

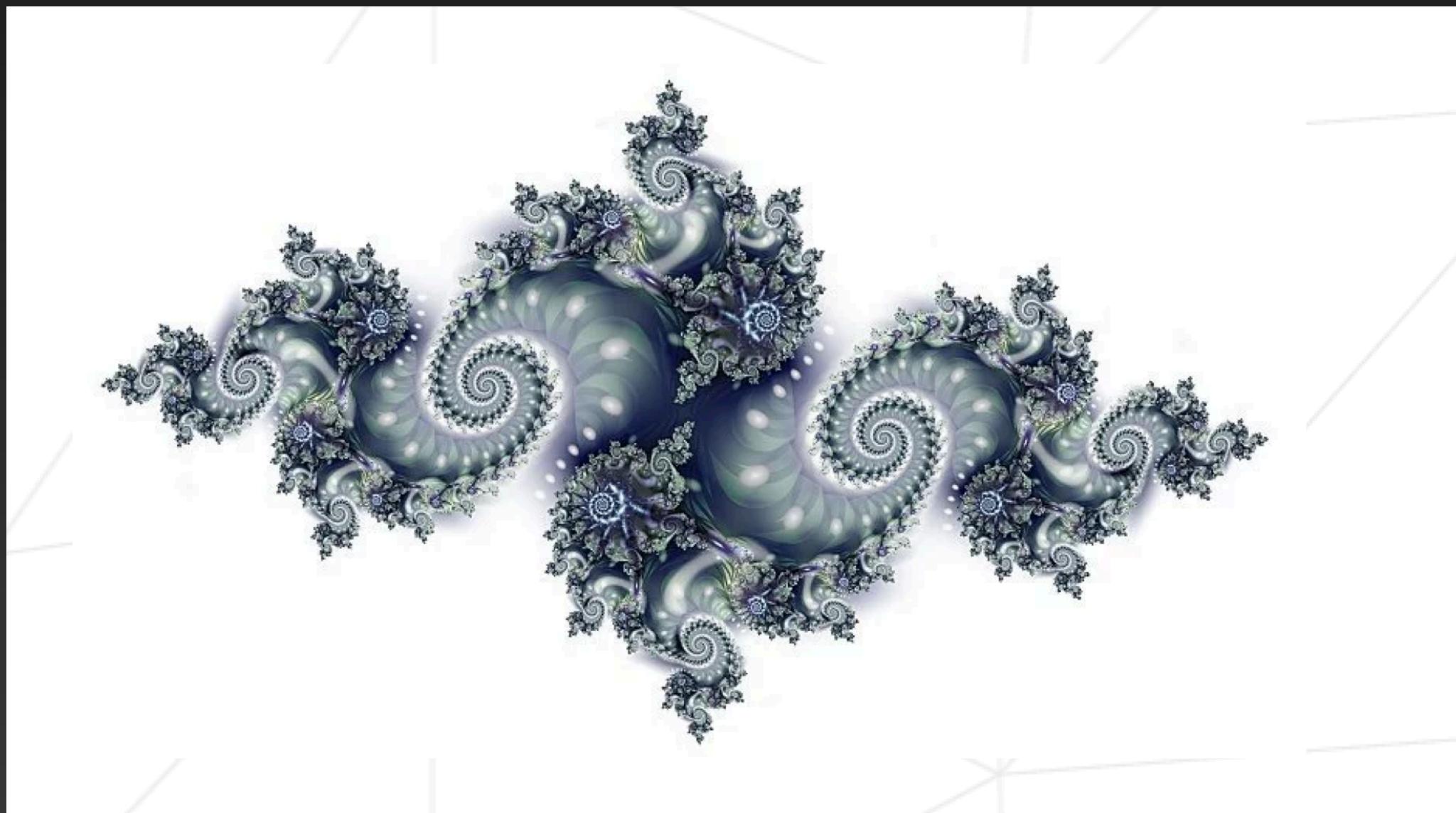
Fractol

Что такое fractal?

Фрактал это слово, придуманное в 1975 году Бенуа Мандельбротом.

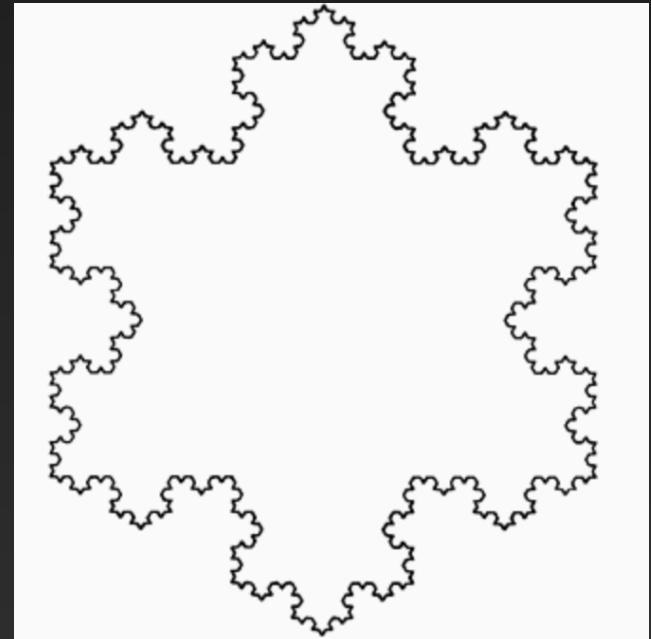
От латинского «*fractus*» - дробленый, сломанный, разбитый.

На языке математики фрактал - это множество со свойством самоподобия. Другими словами, каждый элемент множества является точной или приближенной копией целого.

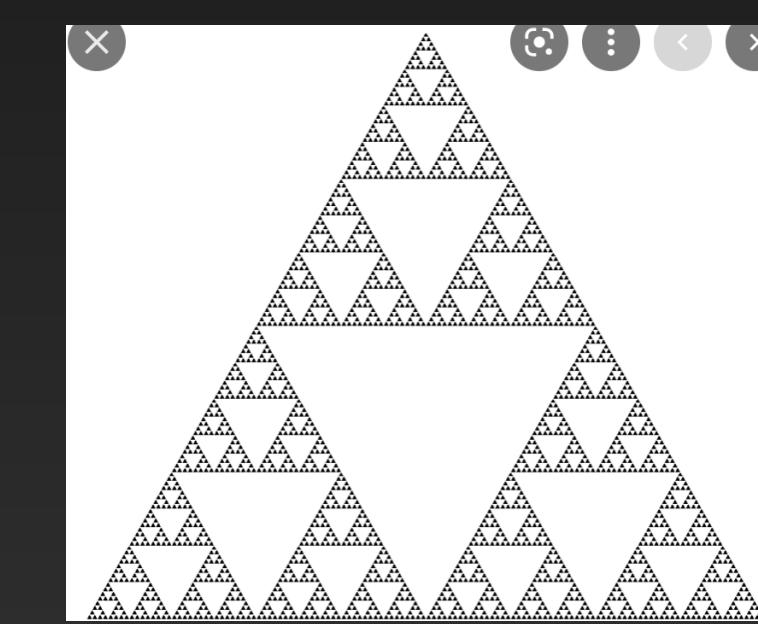


Виды фракталов

Геометрические
(конструктивные)

