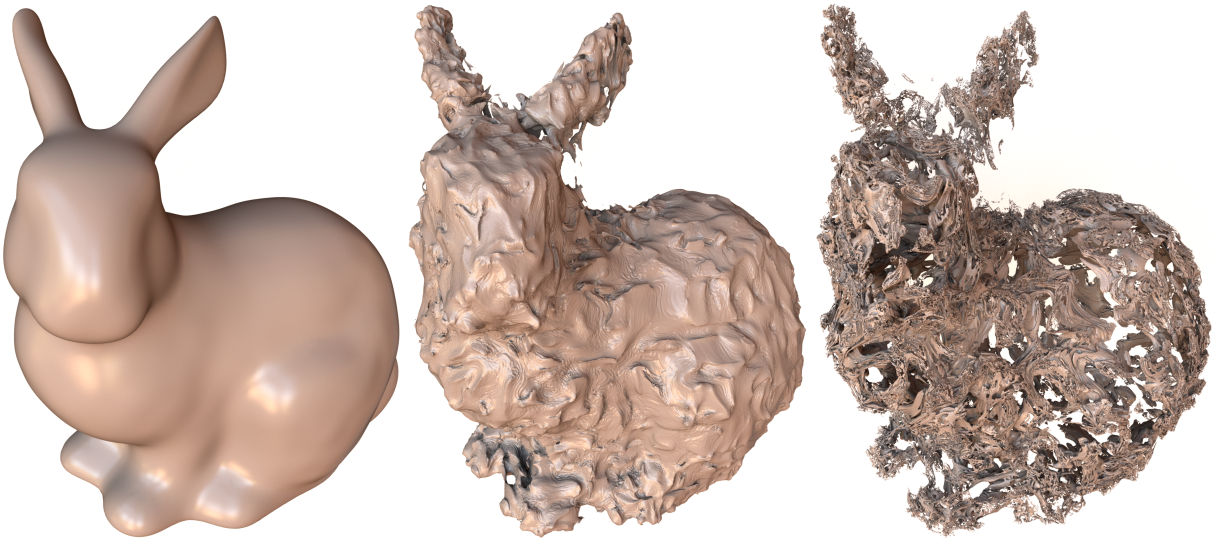
Eurographics Symposium on Geometry Processing 2015
Mirela Ben-Chen and Ligang Liu
(Guest Editors)

Volume 34 (2015), Number 5

Quaternion Julia Set Shape Optimization

Theodore Kim

Media Arts and Technology Program
University of California, Santa Barbara



(a) (b) (c)

Figure 1: *(a) The original Bunny. (b) Julia set of the 331-root rational map found by our optimization. (c) Highly intricate surface obtained by translating the roots by 1.41 in the z direction. Image is high-resolution; please zoom in to see details.*

Abstract

We present the first 3D algorithm capable of answering the question: what would a Mandelbrot-like set in the shape of a bunny look like? *More concretely, can we find an iterated quaternion rational map whose potential field contains an isocontour with a desired shape? We show that it is possible to answer this question by casting it as a shape optimization that discovers novel, highly complex shapes. The problem can be written as an energy minimization, the optimization can be made practical by using an efficient method for gradient evaluation, and convergence can be accelerated by using a variety of multi-resolution strategies. The resulting shapes are not invariant under common operations such as translation, and instead undergo intricate, non-linear transformations.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations

1. Introduction

The problem of robustly generating smooth geometry with guaranteed properties has received much attention in computer graphics. However, the related problem of automatically creating rough, complex, but still highly structured geometries has recently received relatively less attention. The most popular methods include Perlin Noise and its many variants [LLC*10], and similar methods such as hypertexturing [EMP*02]. These methods are widely employed to add details once the overall geometry has been finalized, e.g. as a shader on a pre-existing surface. They are used less during the exploratory, conceptual stage of design, when the overall form of the final geometry is still fluid. Iterated dynamical maps such as those that generate the Mandelbrot set and the closely related Julia sets [Man83] have the potential to assist

at this exploratory stage, as they excel at producing highly complex structures that it would have been difficult or impossible for a designer to conceptualize *a priori*, much less prototype by hand. However, it has never been clear whether it is possible to control these dynamical maps in a way that produce anything other than the now-familiar Mandelbrot "turtle" and Julia "dendrite" shapes.

