Interactive Visualization of Quaternion Julia Sets

John C. Hart Louis H. Kauffman Daniel J. Sandin

Electronic Visualization Laboratory
University of Illinois at Chicago

Abstract

The first half of a two-step quaternion Julia set visualization system is described. This step uses a quaternion square root function to adapt the classic inverse iteration algorithm to the quaternions. The augmented version produces a 3-D Julia set defined by a point cloud that can be interactively manipulated on a graphics workstation. Several cues are assigned to the point cloud to increase depth perception. Finally, a short theorem is proven that extends the domain of the inverse iteration method to a rotational family of quadratic quaternion Julia sets.

1 Introduction

The first step of a two-part quaternion Julia set visualization process is presented. This step produces a point cloud defining the quaternion Julia set that can be manipulated in interactive time. The parameters investigated by this step are passed along to a parallel ray-tracing algorithm [3], the second step, that produces presentation quality renderings of quaternion Julia sets. Areas of interest in the ray-traced images can then be further explored by using the interactive step to provide the viewing parameter.

1.1 Approaching Fractals

With the introduction of fractal geometry, mathematics has presented some interesting but difficult objects to computer graphics. These objects are functional-based but the function is often expensive, requiring many computations per point. This expense has impeded the visualization of these objects until now, as a different approach to fractals makes them as easy to visualize as any other data.

The field of computer graphics has classically dealt with continuous piecewise smooth euclidean shapes. These shapes have properties that can be taken advantage of when rendering. One example is the property that the perspective projection of a 3-D polygon is, at worst, a 2-D polygon. Fractals are difficult objects to display when euclidean objects are expected.

On the other hand, non-euclidean mathematical objects often have an assortment of other properties that may be taken advantage of for more efficient visualization. For example, the inverse iteration algorithm uses the properties of attractivity and symmetry to efficiently generate a fractal shape called a Julia set.

1.2 History

The inverse iteration algorithm for Julia sets in the complex plane, hereby denoted $\mathbb{C}$, first appeared in [6]. The algorithm was used by Alan Norton to illustrate fractal shapes in [6] and [7].

Norton later discovered that Julia sets had interesting extensions in the quaternions. He first visualized these shapes using a boundary tracking method [8]. This method produced striking renderings of never-before-seen shapes but required a large amount of time and memory to operate.

At the same time, John Holbrook developed the first quaternion inverse iteration algorithm. He generated sparse point cloud representations of quaternion Julia sets. In fact, so sparse he called them "starfields" [4, 5]. Dissatisfied, he computed the quaternion Julia sets by sampling every point in a 3-D grid and then rendered the resulting binary voxel array.

Later, a ray-tracing algorithm was developed that produced high quality images of these sets at various levels of detail [3]. Unfortunately, like most ray-tracing algorithms, this method was computationally expensive and not always the most appropriate method for scientific investigation.

The inverse iteration algorithm has now been fully developed for a rotational family of quadratic quaternion Julia sets. It requires nominal space and runs in a few seconds on an average graphics workstation.

2 Quaternion Julia Sets

There is still a lot to be discovered about Julia sets in the quaternions. Currently, most of the investigations deal with the quadratic forms. Even though the book is nearly closed on quadratic dynamics in $\mathbb{C}$, there is still much to be learned about their quaternion counterparts.

2.1 Dynamics and Julia Sets

The dynamics of a function are found by repeated application of a function to an initial starting value. For example, let f be a function and z_0 be an initial value. Then the *orbit* of z_0 is $z_1, z_2, \ldots, z_n$ defined by

$$\begin{aligned} z_1 &= f(z_0), \\ z_2 &= f(f(z_0)) \\ &= f^{\circ 2}(z_0), \\ z_n &= f(f(\ldots f(z_0)\ldots)) \\ &= f^{\circ n}(z_0). \end{aligned}$$

One very popular function is the quadratic

$$f(z) = z^2 + c \tag{1}$$

where z is the iterated variable and c is a fixed parameter.

The set K, called the *filled-in Julia set* of polynomial function f, contains all the initial points z_0 whose orbits under f remain bounded. The *Julia set* of f, denoted J, is the boundary of K. In Figure 1, the Julia set is the black boundary of the four-colored filled-in Julia set. Here, as with most Julia sets of Eq. (1), the boundary, J, is fractal [6].

