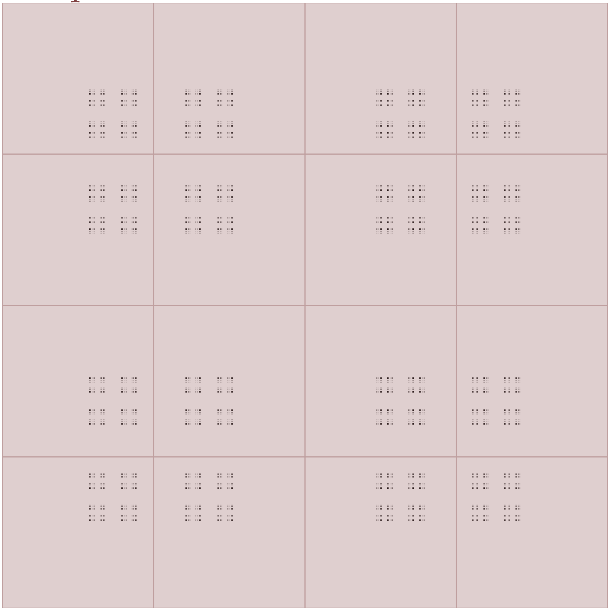
(limit as $r \to 0$)

Example: dust



dust

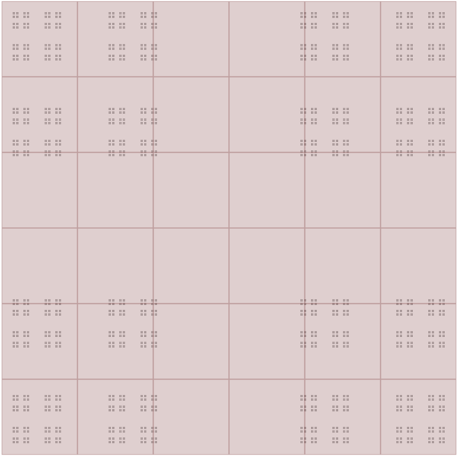
Example: dust



$r = 2^{-1}$

$N = 16$

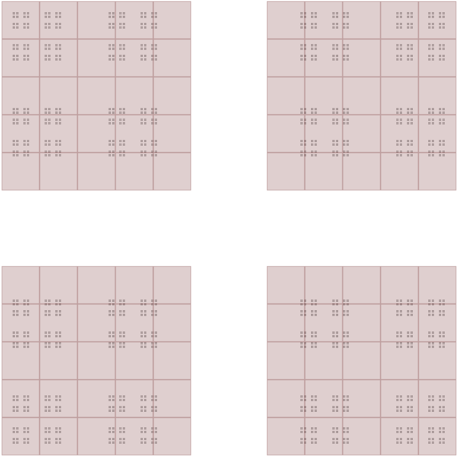
Example: dust



$r = 2^{-2}$

$N = 36$

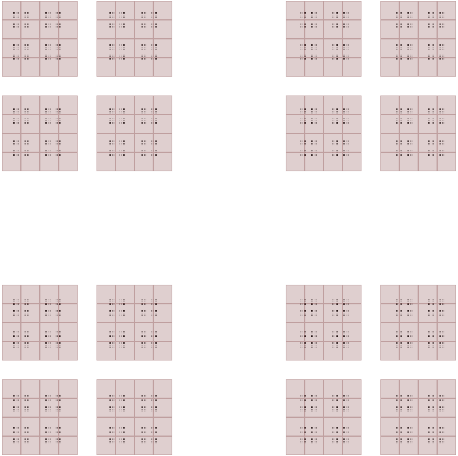
Example: dust



$$r = 2^{-3}$$

$$N = 100$$

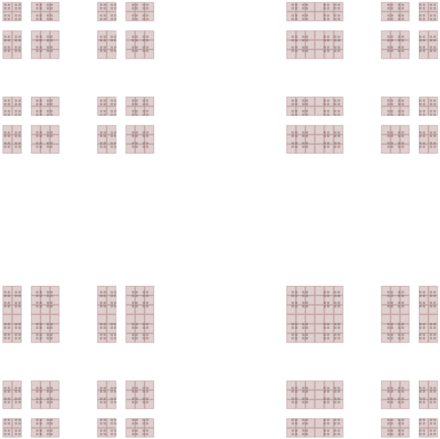
Example: dust



$$r = 2^{-4}$$

$$N = 256$$

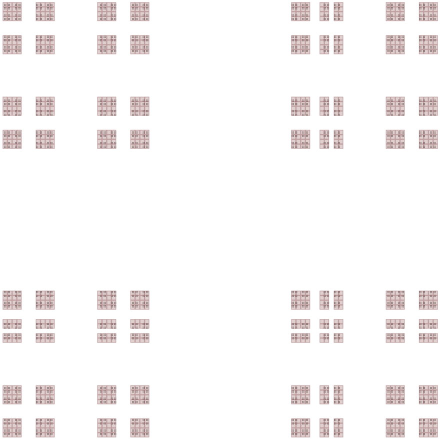
Example: dust



$$r = 2^{-5}$$

$$N = 441$$

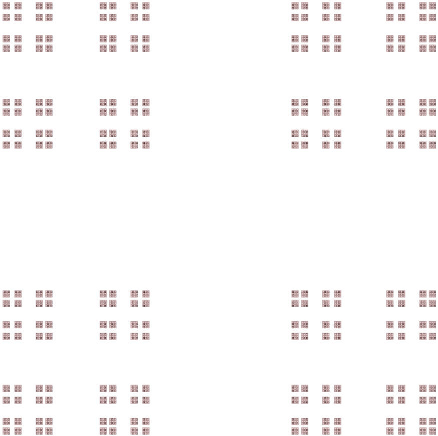
Example: dust



$r = 2^{-6}$

$N = 1024$

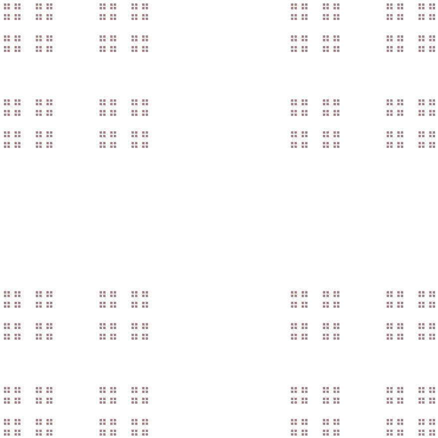
Example: dust



$r = 2^{-7}$

$N = 2304$

Example: dust



$r = 2^{-8}$

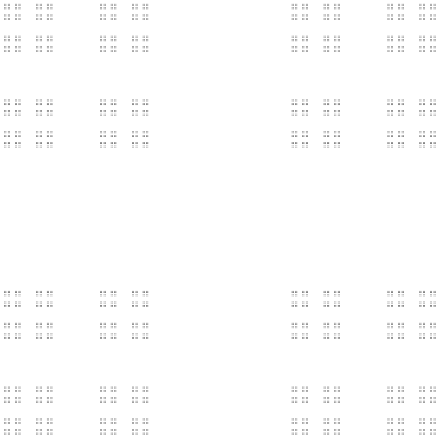
$N = 4096$

Example: dust

$r = 2^{-9}$

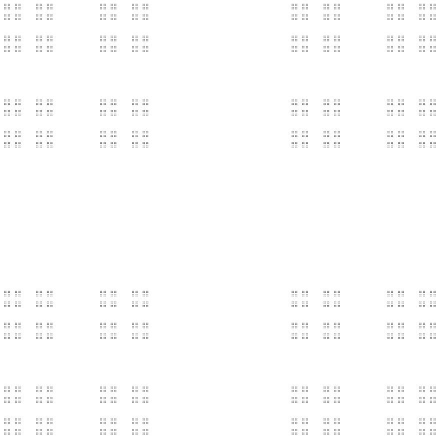
$N = 4096$

Example: dust



$\dim \approx 1.192\ldots$

Example: dust



$$\dim = \frac{\log 4}{\log 3} \approx 1.262\ldots$$

(limit as $r \to 0$)

Julia Sets

Complex Dynamics

Complex Dynamics

Consider the quadratic polynomial:

$$f_c(z) = z^2 + c$$

Complex Dynamics

Consider the quadratic polynomial:

$$f_c(z) = z^2 + c$$

Here $z, c \in \mathbb{C}$, complex numbers.

Complex Dynamics

The quadratic polynomial can be iterated:

$$f_c^n = \underbrace{f_c(f_c(\ldots(f_c(f_c}_{n \text{ times}}(z)))\ldots))$$

Complex Dynamics