In this paper, we show that such control is in fact possible. Fundamentally, we ask the question: *What would a Mandelbrot-like set in the shape of a bunny look like?* We present the first algorithm that is capable of answering this question in 3D (Fig. 1). More specifically, we present an optimization method that finds an approximating quaternion Julia set for a given shape. Our specific contributions follow.

- The polynomial Julia sets from previous works contain insufficient degrees of freedom to fit a shape. We introduce the needed degrees of freedom by using the Julia sets of *rational* functions, containing *hundreds* of roots.
- We design an analytically differentiable energy function that characterizes the fit of an iterated quaternion rational map relative to an arbitrary shape.
- The energy function is extremely non-linear, so we propose a multi-level optimization method that uses both spatial and variable coarsening to find minima.
- We present a monopole approximation method that yields effective initial guesses for the optimization.
- A naïve implementation requires $O(R^2)$ time to compute the gradient of R roots. We show that it is possible to compute this quantity in $O(R)$ time.

Our algorithm discovers highly complex shapes that would have been difficult to conceptualize beforehand, and whose character we have never seen before. Each output serves as an initial point in a parameter space that can subsequently be explored using simple operations such as translation. In quaternion space, these simple operations instead induce intricate and highly non-linear transformations (Fig. 1(c)).

In addition to their aesthetic possibilities, these shapes have potential engineering applications. Currently, iterated function systems (IFS) are used to generate shapes that maximize the surface area of heat exchangers [CC02] and radio antennae [WG03]. Instead of limiting these designs to the convex hull of the IFS [Vas13], Julia sets could be used to generate shapes that fit any arbitrary form factor.

2. Related Work

Julia sets were initially investigated early on by Julia [Jul18] and Fatou [Fat17], and gained wider attention when later examined by Mandelbrot [Man80] and Douady and Hubbard [DH82]. Numerous works followed, and the texts by Barnsley [Bar88] and Peitgen and Richter [PR86] are excellent references on developments during the ensuing years.

In computer graphics, Julia sets were extended to 3D in the pioneering work by Norton [Nor82], which took a 3D slice from a 4D quaternion function. Subsequently, ray-tracing methods were developed to visualize these quaternion Julia sets [HSK89]. Interest has been sustained by hobbyists, yielding recent variations such as the Mandelbulb [Whi09], but graphics research on this topic has recently been dormant. Fractal algorithms occasionally appear in geometry processing [Gol04, SLG05], but when the Mandelbrot or Julia sets appear in graphics, it is usually in the form of compute-intensive system tests [Cra05, SK10, PM12].

In contrast, theoretical work on Julia sets has remained robust. Recent interesting results include locating Julia sets that are not computable [BY07], the construction of the Laplacian over Julia sets [FS12] and the construction of 2D Julia sets that approximate an arbitrary shape [Lin14]. This last work is closest to our own, and presents an analytical method for placing rational roots in the complex plane so that they approximate an arbitrary 2D shape. Unfortunately, the analysis does not generalize directly to higher dimensions [Lin13], which motivates our optimization approach.

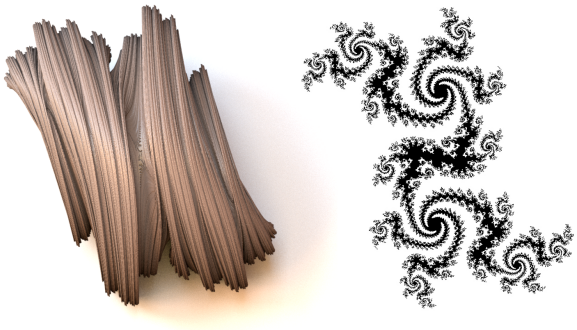


Figure 2: Left: *3D rendering of Julia set with* $\mathbf{c} = [0.285\ 0.485\ 0\ 0]^\top$ Right: *2D slice from the same set. This is a 2D example of a "filled" Julia set. The Julia set itself is the boundary of the black region.*