Figure 1: The filled-in Julia set of $f(z) = z^2 + 0.2809 + 0.53i$.

2.2 The Quaternions

The quaternions were introduced by Hamilton in the mid-19th century as a method of performing 3-D multiplication [1]. As the name suggests, the addition of a fourth dimension was necessary for Hamilton to get the multiplication properties he desired.

A quaternion value $z = \alpha + \beta\mathbf{i} + \gamma\mathbf{j} + \delta\mathbf{k}$ is a four-tuple of independent real values $(\alpha, \beta, \gamma, \delta)$ assigned to one real axis and three imaginary axes: $\mathbf{i}, \mathbf{j}, \mathbf{k}$ such that

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

and

$$\mathbf{ij} = \mathbf{k};\ \mathbf{jk} = \mathbf{i};\ \mathbf{ki} = \mathbf{j} \tag{3}$$

but

$$\mathbf{ji} = -\mathbf{k};\ \mathbf{kj} = -\mathbf{i};\ \mathbf{ik} = -\mathbf{j}. \tag{4}$$

The symbol $\mathbb{H}$ (after Hamilton) denotes the quaternions to avoid confusion with the rationals.

The values of the individual components of a quaternion number $z = \alpha + \beta\mathbf{i} + \gamma\mathbf{j} + \delta\mathbf{k}$ are denoted $\mathrm{Re}(z) = \alpha$,$\mathrm{Im}_i(z) = \beta$,$\mathrm{Im}_j(z) = \gamma$, $\mathrm{Im}_k(z) = \delta$. The quaternions are an extension of $\mathbb{C}$. Thus, every complex number is a degenerate quaternion number.

Unlike the complex numbers, quaternion multiplication is not commutative. This becomes important when writing polynomials. For example, in $\mathbb{H}$, the expression az^2 is not equivalent to z^2a nor is it equivalent to zaz.

2.2.1 Rotational Families

Let

$$g_\theta(z) = e^{i\theta} z \tag{5}$$

be a function that rotates point z by θ radians counter-clockwise about the origin in $\mathbb{C}$. A new func-

tion

$$\begin{aligned} F_\theta(z) &= g_\theta \circ f \circ g_\theta^{-1}(z) \\ &= e^{-i\theta} z^2 + e^{i\theta} c \end{aligned} \tag{6}$$

adds a new parameter to Eq. (1) which causes the resulting Julia set to be rotated by θ radians counterclockwise about the origin in $\mathbb{C}$ [9].

What is perhaps unexpected is that when Eq. (6) is iterated in $\mathbb{H}$, completely different shapes occur for the same c but different θ. These shapes all share the same intersection with $\mathbb{C}$ (except for a rotation) but the topology of their extensions in $\mathbb{H}$ change dramatically.

The quaternion Julia set in Figure 5 is an extension to the one in Figure 1 but Eq. (6) is used with $\theta = 90°$. This causes the $\mathbf{i}, \mathbf{j}$ plane to intersect the complex Julia set at its thinnest section and provides perhaps the most characteristic quaternion extension of the complex Julia set.

A closer inspection of Eq. (6) reveals that commutivity was expected. For use in $\mathbb{H}$, it should have been derived

$$F_\theta(z) = e^{i\theta}(e^{-i\theta} z e^{-i\theta} z) + e^{i\theta} c. \tag{7}$$

This lack of commutivity is why the two parameter function Eq. (6) produces the entire family of quaternion Julia sets for any quadratic functions parameterized by a single complex constant. For example, the complex Julia set of

$$f(z) = \lambda z(1 - z) \tag{8}$$

for a given parameter λ produces the same shape as the complex Julia set of Eq. (1) given the right parameter $c \in \mathbb{C}$. But their counterparts in $\mathbb{H}$ are quite different. Both quaternion Julia sets can be produced by Eq. (6) given the parameter c from Eq. (1) and the proper rotation θ.

2.3 Useful Properties

Several properties of quaternion Julia sets have been discovered. The properties that follow were essential for the development of the efficient visualization method using the inverse iteration algorithm. However, knowledge of their details and proofs are not necessary to implement the algorithm.