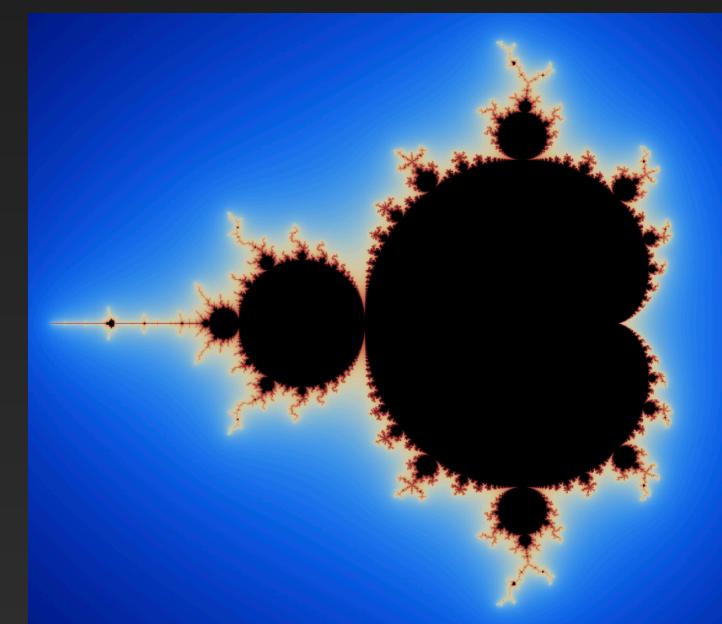


Снежинка Коха

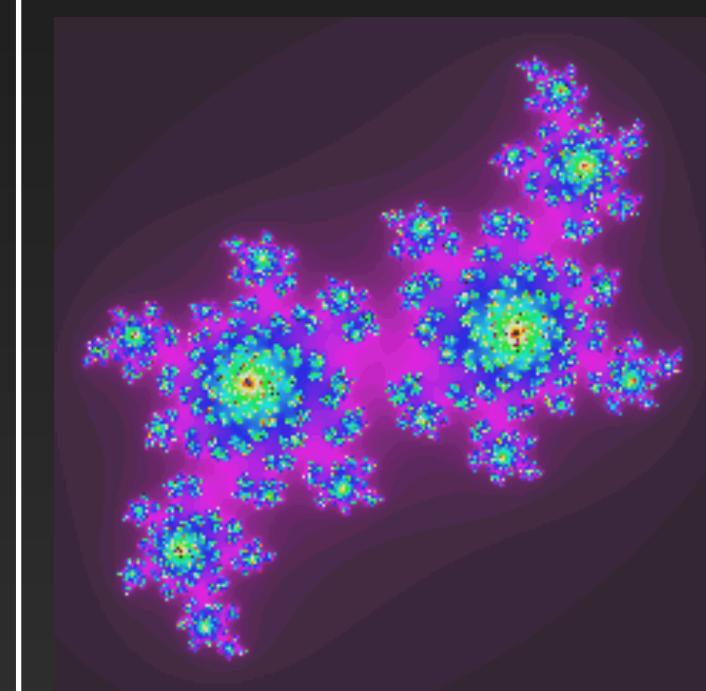


Треугольник Серпинского

Алгебраические
(конструктивные)

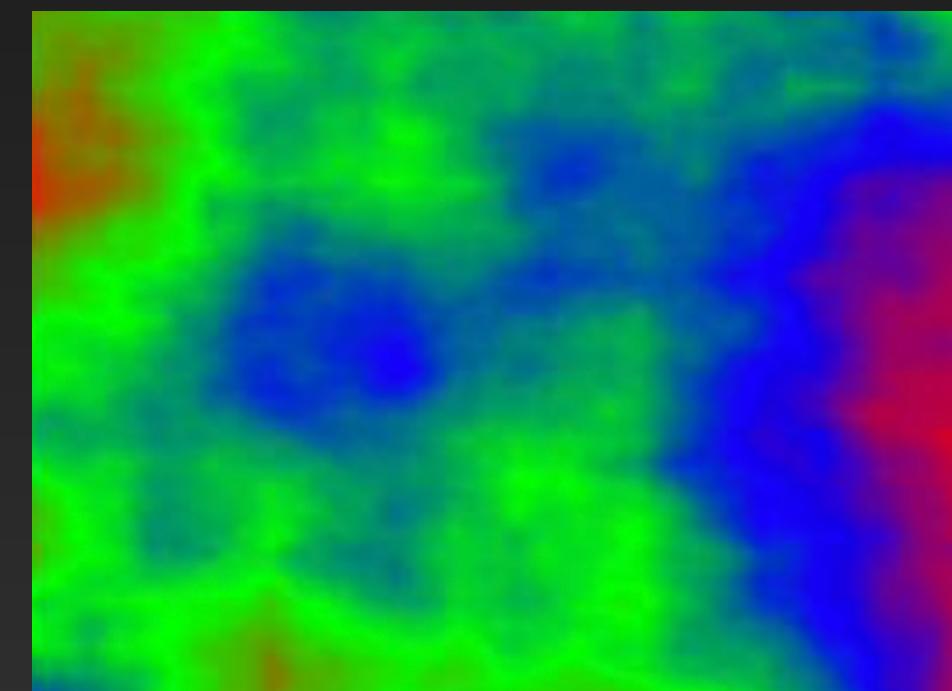


Мандельброт



Множество Жюлиа

Стохастический
(конструктивные)

