COMP3271 Computer Graphics

2D Fractal Rendering

2019-20

Objectives

Understand the framebuffer and the viewport space

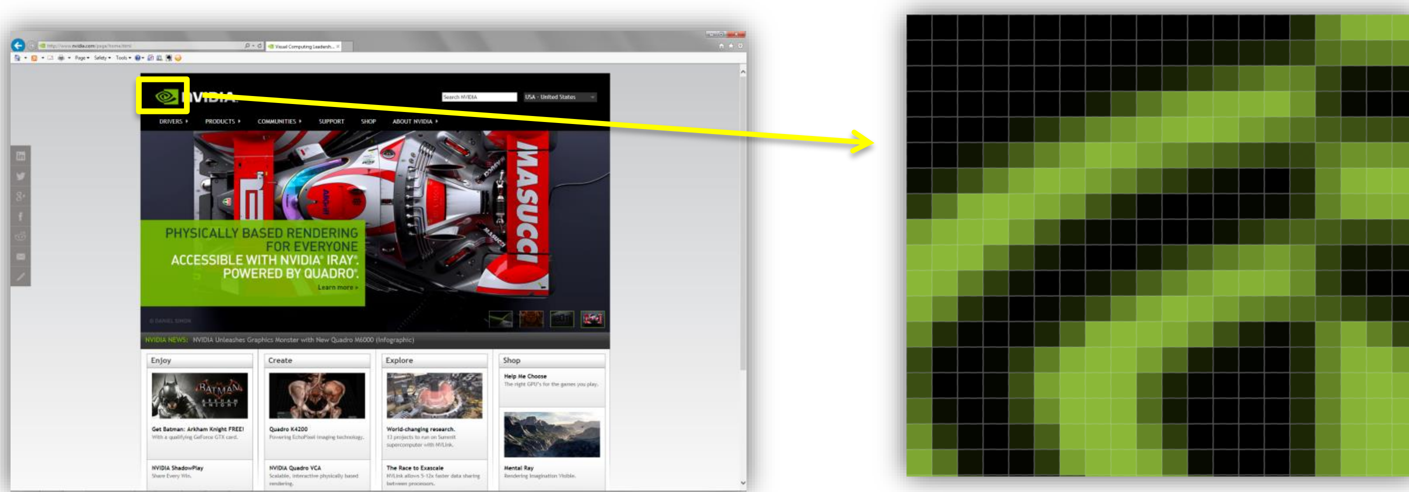
Create a CG generated image by turning abstract formulas into visualization.

Framebuffer

A **framebuffer** is some physical memory (RAM) used to store the entire image to be displayed on a screen.

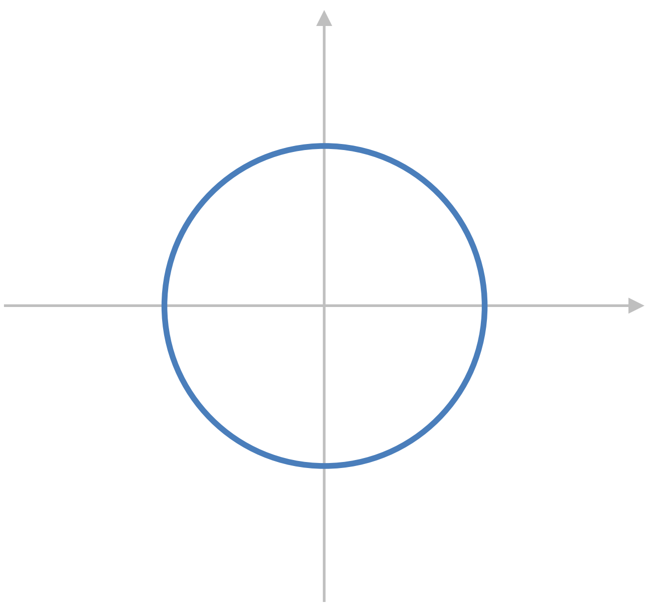
Each element in a framebuffer is called a **pixel** (picture element).

The screen size (or window size) determines the resolution of the framebuffer.

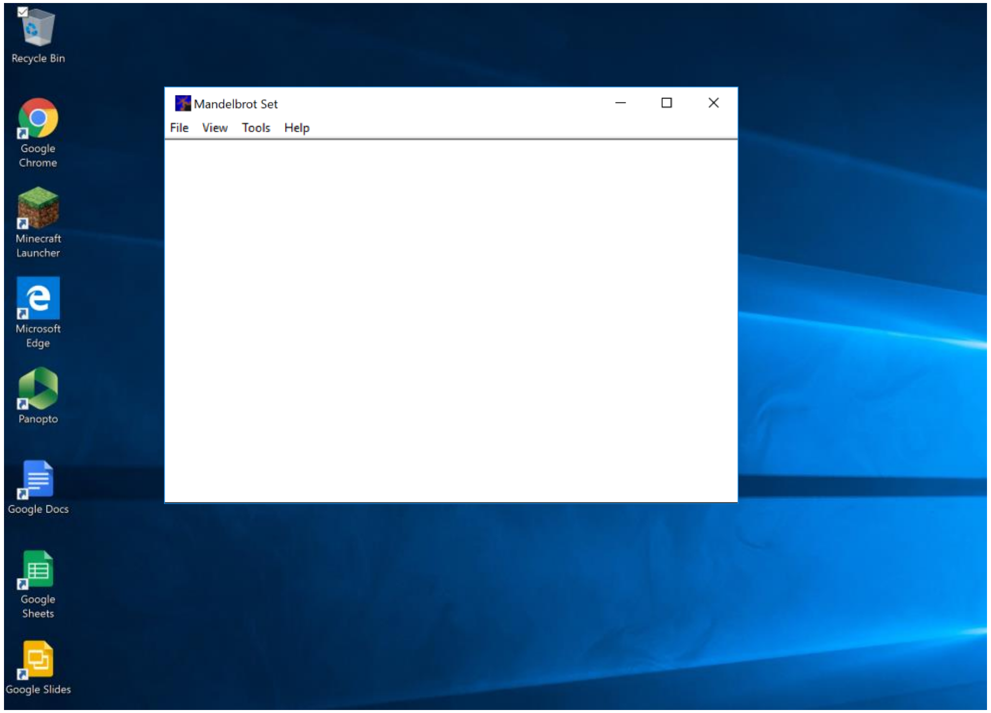


Drawing on a canvas

Consider drawing a unit circle with center located at the origin in a canvas of 800x600 pixels.



World coordinate system

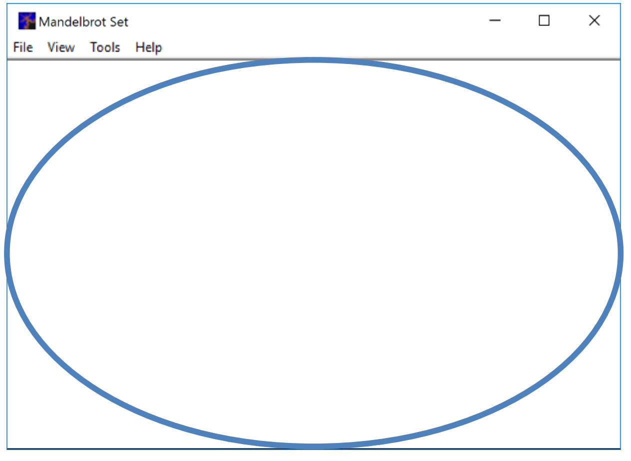


The region for drawing the image is called the viewport in OpenGL

Drawing on a canvas

Which part of the world will be drawn?