2.3.1 Strange Attractors

The Julia set is repulsive with respect to f which means that the orbits of points near the set move farther away from it. Conversely, the Julia set is attractive with respect to the inverse

$$f^{-1}(z) = \sqrt{z - c} \tag{9}$$

because the orbits under f^{-1} of points far away from J come closer to it [10]. This means that Julia sets are a specific form of strange attractor. These shapes commonly rise from physical differential equations such as the popular Lorenze attractor.

Each evaluation of Eq. (9) will have some error in its result. This is often a problem in dynamical systems since a small error can become quite large after many iterations of a function (a property called the Butterfly Effect). This will not a problem here because the strong attraction to J is such that any error produced by Eq. (9) is reduced by the next application of f.

2.3.2 Collapsing Two-Spheres

In $\mathbb{R}$ as well as $\mathbb{C}$, squaring is a two-to-one mapping everywhere but the origin. This is not so in $\mathbb{H}$, as suggested by Eq. (2). Here the squaring mapping is capable of mapping an infinite number of values to a single point.

Let

$$\mathbf{S}^2 = \{z : z \in \mathbb{H}, |z| = 1, \mathrm{Re}(z) = 0\} \tag{10}$$

be a unit two-sphere of totally imaginary points. Then

$$z^2 : \mathbf{S}^2 \mapsto -1. \tag{11}$$

In general, any such sphere of radius $\rho \in \mathbb{R}_+$ (the positive real axis), when squared, "collapses" to a single point in $\mathbb{R}_-$ (the negative real axis), specifically $-\sqrt{\rho}$.

2.3.3 j, k Equivalence

This property is less than obvious and so is presented in the form of a theorem.

Theorem 1 *The dynamics of the function*

$$F_\theta(z) = e^{-i\theta} z^2 + e^{i\theta} c \tag{12}$$

for $c \in \mathbb{C}$ is independent of the angle ϕ in

$$z = \alpha + \beta\mathbf{i} + \rho e^{\mathbf{i}\phi}\mathbf{j}. \tag{13}$$

Hence the 4-D Julia set is obtained as a set of revolution about $\mathbb{C}$ of the 3-D Julia set.

A simple algebraic proof may be found in Appendix B. The hypothesis of this theorem requires that $c \in \mathbb{C}$. This causes no loss of generality on c since a theorem in [2] shows that the dynamics of Eq. (1) for $c \in \mathbb{H}$ are the same as with $c \in \mathbb{C}$ except for a rotation about the real axis.

The consequences of Theorem 1 are that the dynamics under Eq. (1) are the same in 3-D as they are in 4-D. Thus, when necessary, a 3-D quaternion value suffices for computation. This also means that if $z \in J$ then its reflection with respect to $\mathbb{C}$, algebraically $z - 2\mathrm{Im}_j z$, is also in J.

3 Visualization Algorithms

Visualization algorithms for fractals most always involve two phases: A *generation* phase that involves some form of production or detection of the fractal object and a *surface normal determination* phase that deals with the problem of assigning a surface normal to a point on a non-differentiable surface.

3.1 Point Cloud Generation

The algorithm presented by this paper is an extension of the classical inverse iteration algorithm to handle quaternion values. This involves more than just changing the square root function. Several other properties of $\mathbb{H}$ can be used to make the algorithm run more efficiently.

3.1.1 Classical Inverse Iteration

Inverse iteration was first used to produce Julia sets by Mandelbrot [6]. It was later developed by Peitgen and Richter [10] and then Peitgen and Saupe [11] where several properties of Julia sets were taken advantage of to make it run more efficiently.

The basic inverse iteration algorithm, Figure 2, utilizes the attractivity of the Julia set under the inverse equation.

The parameter N, the number of points generated, should be set so that enough points are plotted to reasonably define the set but still run in a decent amount of time.

If the initial point is chosen at random then several iterations of the inverse function will bring the point very close to the set. The value M may be set to zero if z_0 is chosen sufficiently close to J.

This algorithm is capable of generating complex Julia sets in real time. In fact, using lookup tables,

1. Select an arbitrary initial point z_0.
2. For $i \in \{1, 2, 3, \ldots, N\}$
   (a) Let z_i be chosen at random from the two possible results of $f^{-1}(z_{i-1})$.
   (b) If $i > M(\approx 50)$ then plot z_i and $-z_i$.
3. End for.