The quadratic polynomial can be iterated:

$$f_c^n = \underbrace{f_c(f_c(\ldots(f_c(f_c}_{n \text{ times}}(z)))\ldots))$$

Or in more manageable notation:

$$\begin{aligned} f_c^0 \quad (z) &= \qquad z \\ f_c^{n+1}(z) &= f_c^n(f_c\,(z)) \end{aligned}$$

Complex Dynamics

What is the behaviour of $f_c^n(z)$ as $n \to \infty$?

Complex Dynamics

When $c = -2$ there are 2 distinct cases:

Complex Dynamics

When $c = -2$ there are 2 distinct cases:

- $f_c^n(z) \to \infty$ as $n \to \infty$

Complex Dynamics

When $c = -2$ there are 2 distinct cases:

- $f_c^n(z) \to \infty$ as $n \to \infty$
- none of the above

When $c = 0$ there are 3 distinct cases:

Complex Dynamics

When $c = 0$ there are 3 distinct cases:

- $f_c^n(z) \to \infty$ as $n \to \infty$

Complex Dynamics

When $c = 0$ there are 3 distinct cases:

- $f_c^n(z) \to \infty$ as $n \to \infty$
- $f_c^n(z) \to 0$ as $n \to \infty$

Complex Dynamics

When $c = 0$ there are 3 distinct cases:

- $f_c^n(z) \to \infty$ as $n \to \infty$
- $f_c^n(z) \to 0$ as $n \to \infty$
- none of the above

Complex Dynamics

When $c = 0$ there are 3 distinct cases:

- $f_c^n(z) \to \infty$ as $n \to \infty$
- $f_c^n(z) \to 0$ as $n \to \infty$
- none of the above

- $|z| > 1$

Complex Dynamics

When $c = 0$ there are 3 distinct cases:

- $f_c^n(z) \to \infty$ as $n \to \infty$
- $f_c^n(z) \to 0$ as $n \to \infty$
- none of the above

