Ray Tracing Deterministic 3-D Fractals

John C. Hart*, Daniel J. Sandin*, Louis H. Kauffman†

*Electronic Visualization Laboratory
†Dept. of Mathematics, Statistics and Computer Science

University of Illinois at Chicago

Abstract

As shown in 1982, Julia sets of quadratic functions as well as many other deterministic fractals exist in spaces of higher dimensionality than the complex plane. Originally a boundary-tracking algorithm was used to view these structures but required a large amount of storage space to operate. By ray tracing these objects, the storage facilities of a graphics workstation frame buffer are sufficient. A short discussion of a specific set of 3-D deterministic fractals precedes a full description of a ray-tracing algorithm applied to these objects. A comparison with the boundary-tracking method and applications to other 3-D deterministic fractals are also included.

CR Categories and Subject Descriptors: I.3.7 [Computer Graphics]: Three Dimensional Graphics and Realism — Color, shading, shadowing and texture.

General Terms: Algorithms, Theory.

Additional Keywords and Phrases: fractal, quaternions, distance estimate, ray tracing, surface determination.

1 Introduction

Computer graphics has greatly aided the investigation of the dynamics of iterative functions. Standard 2-D frame buffer techniques have provided sufficient visual information about the structures since most of the research has concentrated on the dynamics of complex variables. However, recent investigations into higher-dimensional dynamical systems [14,15,3,5,17] have shown the need for 3-D visualization tools that will give researchers a better understanding of these objects.

One such method is ray tracing, but this method is prohibitively slow unless an efficient ray-surface intersection computation is used. While these functions are available for Euclidean surfaces, they do not exist (yet) for fractal ones. However, using an unusual construction called the unbounding volume, made possible by a recent advance in the study of dynamics, swift ray tracing of these deterministic fractal objects is possible.

Prior to the description of the algorithm, a specific family of 3-D deterministic fractals, quaternion Julia sets, is outlined. The generation algorithm is then developed using this family as example. Rendering procedures specific to fractal surfaces are then discussed. Finally, the algorithm is compared with its predecessor [14] and applications to other families of 3-D deterministic fractal objects are shown.

2 Dynamics in the Quaternions

The dynamics of quadratic functions have been observed mainly in the complex plane. However, as shown first in 1982 [14], they exist in the 4-D space of the quaternions as well. A discussion of the special properties of Julia sets in the quaternions, for which the ray-tracing algorithm was developed to visualize, is preceded by an introduction to dynamics and quaternion algebra.

©1989 ACM-0-89791-312-4/89/007/0289 $00.75

2.1 Dynamics of Quadratic Functions

The examples used in this paper are derived from the quadratic function

$$f_\mu(z) = z^2 + \mu, \tag{1}$$

where z is the iterated variable and μ is a constant parameter of the equation.

The dynamics of a function f are expressed as the n-fold application of function f to an initial value z. The result is denoted as $f^n(z)$ and should not be confused with simply raising the result of $f(z)$ to the nth power.

The resulting value $f^n(z)$ is used to classify the initial point z depending on its attraction to infinity. Two sets may be constructed under this classification. The filled-in Julia set $\mathcal{K}_\mu$ and the Mandelbrot set $\mathcal{M}$.

Definition 1 $\mathcal{K}_\mu = \{z : \lim_{n\to\infty} f_\mu^n(z) \not\to \infty\}$

Definition 2

$$\mathcal{M} = \{\mu : \lim_{n\to\infty} f_\mu^n(z_c) \not\to \infty, f'_\mu(z_c) = 0\}$$

Note that z_c is the critical point of the function. There is only one critical point of eq. (1) and it is always 0. Several critical points are common for polynomials of degree 3 or greater.

The interesting property that Julia and Mandelbrot sets share is that they are both fractal [12] possessing detail at every level of magnification.

2.2 The Quaternions

The values z and μ are commonly defined as real or complex. However, these values may be defined in any algebraic system closed under addition and multiplication. One such system, the quaternions [7], possesses the additional benefit of having four dimensions.