Figure 2: The classical inverse iteration algorithm.

we have created a real time complex Julia set exploration game called "Complex Shapes From Simple Rules" that runs on a PC. It is part of the Interactive Image, an installation of educational games that teach various computer graphics concepts.

3.1.2 Quaternion Inverse Iteration

The quaternion inverse iteration algorithm, Figure 3, utilizes the collapsing two-sphere property to provide initial values that give a more even distribution of points across the quaternion Julia set.

The property of collapsing two-spheres is used to define the initial values of the backward orbits. Since the only way to forward iterate from $\mathbb{H}$ into $\mathbb{C}$ is to hit a totally imaginary two-sphere, this is the best place to start to get a quaternion backward orbit.

Since the initial values of the process are taken from a circle, this algorithm can be thought of as inverse iteration of loops. These loops, however, become quite convoluted in the deeper branches of the inverse iteration tree.

The square root returns at least two values, a positive and a negative, so both are plotted. Theorem 1 allows us to reflect each point across $\mathbb{C}$. These two symmetries allow each point to be plotted as four in the resulting point cloud. The production of four points from each iteration reduces the number of quaternion square root calls, which are quite expensive.

The enhancements made to classical inverse iteration [10, 11] have been investigated and were found to be less than fruitful. Some forms of "tree pruning" have had a slight effect on the distribution of points in the quaternions but overall the enhancements to classical inverse iteration do not improve the quaternion version.

One more improvement that keeps the values in $\mathbb{H}$

1. Compute
$$\rho = \min_{y>0}\{y : y\mathbf{i} \notin K\} \tag{14}$$
by extending a ray from the origin up the $\mathbf{i}$ axis to find the least positive $\mathbf{i}$ value not in the filled-in Julia set.

2. For each value $\theta \in [0, 2\pi)$ let
$$z_0 = \rho\cos\theta\mathbf{i} + \rho\sin\theta\mathbf{j} \tag{15}$$
be a point on a circle of radius ρ about the origin in the $\mathbf{i},\mathbf{j}$ plane ...

   (a) For $i \in \{1, 2, 3, \ldots, N\}$ ...

      i. Compute the next point

         A. Let $z_i = f^{-1}(z_{i-1})$ be the next point in the iteration unless ...

         B. $(z_{i-1} - c) \in \mathbb{R}_-$ in which case z_i should be chosen at random from the circle
$$\mathbf{S}^1 = \{z : |z| = \sqrt{c - z_{i-1}}, \mathrm{Re}(z) = \mathrm{Im}_k(z) = 0\} \tag{16}$$

      ii. Plot $z_i, -z_i$.

      iii. Plot the reflections $z_i - 2\mathrm{Im}_j(z_i)$ and $-z_i + 2\mathrm{Im}_j(z_i)$.

   (b) ...End of "for."

3. End of "for each."

Figure 3: The quaternion inverse iteration algorithm.

is to start over from z_0 when $z_i \in \mathbb{C}$. Once a point is in $\mathbb{C}$, application of f^{-1} keeps it there until it hits $\mathbb{R}_-$. Waiting for that event may take a considerable amount of time.

3.1.3 Inverse Iteration of Rotated Quaternion Julia Sets

When Eq. (9) is applied to a point with zero $\mathbf{k}$ component, then the result also has zero $\mathbf{k}$ component. This is not true for the inverse of Eq. (6). This means that the inverse iteration method only applies to the Julia sets of Eq. (6) when $\theta = 0$.

Here Theorem 1 comes to the rescue, showing how any point in J with non-zero $\mathbf{k}$ component can be converted into a point in J with zero $\mathbf{k}$ component. This is done simply by setting
$$z = \mathrm{Re}(z) + \mathrm{Im}_i(z)\mathbf{i} + \sqrt{\mathrm{Im}_j(z)^2 + \mathrm{Im}_k(z)^2}\ \mathbf{j} \tag{17}$$
for any $z \in J$.

3.2 Point Cloud Display

A simple orthogonal projection can be obtained by displaying $\mathrm{Re}(z)$ horizontally and $\mathrm{Im}_i(z)$ vertically. One small 3-D cue is given in that sparse areas of the projection are surfaces with a large dot product with respect to the viewing vector while the denser areas show smaller dot products. This simulates a light source at the eye position and is a nice way to display the shapes when a single-bit black and white display is used.