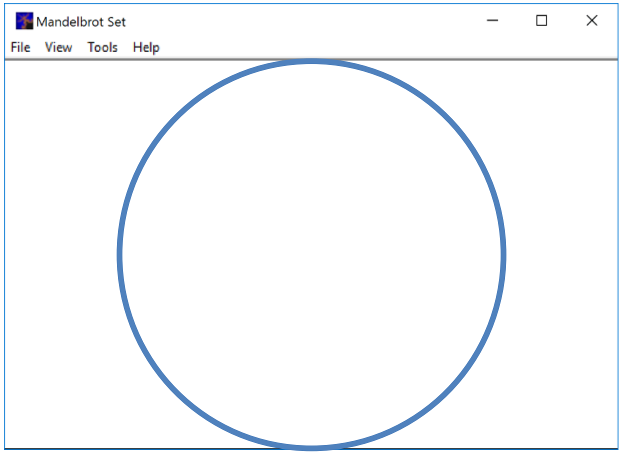
- By defining a screen space in the viewport



(1,1)

(-1,-1)

Mapping from world coordinates to screen coordinates

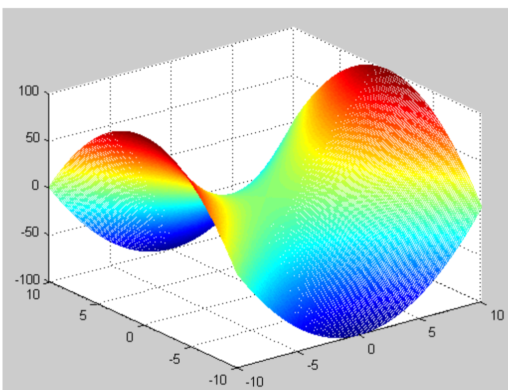


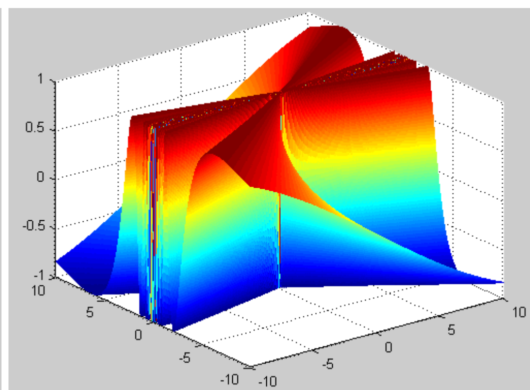
How to maintain the aspect ratio of the circle?

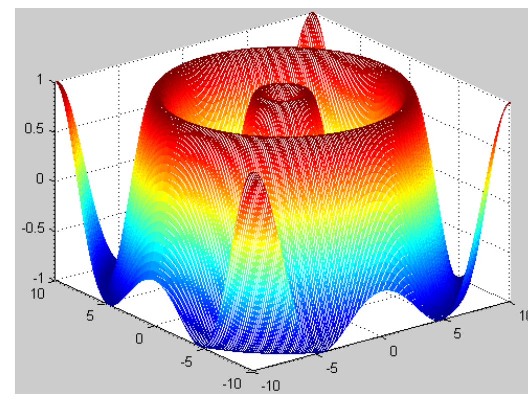
To decide the color of each pixel, you need a correspondence of the world coordinates and the screen coordinates of the pixels

Graphics from Formulas

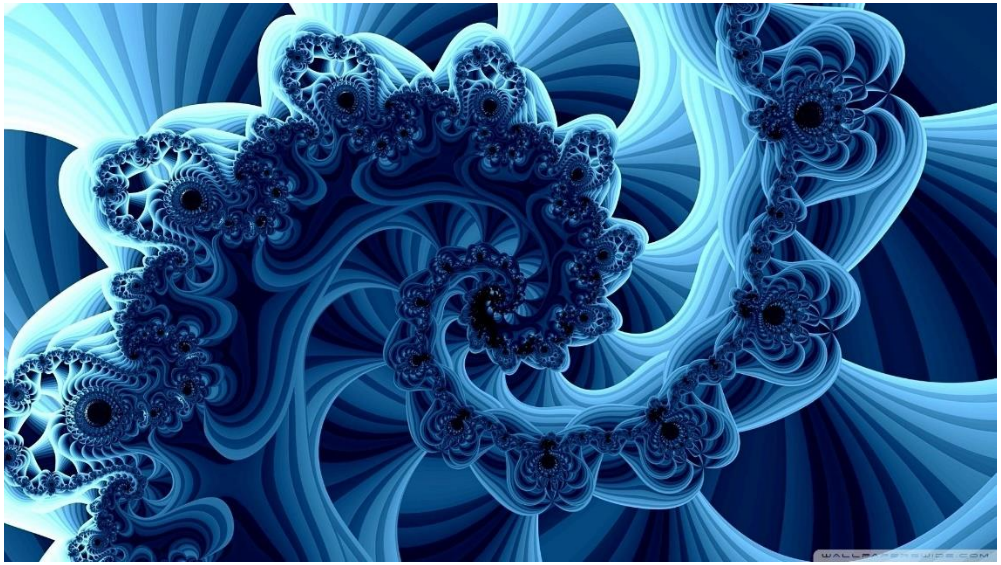
$$z = y^2 - x^2 \qquad z = \sin(\frac{x}{y}) \qquad z = \sin(\sqrt{x^2 + y^2})$$

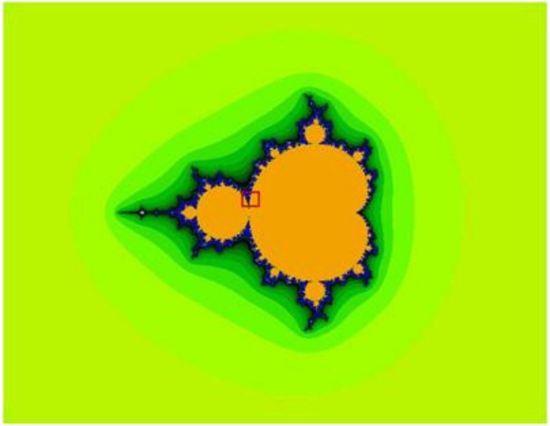


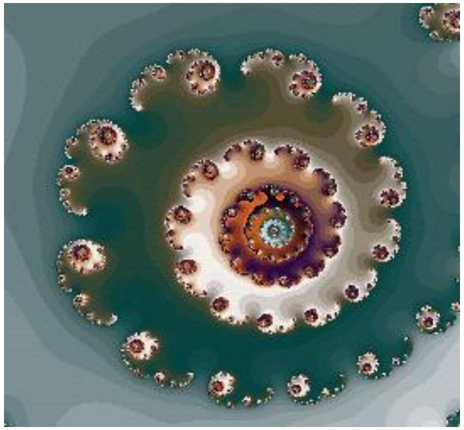


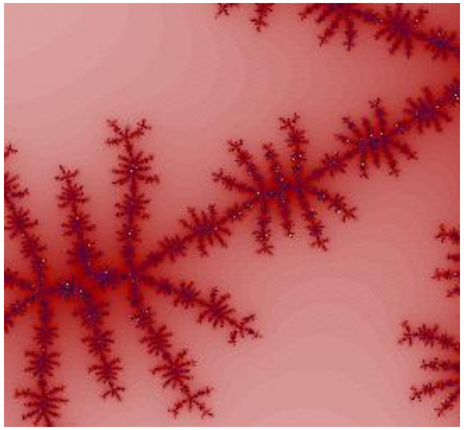


Fractal Rendering





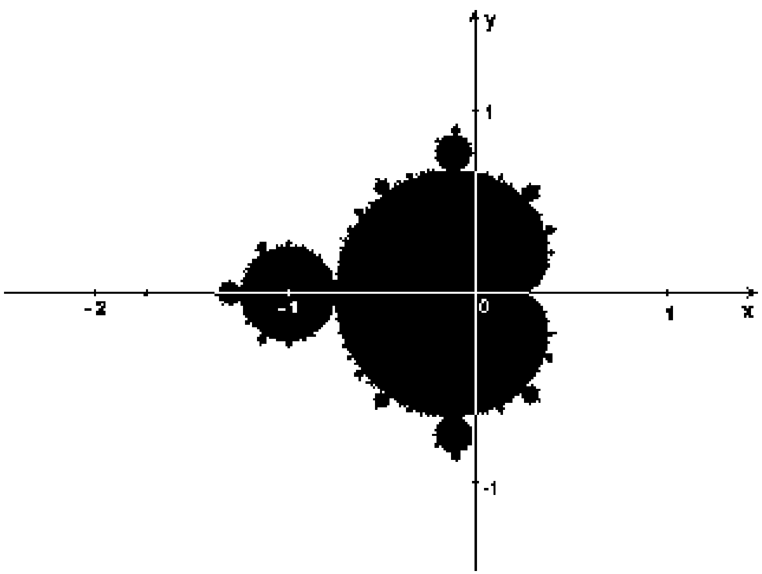




Fractal

Fractal is a set of points that has irregular shapes or boundaries of fractional dimensions.