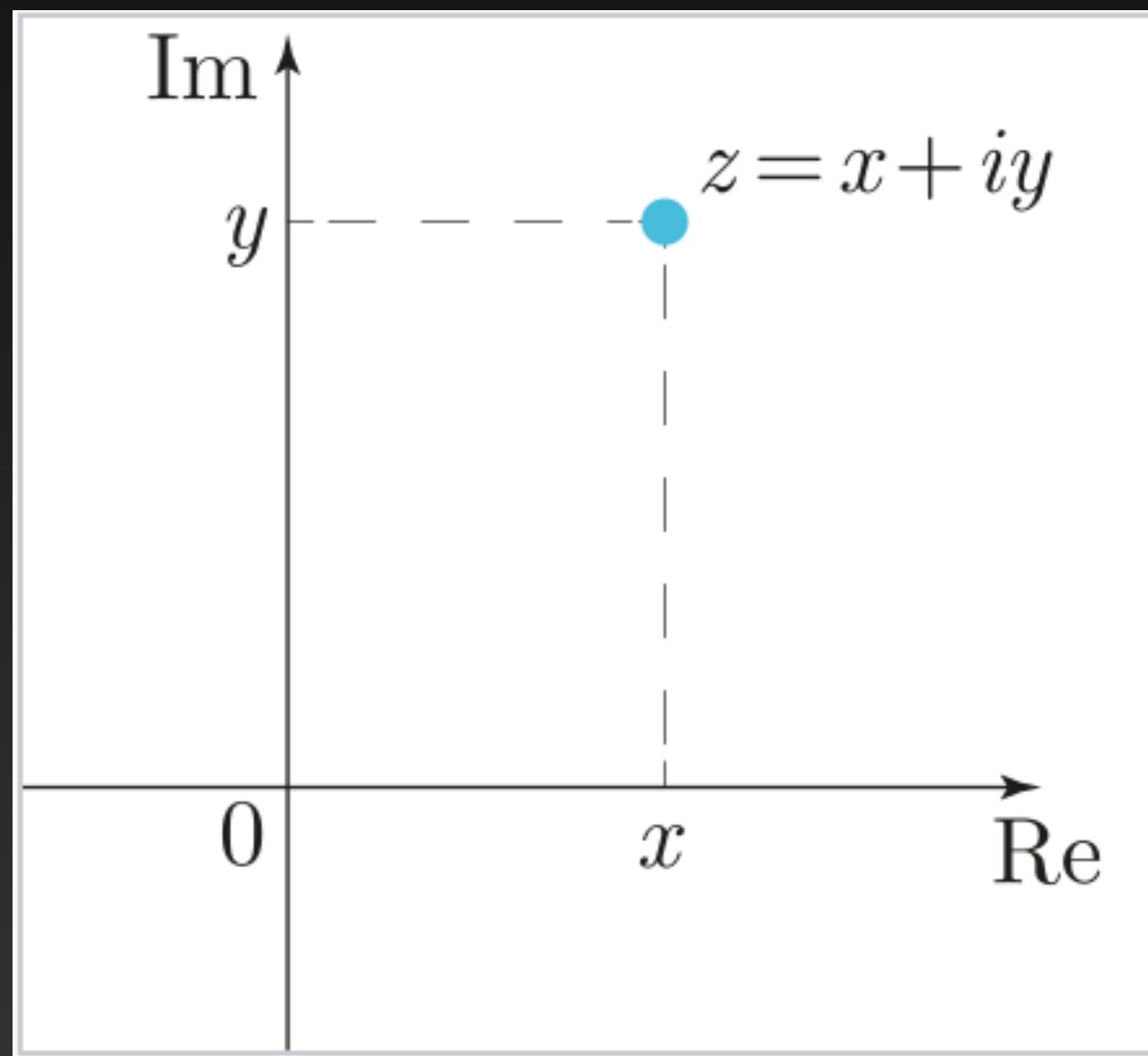


Задаются на комплексной плоскости

Комплексные числа

ОХ - Real part (действительная часть)

ОY - Imaginary part (мнимая часть)

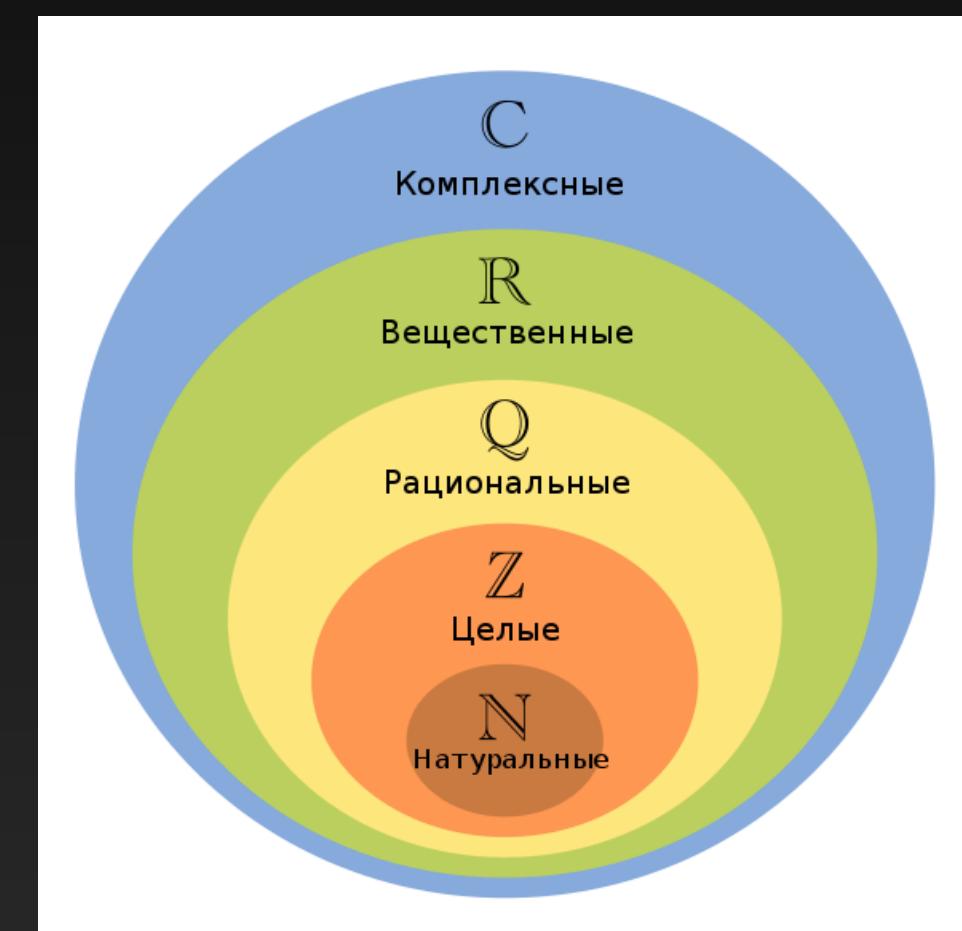


При рассмотрении ряда задач действительных чисел не хватает. Например, уравнение $x^2 + 4 = 0$ не имеет решения на множестве действительных чисел. Поэтому понятие числа нужно расширить.