- $|z| > 1$
- $|z| < 1$

Complex Dynamics

When $c = 0$ there are 3 distinct cases:

- $f_c^n(z) \to \infty$ as $n \to \infty$
- $f_c^n(z) \to 0$ as $n \to \infty$
- none of the above

- $|z| > 1$
- $|z| < 1$
- $|z| = 1$

When $c = -1$ there are 4 distinct cases:

Complex Dynamics

When $c = -1$ there are 4 distinct cases:

- $f_c^n(z) \to \infty$ as $n \to \infty$

Complex Dynamics

When $c = -1$ there are 4 distinct cases:

- $f_c^n(z) \to \infty$ as $n \to \infty$
- $f_c^{2n}(z) \to 0$ as $n \to \infty$

Complex Dynamics

When $c = -1$ there are 4 distinct cases:

- $f_c^n(z) \to \infty$ as $n \to \infty$
- $f_c^{2n}(z) \to 0$ as $n \to \infty$
- $f_c^{2n}(z) \to -1$ as $n \to \infty$

Complex Dynamics

When $c = -1$ there are 4 distinct cases:

- $f_c^n(z) \to \infty$ as $n \to \infty$
- $f_c^{2n}(z) \to 0$ as $n \to \infty$
- $f_c^{2n}(z) \to -1$ as $n \to \infty$
- none of the above

The initial cases are Fatou components $F_m(f_c)$.

Complex Dynamics

The initial cases are Fatou components $F_m(f_c)$.

- Within each Fatou component the behaviour is the same.

The initial cases are Fatou components $F_m(f_c)$.

- Within each Fatou component the behaviour is the same.
- Moreover, nearby points stay nearby under iteration.

The initial cases are Fatou components $F_m(f_c)$.

- Within each Fatou component the behaviour is the same.
- Moreover, nearby points stay nearby under iteration.
- (In fact, they usually get closer.)

The last case (none of the above) is the Julia set $J(f_c)$.

Complex Dynamics

The last case (none of the above) is the Julia set $J(f_c)$.

- Nearby points get further apart under iteration.

The last case (none of the above) is the Julia set $J(f_c)$.

- Nearby points get further apart under iteration.
- But they stay within the Julia set.

Complex Dynamics

The last case (none of the above) is the Julia set $J(f_c)$.

- Nearby points get further apart under iteration.
- But they stay within the Julia set.
- The Julia set is the boundary of the Fatou components.

Image Generation

Image Generation

What do Julia sets look like?

Image Generation

The first step is to determine the number of Fatou components.

Image Generation

The first step is to determine the number of Fatou components.

- There is always one component F_{-1} with $f_c^n(z) \to \infty$.

Image Generation

The first step is to determine the number of Fatou components.

- There is always one component F_{-1} with $f_c^n(z) \to \infty$.
- Call the number of other components p.

Image Generation

The first step is to determine the number of Fatou components.

- There is always one component F_{-1} with $f_c^n(z) \to \infty$.
- Call the number of other components p.
- Then there are components $F_m, 0 \leq m < p$ with $f_c^{pn}(z) \to z_{*m}$.

Image Generation

The first step is to determine the number of Fatou components.

- There is always one component F_{-1} with $f_c^n(z) \to \infty$.
- Call the number of other components p.
- Then there are components $F_m, 0 \leq m < p$ with $f_c^{pn}(z) \to z_{*m}$.

The algorithm for determining p from c is quite involved, so I won't go into it now.

Image Generation

The second step is to determine the attractor of F_0:
(if $p = 0$, skip this step)

Image Generation

The second step is to determine the attractor of F_0:
(if $p = 0$, skip this step)

- Define $z_{*0} = \lim_{n \to \infty} f_c^{pn}(0)$.

Image Generation

The second step is to determine the attractor of F_0:
(if $p = 0$, skip this step)

- Define $z_{*0} = \lim_{n\to\infty} f_c^{pn}(0)$.
- z_{*0} satisfies $f_c^p(z_{*0}) = z_{*0}$.

Image Generation

The second step is to determine the attractor of F_0:
(if $p = 0$, skip this step)

- Define $z_{*0} = \lim_{n\to\infty} f_c^{pn}(0)$.
- z_{*0} satisfies $f_c^p(z_{*0}) = z_{*0}$.
- The solution can be found using Newton's method.

Image Generation

Now we can iterate $f_c(z)$ with z set by the coordinates of the pixel within the image, to determine which Fatou component z is in:

Image Generation

Now we can iterate $f_c(z)$ with z set by the coordinates of the pixel within the image, to determine which Fatou component z is in:

- We need two numbers, a large E for detecting attraction to ∞ and a small e for detecting attraction to z_{*0}.

Image Generation

Now we can iterate $f_c(z)$ with z set by the coordinates of the pixel within the image, to determine which Fatou component z is in:

- We need two numbers, a large E for detecting attraction to ∞ and a small e for detecting attraction to z_{*0}.
- If $|f_c^n(z)| > E$, then $z \in F_{-1}$.

Image Generation

Now we can iterate $f_c(z)$ with z set by the coordinates of the pixel within the image, to determine which Fatou component z is in:

- We need two numbers, a large E for detecting attraction to ∞ and a small e for detecting attraction to z_{*0}.
- If $|f_c^n(z)| > E$, then $z \in F_{-1}$.
- If $|f_c^n(z) - z_{*0}| < e$, then $z \in F_{n \bmod p}$.