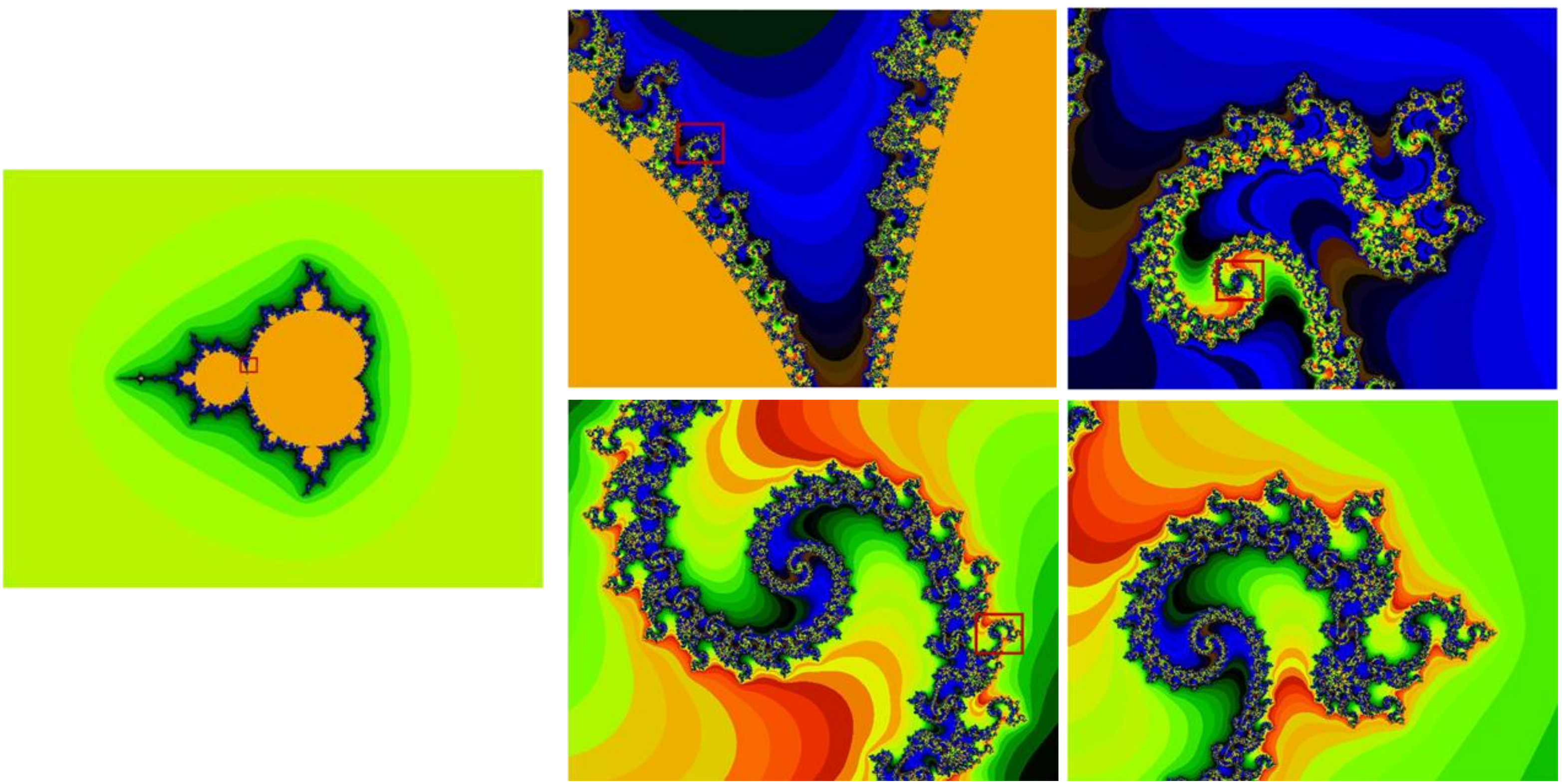
- For example, the Mandelbrot fractal is a fractal defined in the complex plane:



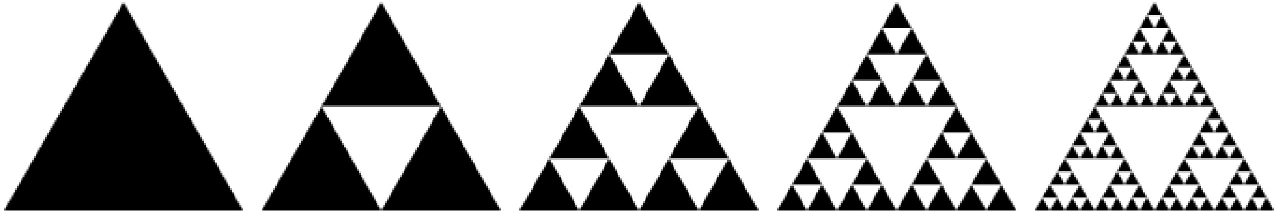
Fractal

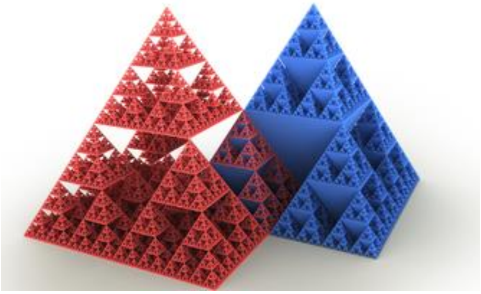
Self-similarity is a characteristic property of fractals.

Fractals may be exactly the same at every scale, or nearly the same at different scales.

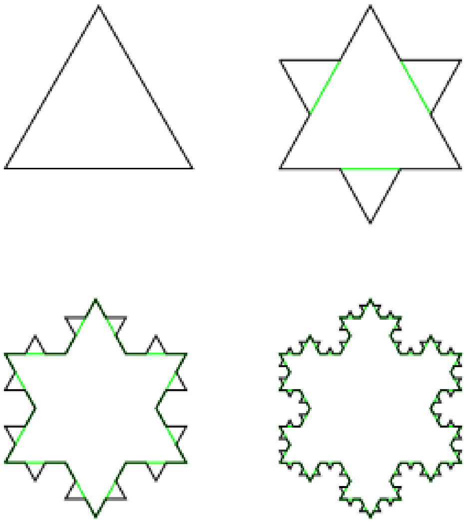


Fractal – More Examples

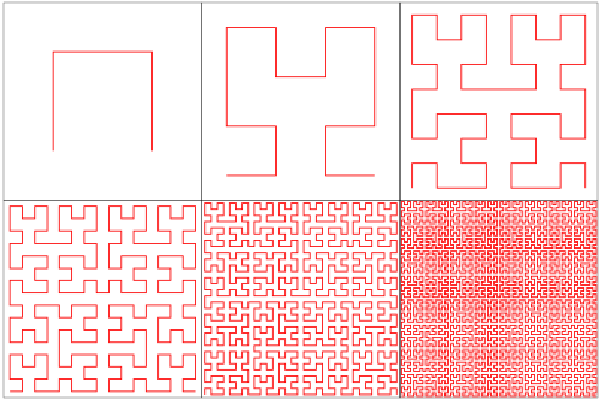




Sierpinski triangle ($d = \log_2 3 \approx 1.58496$)



Koch snowflake
($d = \ln 4/\ln 3 \approx 1.26186$)



Hilbert curve (d = 2)

Romanesco broccoli (d = ~2.7)

Review: Complex number

$$z = \underbrace{a}_{\text{real part}} + \overbrace{i}^{\text{imaginary unit}}\underbrace{b}_{\text{imaginary part}}$$

- Operations:

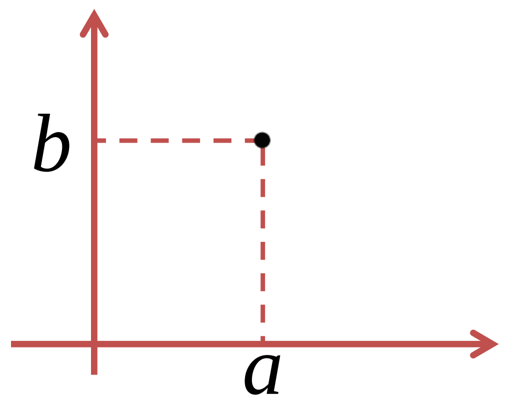
"+": $(a+bi)+(c+di)=(a+c)+(b+d)i$

"−": $(a+bi)-(c+di)=(a-c)+(b-d)i$

"×": $(a+bi)(c+di)=ac+bci+adi+bdi^2=(ac-bd)+(bc+ad)i$

- Magnitude:

$$|z| = \sqrt{a^2 + b^2}$$



Complex plane

What is Mandelbrot Set ?

