Hypercomplex Iterative Methods for Fractal Generation and Visualization

Samuel J. Monson

June 9, 2023

Abstract

Fractal generation and rendering has been a topic of interest in the computer graphics field for decades, with numerous techniques developed for exploring the complex and often beautiful world of fractals. In this paper, we focus on the use of quaternion iterative methods for generating fractals, utilizing the unique properties of quaternions to create visually stunning and mathematically intriguing fractals. We explore the definition and properties of quaternions. Using these properties, we define quaternion iterative methods. Finally, We discuss the challenges of rendering fractals in three dimensions, including the use of ray marching and distance estimation.

Contents

1 Introduction **3**
1.1 Fractals 3
1.2 Quaternions 4
1.3 Rendering 5

2 Background **5**
2.1 Iteration 5
2.2 Complex Dynamical Systems 6
2.2.1 The Julia Set 7
2.3 Quaternions 8
2.3.1 Basic Theory 8
2.3.2 Polynomials 9
2.4 Ray Tracing 10

3 Main Results **11**
3.1 Quaternion Julia Set 11
3.2 Ray Tracing Quaternion Fractals 11
3.2.1 Mapping Quaternions to Real Space 12
3.2.2 Ray Marching and Distance Estimation 13

4 Conclusion **14**

1 Introduction

The path towards understanding hypercomplex fractals encompasses various topics that are not typically covered in a standard mathematics education. It includes components of fractals, hypercomplex quaternion numbers, and computer graphics. Although each of these subjects build on existing mathematical principles, they are extensive and could each be studied separately. Fractals, in particular, are a vast topic that spans numerous disciplines. Thus, we will begin by providing a brief overview of each topic.

1.1 Fractals

Fractals, derived from the Latin word “fractus” meaning broken or irregular, are geometric structures characterized by self-similarity. They possess a unique property wherein they exhibit the same pattern regardless of the scale at which they are observed. This recursive nature allows fractals to possess an infinite level of detail, providing an infinite reservoir of complexity within finite boundaries. The study of fractals combines elements of mathematics, physics, computer science, and other disciplines to unravel the fundamental principles that govern their formation and behavior.

The idea of fractals mostly originates from the groundbreaking work of mathematician Benoit Mandelbrot. Mandelbrot introduced the term “fractal” and developed a comprehensive framework for their study. His seminal work, “The Fractal Geometry of Nature,” laid the foundation for understanding the inherent fractal nature of various natural phenomena [Man83]. Since then, researchers have expanded upon Mandelbrot’s ideas, exploring fractals in different contexts and uncovering their presence in a myriad of systems, ranging from the microscopic to the cosmic.

One of the fundamental aspects of fractals lies in their capacity to generate an infinite number of shapes and patterns through simple iterative processes. Fractals often emerge from recursive algorithms that repeatedly apply a set of rules to generate self-similar structures. By manipulating these rules, we can create an array of intricate fractal forms, each exhibiting its unique properties and characteristics. This ability to generate complexity from simplicity has profound implications for fields such as computer graphics and visualization, where fractals serve as valuable tools for generating realistic landscapes, simulating natural phenomena, and modeling complex structures.

Moreover, fractals play a crucial role in understanding the dynamics and behavior of complex systems. Natural processes, such as the growth of biological organisms, the flow of fluids, and the formation of geological structures, often exhibit fractal characteristics. Fractals have also proven valuable in understanding phenomena such as chaos theory, network theory, and the behavior of financial markets, providing a fresh perspective on complex systems that defy traditional linear analysis. [Gle88, BDM$^+$88]

1.2 Quaternions

Quaternions are a 4-dimensional numbering system that extends the complex number. Developed by the Irish mathematician William Rowan Hamilton in the mid-19th century [Ham66], quaternions have since found some applications in various scientific and engineering disciplines, but their most extensive use is in computer graphics where they provide a powerful framework for representing and manipulating rotations in three-dimensional space.

Quaternions consist of a real part and three imaginary parts, adding an additional two imaginary components to complex numbers. Quaternions retain most of the algebraic properties of complex numbers, and like complex numbers, they also retain most of the properties of real numbers. However, one intriguing difference between quaternions and complex numbers is that quaternions exhibit non-commutative multiplication, meaning that the order of their multiplication matters.

For our purposes quaternions offer a great opportunity to visualize complex fractals in 3-dimensions. They naturally extend the complex numbers and enable us to use similar fractal formulas that we use for complex fractals. However, we cannot prove some theorems that apply to complex fractals for quaternion fractals due to the loss of commutativity. This challenges the rigorous definition of quaternion fractals compared to their complex counterparts.