3. Julia Set Preliminaries

Notation: In this paper, we use the following notation. Greek symbols, e.g. α, and unbolded, capitalized symbols denote scalars, e.g. T. Unbolded, lowercase symbols are used exclusively as counting indices, e.g. i, j, or n. Julia sets and other sets will be denoted in blackboard bold, e.g. $\mathbb{J}$. Lowercase bold denotes a quaternion, e.g. $\mathbf{q} = A + Bi + Cj + Dk = [A\ B\ C\ D]^\top$. The one exception is $\mathbf{x}$, which denotes a 3D Cartesian coordinate, $\mathbf{x} = [X\ Y\ Z]^\top$. When we represent a Cartesian coordinate using a quaternion, we will denote it as $\mathbf{q}_\mathbf{x}$. The first three coordinates of $\mathbf{q}$ are then set to $\mathbf{x}$, e.g. $\mathbf{q}_\mathbf{x} = [X\ Y\ Z\ 0]^\top$, or equivalently, $\mathbf{q}_\mathbf{x} = [\mathbf{x}\ 0]^\top$. This differs from the more conventional $\mathbf{q}_\mathbf{x} = [0\ \mathbf{x}]^\top$, but we have found this necessary, because relatively mundane shapes are obtained if the real component is not varied. Functions of a quaternion that yield another quaternion use

capital bold, e.g. $\mathbf{P}(\mathbf{q}) = \mathbf{q}\mathbf{q} = \mathbf{q}^2$. We define quaternion division as $\frac{\mathbf{p}}{\mathbf{q}} = \mathbf{p}\mathbf{q}^{-1}$.

We can separate $\mathbf{q}$ into real and imaginary components, $\mathbf{q} = [A\ \mathbf{v}]^\top$, where $\mathbf{v} = [B\ C\ D]^\top$. The logarithm is then:

$$\log \mathbf{q} = \begin{bmatrix} \log \|\mathbf{q}\| \\ \frac{\mathbf{v}}{\|\mathbf{v}\|} \cos^{-1}\left(\frac{A}{\|\mathbf{q}\|}\right) \end{bmatrix},$$

where $\|\cdot\|$ denotes magnitude. The exponential is:

$$e^{\mathbf{q}} = e^A \begin{bmatrix} \cos \|\mathbf{v}\| \\ \frac{\mathbf{v}}{\|\mathbf{v}\|} \sin \|\mathbf{v}\| \end{bmatrix}.$$

A quaternion power then combines the two: $\mathbf{q}^\alpha = e^{\alpha \log \mathbf{q}}$.

3.1. Computing a Julia Set

The simplest and most common expression for generating a geometrically complex Julia set is the quadratic polynomial

$$\mathbf{P}(\mathbf{q}) = \mathbf{q}^2 + \mathbf{c}, \tag{1}$$

where $\mathbf{c}$ is a user-defined constant (Fig. 2). Each point in space $\mathbf{x} \in \mathbb{R}^3$ is mapped to a quaternion $\mathbf{q}_\mathbf{x} = [\mathbf{x}\ 0]^\top$, and recursively evaluated:

$$\mathbf{P}_1(\mathbf{q}_\mathbf{x}) = \mathbf{P}(\mathbf{q}_\mathbf{x}), \quad \mathbf{P}_2(\mathbf{q}_\mathbf{x}) = \mathbf{P}(\mathbf{P}(\mathbf{q}_\mathbf{x})) \quad \ldots \tag{2}$$

We denote the nth recursion as $\mathbf{P}_n(\mathbf{q}_\mathbf{x})$. The *filled Julia set* $\mathbb{J}$ is then the set of all $\mathbf{x} \in \mathbb{R}^3$ where the magnitude of $\mathbf{P}_n(\mathbf{q}_\mathbf{x})$ does not approach infinity as $n \to \infty$ (see e.g. [HSK89]):

$$\mathbb{J} = \{\mathbf{x} : \lim_{n\to\infty} \|\mathbf{P}_n(\mathbf{q}_\mathbf{x})\| \nrightarrow \infty\}. \tag{3}$$

An approximation of $\mathbb{J}$ is computed by evaluating $\mathbf{P}_n(\mathbf{q}_\mathbf{x})$ over a grid with a fixed (preferably large) n. Each $\|\mathbf{P}_n(\mathbf{q}_\mathbf{x})\|$ is checked against a radius r in lieu of ∞. The code is simple enough that it is often an introductory example for GPUs [SK10]. Our goal is to find an expression whose $\mathbb{J}$ coincides with the interior of a desired shape.