Image Generation

Now we can iterate $f_c(z)$ with z set by the coordinates of the pixel within the image, to determine which Fatou component z is in:

- We need two numbers, a large E for detecting attraction to ∞ and a small e for detecting attraction to z_{*0}.
- If $|f_c^n(z)| > E$, then $z \in F_{-1}$.
- If $|f_c^n(z) - z_{*0}| < e$, then $z \in F_{n \mod p}$.
- If $n > N$, where N is a maximum iteration count necessary for finite computers, then we give up, and don't know much about z.

Image Generation

While iterating, keep track of the derivative:

Image Generation

While iterating, keep track of the derivative:

- $\frac{\partial}{\partial z} f_c^0(z) = 1$

Image Generation

While iterating, keep track of the derivative:

- $\frac{\partial}{\partial z} f_c^0(z) = 1$
- $\frac{\partial}{\partial z} f_c^{n+1}(z) = 2 f_c^n(z) \frac{\partial}{\partial z} f_c^n(z)$

Image Generation

While iterating, keep track of the derivative:

- $\frac{\partial}{\partial z} f_c^0(z) = 1$
- $\frac{\partial}{\partial z} f_c^{n+1}(z) = 2 f_c^n(z) \frac{\partial}{\partial z} f_c^n(z)$
- In imperative programming language pseudo-code:

```
z := pixel coordinates
d := 1
for n := 0 to N
  d := 2 * z * d
  z := z * z + c
```

Image Generation

Why do we need the derivative?

Image Generation

Why do we need the derivative?

- The Julia set is the boundary of the Fatou components.

Image Generation

Why do we need the derivative?

- The Julia set is the boundary of the Fatou components.
- How to detect the boundary, if there is only one Fatou component?

Image Generation

Why do we need the derivative?

- The Julia set is the boundary of the Fatou components.
- How to detect the boundary, if there is only one Fatou component?
- Even if there are more components, the boundary might be very thin in places.

Image Generation

The derivative provides an estimate of the distance to the Julia set:

Image Generation

The derivative provides an estimate of the distance to the Julia set:

$$d = \frac{|f_c^n(z)| \log |f_c^n(z)|}{\left| \frac{\partial}{\partial z} f_c^n(z) \right|}$$

Image Generation

The derivative provides an estimate of the distance to the Julia set:

$$d = \frac{|f_c^n(z)| \log |f_c^n(z)|}{\left| \frac{\partial}{\partial z} f_c^n(z) \right|}$$

If d is small compared to the pixel size, the Julia set passes through the pixel.

Image Generation

The derivative provides an estimate of the distance to the Julia set:

$$d = \frac{|f_c^n(z)| \log |f_c^n(z)|}{\left| \frac{\partial}{\partial z} f_c^n(z) \right|}$$

If d is small compared to the pixel size, the Julia set passes through the pixel.
This formula is only valid for F_{-1} with $f_c^n(z) \to \infty$, so it's best to make E as large as reasonably possible.

Image Generation

The derivative provides an estimate of the distance to the Julia set:

$$d = \frac{|f_c^n(z)| \log |f_c^n(z)|}{\left| \frac{\partial}{\partial z} f_c^n(z) \right|}$$

If d is small compared to the pixel size, the Julia set passes through the pixel.
This formula is only valid for F_{-1} with $f_c^n(z) \to \infty$, so it's best to make E as large as reasonably possible.
(There are other formulae for the other cases, but I couldn't get them to work reliably.)

Image Generation

Now we know the Fatou component F_m for each pixel, and for $m = -1$ we also have a distance estimate d. This is enough to generate an image:

Image Generation

Now we know the Fatou component F_m for each pixel, and for $m = -1$ we also have a distance estimate d. This is enough to generate an image:

- If $m = -1$ and d is small, then colour the pixel black.

Image Generation

Now we know the Fatou component F_m for each pixel, and for $m = -1$ we also have a distance estimate d. This is enough to generate an image:

- If $m = -1$ and d is small, then colour the pixel black.
- If $m = -1$ and d is large, then colour the pixel white.

Image Generation

Now we know the Fatou component F_m for each pixel, and for $m = -1$ we also have a distance estimate d. This is enough to generate an image:

- If $m = -1$ and d is small, then colour the pixel black.
- If $m = -1$ and d is large, then colour the pixel white.

Compare our m with the m for neighbouring pixels:

Image Generation

Now we know the Fatou component F_m for each pixel, and for $m = -1$ we also have a distance estimate d. This is enough to generate an image:

- If $m = -1$ and d is small, then colour the pixel black.
- If $m = -1$ and d is large, then colour the pixel white.

Compare our m with the m for neighbouring pixels:

- If any are different, then colour the pixel black.

Image Generation

Now we know the Fatou component F_m for each pixel, and for $m = -1$ we also have a distance estimate d. This is enough to generate an image:

- If $m = -1$ and d is small, then colour the pixel black.
- If $m = -1$ and d is large, then colour the pixel white.

Compare our m with the m for neighbouring pixels:

- If any are different, then colour the pixel black.
- If all are the same, then colour the pixel white.