Mandelbrot Set — the set of all complex numbers c such that z_n is finite as n goes to infinity.

$$M = \{c | z_n \not\to \infty, z_n = z_{n-1}^2 + c, z_0 = 0\}$$

$$c \Rightarrow \{z_0, z_1, z_2, z_3 \cdots\}$$

Example.1

For c =1+0*i, the sequence is
{0, 1, 2, 5, 26,…}

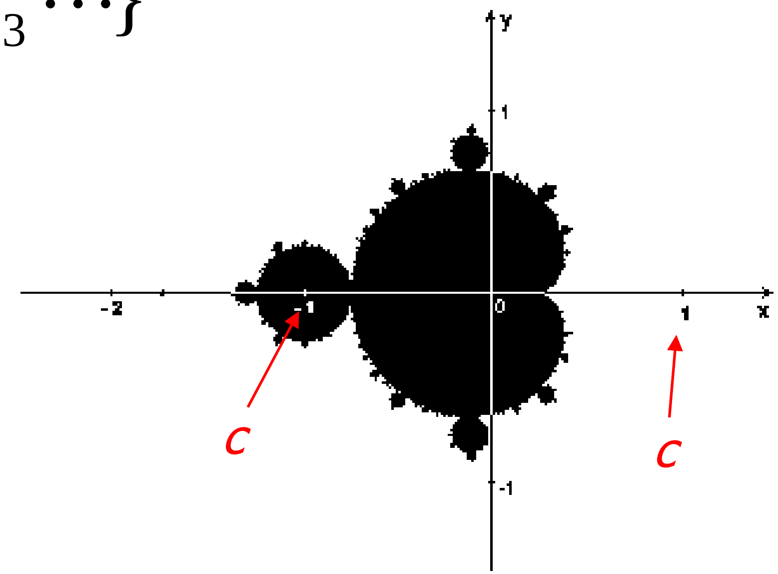
Example.2

For c =-1+0*i, the sequence is
{0, -1, 0, -1, 0,…}

Exercise:

c =-1+i {0, -1+i, -1-i, -1+3*i, -9-5*i, 55+91i…}

c = 1–i {0, 1-i, 1-3*i, 7-7*i, 1+97*i, -9407+193*i…}



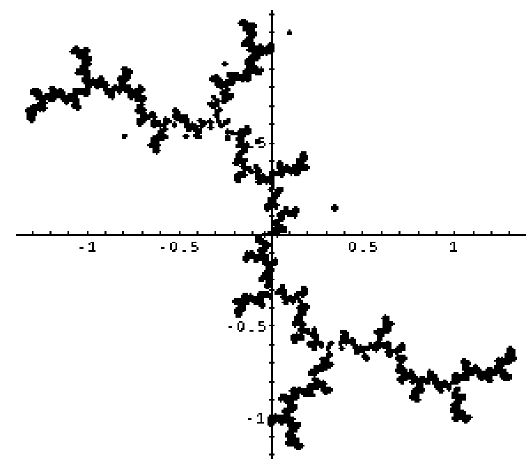
boundary of M (d = 2)

What is Julia Set?

Associate with each (complex) parameter value c, there is a Julia set J_c which is defined as the boundary of the set:

$$A_c = \{z | z_n \not\to \infty, z_n = z_{n-1}^2 + c, z_0 = z\}$$

$$c, z \Rightarrow \{z_0, z_1, z_2, z_3 \cdots\}$$



The Julia set for some fixed *c*

Each Julia Set corresponds to a complex number c

Example

For *c* =1+0*i, z =1+0*i the sequence is {1, 2, 5, 26, 677…}

Mandelbrot Set and Julia Set

Both sets use the same rule for generating a sequence:

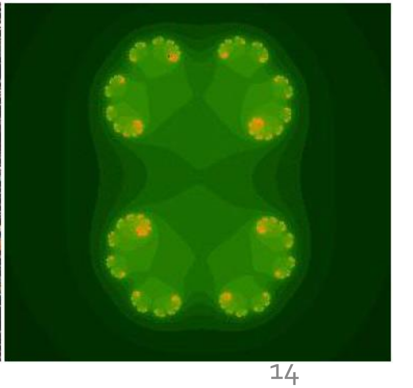
$$z_n = z_{n-1}^2 + c$$

Differed by:

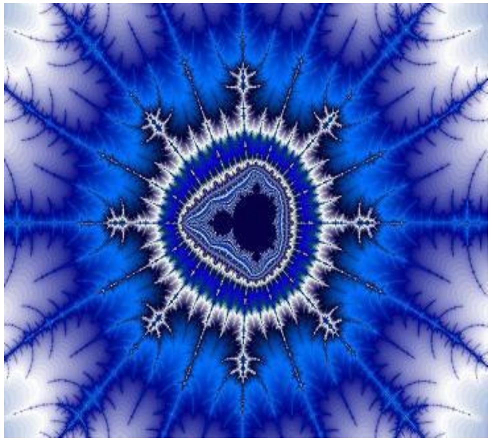
	Mandelbrot	Julia
c	variable	fixed
z_0	fixed (z_0=0)	variable

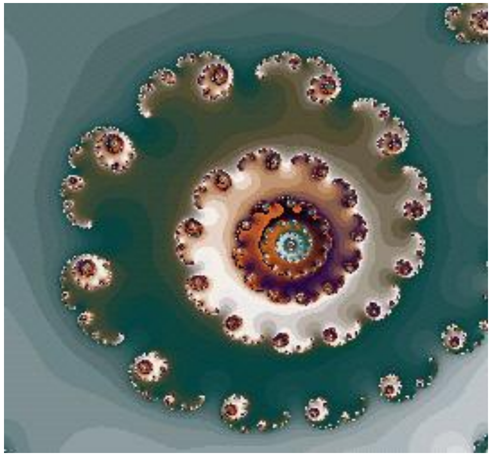
Properties:

- The boundary of M is self-similar
- J_c is connected iff $c \in M$



What do these fractals look like?





How is the rendering or visualization done?

- Use color coding to display the dynamic behavior of the iteration used to generate the fractal

Complex Number Representation

Represent a complex number using *x*, *y* coordinates

$$z_n = z_{n-1}{}^2 + c \quad n = 1,2,3\ldots$$

$$z_{n-1} = x_{n-1} + iy_{n-1}$$

$$z_n = x_n + iy_n$$

$$c = a + ib$$

$$\Rightarrow \begin{cases} x_n = x_{n-1}{}^2 - y_{n-1}{}^2 + a \\ y_n = 2x_{n-1}y_{n-1} + b \end{cases}$$

Computational Issues

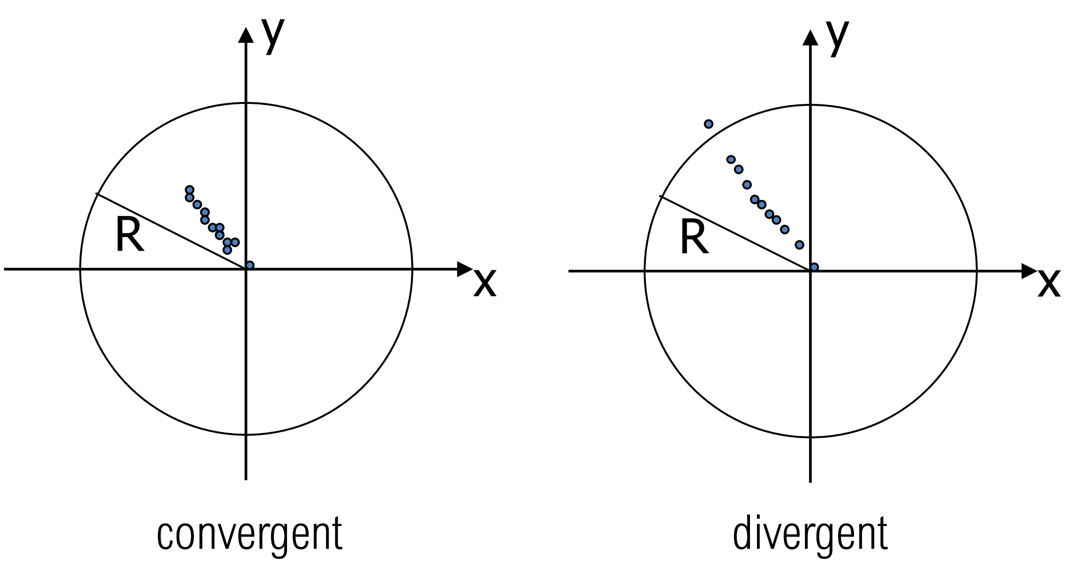
As the definitions of the Mandelbrot set and Julia sets involve ∞, computationally it is impractical to apply these definitions directly.