A more sophisticated workstation allows a perspective projection along with animation which gives a much better sense of the depth of the point cloud. Perspective along with other dynamic transformations show closer points moving faster than farther points.

3.2.1 Stereo

Stereo can be integrated into the orthogonal projection easily by the following formula

$$\begin{aligned} x' &= \mathrm{x} && \text{(left eye image)},\\ x' &= \mathrm{x} + 0.12\ \mathrm{z} && \text{(right eye image)}, \end{aligned}$$

$$y' = y \qquad \text{(both images)}.$$

This is computed for positive x axis pointing to the viewer's right and positive z axis pointing away from the viewer.

The viewer can percieve a true 3-D image without any special equipment by crossing his/her eyes until the left and right images fuse into one. Focusing may take a moment; sometimes squinting helps.

Some people that have difficulty crossing their eyes may find it helpful to place a finger between the two images. Then focus on the finger while drawing it closer to the face until the left and right images merge.

The quaternion Julia set in Figure 4 is displayed as a stereo pair. It was computed on a laptop PC compatible in less than a minute.

The factor 0.12 is the disparity and can be adjusted to change the depth of the stereo image. These equations force the center of focus of the stereo projection to be at the $z = 0$ plane so half of the points will be in front of the screen and the other half will be behind.

A more correct stereo display involves two perspective transformations performed after a translation to the left and right by some disparity factor. Although rotation can produce a stereo effect, it is not valid stereopsis whereas off-axis perspective projections are.

3.2.2 Depthcueing

Another simple 3-D cue is obtained by depthcueing. This is a technique where the color of an object fades for points farther away as if viewing the point cloud through a fog.

An alternative but similar cue is to use a modulo color scheme based on a coordinate. For example, if valid color indices range from 0 to 255 inclusively, then the function

$$C(z) = \lfloor 256(\mathrm{Re}(z) \bmod 1) \rfloor \tag{18}$$

will assign colors that vary across the point cloud in the direction of the real axis. Computationally, some modulo functions return unexpected values for negative values. In such a case, adding 10 to z does not change its modulo with respect to 1 and insures the points to be positive.

3.2.3 Computation of Surface Normals

A forward iterative method for computing the surface normal of a point on the quaternion Julia set exists [3] but it is computationally expensive. The technique is to define some scalar field that is zero at K and diverges as it approches infinity. Two continuos fields that work well are the potential and the estimated distance. The surface normal is then computed as the gradient of this field at any given point.

3.2.4 Rendering

This method can produce renderable point clouds and an easy method exists for rendering them [2]. During inverse iteration, a z-buffer is maintained and each resulting point is added to the z-buffer. After all the points have been produced the visible points in the z-buffer are rendered by computing their surface normals and lighting them according to the Lambertian lighting model.

The surface normal of a z-buffered point may be found by forming two other vectors [8]. Let

$$\begin{aligned} \vec{X} &= (1, 0, z(x+1, y)), \\ \vec{Y} &= (0, 1, z(x, y+1)) \end{aligned}$$

where z is the z-buffer indexed by x and y. The surface normal may then be computed as

$$\vec{N} = \frac{\vec{X}}{|\vec{X}|} \times \frac{\vec{Y}}{|\vec{Y}|}. \tag{19}$$

If a shadow buffer is maintained by projecting each point with respect to each light source and adding them to a separate z-buffer dedicated to each light source, then the object can be rendered with shadows.

4 Conclusion

The inverse iteration algorithm is much simpler and faster than the other quaternion Julia set visualization algorithms. It also does not require an expensive frame buffer. We expect that these factors will make the algorithm very popular to those wanting to investgate these objects but without the resources available to a research mathematician.

4.1 Comparisons

The inverse iteration method runs in object-time but takes image-space. This requires much less memory

Figure 4: A disconnected quaternion Julia set of $f(z) = z^2 + 1.2i$ generated by inverse iteration on a laptop PC compatible and displayed as a stereo pair.

than does Norton's boundary tracking method and significantly less memory than Holbrook's exhaustive method. The ray-tracing method not only runs in image-space but in image-time as well.

Inverse iteration is the best method for quickly visualizing the global shape of the Julia set whereas ray tracing is the best method for investigating the finer details of the set. Boundary tracking also provides a global view that is significantly better than inverse iteration but at the expense of much more time and memory.

