Capturing the Complexity of the Julia Set

Colin I. Kim

Groton School, Groton, MA; ckim23@groton.org

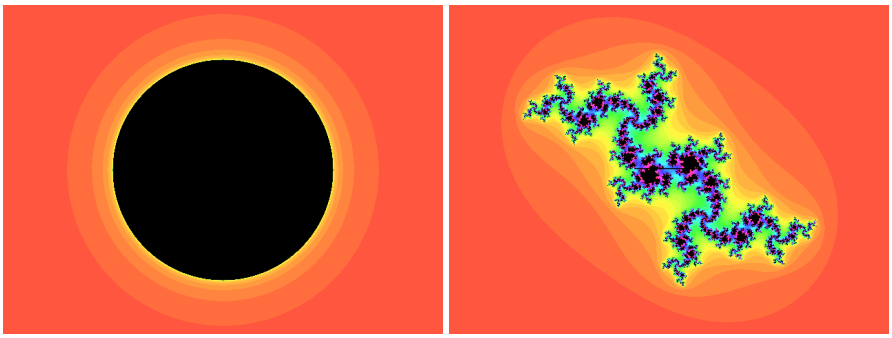
Abstract

The Julia set has the potential to generate beautifully elaborate images from simple functions. Yet, two-dimensional computer-generated graphics are only able to depict such complexity through animation. I attempt to capture the essential aesthetic intricacies of the Julia set in a three-dimensional perspective, modeling an art piece that organically morphs according to the Julia set while also acting as a multifunctional object.

Introduction

Fractals are an area of mathematics that exhibit unique natural complexity and aesthetic appeal, leading to a flurry of opportunities for artistic reinterpretations. As Benoit Mandelbrot, who studied the Mandelbrot set and pioneered fractal geometry, put it in 1989, "simple rules and short algorithms can generate structures of great richness" [3].

The Julia set of a function is the collection of points on the complex plane that form the boundary between the points that converge to infinity under the function and the points that do not. Let us take the function $f(z) = z^2$. It is easy to see that a point that has a distance to the origin less than 1 will never converge to infinity under $f(z)$: if $z = a + bi$, its distance to the origin is $\sqrt{a^2 + b^2}$, and $f(z)$'s distance to the origin is $a^2 + b^2$. It follows that if the distance between z and the origin is less than 1, under iterations of $f(z)$ the distance will continue to decrease. On the other hand, if the distance between z and the origin is greater than 1, under iterations of $f(z)$ the point will converge to infinity. In this case, our Julia set consists of the points that are exactly 1 apart from the origin, or a circle around the origin with a radius of 1. This seems to be a seemingly straightforward example, but similar-looking functions are able to produce wildly various Julia sets. Figure 1 shows graphic depictions of the Julia sets of two different functions: one is based on our previously discussed $f(z) = z^2$, whilst the other is based on a function that simply adds a complex constant to z^2. However, the two images look nothing alike. In fact, adding any complex c value to $f(z) = z^2 + c$ has the potential to completely change the Julia set's appearance [1].



(a) $f(z) = z^2$ **(b)** $f(z) = z^2 - 0.23 + 0.74i$

Figure 1: *Julia sets colored according to degree of escape, made by generator [4]*

Despite their unique look and intricate patterns, artistic interpretations of the sets seem to be limited. Including the two images shown above, most visual interpretations of the Julia set only vary in their use of color (in the examples above, a point's color is determined by how many iterations it needs to escape, thereby creating a gradient around the set) [5]. Some depictions exhibit an animated element, which allows the pieces to truly capture the dynamic nature and unpredictability of the Julia set by showing how it shifts according to values of c. An interesting idea was used by Irving and Segerman, who tweened along time to generate models of geometric fractal curves [2]. I decided to approach my piece in the perspective of adding in a new dimension, depth, by compiling Julia sets along a parameter to create a three-dimensional model.

Implementation

I began to construct my model first by writing a Python program that generated the filled Julia set for a given $f(z) = z^2 + c$. For this project, I used John Zelle's Graphics library, an open-source library for constructing simple graphic images. I translated the complex plane, with real and imaginary values ranging from -2 to 2, into a window 500 pixels long and wide. For each and every point in this plane, I computed iterations of $f(z)$, determining what color that point should appear as based on the point's behavior (escaping). Because I wanted my model to be solid, I had to color in both the Julia set itself and all the points inside it: these points were filled black, whilst every other point on the plane was filled white. Below is the completed program.

```
from graphics import *

def iterations(x, y, a, b, re, im, win):
    for i in range(20):
        x1=(x*x)-(y*y)+re
        y1=(2*x*y)+im
        if (x1*x1)+(y1*y1)>4:
            win.plot(a, b, color="white")
            return;
        x=x1
        y=y1
    win.plot(a, b, color="black")

def main():
    re=float(input("input␣real␣part:␣"))
    im=float(input("input␣imaginary␣part:␣"))
    win=GraphWin("Julia␣Set", 500, 500, autoflush=False)
    win.setCoords(0, 500, 500, 0)
    for a in range(0, 500):
        for b in range(0, 500):
            x=-2+(float(a)/125)
            y=2-(float(b)/125)
            iterations(x, y, a, b, re, im, win)
    update()
    win.getMouse()

main()
```