Definition 3 *A quaternion value q is a four-tuple consisting of one real part and three imaginaries*

$$q = q_1 + q_i\boldsymbol{i} + q_j\boldsymbol{j} + q_k\boldsymbol{k}$$

where $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ are imaginary units,

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

Algebraic operations can be defined in the quaternions by treating the quaternion values as polynomials of three variables $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$. For example, the coefficients of the sum of two quaternion values may be found by adding their corresponding coefficients. Quaternion multiplication is also similar to polynomial multiplication but with the special cases

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

and

$$\boldsymbol{ji} = -\boldsymbol{k};\ \boldsymbol{kj} = -\boldsymbol{i};\ \boldsymbol{ik} = -\boldsymbol{j}, \tag{4}$$

revealing an unfortunate side effect of the quaternions: noncommutative multiplication.

2.3 Julia Sets in the Quaternions

By using the rules of quaternion algebra, eq. (1) can be iterated in the quaternions and Julia sets may be computed. Since the complex plane is a subset of the quaternions, the same complex Julia sets exist in the quaternions but often have extensions outside the complex plane. In fact, if μ has an imaginary component, then the extensions are nontrivial, containing more information than their complex subsets.

A subset of these extensions can be visualized in 3-D by examining the intersection of the 4-space with a 3-space such as that spanned by $1, \boldsymbol{i}, \boldsymbol{j}$ at $0\boldsymbol{k}$. It should be mentioned that the Julia sets of eq. (1) in the 3-space spanned by $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ at 0 are always concentric spheres centered at the origin [8].

An interesting property about quaternion Julia sets is that given two complex Julia sets differing only by a rotation about the origin, their supersets in 3-D may have completely different shapes. The rotation of the Julia set in the complex plane is computed by incorporating the homeomorphism

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

into the iterated function such that

$$f_{\mu,\theta}(z) = g_\theta(f_\mu(g_\theta^{-1}(z))) \tag{6}$$

which suffices to rotate the positive real axis into the positive imaginary axis and so forth in a counterclockwise manner about the origin in the complex plane. The resulting Julia set is merely rotated in the complex plane but appears quite differently in the quaternions since its intersection with the imaginary 3-space is changed. See [15,8] for details.

3 Ray Tracing 3-D Julia Sets

Ray tracing is one of the more realistic methods of rendering objects. Easily accounting for hidden surfaces and self-shadowing, it also provides a method for displaying reflection, transparency and refraction. Mathematical objects may be ray traced by detecting

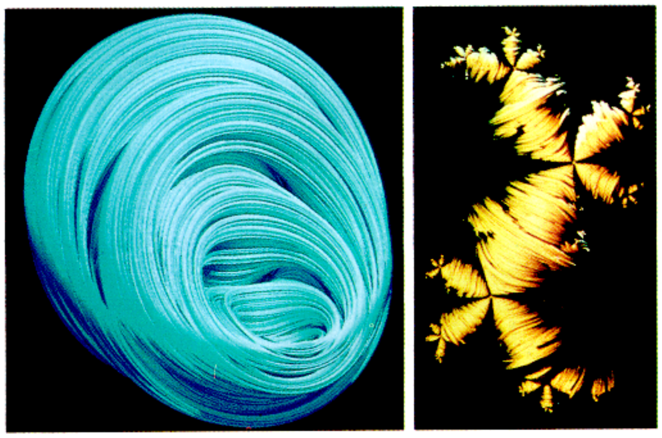

Figure 1: A quaternion Julia set before and after a quarter turn in the complex plane.

their boundaries during a ray-casting step and rendering the surfaces by allowing the ray to be deflected off to a light source.

A naive method to ray trace a quaternion Julia set is to sample each point at a given resolution along each ray. This is not an entirely ridiculous method since it is the basis of some volumetric rendering algorithms [10]. However, it is not practical when applied to fractal objects since each point's classification may rely on a large number of function iterations. With the use of a new ray-tracing mechanism, the amount of sample points per ray is greatly reduced.