The boundary tracking method is the most efficienct algorithm when a single Julia set is to be animated. The inverse iteration method is more efficient at animating Julia sets undergoing changes due to varying a parameter but the ray-tracing method is the most efficient algorithm for this type of animation when a well-defined image is desired.

The inverse iteration method is by far the best way to visualize disconnected Julia sets in $\mathbb{C}$ or $\mathbb{H}$. The disconnected Julia set in Figure 4 is impossible to produce by boundary tracking and would be quite difficult to accurately display using the ray tracing method.

4.2 Implementations

This method was originally implemented on a laptop where the sets were displayed in single-bit black and white as stereo pairs. The stereo images would generate visually and one could watch as the inverse iteration fills in the quaternion Julia set.

Our current version runs on a Personal Iris 4D/20 which allows interactive positioning of the point cloud and almost interactive parameter changes. The user is allowed to rotate and translate the point cloud to provide the most informative viewing angle. Currently, 12,000 points take less than one second to generate. The display is real-time.

4.3 Acknowledgements

Special thanks are due to Alan Norton for his extensive communication with this group. Thanks also to John Holbrook for his communication.

Steven Lecompte did the initial programming for this project when it first started in 1987. Thanks are also due to Sumit Das for his help on the user interface.

The research for this project was performed at the Electronic Visualization Laboratoy under the co-direction of Thomas A. DeFanti and Daniel J. Sandin with Maxine D. Brown (Assistant Director) and Irving Moy (Systems Specialist).

References

[1] W. R. Hamilton. *Elements of Quaternions*, volume 1–2. Chelsea Publishing Company, New York, 3rd edition, 1969.

[2] J. C. Hart. Image space algorithms for visualizing quaternion Julia sets. Master's thesis, EECS Dept., University of Illinois at Chicago, 1989.

[3] J. C. Hart, D. J. Sandin, and L. H. Kauffman. Ray tracing deterministic 3-D fractals. *Computer Graphics*, 23(3):289–296, 1989.

[4] J. A. R. Holbrook. Quaternionic astroids and starfields. *Applied Mathematical Notes*, 8(2):1–34, 1983.

[5] J. A. R. Holbrook. Quaternionic Fatou-Julia sets. *Annals of Science and Math Quebec*, 1987(1):79–94, 1987.

[6] B. B. Mandelbrot. Fractal aspects of the iteration of $z \rightarrow \lambda z(1-z)$ for complex λ and z. *Annals New York Academy of Sciences*, 357:249–259, 1980.

[7] B. B. Mandelbrot. *The Fractal Geometry of Nature*. Freeman, San Francisco, 2nd edition, 1982.

[8] A. Norton. Generation and rendering of geometric fractals in 3-D. *Computer Graphics*, 16(3):61–67, 1982.

[9] A. Norton. Julia sets in the quaternions. *Computers and Graphics*, 13(2):267–278, 1989.

[10] H.-O. Peitgen and P. H. Richter. *The Beauty of Fractals*. Springer-Verlag, New York, 1986.

[11] H.-O. Peitgen and D. Saupe, editors. *The Science of Fractal Images*. Springer-Verlag, New York, 1988.

A Derivation of the Quaternion Square Root

Recall that the simple formula for the square root of a complex number is given by

$$\sqrt{a+b\mathbf{i}} = \sqrt{\frac{1+a}{2}} + E\sqrt{\frac{1-a}{2}} \qquad (20)$$

when $a^2+b^2=1$. Here E denotes the sign of b if it is non-zero, and E is zero otherwise.

Similarly, a quaternion in 4-D can be written in the form $a+b\mathbf{u}$ where $a, b \in \mathbb{R}$ and $\mathbf{u} = u_1\mathbf{i}+u_2\mathbf{j}+u_3\mathbf{k}$ and $|\mathbf{u}| = 1$. It then follows directly from the algebraic structure of $\mathbb{H}$ that $\mathbf{u}^2 = -1$. Thus, in and of itself each quaternion belongs to a complex plane in four-space that is determined by the real axis and the direction of the unit vector $\mathbf{u}$ in the complementary three-space.

For $\mathbb{H}$, the consequence of the preceding remark is that a power mapping such as Eq. (1) can be computed entirely via the complex numbers formalism of $\mathbf{u}^2 = -1$. By the same token, the core of the formula for the inverse of this power mapping is a square root mapping, and this can be expressed in a form identical to the formula for the square root in complex numbers as shown above.