- In other words, we cannot iterate the function infinite times to see if a point (x_n, y_n) goes to ∞

Determining Divergence

Escape Time:

Definition 1: *Given* $R > 0$*, the* escape time *of a point* z *with respect to* J_c *is the smallest positive integer* k *such that* $|z_k| > R$*, where* $z_n = z_{n-1}^2 + c$*,* $z_0 = z$*.*



Intuitively, the escape time is the number of iterations for an initial point z to get out of a pre-specified range.

Theoretically, the escape time can be any integer from 0 to ∞

The smaller the escape time, the more rapidly the point goes to infinity.

Rendering Procedure

- Define a 2D coordinate system in a window (viewport) so that each pixel is associated with some coordinates (x, y)
- For each complex point $z = (x_0, y_0)$, determine the escape time for z which is clamped off by a fixed integer K (i.e., escape time can only be from 0 to K)
- Assign a color to the pixel corresponding to z depending on its escape time (color coding)

Rendering Procedure

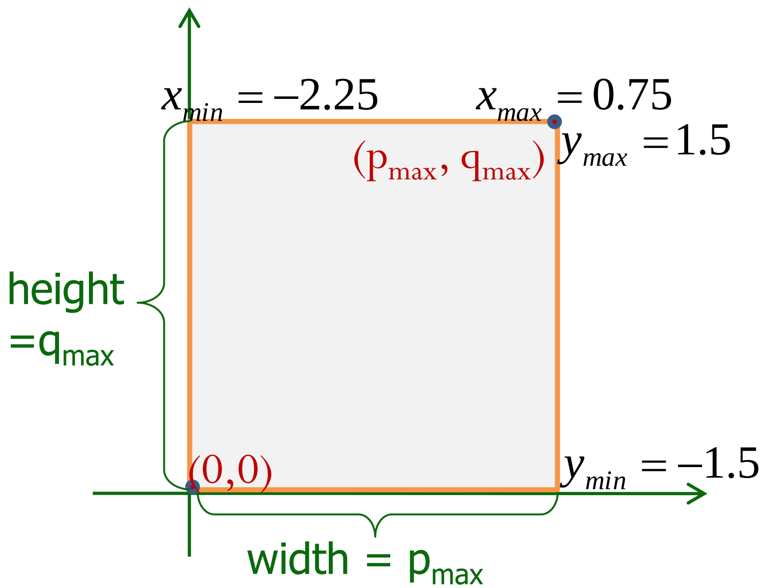
Same for rendering the Mandelbrot set.

- Escape time for Mandelbrot set:

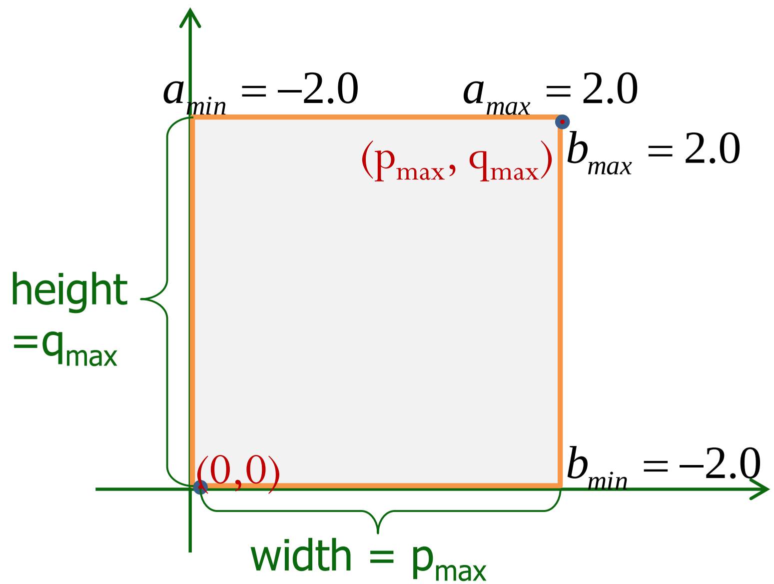
Definition 2: *Given* $R > 0$*, the* escape time *of a point* c *with respect to the Mandelbrot fractal* M *is the smallest positive integer* k *such that* $|z_k| > R$*, where* $z_n = z_{n-1}^2 + c$*,* $z_0 = 0$*.*

Defining the Viewport

Suppose that the display window is $[0, p_{max}]$ x $[0, q_{max}]$ in pixels and let $[x_{min}, x_{max}]$ x $[y_{min}, y_{max}]$ be the area to be displayed in the complex plane.



Mandelbrot set

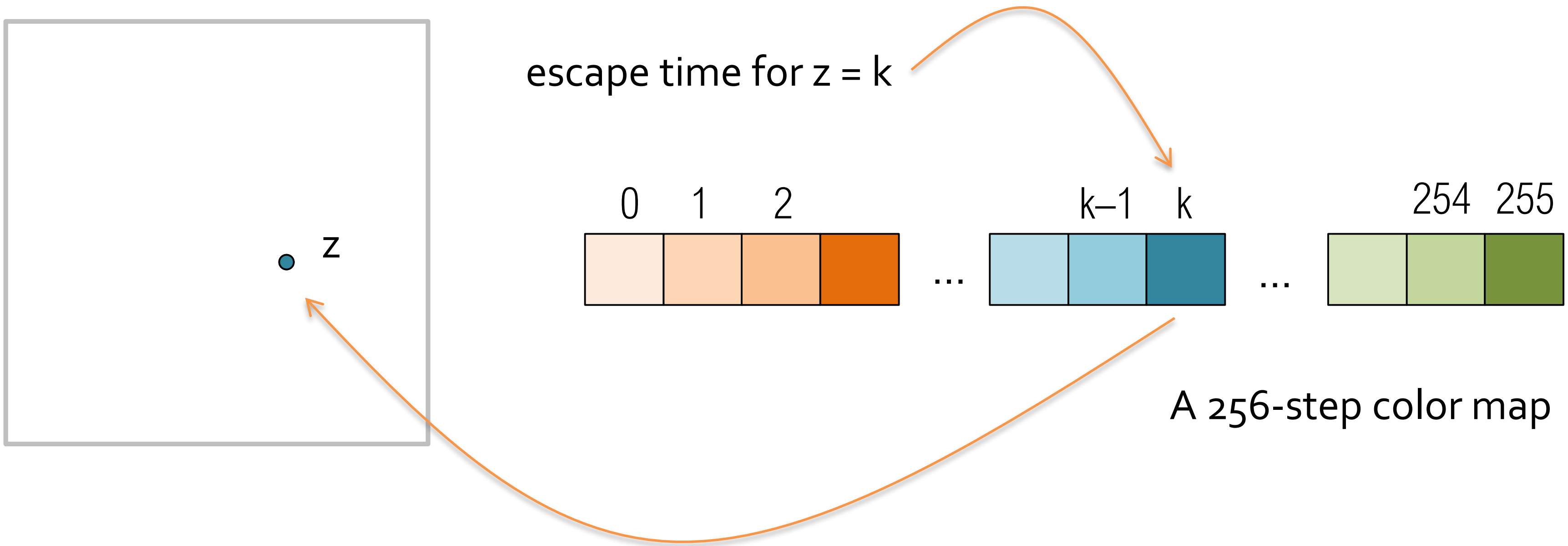


Julia set

Note that there might be display distortion in the complex coordinates, since the scalings in the x- and y-dimension is not 1:1.