3.1 Unbounding Volumes

One method of increasing the speed of ray tracing is the use of ***bounding volumes.*** A bounding volume, usually a sphere or ellipsoid, envelopes several surfaces such that if a ray does not intersect the bounding volume it does not intersect the surfaces contained in it.

Bounding volumes are quite useful in hastening ray tracing of stochastically-defined fractal surfaces [9,4]. Unfortunately their application to deterministic fractals has not been as successful. However, with the discovery of the distance estimate, we can increase the speed of ray tracing deterministic fractals using ***unbounding volumes.***

Unbounding volumes are defined as volumes that do ***not*** contain any part of the object. Thus, given any point outside the object, the ideal bounding volume is the largest volume that does not intersect the object centered at the point. If this volume is a sphere then its radius is the distance to the object. Given a point and a deterministic fractal object, its exact distance cannot be computed efficiently but it can be estimated in time proportional to the time it takes to determine if the point is external to the object.

The lower bound of the distance from a point external to the deterministic fractal set is given as

$$d(z) = \frac{\sinh G(z)}{2e^{G(z)}|G'(z)|} \log G(z), \tag{7}$$

where $G(z)$ is the electrostatic potential at point z and $G'(z)$ is the gradient of this potential. For the quadratic family, the approximation

$$d(z) = \frac{|f^n(z)|}{2|f'^n(z)|} \log f^n(z) \tag{8}$$

is sufficiently accurate [13,6]. See [17] for the computation of $f'(z)$.

By using a distance estimate we can define an unbounding volume of a deterministic fractal set as a sphere that is guaranteed not to intersect[1] nor contain the set in question. Since the distance estimate may be much smaller than the distance along the ray to the object, several repeated distance calculations must be made as the ray is traversed from eye to surface.

3.2 Ray Traversal

Given the set of unbounding spheres completely surrounding an object, a ray is traversed from the eye through the projection plane to the object, testing and incrementing at each point along the ray. By incrementing by the radius of the unbounding sphere, we leap along the ray until we approach the surface.

As the current point on the ray approaches the surface, the unbounding spheres get smaller and smaller. To hasten convergence, a small number ϵ is defined as the minimum ray increment. This increment should be set to give the best depth resolution with respect to the resolution of the projection plane.

The ray traversal equation may be stated inductively given the eye position r_0 and a point in the projection plane $p_{x,y}$ as

$$r_{n+1} = r_n + m \max d(r_n), \epsilon. \tag{9}$$

where m is a unit slope vector of the ray

$$m = \frac{p_{x,y} - r_0}{|p_{x,y} - r_0|}. \tag{10}$$

[1]With the exception of a single point.

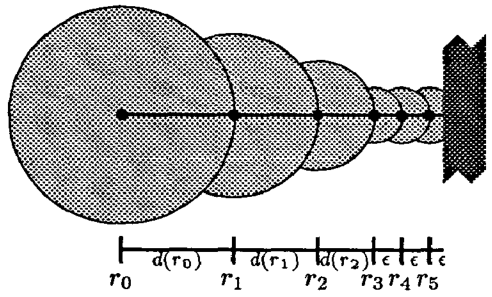

Figure 2: Ray traversal using unbounding spheres.

3.3 Thin Objects

Often the extensions of $\mathcal{K}$ into 3-D are very thin, such as when $\mathcal{K}$ is a dendrite. This creates the possibility that incrementing by ϵ may traverse the ray completely through the object.

This problem has also had manifestations in the 2-D study of these images such as the complex Mandelbrot set. To show that the "islands" off the main continent of $\mathcal{M}$ are connected to it, the Mandelbrot set may be defined computationally as

$$\mathcal{M}_\epsilon = \{z : d(z) < \epsilon\} \tag{11}$$

where $d()$ is the distance estimate as defined in eq. (7). The result is the "hairy" Mandelbrot set [13] revealing its dendritic structures[2].