Image Generation

Now we know the Fatou component F_m for each pixel, and for $m = -1$ we also have a distance estimate d. This is enough to generate an image:

- If $m = -1$ and d is small, then colour the pixel black.
- If $m = -1$ and d is large, then colour the pixel white.

Compare our m with the m for neighbouring pixels:

- If any are different, then colour the pixel black.
- If all are the same, then colour the pixel white.

This image has black pixels near the Julia set, and white pixels elsewhere.

Image Generation

Now we know the Fatou component F_m for each pixel, and for $m = -1$ we also have a distance estimate d. This is enough to generate an image:

- If $m = -1$ and d is small, then colour the pixel black.
- If $m = -1$ and d is large, then colour the pixel white.

Compare our m with the m for neighbouring pixels:

- If any are different, then colour the pixel black.
- If all are the same, then colour the pixel white.

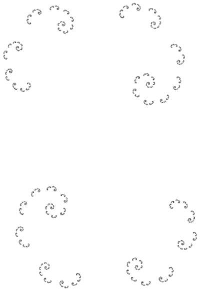
This image has black pixels near the Julia set, and white pixels elsewhere.
An extension is to colour pixels by the value of m, instead of white, as in the following examples.

Examples

Example: needle dust

$c = -2.1$
$+ 0.0i$
$\dim \approx 0.800\ldots$

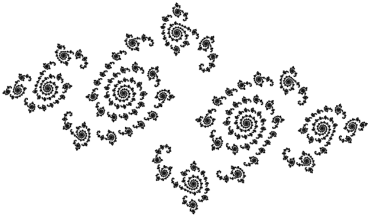
Example: elephant dust



$$c = +0.5 \\ +0.1i$$

$$\dim \approx 1.167\ldots$$

Example: seahorse dust



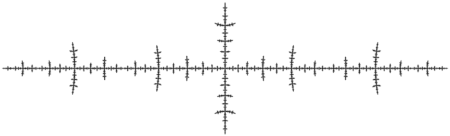
$$c = -0.75 + 0.25i$$

$$\dim \approx 1.609\ldots$$

Example: needle tip dendrite

$$c = -2$$
$$+ 0i$$
$$\dim \approx 0.999\ldots$$

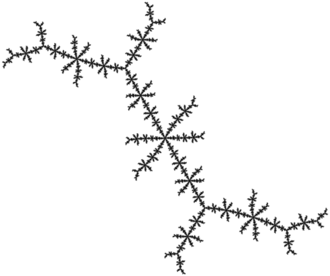
Example: 2-way hub dendrite



$$c = -1.54368 + 0i$$

$\dim \approx 1.450\ldots$

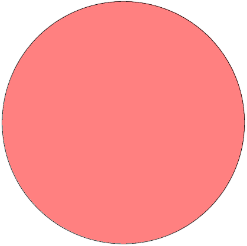
Example: 3-way hub dendrite



$$c = -0.10109 + 0.95628i$$

$\dim \approx 1.564\ldots$

Example: period 1

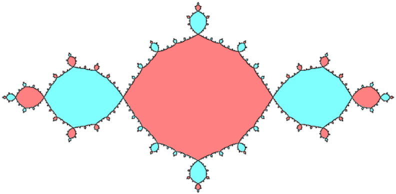


$c = +0$

$+0i$

$\dim \approx 1.086\ldots$

Example: period 2

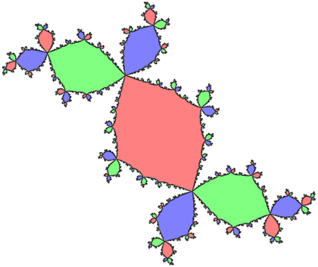


$c = -1$

$+ 0i$

dim $\approx 1.331\ldots$

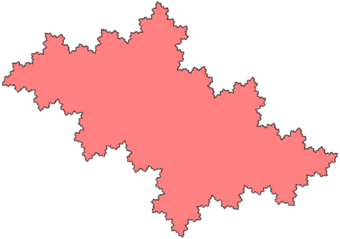
Example: period 3



$c = -0.12256$
$+0.74486i$

$\dim \approx 1.443\ldots$

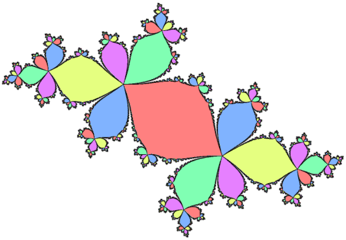
Example: period 1 near 2 over 5



$$c = -0.45400 + 0.49378i$$

$$\dim \approx 1.257\ldots$$

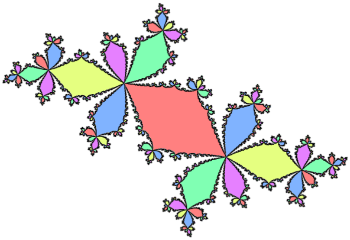
Example: period 5 near 2 over 5



$c = -0.48734$
$+0.53932i$

dim $\approx 1.485\ldots$

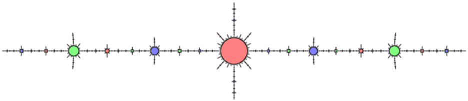
Example: period 5



$$c = -0.50434 + 0.56276i$$

$\dim \approx 1.550\ldots$

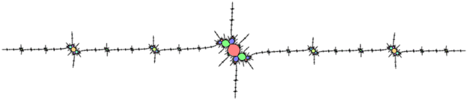
Example: period 3 island



$c = -1.75487$
$+ 0i$

$\dim \approx 1.309\ldots$

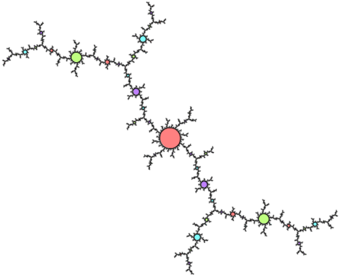
Example: period 3 island 1 over 3 bulb



$$c = -1.75778 + 0.01379i$$

dim $\approx 1.455\ldots$

Example: period 4 island



$$c = -0.15652 + 1.03224i$$

$$\dim \approx 1.463\ldots$$

Fractal Dimension of Julia Sets

Concept

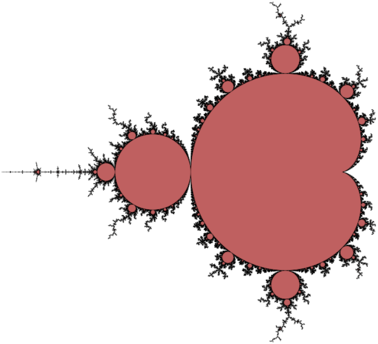
Concept

