Mandelbrot set with zoom in animation

Nikita Kalinskiy
Software Engineering Bachelors
Innopolis University
Innopolis, Russia
n.kalinskiy@innopolis.ru

***Abstract*—This paper proposes a technique how to create a Mandelbrot fractals set with an option to make a zoom in GIF file to dive deeper into the amazing fractals world. The progam described here is written in Python using NumPy to make parallel computations and Matplotlib to visualize and combine generated images in an animated GIF file.**

***Index Terms*—Mandelbrot set, fractals**

I. Introduction

Almost every person has seen strange psychedelic but so allure pictures of Mandelbrot set geometric patterns that are usually called **fractals** at least once "Fig. 1", "Fig. 2". They look like something created specially for a sci-fi movie or by an artist whose imagination ran away with him. But in fact those geometric patterns were visualized by professor Benua Mandelbrot in 1975 when he used a computer to calculate them using formula that had been determined by Pierre Fatou in 1905. In the next section we will describe the method used to create a zoom in animation of a Mandelbrot set[1].

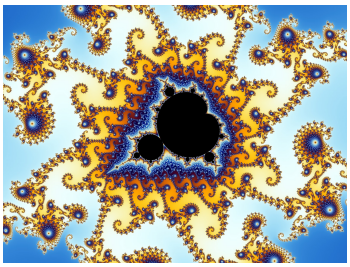

Fig. 1. Mandelbrot set 2.

II. Method description

Let c be some complex number. Consider the sequence of numbers $z0, z1, z2, ...$, which is constructed as follows:

$$z_0 = 0, \qquad z_{k+1} = z_k^2 + c, \qquad k = 0, 1, 2, ... \qquad (1)$$

At each step, we take the previous number, square it and add c. Depending on the value of c, the sequence of numbers $\{z_k\}$ may be limited or unlimited. If it is limited, we say that c belongs to the Mandelbrot set M.

Since the number c is complex, it has real and imaginary parts. Each complex number is defined by a point on the Cartesian plane: the horizontal part is the real part, and the imaginary part is the vertical one. Thus, the set M is a set on the real plane.

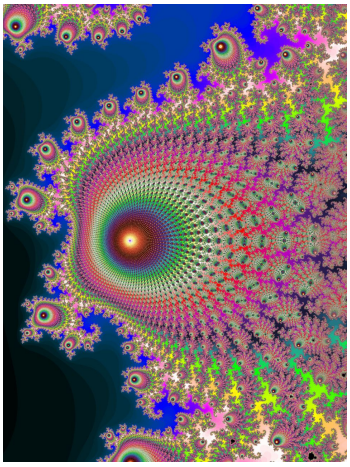

Fig. 2. Mandelbrot set 2.

As we want to create a zoom in animation, we can not use default for loop to create a bunch of pictures from M since it will take hours to generate them. That is why NumPy library is used: instead of doing nested loops, we will create a two-dimensional array, which will contain all the values of c that we want to process, and we will calculate the sequence $\{zk\}$ for each of them in parallel:

```
def mandelbrot(rmin, rmax, rpoints, imin, imax,
               ipoints, max_iterations=200,
               infinity_border=10):
    image = np.zeros((rpoints, ipoints))
    r, i = np.mgrid[rmin:rmax:(rpoints*1j),
                    imin:imax:(ipoints*1j)]
    c = r + 1j*i
    z = np.zeros_like(c)
    for k in range(max_iterations):
        z = z**2 + c
        mask = (np.abs(z) > infinity_border)
               & (image == 0)
        image[mask] = k
        z[mask] = np.nan
    return -image.T
```

In the code above $c = r+i$, where r is real part in an interval of $[rmin, rmax]$ and i is an imaginary part in $[imin, imax]$. $Infinityborder$ is a limit and if we reach it we consider that

we are moving towards infinity and stop computing z.

To visualize the result, we will use *matplotlib* and create and figure from the image:

```
plt.figure(figsize=(10, 10))
image = mandelbrot(-2.5, 1.5, 1000,
                   -2, 2, 1000)
plt.xticks([])
plt.yticks([])
plt.imshow(image, cmap='flag',
           interpolation='none')
```

, which gives us a figure like "Fig. 3" or "Fig. 4" if we set r and i intervals to $[-0.74877, -0.74872]$ and $[0.065103, 0.065053]$ correspondingly and $max_iterations$ to 2048.

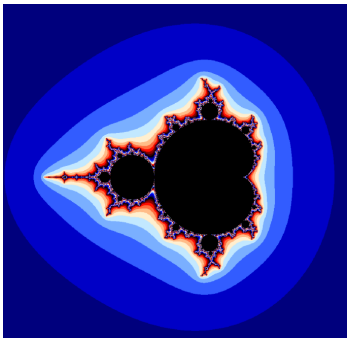

Fig. 3. Generated picture of Mandelbrot set.

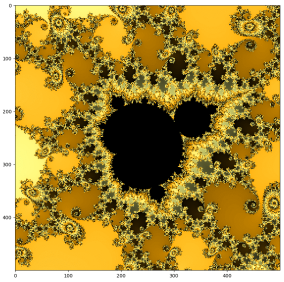

Fig. 4. Another generated picture of Mandelbrot set.

Finally, to create an animation we will define two more functions that are passed to a $matplotlib.animation.FuncAnimation()$ function. The first one is just an *init* function to add an empty image to the beginning of the images cache. The second is for creating a fractal image with a scaled factor that implements our zoom functionality:

```
from matplotlib import animation, rc
rc('animation', html='html5')

images = []
max_frames = 300
max_zoom = 300

def init():
    return plt.gca()

def animate(i):
    if i > max_frames // 2:
        # fetching images from cache
        plt.imshow(images[max_frames//2-i],
                   cmap='flag')
        return

    r_center = -0.793191078177363
    i_center = 0.16093721735804
    zoom = (i/max_frames*2)**3*max_zoom+1
    scalefactor = 1 / zoom
    rmin_ = (rmin - r_center) *
                scalefactor + r_center
    imin_ = (imin - i_center) *
                scalefactor + i_center
    rmax_ = (rmax - r_center) *
                scalefactor + r_center
    rmax_ = (imax - i_center) *
                scalefactor + i_center
    image = mandelbrot(rmin_, rmax_, 500,
                       imin_, imax_, 500)
    plt.imshow(image, cmap='flag')
    images.append(image)

    return plt.gca()


animation.FuncAnimation(fig, animate,
                        init_func=init,
                        frames=max_frames,
                        interval=150)

# saving our animation to GIF file
# with a help of ImageMagick tool
anim.save('mandelbrot.gif',
          writer='imagemagick')
```

III. Conclusions

To conclude, it is so amazing that such beautiful and allure geometric figures can be expressed by a single and simple math equation. And that anyone nowadays can create an almost infinite(limited only by their curiosity and time) sequence of fractals that can draw you in an psychedelic rabbit hole of a like nothing on earth art.

References

[1] Wikipedia contributors. (2019, October 25). Mandelbrot set. In Wikipedia, The Free Encyclopedia. Retrieved 10:54, November 10, 2019

[2] My source repo: https://github.com/KN878/MandelbrotZoom