1.3 Rendering

Moving from the 2-dimensional complex fractals to producing 3-dimensional renders of hypercomplex fractals introduces a lot of challenges. With complex fractals, it is sufficient to calculate an iteration for each screen pixel. While this posed difficulties for computers in the 1980s, modern machines can perform this operation in microseconds if coded appropriately. However, when it comes to hypercomplex fractals in 3-dimensions, things get more difficult. In general, 3-dimensional rendering is accomplished through the rasterization process, which projects shapes geometrically onto a viewport plane. Unfortunately, fractals are implicit surfaces, which makes geometric transformation impossible.

Instead of rasterization, we use a rendering technique called ray tracing. Ray tracing simulates the behavior of light as it interacts with objects in a virtual environment. This process follows rays of light from a virtual camera that interact with surfaces through reflection, refraction, and absorption. Ray tracing has been used for many years in cinema to create photorealistic imagery, but recent advances in computing power have made real-time ray tracing possible, leading to a craze of ray tracing in video games. What once took hours on the greatest graphics computing hardware of 1996 can now be accomplished on a personal computer many times faster thanks to these advances.

2 Background

2.1 Iteration

The basic technique for generating fractals is iteration, which involves repeatedly applying a function to a point in space and observing the resulting trajectories. The simplest way to define iteration is with function composition. Given some function f, the k-th iteration of f is f composed k times. Large values of k can become tedious to write, thus to simplify notation we can define iteration as an operation.

Definition 2.1 (Function Iteration Operation)
For some function f and all $k \in \mathbb{Z}^+$,

$f^0 := \mathbf{I}$

$f^{k+1} := f \circ f^k$

For our operation it is also helpful to define our 0th value as the input of our function. Thus for the function $f(x) = x + 1$ and input of $x = 1$ will result in,

$$\begin{aligned}
f^0(1) &= 1 \\
f^1(1) &= 1 + 1 = 2 \\
f^2(1) &= (1 + 1) + 1 = 3 \\
f^3(1) &= ((1 + 1) + 1) + 1 = 4 \\
&\vdots
\end{aligned}$$

2.2 Complex Dynamical Systems

We can generate dynamical systems by using iteration. One interesting case is the quadratic function

$$f(z) = z^2 + c \tag{2.1}$$

where z is some iterative variable and c some fixed constant.

This example produces some interesting behavior. Namely, for most values of z there is some finite $n \in \mathbb{Z}^+$ values in the set $\mathcal{P} = \{f^m(z) : 0 \leq m < \infty\}$. We call this set the *cycle* of the point z. The value of n is called the *period* of our cycle. For example with the equation

$$f(z) = z^2 - 1 \tag{2.2}$$

an initial point of $f(0)$ has a period of 2, since

$$\begin{aligned} f^0(0) &= 0^2 - 1 = -1 \\ f^1(0) &= (-1)^2 - 1 = 0 \\ f^2(0) &= 0^2 - 1 = -1 \\ f^3(0) &= (-1)^2 - 1 = 0 \\ &\vdots \end{aligned} \tag{2.3}$$

A periodic point with a period of 1 is called a *fixed point*. Points whose orbits become cycles are called *preperiodic* and points that do not start in a cycle but eventually enter one are called *strictly preperiodic*. [DKS02] We can classify cycles using eigenvalues given by the derivative of the n-th point in the cycle. For a given eigenvalue, λ, we give the cycle the following classification:

$$\begin{cases} \lambda = 0 & \text{superattractive} \\ \lambda < 1 & \text{attractive} \\ \lambda = 1 & \text{neutral} \\ \lambda > 1 & \text{repelling} \end{cases}$$

Attractive cycles end up drawing in many preperiodic and strictly preperiodic points. We can define the *basis of attraction*, $A_c(z)$ as the set of all points that approach the cycle containing the periodic point z. Thus for any point z_0,

$$A_c(z) = \left\{ z_0 : f^k(z_0) = z \text{ for some } k > 0 \right\} \tag{2.4}$$

For any polynomial function we will find that infinity is an attractive fixed point and thus,

$$A_c(\infty) = \left\{ z_0 : f^k(z_0) \to \infty \text{ as } k \to \infty \right\} \tag{2.5}$$

See [BDM+88] for a visual proof.