The Mandelbrot set is a c-plane plot of whether $J(f_c)$ is connected.

Concept

The Mandelbrot set is a c-plane plot of whether $J(f_c)$ is connected. Some visualisations of the Mandelbrot set use psychedelic colours, but the mathematical object is binary.

Concept



The Mandelbrot set

Concept

What would a c-plane plot of $\dim J(f_c)$ look like?

Concept

What would a c-plane plot of $\dim J(f_c)$ look like?
$0 \leq \dim J(f_c) \leq 2$, so a spectrum of colours would be necessary.

Implementation

Implementation

The implementation is written in C99:

Implementation

The implementation is written in C99:

- C99 supports complex numbers.

Implementation

The implementation is written in C99:

- C99 supports complex numbers.
- Low-level and efficient.

Implementation

The implementation is written in C99:

- C99 supports complex numbers.
- Low-level and efficient.
- Most libraries have C interfaces.

Implementation

The implementation uses OpenGL for graphics:

Implementation

The implementation uses OpenGL for graphics:

- OpenGL supports programmable graphics hardware (GPUs).

Implementation

The implementation uses OpenGL for graphics:

- OpenGL supports programmable graphics hardware (GPUs).
- GPUs are very good at parallel number-crunching tasks, like rendering a Julia set.

Implementation

The implementation uses OpenGL for graphics:

- OpenGL supports programmable graphics hardware (GPUs).
- GPUs are very good at parallel number-crunching tasks, like rendering a Julia set.
- Mipmap generation reduces images to progressively coarser pixel resolutions.

Implementation

The implementation uses OpenGL for graphics:

- OpenGL supports programmable graphics hardware (GPUs).
- GPUs are very good at parallel number-crunching tasks, like rendering a Julia set.
- Mipmap generation reduces images to progressively coarser pixel resolutions.
- Occlusion queries can be used for counting pixels.

Implementation

The implementation uses a few fragment shaders:

Implementation

The implementation uses a few fragment shaders:

- to compute Fatou components and distance estimates;

Implementation

The implementation uses a few fragment shaders:

- to compute Fatou components and distance estimates;
- to post-process the Fatou component index and distance estimate into a black and white image of the Julia set;

Implementation

The implementation uses a few fragment shaders:

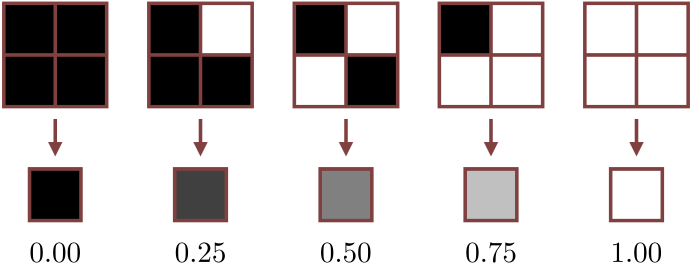
- to compute Fatou components and distance estimates;
- to post-process the Fatou component index and distance estimate into a black and white image of the Julia set;
- to discard pixels below a threshold.

Implementation

Mipmap reduction **averages** groups of pixels:

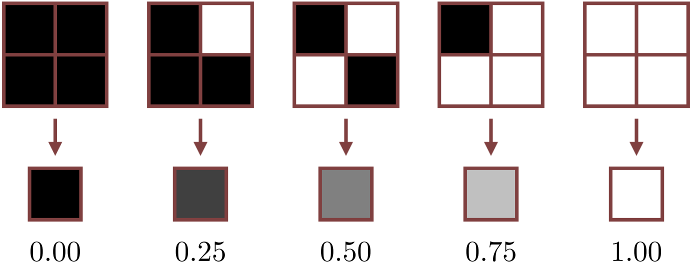
Implementation

Mipmap reduction **averages** groups of pixels:



Implementation

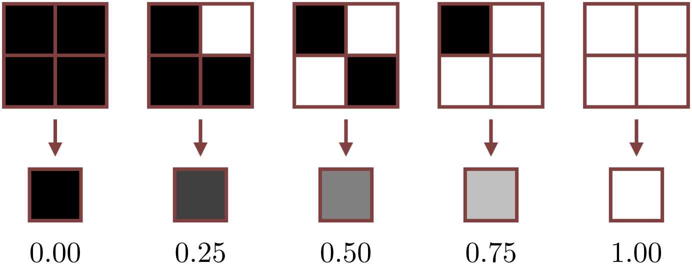
Mipmap reduction **averages** groups of pixels:



Box-counting should count if **any** subpixel was black.