This code allowed me to then extract images of every possible individual Julia set. As an experiment, I decided to fix the imaginary value of c to 0 and vary the real value in intervals of 0.01. Knowing that a larger

number of images was required for the finished model to fully showcase the extent to which the Julia set can change, I decided to take the Julia sets for $c = 0$ to $c = -1$. Using the program, I was able to obtain a png image of each of the black-and-white Julia sets, 100 images in total. Three of these images are illustrated in Figure 2.

(a) *Image for* $c = 0$ **(b)** *Image for* $c = -0.5$ **(c)** *Image for* $c = -1$

Figure 2: *Computer-generated Julia set images for varying c values*

To enable their scaling and modifying in 3D-modeling software, I used Inkscape to trace the bitmaps of each image and exported them as svg (scalar vector) images. Unlike png and jpg files, or raster graphics, which are composed of a set amount of pixels, svg files are defined by a set of equations that enable them to be zoomed in and out without losing line smoothness and quality. More relevantly to my intentions, this meant that I was able to use them to create 3D objects that I could then freely resize and reshape however I wanted to.

I imported my vector graphics into Fusion 360, a 3D-modeling software, where I extruded them by 1 millimeter each, thus creating thin Julia set "slices" that I then stacked on top of each other. I found out here that my experiment had been successful: while Julia sets are normally difficult to predict, as small changes in c can lead to completely different images, keeping the imaginary component of c at a constant 0 allowed the Julia sets to transition smoothly between each other, creating a final 3D model with nicely curved edges and no jagged appendages. However, while the bottom of the model ($c = 0$) began out as a circle, as the c values decreased the slices' shapes became increasingly chaotic, with more and more complex webs of circles beginning to expand from the middle. Towards the top of the model, thus, I was able to perceive unattractively jagged edges. The problem was that as c diverged more from zero, the degree of variation between two adjacent Julia sets seemed to increase. For example, $c = 0$ and $c = -0.01$ created two images with nearly no perceivable difference whatsoever, whereas $c = -0.99$ and $c = -1$ could easily be distinguished from each other, especially in their intricate edge segments.

Because the shape that the Julia set was approaching in itself was intriguing, I decided to cut the model at a point where the top face was recognizable and the differences between the slices was not too drastic. $c = -0.85$ proved to be the right number; whilst the final image was different enough from the original circle, the sizes of the smaller sections were at a reasonable state (and the variation between slices seemed to grow drastically at around $c = -0.9$). On the Mandelbrot set, a map for all Julia sets, moving between -0.85 and 0 along the real axis allows us to stay within the largest cardioid and bulb, thereby avoiding the chaos that arises as we approach the edges of the bulbs.

My next course of actions was to decide what functional purpose my object was going to serve, as I didn't want my project to simply end at a small sculpture. Looking at the extra circles that lined the edges of the top face, I realized that the model would be able to function as a symmetrical vase that allowed flowers to rest within these circles, thereby creating perfectly organized bouquet arrangements.

Results

I finally 3D-printed four versions of the finished model, shown in Figure 3b. All four were printed with acrylic resin on a Stratasys J750. I first printed three versions of the original solid model with different dimensions, then decided to use the dimensions I preferred in printing the final vase model. After converting the original model into a NURBS model, I was able to cut individual holes into the model using CAD programs, which resulted in the final model in Figure 3a. Due to the Julia set's natural symmetry, the objects are well balanced, allowing them to naturally remain upright, and the proportions of the vase can be digitally adjusted to hold any objects in it. I am excited to explore further modifications, including modular designs and more chaotic versions exploring the further edges of the Mandelbrot set.

(a) *The final vase.* **(b)** *All four prints.*

Figure 3: *The completed prints.*

Acknowledgments

I would like to thank Mr. Gnozzio, who opened my eyes to the coolness of mathematics and helped me throughout this entire process. For my parents, who have supported me in everything whilst living so far away, I am eternally grateful.

References

[1] K. Falconer. *Fractal Geometry: Mathematical Foundations and Applications.* John Wiley & Sons, Ltd, 2003.

[2] G. Irving and H. Segerman. "Developing fractal curves." *Journal of Mathematics and the Arts*, vol. 7, no. 3-4, 2013, pp. 103-121.

[3] B. B. Mandelbrot. "Fractal Geometry: What Is It, and What Does It Do?" *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 423, no. 1864, 1989, pp. 3–16.

[4] *Mandelbrot and Julia Set Generator*. http://gzgracez.github.io/Fractals/

[5] T. V. Papathomas and B. Julesz. "Animation with Fractals from Variations on the Mandelbrot Set." *The Visual Computer*, vol. 3, no. 1, 1987, pp. 23–26.