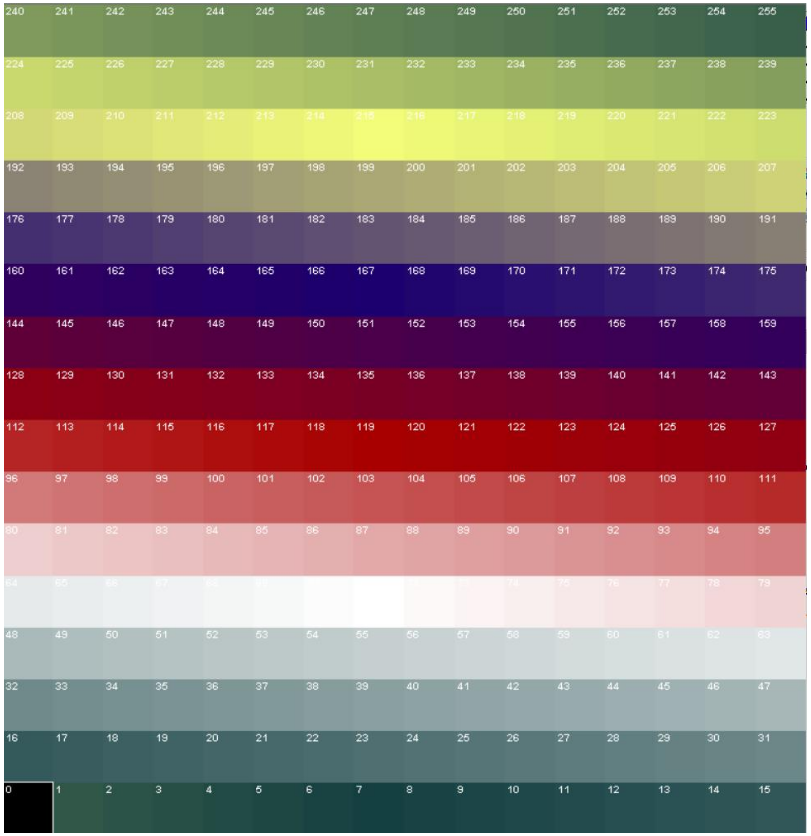
Color Map

The escape time is an integer which is used as an index to a color map to retrieve a color for display.



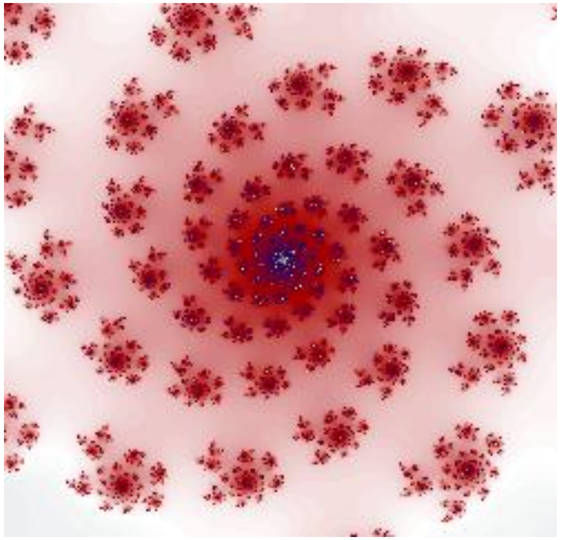
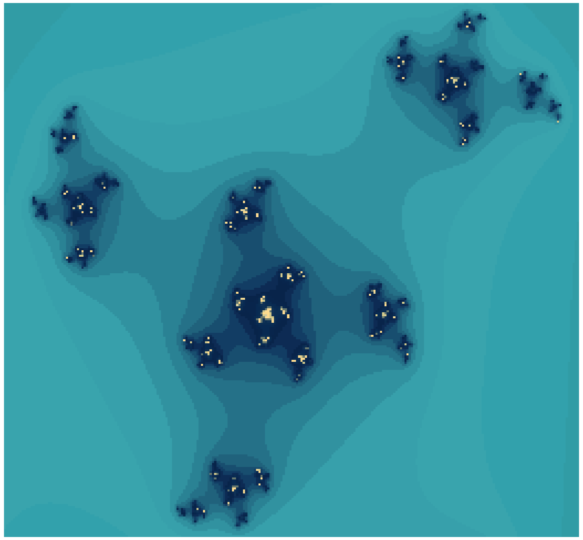
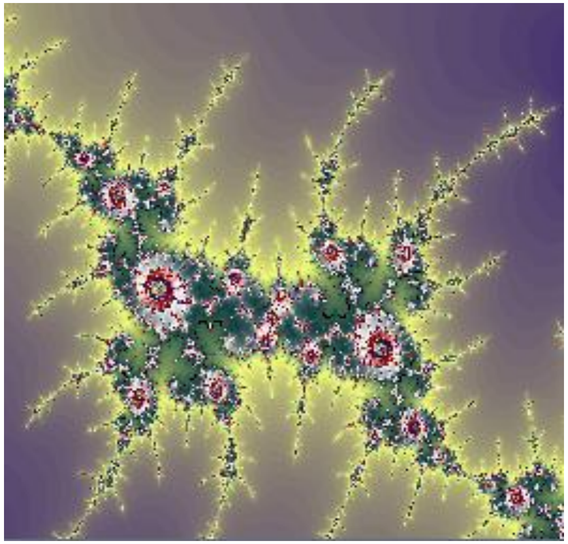
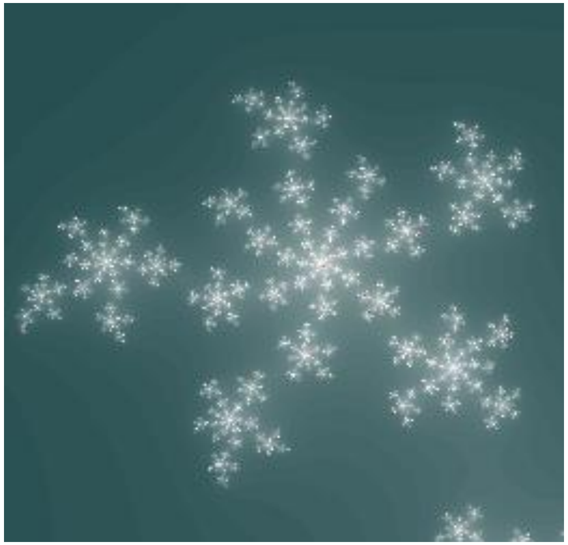
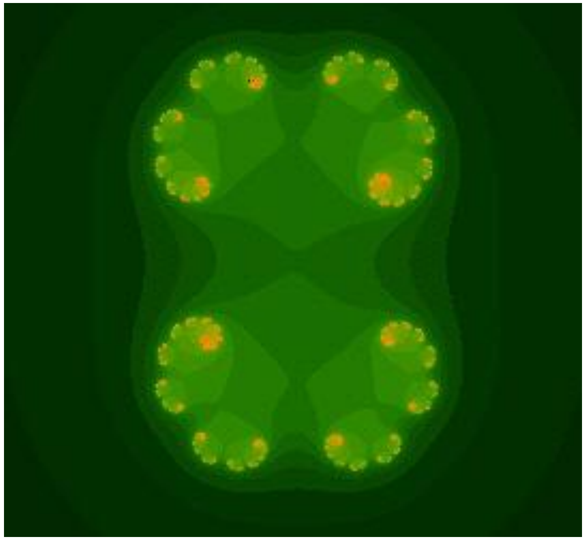
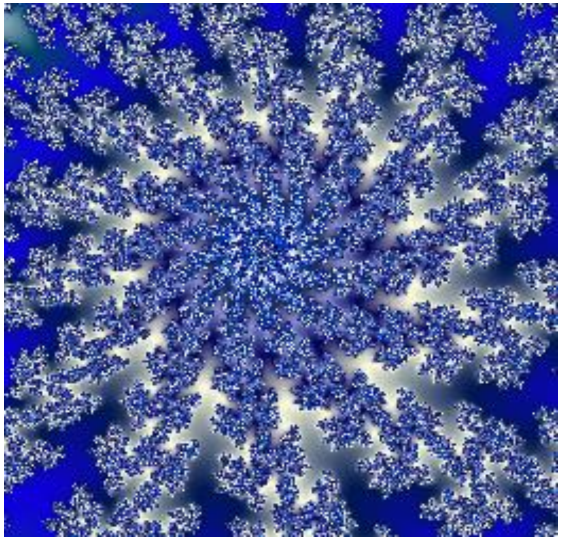
Color Map

The escape time is an integer which is used as an index to a color map to retrieve a color for display.



An example color map used for rendering the fractal sets.

More Rendering Results

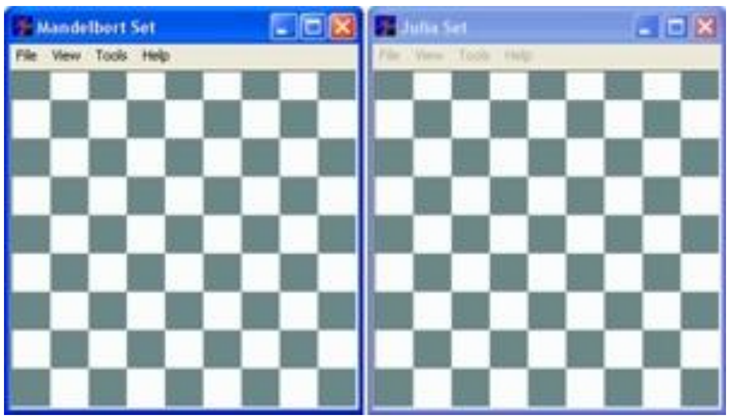
A Warm-up Exercise

You are going to implement two functions to render the Mandelbrot and Julia sets.