Implementation

Mipmap reduction **averages** groups of pixels:

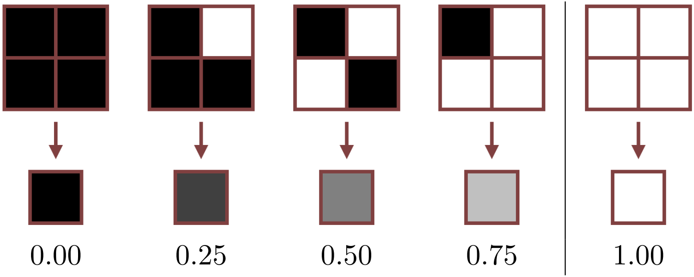


Box-counting should count if **any** subpixel was black.

The solution is to threshold the grey level in each mipmap level.

Implementation

Mipmap reduction **averages** groups of pixels:



Box-counting should count if **any** subpixel was black.

The solution is to threshold the grey level in each mipmap level.

The threshold should between the lightest grey and white.

Implementation

Box-counting is performed with occlusion queries:

Implementation

Box-counting is performed with occlusion queries:

- First clear the depth buffer.

Implementation

Box-counting is performed with occlusion queries:

- First clear the depth buffer.
- Then draw the Julia set, discarding pixels below a threshold.

Implementation

Box-counting is performed with occlusion queries:

- First clear the depth buffer.
- Then draw the Julia set, discarding pixels below a threshold.
- Then draw again, but further away.

Implementation

Box-counting is performed with occlusion queries:

- First clear the depth buffer.
- Then draw the Julia set, discarding pixels below a threshold.
- Then draw again, but further away.
- The depth test prevents pixels that were rendered the first time from being drawn, so only the previously discarded pixels pass.

Implementation

Box-counting is performed with occlusion queries:

- First clear the depth buffer.
- Then draw the Julia set, discarding pixels below a threshold.
- Then draw again, but further away.
- The depth test prevents pixels that were rendered the first time from being drawn, so only the previously discarded pixels pass.
- The occlusion query counts the number of passed pixels in the second draw.

Implementation

Performance:

Implementation

Performance:

- Faster than a CPU-based implementation.

Implementation

Performance:

- Faster than a CPU-based implementation.
- But it's still time-consuming.

Implementation

Performance:

- Faster than a CPU-based implementation.
- But it's still time-consuming.
- The final image took over 5 hours to render.

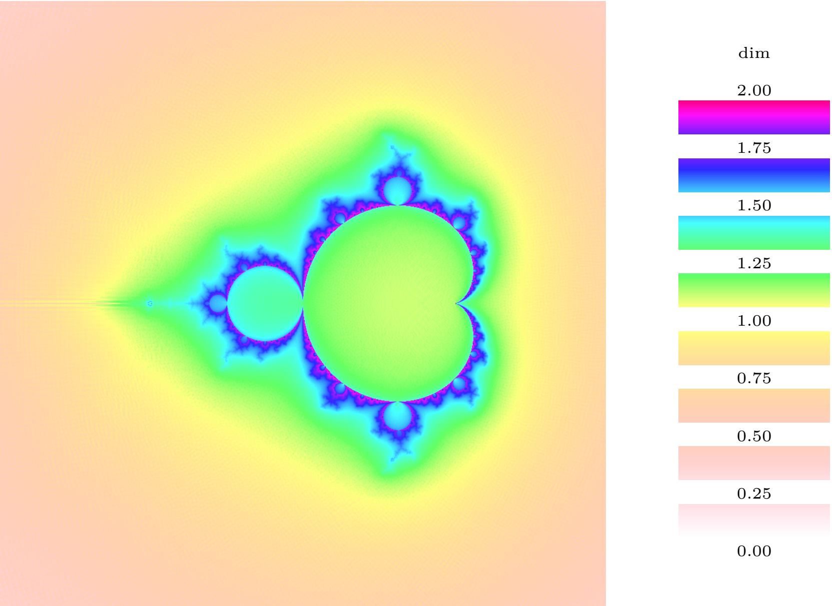
Implementation

Performance:

- Faster than a CPU-based implementation.
- But it's still time-consuming.
- The final image took over 5 hours to render.
- Watching the image appear pixel-by-pixel brings back memories of rendering fractals a couple of decades ago…

Results

Results



Results

But is it accurate?

Results

But is it accurate?
No.

Results

But is it accurate?
No.
But it's pretty close.

Results

Recall the formula I actually used:

$$\dim = \frac{1}{2}\log_2 \frac{N_{2r_0}}{N_{8r_0}}$$

$$r_0 = \text{pixel size}$$

Results

Recall the formula I actually used:

$$\dim = \frac{1}{2} \log_2 \frac{N_{2r_0}}{N_{8r_0}}$$

$$r_0 = \text{pixel size}$$

This formula is based on simple linear regression of $\log N$ against $\log r$.