$$x_1 = -2i$$

$$x_2 = 2i$$

$$i = \sqrt{-1} \longrightarrow i^2 = -1$$

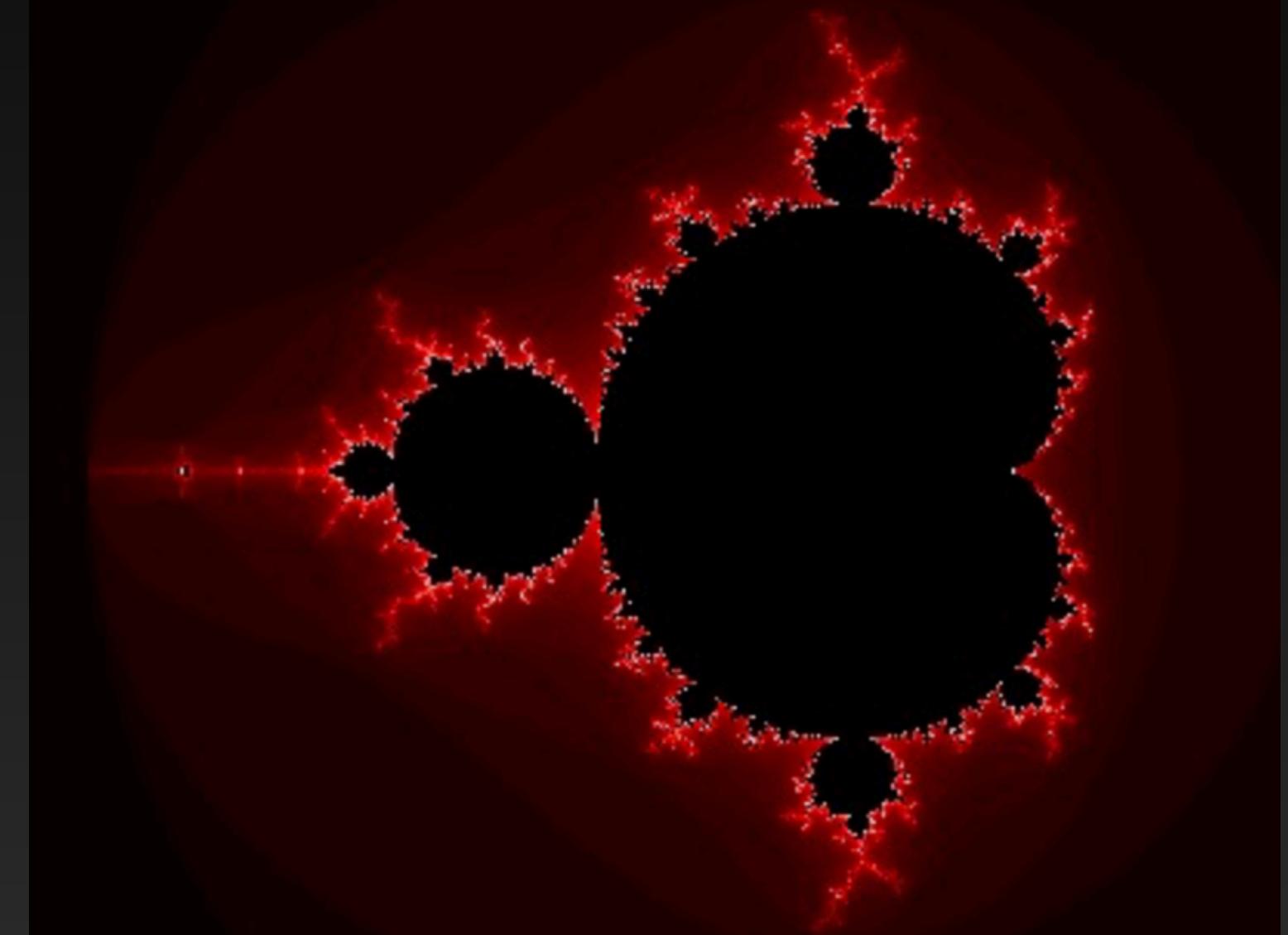


Множество Мандельброта

Множество точек с на комплексной плоскости, для которых последовательность Z_n , определяемая итерациями $Z_0 = 0$, $Z_1 = Z_0^2 + c$, ..., $Z_{n+1} = Z_n^2 + c$, конечна (то есть не уходит в бесконечность).

Визуально выглядит как набор бесконечного количества различных фигур, самая большая - кардиоида (от греческих слов — «сердце» и «вид»).

Простыми словами: нужно проверить комплексные числа по правилу: берем 0, возводим в квадрат и прибавляем к комплексному числу, которое проверяем. Потом полученное число умножаем на самого себя и опять прибавляем к нашему числу и т.д. С каждой итерацией появляется новое комплексное число. Ушло в бесконечность? Нет - тогда добавляем его в множество.



<http://warp.povusers.org/Mandelbrot/>

http://mech.math.msu.su/~shvetz/54/inf/perl-problems/chMandelbrotSet_sIdeas.xhtml

<https://elementy.ru/posters/fractals/Mandelbrot>

Множество Мандельброт

<https://lodev.org/cgtutor/juliamandelbrot.html>

Алгоритм - escape-time - все множество находится внутри круга $R = 2$

Повторяем iteration раз:

Ensemble de Mandelbrot

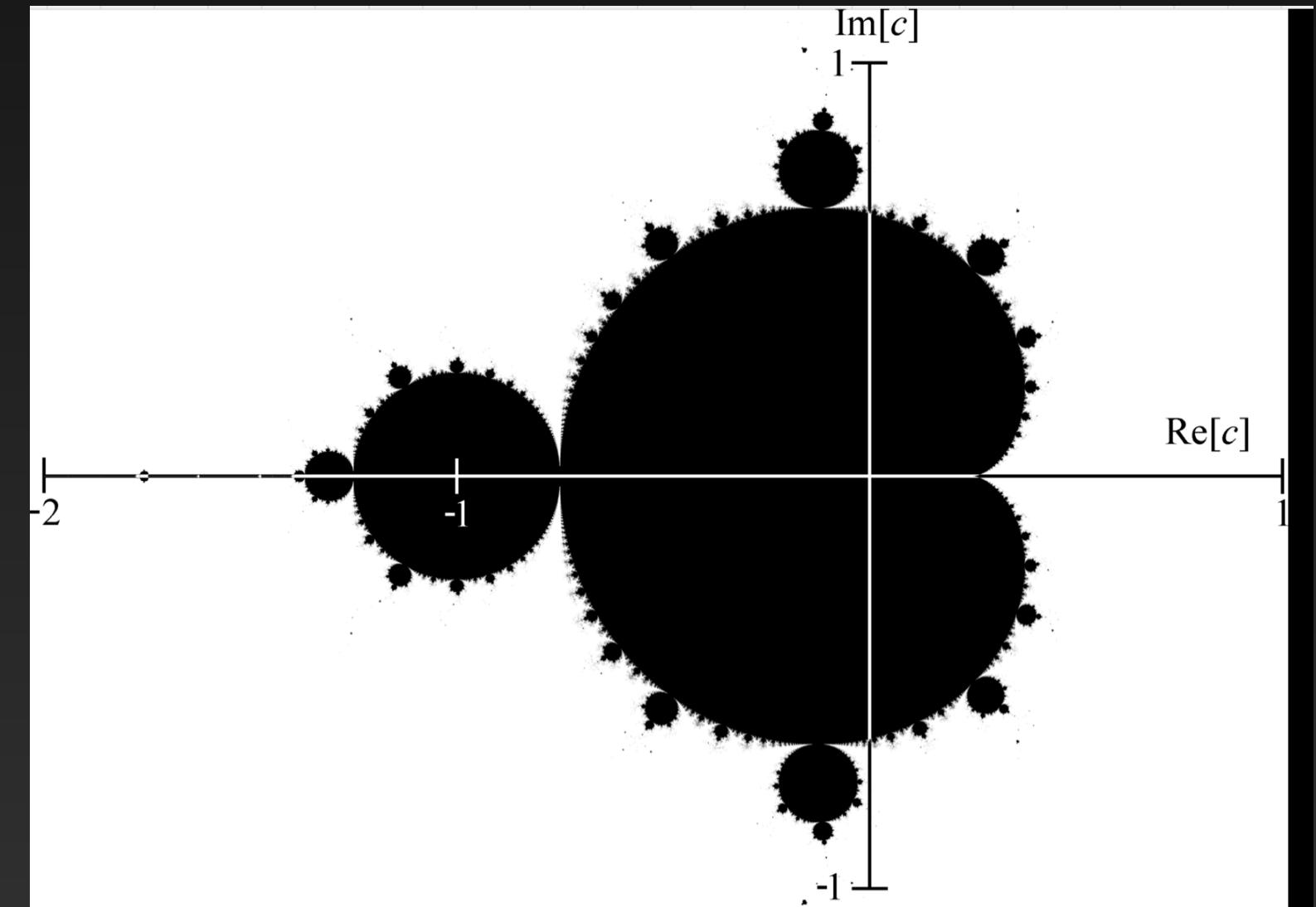
Pour les articles homonymes, voir [Mandelbrot](#).

En [mathématiques](#), l'**ensemble de Mandelbrot** est une [fractale](#) définie comme l'[ensemble](#) des points c du [plan complexe](#) pour lesquels la [suite de nombres complexes](#) définie par récurrence par :

$$\begin{cases} z_0 = 0 \\ z_{n+1} = z_n^2 + c \end{cases}$$

est bornée.

$$(a+bi)^2 = a^2 + 2abi + (bi)^2 = a^2 + 2abi - b^2$$



https://en.wikipedia.org/wiki/Plotting_algorithms_for_the_Mandelbrot_set

https://fr.wikipedia.org/wiki/Ensemble_de_Mandelbrot#/media/Fichier:Cosine_mandelbrot_set_2.png

<https://jonisalonen.com/2013/lets-draw-the-mandelbrot-set/>

<http://sdz.tdct.org/sdz/dessiner-la-fractale-de-mandelbrot.html>

Mandelbrot- and Julia-style Iteration

The Julia sets and Mandelbrot set discussed in this article both use the same iteration function:

$$z \rightarrow z^2 + c$$

The difference between them is the nature of z_0 and c where z_0 is the first value in the sequence and c is the value added to z^2 as we apply the iteration. For the Julia sets, we have:

$$z_0 = z; c = \text{CONSTANT}$$

For the Mandelbrot set we have:

$$z_0 = 0; c = z$$

In both cases, z is the point in the complex plane under consideration. This leads to the observation that we can obtain different fractal images from the same iteration function by using either the initial conditions of a Julia set or the initial conditions of the Mandelbrot set. For convenience, I call the first a Julia-style iteration and the second a Mandelbrot-style iteration. This may not be entirely valid from a mathematical viewpoint, since Julia sets and Mandelbrot sets have very formal definitions, but it is a useful abstraction from a programming viewpoint.

All fractal images produced by this application use either a Mandelbrot-style iteration or a Julia-style iteration, possibly with minor variations (for example, the Nova fractal is obtained by a Mandelbrot-style iteration, but with $z_0 = \text{CONSTANT}$).

[Back to the index](#)

[Демо на code.org](#)

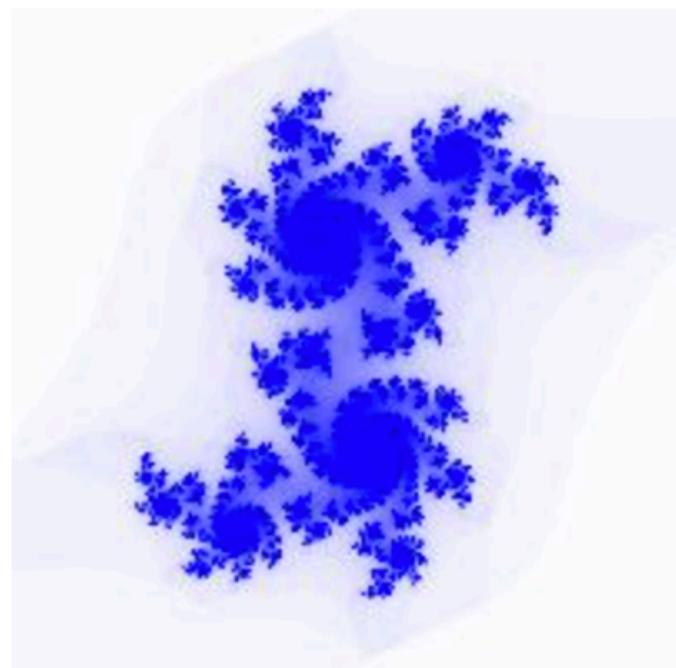
Фрактал Жюлиа

Рассмотрим ту же последовательность комплексных чисел, что и для множества Мандельброта:

$$z_{k+1} = z_k^2 + c, k = 0, 1, 2, \dots$$

Исходные данные, этапы построения и условия остановки – те же, что и для фрактала Мандельброта, за исключением:

- значение c фиксируется: $c = 0,36 + 0,36i$
- начальное значение z_0 перебирается дискретно в области $C \in [-1; 1] + [-1; 1]i$



Рассматривая множество в общем виде: $z_{k+1} = z_k^N + c$ и изменяя N и c , можно получать разнообразные фрактальные множества:



Mandelbrot 3D

3D - Slack - <https://www.mitchr.me/SS/mandelbrot/index.html>
<https://www.mitchr.me/SS/mandelbrot/index.html>



Это обеспечивает
простое создание окна, инструмент рисования, изображение и основные события

<https://gist.github.com/KokaKiwi/4052375>

MAN

mlx_ptr – идентификатор подключения к экрану

win_ptr - является идентификатором окна

size_line - это количество байтов, используемых для хранения одной строки изображения в памяти. Эта информация нужна для перехода от одной строки к другой на изображении.