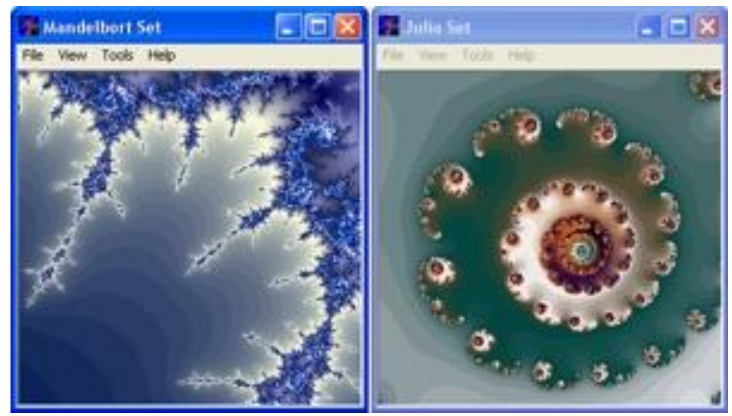
A template project is given to you

- Windows platform project based on based on MS Foundation Class (MFC). Need Visual Studio 2019 Community, a free IDE for Visual C++, to compile.
- Download from the course webpage.
- Double-click the file Fractals_2019.sln to open the project.
- A sample program Fractals.exe in the folder "Fractals_solution".
- Template includes:
    - An interface with functions like resize, zoom in/out, select c, color map import/ edit/ export and file open/save.
    - OpenGL init and projection setup.

About the Template

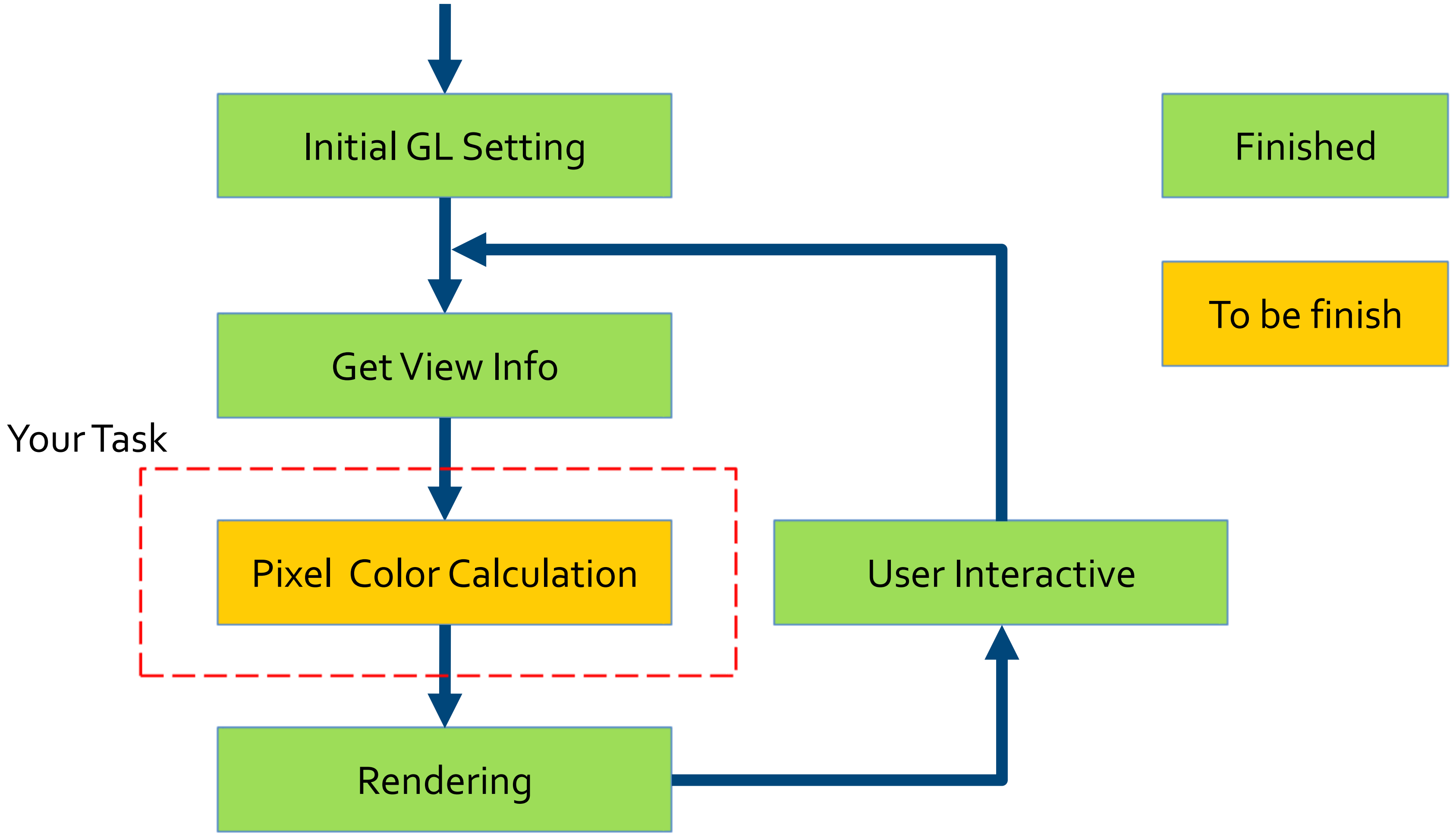


Initial View



Finished View

About the Template

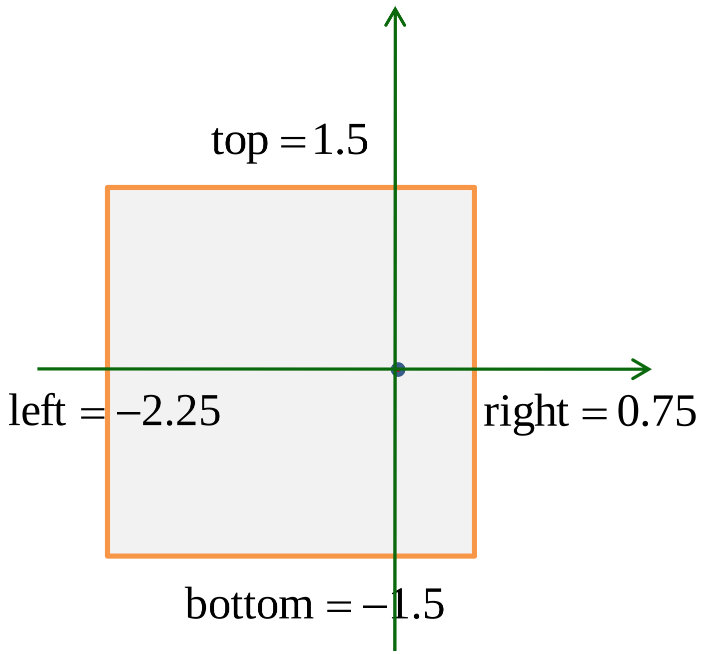


Your Task

Fill in two functions Mandelbrot() and Julia() in **code.cpp.**

void **Mandelbrot**(double left, double right, double bottom, double top, int winwidth, int winheight, unsigned char *map);

- **left, right, bottom** and **top**
  - display region of the complex plane
- **winwidth**, **winheight**
  - dimension of the window.



Your Task

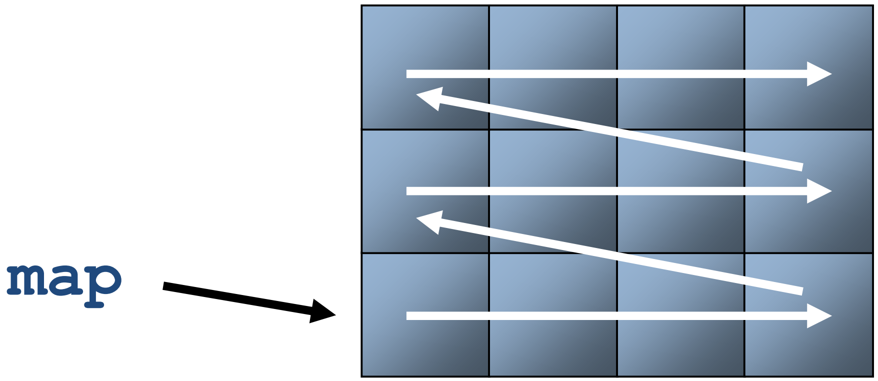
“map” here is an array representing pixels of the fractals image, **from left to right, bottom to top**.