A similar technique is used to ray trace dendritic sheets in 3-D. By terminating ray traversal when the distance is less than the minimum ray increment, thickness is added to the object while maintaining its structure and detail.

3.4 Avoiding Bad Distance Estimates

When the approximation to eq. (7) is used, it is inaccurate when z is far from the set. This can result in exaggerated distance estimates which could possibly push the ray far into the interior of the object.

To avoid these bad estimates a single bounding volume may be used to contain the fractal set if it can be bounded. Another alternative is to set a maximum distance to increment along the ray.

4 Rendering Fractal Surfaces

The deterministic family of fractals has provided computer graphics with the most complicated borders. The surfaces defined by these borders in 3-D, although quite chaotic, often reveal the structure of the object. A proper rendering of a fractal surface should reveal its order while hinting at its chaos.

[2]Note that these hairs may be seen very clearly as the set $\mathcal{M}_\epsilon - \mathcal{M}$.

Since the surface of a fractal is infinitely convoluted, its normal can only be approximated. The approximated normal signifies the structure of the surface while at finer resolutions the light is scattered in all directions. Thus the surfaces should be diffusely shaded using the Lambertian model.

Also, to achieve the most information from each view, it is often better to use axle light instead of ambient light. By defining a point light source at the eyepoint, every viewable point on the object will receive light and thus even heavily-shadowed sections of the object will reveal information about themselves.

4.1 Normal Approximation

One reason fractal lines, such as the border of $\mathcal{K}$, are so interesting is that they are nondifferentiable. The slope at any point is undefined because closer examination shows that the point has different surroundings. Hence, when expanded to surfaces, 3-D fractal surfaces are nondifferentiable and thus have no exact normal defined.

In order to realistically render these surfaces a shading model must be used which requires the definition of a surface normal. Normals may be approximated by examining a point's relationship with its surroundings. Two approximations have been found to work quite well: Z-buffer neighbors, previously discussed in [14], and the gradient.

4.1.1 Z-buffer Neighbors

As shown in [14], the surface normal of a fractal surface may be approximated as the cross product of two vectors embedded in the surface. Given a buffer of visible z-values Z we can define three points

$$X = \{\epsilon, 0, Z_{x+\epsilon,y} - Z_{x,y}\} \tag{12}$$
$$Y = \{0, \epsilon, Z_{x,y+\epsilon} - Z_{x,y}\} \tag{13}$$
$$O = \{0, 0, Z_{x,y} - Z_{x,y}\} \tag{14}$$

where ϵ is the width of an element in the z-buffer. The surface normal may then be approximated as the normal of the plane defined by these three points.

4.1.2 Gradient Computation

The previous method is a useful normal determination tool if a Z-buffer is maintained during rendering. Ray tracing does not require a Z-buffer so a normal approximation method using a single point in 3-space would be more useful.

This can be accomplished by computing the gradient of a point on the surface. The gradient may be computed in a 3-D density map as

$$\begin{aligned} N_x &= D_{x+\epsilon,y,z} - D_{x-\epsilon,y,z} \\ N_y &= D_{x,y+\epsilon,z} - D_{x,y-\epsilon,z} \\ N_z &= D_{x,y,z+\epsilon} - D_{x,y,z-\epsilon} \end{aligned} \qquad (15)$$

where $D_{x,y,z}$ is the density at the point x, y, z.

The density function of a deterministic fractal is defined on its exterior and can be any continuous function based loosely on the distance to the set. Two useful functions are the potential $G()$ and the estimated distance $d()$. The latter should be used when possible since it is more closely associated with the actual distance although the former works when a distance estimate is not defined.

Other gradient functions may be defined based on the number of samples taken. The 6-point gradient may be augmented by adding samples from points sharing edges producing an 18-point gradient. By including points sharing corners, a 26-point gradient results.