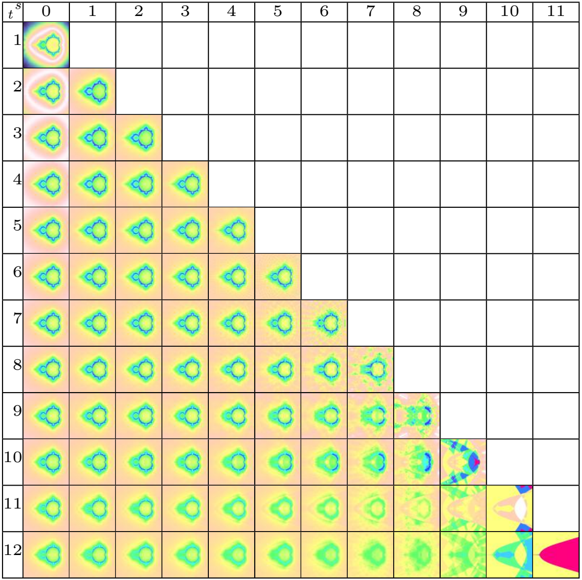
Results

Recall the formula I actually used:

$$\dim = \frac{1}{2} \log_2 \frac{N_{2r_0}}{N_{8r_0}}$$
$$r_0 = \text{pixel size}$$

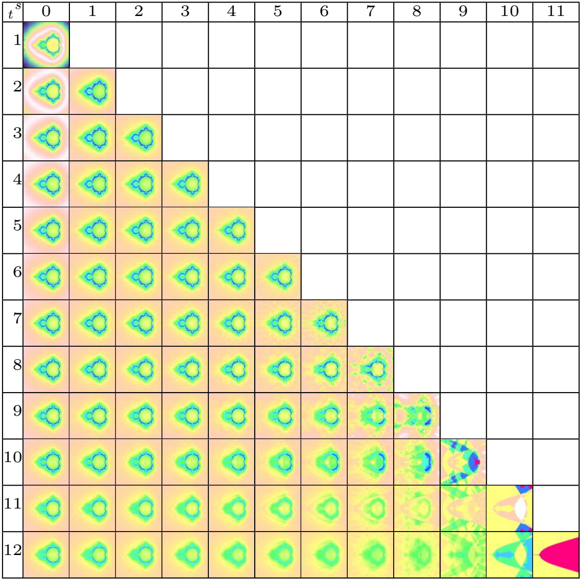
This formula is based on simple linear regression of $\log N$ against $\log r$. I tried all possibilities of $0 \leq s < t \leq 12$ for a regression range between $2^s r_0$ and $2^t r_0$.

Results



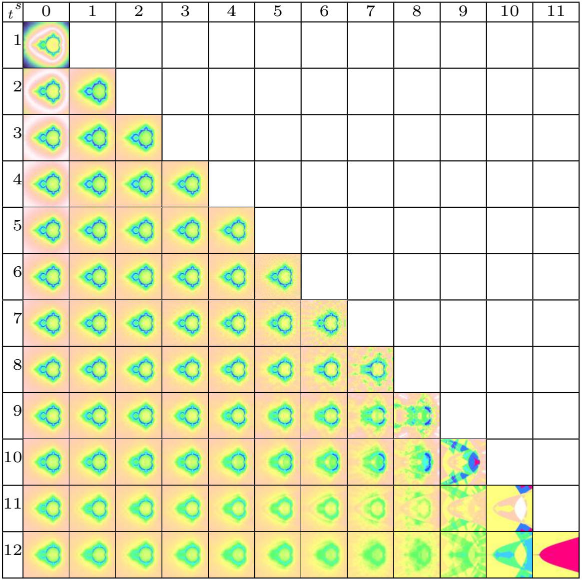
When $s = 0$ and t is small, the dimension calculated is wrong because the Julia set is too inexact at the resolution of the pixel grid.

Results



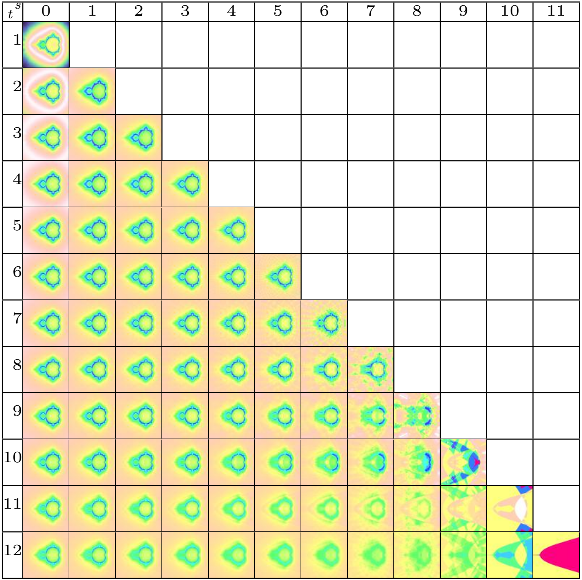
Increasing s a little reduces this artifact of pixel resolution, but t needs to stay small or the results go bad again.

Results



When both s and t are large, the results are nonsense.

Results



The best trade-off seems to be at $s = 1$ and $t = 3$, which gives the formula I actually used.

The End

The End.