2.2.1 The Julia Set

Consider the function $f_c : \mathbb{C} \to \mathbb{C}$; $f_c(z) = z^2 + c$ for some $c \in \mathbb{C}$. Since f_c is polynomial, there exists a set $A_c(\infty)$. The set $A_c(\infty)$

will have a natural boundary between points that are attracted to the period of infinity and points that create their own cycles that do not go to infinity. Since we have at least the fixed points of $z^2 + c = z$, this boundary will always exist. This boundary is known as the *Julia set* of f_c and can be written as $\partial A_c(\infty)$ or J_c.

A third set of interest is known as the *filled-in* Julia set, which we can derive by subtracting the $A_c(\infty)$ from the set of complex numbers. We can denote this set K_c and define it with,

$$K_c = \mathbb{C} \setminus A_c(\infty) = \left\{z_0 \in \mathbb{C} : |f_c^k(z_0)| \text{ is not infinite for all } k\right\}$$

where the $|f_x^k(z_0)|$ denotes the order of the cycle containing z_0. Thus, by our notation,

$$\partial K_c = J_c = \partial A_c(\infty)$$

For most values of c the Julia set, J_c exhibits fractal nature that can change quite drastically as c varies. We can classify filled-in Julia sets based on whether the border, J_c, is one continuous space or a set of infinitely many points. We call the former a *connected* Julia set and the latter a *Cantor* set.

2.3 Quaternions

To create and understand hypercomplex fractals, we require a hypercomplex space to operate within. William Rowan Hamilton, an Irish physicist and mathematician, invented the first hypercomplex space, the quaternion, in 1843 [Ham66]. Other hypercomplex spaces such as the tessarines, coquaternions, biquaternions, and octonions (Cayley Numbers) followed the quaternions. However, this paper will solely focus on quaternions as they provide the simplest and most intuitive extension to complex space.

2.3.1 Basic Theory

We can denote the set of quaternions as $\mathbb{H}$. We can visualize the set $\mathbb{H}$ with the direct product $\mathbb{H} = \mathbb{R} \oplus \mathbb{P}$ where $\mathbb{P}$ is a 3-dimensional Euclidean vector space. In this configuration a quaternion can be written as

$$q = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k} \tag{2.6}$$

for some $w, x, y, z \in \mathbb{R}$ where $q \in \mathbb{H}$ and $\mathbf{i}, \mathbf{j}, \mathbf{k}$ represent linearly independent unit vectors such that under quaternion multiplication

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{i}\mathbf{j}\mathbf{k} = -1 \tag{2.7}$$

From (2.7) we can derive a few other interesting relationships. First that

$$\mathbf{i}\mathbf{j} = \mathbf{k};\ \mathbf{j}\mathbf{k} = \mathbf{i};\ \mathbf{k}\mathbf{i} = \mathbf{j} \tag{2.8}$$

but also

$$\mathbf{j}\mathbf{i} = -\mathbf{k};\ \mathbf{k}\mathbf{j} = -\mathbf{i};\ \mathbf{i}\mathbf{k} = -\mathbf{j} \tag{2.9}$$

The combination (2.8) and (2.9) shows us that quaternion multiplication is noncommutative since we have different definitions for $\mathbf{i}\mathbf{j}$ and $\mathbf{j}\mathbf{i}$.

We can call a quaternion with a non-zero w component and zeroed x, y, z components a *real quaternion* since it is analogous to a real number. Quaternions with a zeroed w component and some non-zero combination of x, y, z components are called *pure quaternions*. Because (2.7) only defines relationships between quaternion units, it is apparent that real quaternions are not affected by their noncommutative nature. In fact, real quaternion can be treated for all intents as real numbers just like real complex numbers.

2.3.2 Polynomials

Because of their noncommutativity, quaternion polynomials cannot be as simply described as other polynomials. Thus, a quaterntion polynomial can be defined as

$$p(z) = \sum_{k=0}^{l} \sum_{i=0}^{m} p_{i0} z p_{i1} z \cdots z p_{ik} \tag{2.10}$$

where $l, m \in \mathbb{N}$, $p_{ij} \in \mathbb{H}$ and z is indeterminate. In this case, l represents the highest degree of our polynomial.

2.4 Ray Tracing

The algorithm for ray tracing involves simulating the paths of rays of light as they interact with surfaces in a scene to generate realistic 2-dimensional images of 3-dimensional objects. A ray is defined as a point in 3-space, known as the *origin*, plus a normalized vector known as the *direction*. The set of rays can be represented with the tuple $\mathcal{R} = (\mathbb{R}^3, \mathbb{P})$.