A.1 Power Mapping in Any Dimension

Regard Euclidean space of dimension $N+1$ as the set of points of the form $a+b\mathbf{u}$ where a and b are real numbers, $\mathbf{u}$ is a unit length vector in $N-space$, and $b\mathbf{u}$ denotes the usual multiplication of this vector by the scalar b. Thus, if $\mathbf{u} = (u_1, u_2, \ldots, u_N)$, then $a+b\mathbf{u}$ is identical with the point $(a, bu_1, bu_2, bu_3, \ldots, bu_N)$ in $(N+1)$-space. Define formally, the product $u^2 = -1$ (for u of unit length), and define $(a+b\mathbf{u})^n$ — the nth power mapping in Euclidean $N+1$ space — exactly as though $a+b\mathbf{u}$ were a complex number. Note that we do not define the products (uv) where u and v ae linearly independent directions in N-space. There are many choices for such products, but only in $\mathbb{C}$ and $\mathbb{H}$ do such products enjoy an associative algebraic structure that is analogous to that of ordinary arithmetic.

If $N=2$, then we obtain a representation of $\mathbb{R}^3$ (Euclidean three-space) as the set of points of the form $a+b\mathbf{u}$ where a and b are real, and $\mathbf{u}$ is a unit direction in the Euclidean plane. Thus $\mathbf{u}$ is specified by an angle θ.

The core of the simplest computation of three dimensional Julia sets is the function for the square root of a point in three space with respect to this second order power mapping. Let this point be specified by $a+b\mathbf{u}(\theta)$ where a and b are real, and $\mathbf{u}(\theta)$ denotes the point $(\cos\theta, \sin\theta)$ in the plane.

Letting $x = a, y = b\cos\theta, z = b\sin\theta$ we can form $\sqrt{a+b\mathbf{u}}$ as follows:

1. Let $R = \sqrt{x^2+y^2+z^2}$.

2. Let $x' = \frac{x}{R}, y' = \frac{y}{R}, z' = \frac{z}{R}$.

3. Let $S = \sqrt{R}$ and $m = \sqrt{y'^2 + z'^2}$.

4. Let $A = \sqrt{\frac{1+x'}{2}}$ and $B = \sqrt{\frac{1-x'}{2}}$.

5. Then $\sqrt{a + b\mathbf{u}} = S[A, B[\frac{y'}{m}, \frac{z'}{m}]]$.

This formula is derived directly from our geometric considerations, and it can be used to program in standard three dimensional coordinates. This makes possible a simple and useful program for 3-D Julia sets, computed by inverse (random for the sign of the root) square root iteration.

B Proof of j, k Equivalence in Julia Sets

Theorem 1 *The dynamics of the function*

$$F_\theta(z) = e^{-i\theta} z^2 + e^{i\theta} c \tag{21}$$

for $c \in \mathbb{C}$ is independent of the angle ϕ in

$$z = \alpha + \beta\mathbf{i} + \rho e^{i\phi}\mathbf{j}. \tag{22}$$

Hence the 4-D Julia set is obtained as a set of revolution about $\mathbb{C}$ of the 3-D Julia set.

Proof: Let

$$\begin{aligned} g_\phi(z) = \mathrm{Re}(z) + \mathrm{Im}_i(z)\mathbf{i} + \\ e^{i\phi}(\mathrm{Im}_j(z)\mathbf{j} + \mathrm{Im}_k(z)\mathbf{k}) \end{aligned} \tag{23}$$

be a rotation of quaternion z about the complex plane that leaves points in $\mathbb{C}$ fixed. The proof consists of showing algebraically that

$$\begin{array}{ccc} & F_\theta & \\ \mathbb{H}^{\frac{3}{4}} & \longrightarrow & \mathbb{H}^{\frac{3}{4}} \\ g_\phi \uparrow & & \uparrow g_\phi \\ \mathbb{H} & \longrightarrow & \mathbb{H} \\ & F_\theta & \end{array} \tag{24}$$

commutes. Here the symbol $\mathbb{H}^{\frac{3}{4}}$ denotes the space spanned by $1, \mathbf{i}$ and $\mathbf{j}$.