4.2 Clarity

Since these objects are fractal, they should reveal more detail when closely inspected. However, if the minimum ray increment ϵ is constant, the surfaces will not reveal a finer structure when examined. A variable-resolution system is required such that closer sections of the object are defined at higher resolutions as suggested in [2].

One method of increasing the depth resolution is to set ϵ to a function of distance from the eye. The clarity function $\Gamma_{\alpha,\delta}$ is defined

$$\Gamma(d) = \alpha d^{\delta} \qquad (16)$$

given d, the Euclidean distance from the eye to the current location on the ray,

$$d = |r_n - r_0|. \qquad (17)$$

The parameter δ is a depth-cueing exponent and α is an empirical proportion defining the depth resolution of the object.

Three effects are defined by varying the depth-cueing exponent. When the clarity function is constant, inverse depth cueing occurs giving the appearance that farther objects have more detail. This may seem useless but is quite adequate when viewing entire fractal objects from a distance. Linear clarity

δ	$\Gamma()$	Effect
0	Constant	Inverse clarity
1	Linear	Even clarity
2	Quadratic	Exaggerated depth

Table 1: Depth-cueing exponents and their effects on renderings

gives an even clarity appearance with close objects appearing as detailed as far. Quadratic depth cueing gives an exaggerated depth cue, blurring distant surfaces.

The α parameter is tuned to balance the equilibrium of order and chaos. Setting α too small will produce a very noisy surface whereas a large α will wash out detail. When defining α a good starting point is to set it to an order of magnitude smaller than the pixel width.

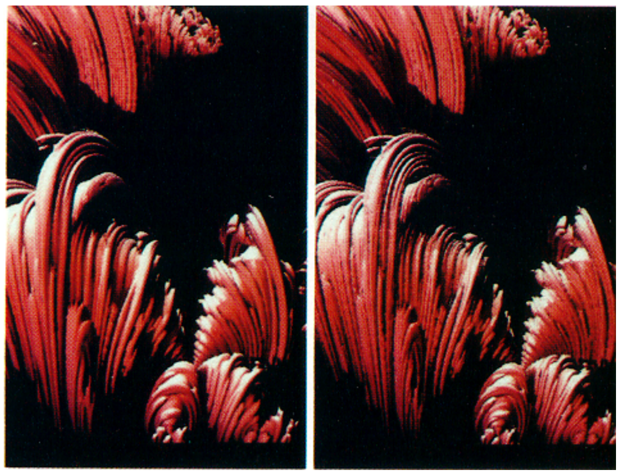

Figure 3: The same Quaternion Julia set rendered twice to show the difference between constant and linear clarity.

Thus, by setting

$$\epsilon = \Gamma_{\alpha,\delta}(|r_n - r_0|) \qquad (18)$$

a variable-resolution rendering system is constructed allowing small details of the surfaces to be investigated without overcomputing the other visible surfaces. The increment ϵ may also be used in the gradient computation as the distance along the axes to sample nearby densities.

5 Application to Other Deterministic Fractals

The quaternions are convenient to observe 3-D dynamics since all three dimensions may be spanned by a single variable. Other 3-D spaces may be constructed using multiple real or complex variables. Complex Julia sets form a 3-D object when they are stacked [14,12]. The cubic connectedness locus is a four-dimensional object when its two parameters are complex [5,17]. Also, Iterated Function Systems may be three-dimensional if they are specified with affine transformations of three real variables [1].

5.1 Julia Set Stacks

A Julia set stack may be specified in 3-D as a slice (i.e. zeroset) of the four dimensional space $C \times C$ defined by the two complex variables z and c from eq. 1. By looking at the z-planes and varying some single-dimension function of c to define the third dimension, the Julia sets may be stacked to form a 3-D object.

There currently is no proven distance underestimate for this set although some images may be generated using empirical formulas based on the Mandelbrot set distance estimate.