3.2. Factored Rationals

While Eqn. 1 generates interesting shapes, we are instead interested in factored rationals of the form

$$\mathbf{R}(\mathbf{q}) = e^C \cdot \frac{(\mathbf{q}-\mathbf{t}_1)^{T_1}(\mathbf{q}-\mathbf{t}_2)^{T_2}\ldots(\mathbf{q}-\mathbf{t}_t)^{T_t}}{(\mathbf{q}-\mathbf{b}_1)^{B_1}(\mathbf{q}-\mathbf{b}_2)^{B_2}\ldots(\mathbf{q}-\mathbf{b}_b)^{B_b}} \tag{4}$$

where the set $\mathbb{T} = [\mathbf{t}_1 \ldots \mathbf{t}_t]$ are the roots of the top polynomial, and the set $\mathbb{B} = [\mathbf{b}_1 \ldots \mathbf{b}_b]$ are the roots of the bottom polynomial. We denote the total number of roots as $R = |\mathbb{T}| + |\mathbb{B}|$ and abbreviate Eqn. 4 to $\mathbf{R}(\mathbf{q}) = e^C \frac{\mathbf{T}(\mathbf{q})}{\mathbf{B}(\mathbf{q})}$. The e^C moves the zero level set to different isosurfaces of the potential function. $\mathbf{R}(\mathbf{q})$ has several properties that make it a natural setting for shape optimization.

Property 1: If an iterate $\mathbf{q}$ is sufficiently close to a top root $\mathbf{t}_i$, then it is inside the filled Julia set $\mathbb{J}$. (Fig. 3)

Intuitively, each root of $\mathbf{T}(\mathbf{q})$ *attracts* the space around it towards the interior of $\mathbb{J}$. More concretely, if $\mathbf{q} = \mathbf{t}_i$, it will translate to the origin, force the magnitude of $\mathbf{R}(\mathbf{q})$ to zero, and future iterations can never approach ∞. Similarly, if $\mathbf{q}$ is close to $\mathbf{t}_i$, it will translate close to the origin, and the magnitude $\|(\mathbf{q}-\mathbf{t}_i)^{T_i}\|$ will be small. If it is sufficiently small, it will dominate the other $\mathbf{R}(\mathbf{q})$ terms and force the magnitude to zero after several iterations.

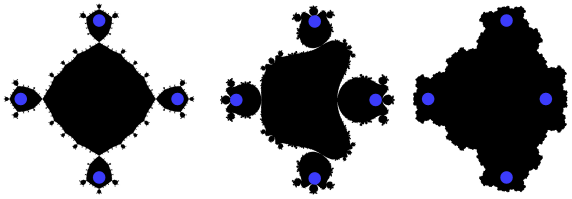


Figure 3: *The $\mathbb{J}$ of a 4-root polynomial, with root locations in blue. The rightmost root is dragged towards the center, and the envelope of $\mathbb{J}$ clearly follows. (Property 1)*

This assumes that $\mathbf{q} = [0\ 0\ 0\ 0]^\top = \mathbf{0}$ is an *attracting fixed point* [Dev92] of the dynamical map. In general, this is not guaranteed. Trivially, setting $\mathbf{c} = [1\ 0\ 0\ 0]^\top$ in Eqn. 1 removes $\mathbf{q} = \mathbf{0}$ as fixed point. However, the special factored form we are interested in, $\mathbf{R}(\mathbf{q})$, effectively sets $\mathbf{c} = \mathbf{0}$, and guarantees the existence of the fixed point.

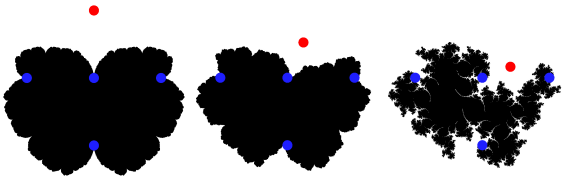


Figure 4: *The $\mathbb{J}$ of a 5-root rational, with the roots in $\mathbb{T}$ in blue and $\mathbb{B}$ in red. As the red root is dragged towards $\mathbb{J}$, $\mathbb{J}$ deforms to exclude it. (Property 2)*

Property 2: If an iterate $\mathbf{q}$ is sufficiently close to a bottom root $\mathbf{b}_i$, then it is outside of $\mathbb{J}$. (Fig. 4)

Intuitively, the roots of $\mathbf{B}(\mathbf{q})$ *repulse* the interior of $\mathbb{J}$. Analogous to the previous property, if $\mathbf{q} = \mathbf{b}_i$, it will translate to the origin, divide by zero, and go instantly to ∞. If $\mathbf{q}$ is close to $\mathbf{b}_i$, it will translate to near the origin, the division will turn its small magnitude into a large one, and if it is sufficiently large, it will dominate $\mathbf{R}(\mathbf{q})$ and shoot it to ∞ after several iterations.