Endian - говорит вам, нужно ли хранить цвет пикселя в изображении в с прямым порядком байтов (Endian == 0) или с обратным порядком байтов (Endian== 1).

В mlx_get_data_addr возвращается адрес, представляющий начало области памяти, где находится изображение.

С этого адреса bits_per_pixel биты представляют цвет первого пикселя в первой строке изображение. Вторая группа биты представляют второй пиксель первой строки и так далее.

Добавлять size_line на адрес, чтобы получить начало второй строки. Вы можете добраться до любого пикселей изображения таким образом.

Mask	Description	Mask	Description
0L	NoEventMask	(1L<<12)	Button5MotionMask
(1L<<0)	KeyPressMask	(1L<<13)	ButtonMotionMask
(1L<<1)	KeyReleaseMask	(1L<<14)	KeymapStateMask
(1L<<2)	ButtonPressMask	(1L<<15)	ExposureMask
(1L<<3)	ButtonReleaseMask	(1L<<16)	VisibilityChangeMask
(1L<<4)	EnterWindowMask	(1L<<17)	StructureNotifyMask
(1L<<5)	LeaveWindowMask	(1L<<18)	ResizeRedirectMask
(1L<<6)	PointerMotionMask	(1L<<19)	SubstructureNotifyMask
(1L<<7)	PointerMotionHintMask	(1L<<20)	SubstructureRedirectMask
(1L<<8)	Button1MotionMask	(1L<<21)	FocusChangeMask
(1L<<9)	Button2MotionMask	(1L<<22)	PropertyChangeMask
(1L<<10)	Button3MotionMask	(1L<<23)	ColormapChangeMask
(1L<<11)	Button4MotionMask	(1L<<24)	OwnerGrabButtonMask

Julia

<http://usefuljs.net/fractals/docs/multibrot.html>

<https://math.berkeley.edu/~kmill/toys/julia/julia.html> - должно быть

Обсуждение #42seoul_global_general

 **donpark** 9 июля 2021 в 9:13 [FDF]
fdf 이제 시작하는데 과제에서 제공해주는 파일
[minilibx_mms_20191025_beta.taz](#)
[minilibx_macos_sierra_20161017.taz](#)
[sources.tgz](#)
이 3개 모두 mlx 파일인가 같은데 fract-ol, so-long, cub3D, miniRT에 있는
[minilibx_mms_20200218_beta.tgz](#) 나 [minilibx_opengl.taz](#)를 사용해서 해도
상관없을까요? (отредактировано)

2 ответа

 **joonpark** 6 мес. назад
저는 클러스터에서 sources.tgz를 사용했고 개인 맥북에서는 minilibx_opengl.taz 썼습니다. 환경에 따라 다를 거 같은데 클러스터면 sources.tgz로 될거 같아요.

 **donpark** 6 мес. назад
감사합니다. 선택하는게 별거 아닌데 참여려웠네요. ^^

 **tde-vlee** 9:15

```
int mandelbrot(int h, int l)
{
    double x0; //scaled x coordinate of pixel (scaled to lie
    in the Mandelbrot X scale (-2.00, 0.47))
    double y0; //scaled y coordinate of pixel (scaled to lie
    in the Mandelbrot Y scale (-1.12, 1.12))
    double x;
    double y;
    int iter;
    int itermax;
    double tempx;

    x0 = (2.47 / 1919 * l - 2.47) + 0.47;
    y0 = (2.24 / 1079 * h - 2.24) + 1.12;
    x = 0;
    y = 0;
    iter = 0;
    itermax = 100;
    while (x * x + y * y < 4 && iter < itermax) // 4 = escape
radius
    {
        tempx = x * x - y * y + x0;
        y = 2 * x * y + y0;
        x = tempx /* + (x0 / 2) */;
        iter++;
    }
    return (palette(iter, itermax));
}
```

h and l are the y and x coordinate of the pixel. The (x0 / 2) is
completely arbitrary and another value can stretch more or less the