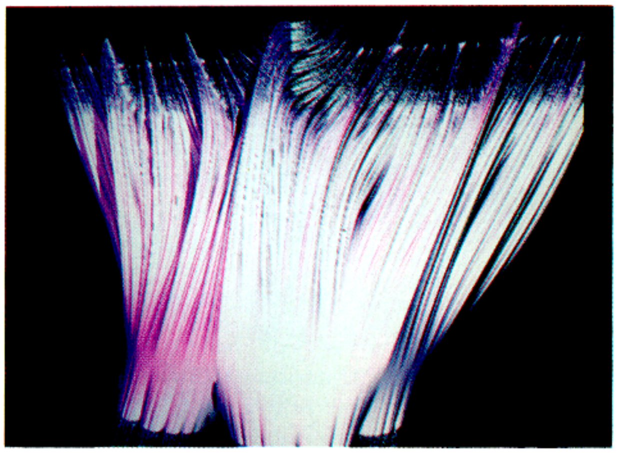
Figure 4: Stack of Julia sets for Im(c) = 0.

5.2 The Cubic Connectedness Locus

The cubic connectedness locus $\mathcal{C}$ is specified by the cubic function

$$f_{a,b}(z) = z^3 - 3a^2 z + b, \qquad (19)$$

where z is a complex variable and a, b are complex parameters. The parameter a is squared in the equation because the two critical points of the equation are $\pm a$.

Since a and b are both complex, a *double* complex plane is constructed that houses the four-dimensional cubic connectedness locus. The locus in this case consists of two components, $\mathcal{C}^+$ and $\mathcal{C}^-$, based on the status of the appropriate critical point.

Definition 4 $\mathcal{C}^+ = \{a, b : \lim_{n\to\infty} f^n_{a,b}(a) \not\to \infty\}$

Definition 5 $\mathcal{C}^- = \{a, b : \lim_{n\to\infty} f^n_{a,b}(-a) \not\to \infty\}$

A picture of this set may be found in [17], presumably created using the technique outlined in [14].

At the moment, a distance estimate does not exist for this set either. However, the cubic connectedness locus has been proven to be connected [5] suggesting that potential measurement and therefore distance estimation may be possible.

5.3 3-D Iterated Function Systems

The most useful forms of deterministic fractals are Iterated Function Systems or IFS's. An IFS can be created to simulate almost any form using the Collage Theorem [1]. Then, given only the resulting set of iterative equations, the form can be reconstructed.

Recently, deterministic IFS Julia and Mandelbrot set functions have been discovered and their exteriors have been categorized according to escape iterations not unlike their quadratic counterparts [1,19]. This suggests that perhaps potential and distance measurements can be made on these sets as well.

6 Comparison with Boundary Tracking

The ray-tracing algorithm's predecessor, Boundary Tracking [14], generates 3-D Julia sets by first locating a starting point on the boundary of the object and then recursively detecting its neighbors until the entire boundary is scanned. To converge, this algorithm must constantly verify that neighboring points have not been previously tested, which requires the efficient storage of all previously generated points. Thus, the Boundary Tracking algorithm runs in object-space and therefore object-time[3].

The main advantage of Boundary Tracking is that objects are only generated once and may be repositioned as often as desired, requiring only re-rendering. Using the ray-tracing technique, repositioning of the object requires re-generation of the viewable sections of the object as well as re-rendering. However, since the image-time ray-tracing algorithm generates these objects more efficiently than the object-time Boundary Tracking algorithm, it is the best choice for parameter-space animations such as varying μ in eq. (1), θ in eq. (5) or the k axis component of the viewable subspace of the quaternions.

[3]See [20] for a discussion of image-space vs. object-space and image-time vs. object-time.

The main disadvantage of Boundary Tracking is that it requires storage of every point in the object. This amount of storage can approach cubic proportions since the number of points defining a surface is $O(r^D)$, the resolution r of the surface raised to its fractal dimension D [12]. The ray tracing technique, using image-space, requires exactly $r^2 + O(1)$ space[4], the storage resources of a graphics workstation frame buffer.