This assumes that $\mathbf{q} = \infty$ is an attracting fixed point of the map, which again is not true in general. A trivial example is the rationals that arise from the Newton-Raphson method, e.g. $\mathbf{N}(\mathbf{q}) = \mathbf{q} - \frac{(\mathbf{q}-\mathbf{t}_1)(\mathbf{q}-\mathbf{t}_2)}{(\mathbf{q}-\mathbf{t}_1)+(\mathbf{q}-\mathbf{t}_2)}$, which due to the absence of this fixed point, generate Julia sets with no bounded envelope [Bar88]. However, for Eqn. 4, this fixed point is guaranteed.

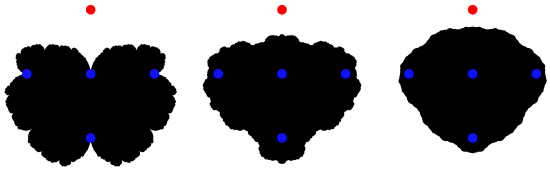


Figure 5: *The multiplicity of the center root is increased from one to three to seven. As its multiplicity grows, so does its attractive influence. (Property 3)*

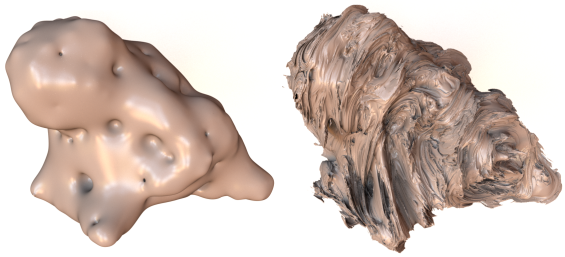


Figure 6: *Frog at 300 optimization iterations. On the left, the broad shape is visible with $n = 1$. On the right, details appear with $n = 4$, but the overall shape remains similar.*

Property 3: The T_i term expands and contracts the attractive region of $\mathbf{t}_i$. The B_i does the same for repulsion with $\mathbf{b}_i$. (Fig. 5)

A $\mathbf{q}$ that was insufficiently close to $\mathbf{t}_i$ for its magnitude to dominate $\mathbf{R}(\mathbf{q})$ can be made dominant by a large T_i. E.g. $\|(\mathbf{q}-\mathbf{t}_i)\| = 0.1$ can be amplified to a dominant $\|(\mathbf{q}-\mathbf{t}_i)^8\| = 1e^{-8}$. The same property applies to B_i.

4. A Shape Optimization Method

Using these properties, we can fit a quaternion Julia set to an arbitrary target shape. Our only assumption is that the shape's signed distance function (SDF), $\phi(\mathbf{x})$, is available.

4.1. The Energy Function

We have observed the following regarding $\mathbf{R}(\mathbf{q})$:

- A single recursion of $\mathbf{R}(\mathbf{q})$, i.e. $\mathbf{R}_1(\mathbf{q})$, is already a useful approximation of $\mathbb{J}$ (Fig. 6). This was not true for Eqn. 1 because of its relatively low quadratic degree: a single recursion yielded a sphere, and more recursions were needed before the overall shape emerged.
- The sign of the potential, $L = \log(\|\mathbf{R}(\mathbf{q})\|)$ can be used to partition $\mathbf{R}(\mathbf{q})$ into inside and outside components.

Using these, we propose the following energy,

$$\psi(\mathbf{R},\phi) = \frac{1}{\eta}\int_{\mathbf{x}\in\mathbb{R}^3} -\tanh(\alpha\log(\|\mathbf{R}(\mathbf{q}_\mathbf{x})\|))\, W(\phi,\mathbf{x})\, d\mathbf{x}. \tag{5}$$