To begin, the algorithm takes a scene containing the objects we wish to visualize. We define a point known as our *eye* or *camera* and we define a grid, known as our *image plane* some set distance and direction away from the eye. In real terms, the image plane represents a computer display and the eye represents the point of perspective relative to the display.

The next step is to generate rays that originate from the eye point and pass through each pixel on the image plane. These rays are traced through the scene by testing for intersections with the objects in the scene. If an intersection is found, the algorithm calculates the color and shading of the object at that intersection point.

To calculate the color and shading, the algorithm considers the properties of the surface material, such as its reflectivity and transparency, as well as the position and intensity of light sources in the scene. This information is used to determine the color of the object at the intersection point, which is then projected onto the corresponding pixel on the image plane.

The process of generating and tracing rays for each pixel on the image plane is repeated until all pixels have been processed, resulting in a complete image of the scene.

We can formally write this algorithm for tracing the object O where B is a maximum ray length condition and δ is some small amount to increment by:

Algorithm 1: Ray Tracing

```
$e \in \mathbb{R}^3$
$R \leftarrow \{r \in \mathcal{R}$ : origin is $e$ and ray intersects image plane$\}$
for $r \in R$ do
    $p_0 \leftarrow$ point of intersection between $r$ and image plane
    $p \leftarrow p_0$
    while $|e - p| < B$ do
        if $p$ intersects $0$ then
            Calculate surface for $p$ and display at $p_0$
            break
        else
            $p \leftarrow p + \delta$
        end
    end
end
```

3 Main Results

3.1 Quaternion Julia Set

Just like the complex case, Julia sets can be generated by quaternion polynomials. Let $p(z)$ be some quaternion polynomial. Then, for any quaternion q, there exists another quaternion $p(q)$. This means we can preform the iterative operation from definition 2.1 on quaternion polynomial functions.

Since we are able to iterate quaternion polynomials, we can apply the same analysis seen in the section 2.2 to quaternions. Assuming the same definitions for fixed points and cycles, we can define our basis of attraction for a Julia set generated by the polynomial function $p(z)$ to be $A_p(\infty)$. Thus we can define a Julia set $J_p = \partial A_p(\infty)$ and the filled-in Julia set

$$K_p = \mathbb{H} \setminus A_c(\infty) = \left\{z_0 \in \mathbb{H} : |p^k(z_0)| \text{ is not infinite for all } k\right\}$$

3.2 Ray Tracing Quaternion Fractals

To render quaternion fractals, we can adapt the ray tracing algorithm to our case:

Algorithm 2: Ray Tracing Fractals

```
$e \in \mathbb{R}^3$
$R \leftarrow \{r \in \mathcal{R}$ : origin is $e$ and ray intersects image plane$\}$
for $r \in R$ do
    $p_0 \leftarrow$ point of intersection between $r$ and image plane
    $p \leftarrow p_0$
    while $|e - p| < B$ do
        if $|f_q^k(p)| \not\to \infty$ as $k \to \infty$ then
            Calculate surface for $p$ and display at $p_0$
            break
        else
            $p \leftarrow p + \delta$
        end
    end
end
```

The algorithm has a few issues that need to be addressed. Firstly, although we have described quaternions as a 4-dimensional numbering system, science has only allowed us to observe up to the third dimension. We will present a solution to this problem in section 3.2.1.

Secondly, the algorithm is computationally expensive to render. For instance, when rendering to a typical 1920x1080 pixel computer display with 2,073,600 rays, assuming a maximum length of 2 for each ray, and incrementing $\delta = 0.0005$ along each ray, we need to calculate a fractal intersection 4000 times per ray. This amounts to a total of 8,294,400,000 tests. In section 3.2.2, we will discuss some optimization techniques to address this issue.