Using color index mode, with the range from 0 to 255.

Each value takes one byte and represents a pixel's color.

It is already allocated by the template.

There are winwidth * winheight bytes in the array.



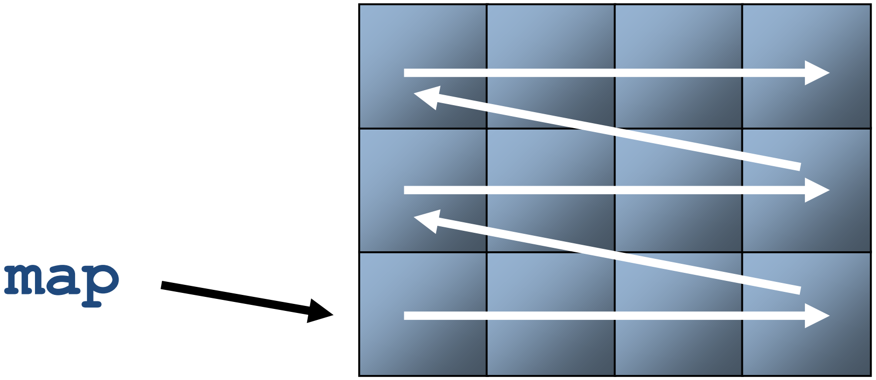
Note! Assign color to Pixel (i, j)

Color index ranges from 0 to 255!

Suppose the escape time for the complex number presented by pixel (i, j) is n, then it should be

map[i+ j*winwidth] = n%256

MOD operation



Procedure: Mandelbrot_Set

begin

For all pixels (p, q) in the window, **do**

begin

Step (1): set

$$\begin{aligned}
\Delta a &= (a_{max} - a_{min})/p_{max}, \quad \Delta b = (b_{max} - b_{min})/q_{max}, \\
a &= a_{min} + p * \Delta a, \quad b = b_{min} + q * \Delta b, && \longleftarrow C=a+bi \\
x_0 &= 0, \quad y_0 = 0, && \longleftarrow z_0=0 \\
n &= 1; && \longleftarrow \text{escape time}
\end{aligned}$$

Step (2): set $x_n = x_{n-1}^2 - y_{n-1}^2 + a$, $y_n = 2x_{n-1}y_{n-1} + b$;

Step (3):

(i) if $n = K$, color the pixel (p, q) with color 0;

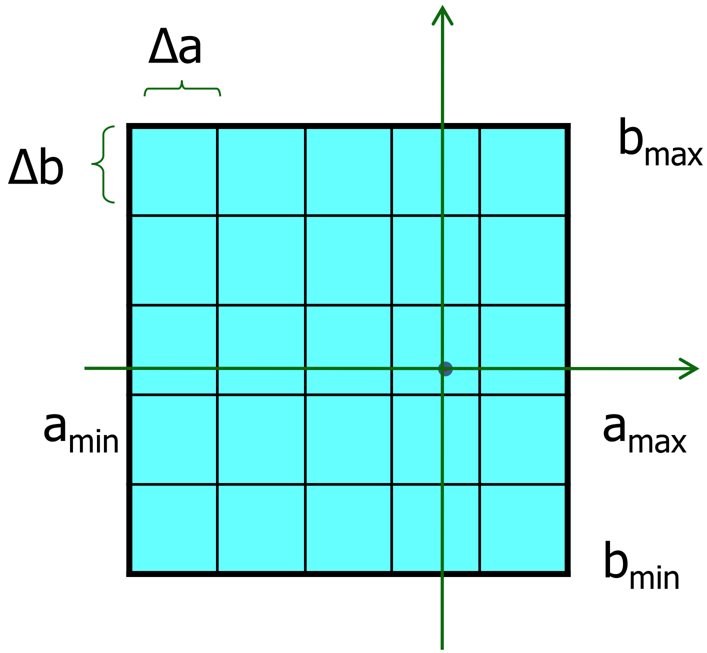
(ii) if $x_n^2 + y_n^2 < R$ and $n < K$, set $n = n + 1$, go back to Step (2);

(iii) if $x_n^2 + y_n^2 \geq R$, color the pixel (p, q) with color n;

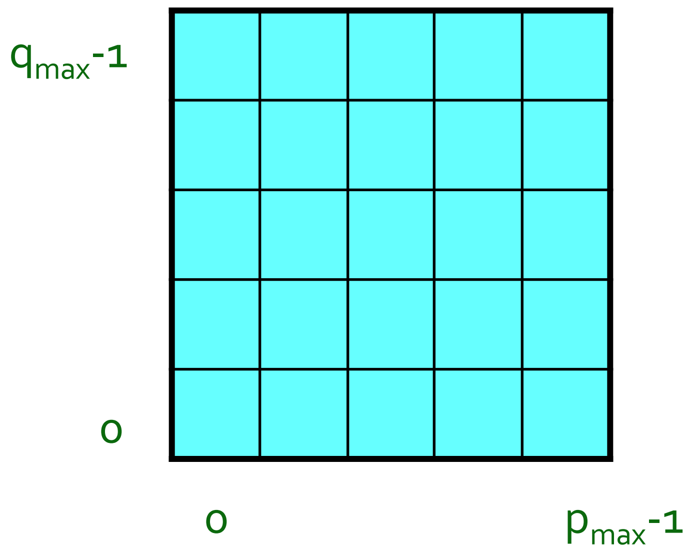
end

end

Complex plane



Window



Your Task

void **Julia**(double left, double right, double bottom, double top, double a, double b, int winwidth, int winheight, unsigned char *map);

- **Left, right, bottom, top, winwidth, winheight, map,** same as Mandelbrot Set.

- **a** , **b** -- in Julia(), define the complex number c=a+ib.

Procedure: Julia_Set (with $c = a + ib$)

begin

For all pixels (p, q) in the window, **do**

begin

Step (1): set

$$\begin{aligned}\Delta x &= (x_{max} - x_{min})/p_{max}, \quad \Delta y = (y_{max} - y_{min})/q_{max}, \\ x_0 &= x_{min} + p * \Delta x, \quad y_0 = y_{min} + q * \Delta y, \\ n &= 1;\end{aligned}$$

Step (2): set $x_n = x_{n-1}^2 - y_{n-1}^2 + a$, $y_n = 2x_{n-1}y_{n-1} + b$;

Step (3):

(i) if $n = K$, color the pixel (p, q) with color 0;

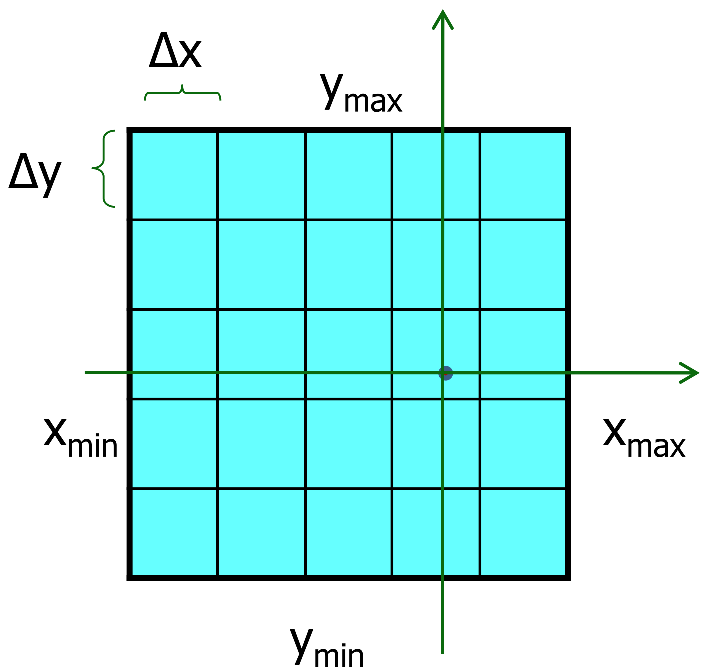
(ii) if $x_n^2 + y_n^2 < R$ and $n < K$, set $n = n + 1$, go back to Step (2);

(iii) if $x_n^2 + y_n^2 \geq R$, color the pixel (p, q) with color n;

end

end

Complex plane



Window