Any arbitrary $z \in \mathbb{H}$ can be expressed

$$z = \alpha + \beta\mathbf{i} + \rho e^{i\phi}\mathbf{j} \tag{25}$$

parameterized by three reals α, β, ρ and an angle ϕ. Notice that

$$g_{-\phi}(z) = \alpha + \beta\mathbf{i} + \rho\mathbf{j} \tag{26}$$

takes z from $\mathbb{H}$ into $\mathbb{H}^{\frac{3}{4}}$. Then

$$\begin{aligned} F_\theta \circ g_{-\phi}(z) = \\ e^{-i\theta}(\alpha^2 - \beta^2 - \rho^2 + 2\alpha\beta\mathbf{i} + 2\alpha\rho\mathbf{j}) + e^{i\theta}c \end{aligned} \tag{27}$$

$$\begin{aligned} = \; & (\alpha^2 - \beta^2 - \rho^2)\cos\theta + 2\alpha\beta\sin\theta + \\ & ((-\alpha^2 + \beta^2 + \rho^2)\sin\theta + 2\alpha\beta\cos\theta)\mathbf{i} + \\ & 2\alpha\rho\cos\theta\mathbf{j} - 2\alpha\rho\sin\theta\mathbf{k} + e^{i\theta}c. \end{aligned} \tag{28}$$

Rotating back we get

$$\begin{aligned} g_\phi \circ F_\theta \circ g_{-\phi}(z) = \\ (\alpha^2 - \beta^2 - \rho^2)\cos\theta + 2\alpha\beta\sin\theta + \\ ((-\alpha^2 + \beta^2 + \rho^2)\sin\theta + 2\alpha\beta\cos\theta)\mathbf{i} + \\ 2\alpha\rho(\cos\theta\cos\phi - \sin\theta\sin\phi)\mathbf{j} - \\ 2\alpha\rho(\sin\theta\cos\phi - \cos\theta\sin\phi)\mathbf{k} + e^{i\theta}c. \end{aligned} \tag{29}$$

which is the same result as

$$\begin{aligned} F_\theta(z) = \\ e^{-i\theta}(\alpha + \beta\mathbf{i} + \rho\cos\phi\mathbf{j} + \rho\sin\phi\mathbf{k})^2 + e^{i\theta}c \end{aligned} \tag{30}$$

$$\begin{aligned} = \; & e^{-i\theta}(\alpha^2 - \beta^2 - \rho^2\cos^2\phi - \rho^2\sin^2\phi + \\ & 2\alpha\beta\mathbf{i} + 2\alpha\rho\cos\phi\mathbf{j} + 2\alpha\rho\sin\phi\mathbf{k}) + e^{i\theta}c \end{aligned} \tag{31}$$

$$\begin{aligned} = \; & e^{-i\theta}(\alpha^2 - \beta^2 - \rho^2 + 2\alpha\beta\mathbf{i} + \\ & 2\alpha\rho\cos\phi\mathbf{j} + 2\alpha\rho\sin\phi\mathbf{k}) + e^{i\theta}c \end{aligned} \tag{32}$$

$$\begin{aligned} = \; & (\alpha^2 - \beta^2 - \rho^2)\cos\theta + 2\alpha\beta\sin\theta + \\ & ((-\alpha^2 + \beta^2 + \rho^2)\sin\theta + 2\alpha\beta\cos\theta)\mathbf{i} + \\ & 2\alpha\rho(\cos\theta\cos\phi - \sin\theta\sin\phi)\mathbf{j} - \\ & 2\alpha\rho(\sin\theta\cos\phi - \cos\theta\sin\phi)\mathbf{k} + e^{i\theta}c. \quad \square \end{aligned} \tag{33}$$

Figure 5: The ray traced quaternion extensions to the Julia set in Figure 1 after a rotation of 90°.

Figure 6: An depthcued point cloud defined by quaternion inverse iteration of the quaternion Julia set in Figure 5.

Figure 7: The Julia set at the origin with $\theta = 0$ surrounding a sphere for reference.

Figure 8: The Julia set at the origin with $\theta = 90°$. Although the Julia set is still a sphere, the distribution of points has changed dramatically.

Figure 9: The Julia set at the origin without the sphere, depthcued as in Figure 6.

Figure 10: "The Pillow." The quaternion Julia set at $c = 0.25$ for $\theta = 90°$.