3.2.1 Mapping Quaternions to Real Space

We can solve the human lack of dimensionality by mapping our quaternion fractals into a 3-dimensional space. We do this by ignoring one dimension. For example, given $q = x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$ we can take the slice $\{w + x\mathbf{i} + y\mathbf{j} + 0\mathbf{k}\}$ and define a function $g : \{w + x\mathbf{i} + y\mathbf{j} + 0\mathbf{k}\} \to \mathbb{R}^3$ such that

$$g(q) = (w, x, y) \tag{3.1}$$

We can then take advantage of the equations (2.8) and (2.9) to rotate the dimension hidden by (3.1) into focus. For example if we multiply q by $\mathbf{k}$ and then take $g(q\mathbf{k})$ we will get

$$\begin{aligned} g\left(q\mathbf{k}\right) &= g\left(w\mathbf{k} + x\mathbf{i}\mathbf{k} + y\mathbf{j}\mathbf{k} + z\mathbf{k}^2\right) \\ &= g\left(w\mathbf{k} - x\mathbf{j} + y\mathbf{i} - z\right) \\ &= g\left(-z + y\mathbf{i} - x\mathbf{j} + w\mathbf{k}\right) \\ &= \left(-z, y, -x\right) \end{aligned}$$

3.2.2 Ray Marching and Distance Estimation

Ray marching is a technique used to optimize ray tracing by reducing the number of iterations needed per ray. Instead of taking fixed steps along the ray, the distance to the fractal surface is estimated at each point and the ray is advanced by that distance. This allows for adaptive step sizes that can be reduced as the distance to the fractal surface decreases, resulting in more accurate rendering with fewer iterations.

To achieve this, we need to be able to estimate the distance to the fractal surface at any given point along the ray. This is where distance estimation comes into play. Distance estimation is a technique used to approximate the distance to the fractal surface at any point in space.

For quaternion fractals of the form $f_p(z) + q$, where $q \in \mathbb{H}$ and $n \in \mathbb{Z}^+$, we can use the formula

$$a\frac{f_p^k(z)}{\left(f_p^k\left(z\right)\right)'} < \delta \tag{3.2}$$

where a is some constant parameter, to determine the lower bound of our distance to the fractal.

With the distance estimate, we can now perform ray marching to efficiently render the fractal. At each point along the ray, we estimate the distance to the fractal surface and advance along the ray by that distance. We repeat this process until we either reach the fractal surface or we exceed a maximum number of iterations. By using adaptive step sizes, we can achieve more accurate rendering with fewer iterations, resulting in faster rendering times.

4 Conclusion

In conclusion, this paper has presented an exploration into the use of quaternion polynomials to generate fractals in 3D space and the adaptation of ray tracing algorithms to render these fractals. By defining quaternion Julia sets and filled-in Julia sets, we have shown how the dynamics of hypercomplex systems can be analyzed and visualized through fractals. Our adapted ray tracing algorithm for rendering quaternion fractals provides a powerful tool for visualizing these complex structures, though it is computationally expensive. Finally, we discussed various optimization techniques to address this issue, such as marching cubes, which can significantly reduce computation time.

References

[BDM$^+$88] Michael F. Barnsley, Robert L. Devaney, Benoit B. Mandelbrot, Heinz-Otto Peitgen, Dietmar Saupe, and Richard F. Voss. *The Science of Fractal Images*. Spinger-Verlag, 1988.

[Buc09] David Bucciarelli. Juliagpu (v1.2). `http://davibu.interfree.it/opencl/juliagpu/juliaGPU.html`, 2009.

[CC93] Lennart Carleson and Theodore W. Camelin. *Complex Dynamics*. Universitext: Tracts in Mathematics. Spinger-Verlag, 1993.

[DKS02] Yumei Dang, Louis H. Kauffman, and Daniel Sandin. *Hypercomplex Iterations: Distance Estimation and Higher Dimensional Fractals*, volume 17 of *Knots and Everything*. World Scientific, 2002.

[Gle88] James Gleick. *Chaos: making a new science*. Penguin Books, 1988.

[Ham66] Sir William Rowan Hamilton. *Elements of Quaternions*. Longsmans, Green, & Co., 1866.

[HSK89] John C. Hart, Daniel J. Sandin, and Louis H. Kauffman. Ray tracing deterministic 3-d fractals. *SIGGRAPH Computer Graphics*, 23(3):289–296, July 1989.

[Man83] Benoit B. Mandelbrot. *The Fractal Geometry of Nature*. Henry Holt and Company, 1983.

[Nyl] Paul Nylander. Hypercomplex fractals. `http://www.bugman123.com/Hypercomplex/index.html`. Accessed: 2023-02-10.

[SC15] Daniel C. Stoll and Hubert Cremer. A brief introduction to complex dynamics. 2015.

[Öd] Torkel Ödegaard. Raytracing 4d fractals, visualizing the four dimensional properties of the julia set. `http://www.codinginstinct.com/2008/11/raytracing-4d-fractals-visualizing-four.html`. Accessed: 2023-02-10.