Mlx_hook

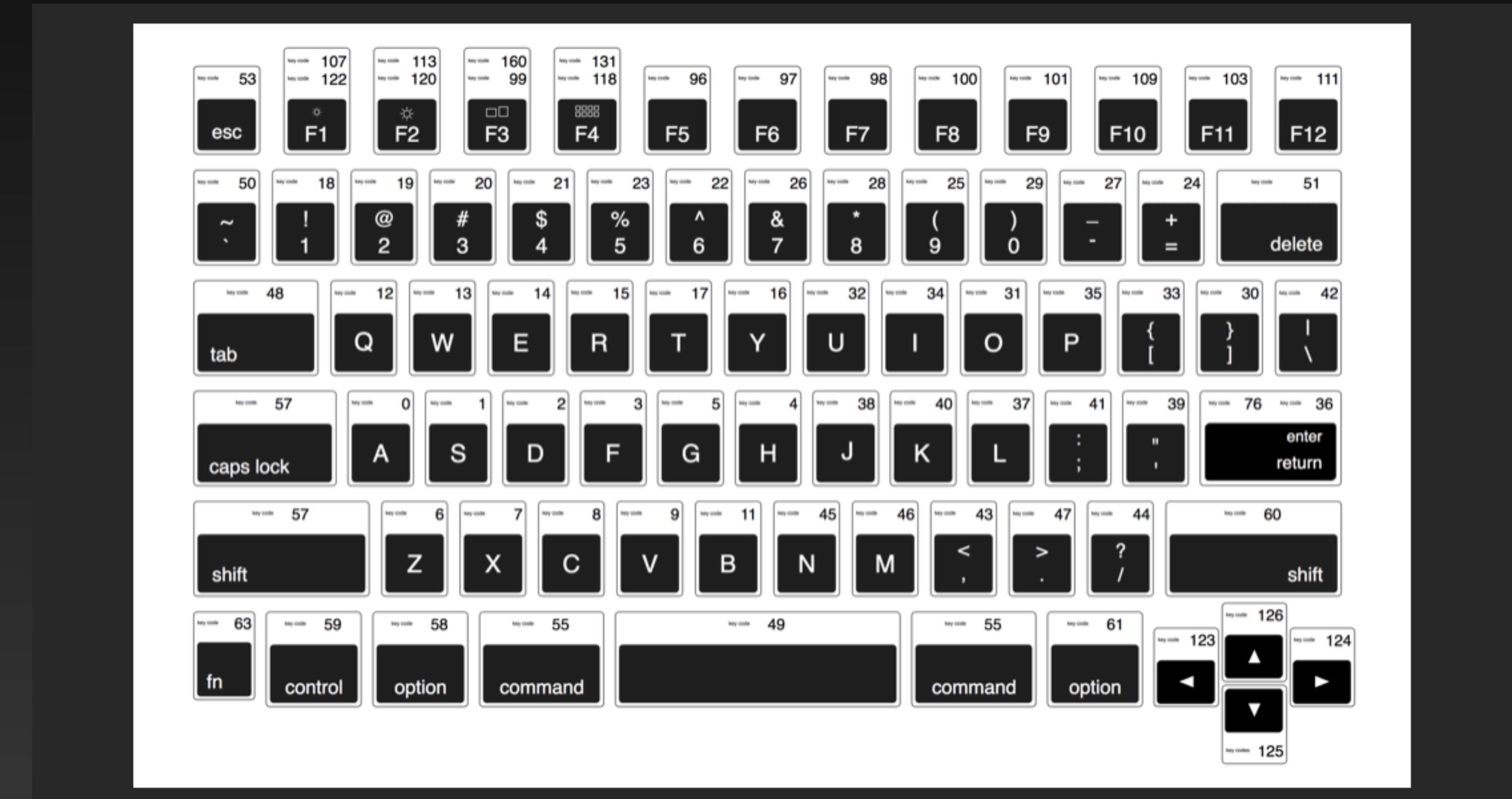
События обрабатываются через маску путем использования X11 -
<https://tronche.com/gui/x/xlib/events/types.html>

MASK /* https://harm-smits.github.io/42docs/libs/mlx_events.html#x11-masks */



Коды клавиш на макбуке

<https://eastmanreference.com/complete-list-of-applescript-key-codes>

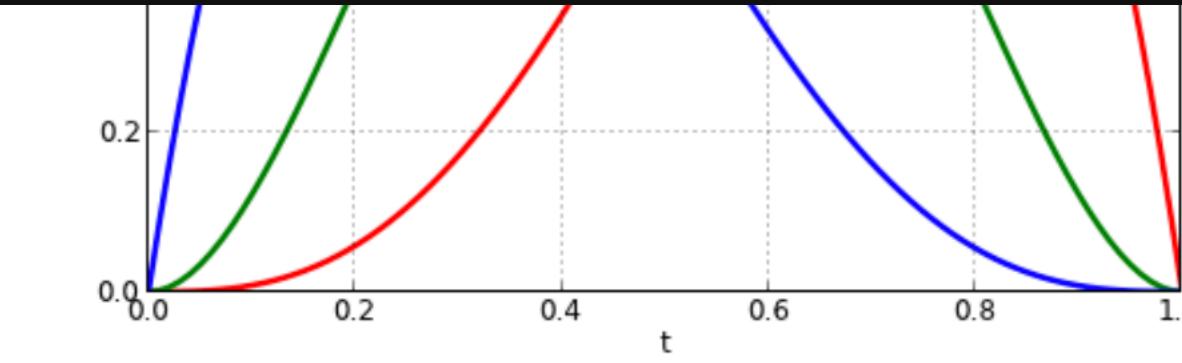
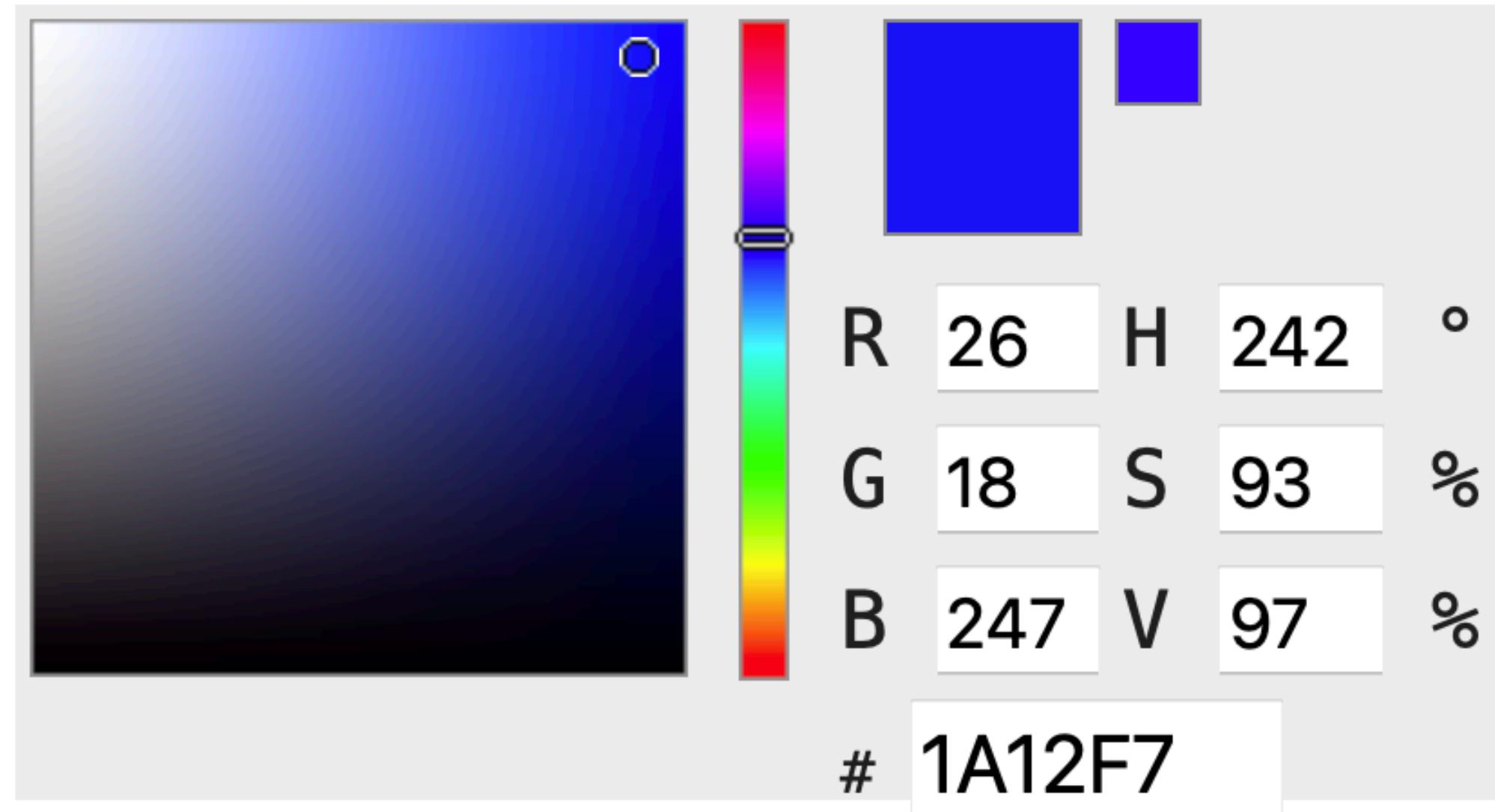