Another problem with Boundary Tracking is constant resolution. However, a variable-resolution Boundary Tracking algorithm was developed to create [16] by generating certain sections of the set already known to be closely examined in the fly-by at higher resolutions. Although Boundary Tracking saves computing time by generating the object only once, the ray-tracing algorithm is the better choice for fly-bys since its dynamic resolution allows a more realistic inspection of the surface.

Finally, ray tracing allows certain special effects such as reflections and refractions. The former produces the interesting effect of revealing only macroscopic images of its environment; the subtle details are lost in the convoluted interreflections of the fractal surface. Refraction as well as simple transparency should be avoided until a reliable internal distance estimate is developed to prevent minimum increments in the interior of the sets.

7 Conclusion

The research on this project began as a method of visualizing quaternion Julia sets in 3-D using the resources of a common computer graphics workstation. The first attempt relied on the Inverse Iteration method of generating Julia sets [11,18] which operated in image-space and object-time but produced less than satisfactory results due to inherent problems of the process itself magnified by the addition of an extra dimension [8]. The ray-tracing solution, being forward iterative, does not experience the problems of the Inverse Iteration method while still requiring only image-space.

7.1 Parallel Implementation

The current implementation of the algorithm runs on an AT&T Pixel Machine with 64 parallel processors each running at about 10 MFLOPS. The architecture of the Pixel Machine, each processor connected only to its portion of the frame buffer, dictates that image-space, image-time algorithms will run at the most optimal level.

[4]If the gradient normal approximation method is used, a Z-buffer is not required. The only other considerable amount of memory used is a small array the size of the maximum allowable iteration count used to optimize the computation of the distance estimate [17].

The ray-tracing code is replicated into 64 equivalent programs running simultaneously as if in a race, each generating and rendering its $\frac{1}{64}$th of the image. Of course, the first operation of each program is to find out which pixel with respect to the entire frame buffer it is working on.

Currently, full screen images (1280 × 1024) take about an hour to generate. When positioning the object, lower image resolutions are used to create a more interactive environment. Also, reducing the depth resolution will increase the speed of the algorithm.

7.2 Acknowledgments

The authors wish to thank Alan Norton and Charlie Gunn for their communications, AT&T for its major grant to the Electronic Visualization Laboratory which supported this research, Maggie Rawlings for her artistic advice with the illustrations, and Tom DeFanti and Maxine Brown for their assistance with the manuscript. This research would not have been possible without the efforts of the faculty, staff and students of the Electronic Visualization Laboratory.

References

[1] Barnsley, M. F. *Fractals Everywhere.* Academic Press, New York, 1988.

[2] Barr, A. H. Ray tracing deformed surfaces. *Computer Graphics 20*, 4 (1986), 287–296.

[3] Blanchard, P. Disconnected Julia sets. *Chaotic Dynamics and Fractals* (1986), 181–201.

[4] Bouville, C. Bounding ellipsoids for ray-fractal intersection. *Computer Graphics 19*, 3 (1985), 45–51.

[5] Branner, B., and Hubbard, J. H. The iteration of cubic polynomials, Part I: The global topology of the parameter space. *Acta Mathematica 160*, 3 (1988), 143–206.

[6] Fisher, Y. *The Science of Fractal Images.* Springer-Verlag, New York, 1988, ch. Exploring the Mandelbrot Set, pp. 287–296.

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

[8] Hart, J. C. *Image Space Algorithms for Visualizing Quaternion Julia Sets.* Master's thesis, University of Illinois at Chicago, 1989.

[9] Kajiya, J. T. New techniques for ray tracing procedurally defined objects. *ACM Transactions on Graphics 2*, 3 (1983), 161–181.

[10] Levoy, M. Display of surfaces from volume data. *IEEE Computer Graphics and Applications 8*, 3 (1988), 29–37.