Цвета RGB

https://www.rapidtables.org/ru/web/color/RGB_Color.html

<https://solarianprogrammer.com/2013/02/28/mandelbrot-set-cpp-11/>

Палитра цветов RGB



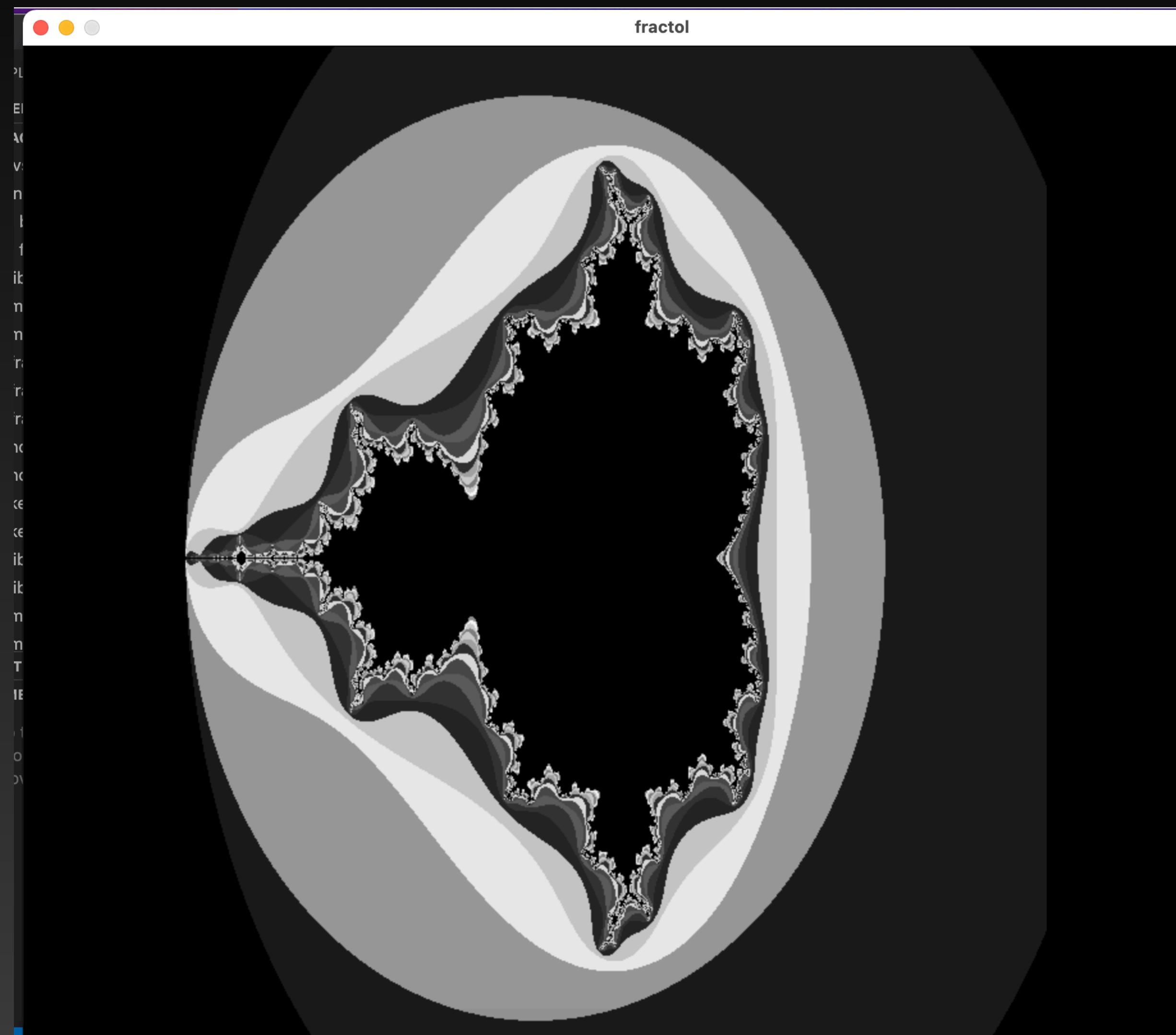
Implementing the above functions in C++ is straightforward:

```
1 std::tuple<int, int, int> get_rgb_smooth(int n, int iter_max) {  
2     // map n on the 0..1 interval  
3     double t = (double)n/(double)iter_max;  
4  
5     // Use smooth polynomials for r, g, b  
6     int r = (int)(9*(1-t)*t*t*t*255);  
7     int g = (int)(15*(1-t)*(1-t)*t*t*255);  
8     int b = (int)(8.5*(1-t)*(1-t)*(1-t)*t*255);  
9     return std::tuple<int, int, int>(r, g, b);  
10 }
```

Let's see now, the effect of using a smooth map from the iteration numbers to the 3D RGB space:



Код, отвечающий за цвет в зависимости от количества итераций (то есть от количества нанесенных точек)



```
31 #include "includes/fractol.h"
32
33 typedef struct s_vars {
34     void    *mlx;
35     void    *win;
36 }           t_vars;
37
38 int ft_close(int keycode, t_vars *vars)
39 {
40     mlx_destroy_window(vars->mlx, vars->win);
41     return (0);
42 }
43
44 int main(void)
45 {
46     t_vars  vars;
47
48     vars.mlx = mlx_init();
49     vars.win = mlx_new_window(vars.mlx, 500, 500, "Hello world!");
50     mlx_hook(vars.win, 2, 1L<<0, close, &vars);
51     mlx_loop(vars.mlx);
52 }
```

TERMINAL PROBLEMS OUTPUT

▼ TERMINAL