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

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

[13] Milnor, J. *Computers in Geometry and Topology.* Marcel Dekker, Inc., 1989, ch. Self-similarity and hairiness in the Mandelbrot set, pp. 211–257.

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

[15] Norton, V. A. Julia sets in the quaternions. To appear in *Computers and Graphics.*

[16] Norton, V. A., and Melton, E. A close encounter in the fourth dimension. *SIGGRAPH Video Review 39* (1988), 30.

[17] Peitgen, H. *The Science of Fractal Images.* Springer-Verlag, New York, 1988, ch. Fantastic Deterministic Fractals, pp. 169–218.

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

[19] Prusinkiewicz, P., and Sandness, G. Koch curve as attractors and repellers. *IEEE Computer Graphics and Applications 8*, 6 (1988), 26–40.

[20] Sutherland, I., Sproul, R., and Schumacker, R. A characterization of ten hidden-surface algorithms. *Computing Surveys 6*, 1 (1974), 1–55.

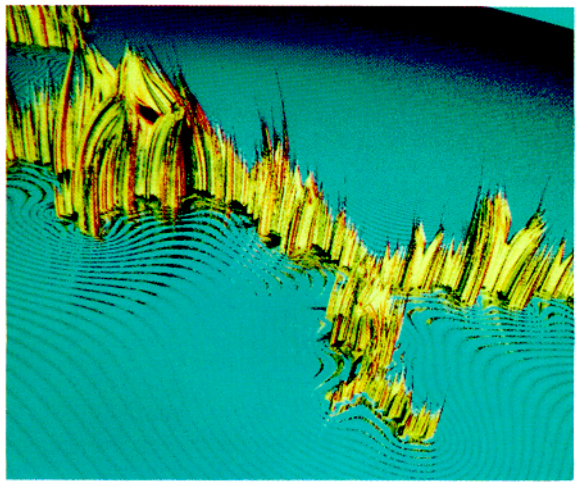
Figure 5: A dendritic quaternion Julia set, set in a sea whose waves are periodic functions of the distance estimate.

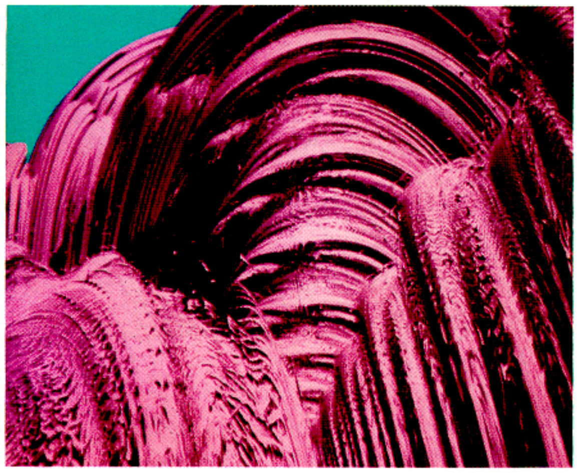
Figure 6: Close up of the surface of the quaternion Julia set shown in fig. 3.

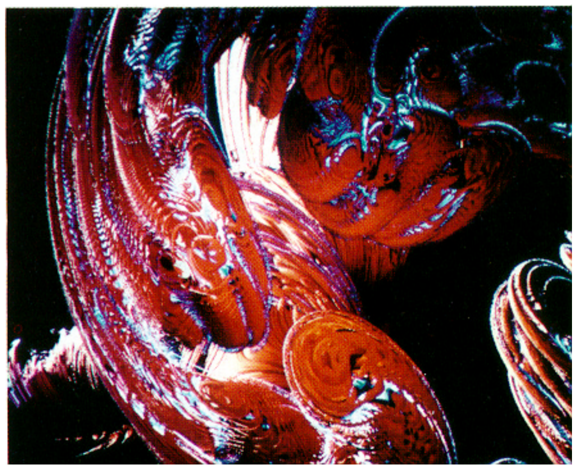
Figure 7: Close up of an interesting section of the Julia set for $\theta = 110^\circ$