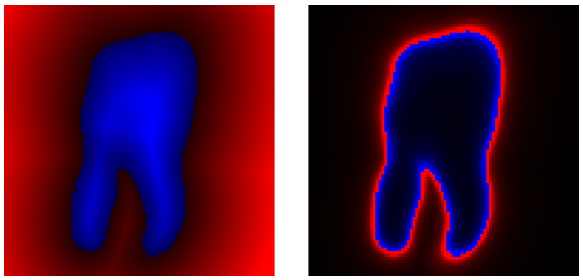


Figure 7: ***Left:*** *Slice through the SDF of the Tooth (Fig.9). Inside is blue, outside is red, and brightness denotes the absolute distance.* ***Right:*** *Inverse SDF of the same slice, Eqn. 6. The weight concentrates on the shape boundary, which is then given more importance by ψ.*

The tanh maps the potential function to the $[1,-1]$ range, and α defines how sharply it approximates a step function. The $W(\phi,\mathbf{x})$ term is an arbitrary weight function that we define as the inverse of the SDF,

$$W(\phi,\mathbf{x}) = \phi(\mathbf{x})^{-1}. \tag{6}$$

The $-\tanh$ maps the inside of $\mathbb{J}$ to 1 and the outside to -1. This is the opposite sign convention as the SDF, so when the two shapes match, they produce a large, negative energy. We normalize the energy using

$$\eta = \int_{\mathbf{x}\in\mathbb{R}^3} |W(\phi,\mathbf{x})|\, d\mathbf{x}. \tag{7}$$

When $\mathbb{J}$ and the target shape match exactly (up to the grid resolution of ϕ), the η^{-1} term forces ψ to -1. If they mismatch exactly, ψ then yields 1. By construction, the energy cannot take on a value outside this range.

We use $\phi(\mathbf{x})^{-1}$ in lieu of $\phi(\mathbf{x})$ because we found that $\phi(\mathbf{x})$ erroneously encourages the optimization to match the shape's medial axis, and the border of the ϕ grid. Fig. 7 discusses this further. Many energies often square the integrand; we preferred not to do this because it worsened the conditioning of an already challenging optimization.

4.2. The Optimization Variables

With the energy function defined (Eqns. 5-7), we now select optimization variables. The C term from §3 is a very effective, as it allows any isosurface of $\mathbf{R}(\mathbf{q})$ to be selected.

The roots of Eqn. 4, $[\mathbf{t}_1 \dots \mathbf{t}_t]$ and $[\mathbf{b}_1 \dots \mathbf{b}_b]$, also appear to be promising candidates for optimization. However, our experiments indicated that this is not the case. In the 2D case [Lin14], the optimal roots were found to lie along the surface of the target curve. The same is likely true in the 3D case, but without a constraint mechanism pinning the roots to the shape surface, the optimizer consistently performs locally optimal but globally counter-productive steps such as

pushing roots outside the bounds of the SDF grid or setting exponents to zero.

Instead of adding complex surface constraints to the optimization, we found it effective to front-load the difficulty of root placement to the initial guess (§4.4), and instead optimize the exponents $[T_1 \dots T_t]$ and $[B_1 \dots B_b]$ in line with Property 3 from §3.2. In addition, we found it very effective to add two global parameters, τ and β, that allow the optimizer to tune all the exponents of $\mathbf{T}$ and $\mathbf{B}$ in tandem:

$$\mathbf{T}(\mathbf{q}) = \prod_{i=1}^{|\mathbb{T}|} (\mathbf{q} - \mathbf{t}_i)^{\tau T_i} \qquad \mathbf{B}(\mathbf{q}) = \prod_{i=1}^{|\mathbb{B}|} (\mathbf{q} - \mathbf{b}_i)^{\beta B_i}. \tag{8}$$

There are now $R+3$ variables to optimize: the global variables $[C\,\tau\,\beta]$ and the root exponents, $[T_1 \dots T_t]$ and $[B_1 \dots B_b]$.

4.3. Building the Energy Gradient

We must now compute the energy gradient for a single cell, $\psi = \psi(\mathbf{R}, \phi, \mathbf{x})$. The full gradient is then a summation over all cells. The derivation is straightforward but tedious, so we describe it in Appendix A.

Fast Gradient Evaluation: The gradient contains terms that take $O(R^2)$ to evaluate, such as:

$$\frac{\partial \mathbf{T}(\mathbf{q})}{\partial \tau} = \sum_{i=1}^{|\mathbb{T}|} \left[\prod_{j=1}^{i-1} \mathbf{t}_j^{\tau T_j}\, T_i \log \mathbf{t}_i \prod_{j=i}^{|\mathbb{T}|} \mathbf{t}_j^{\tau T_j} \right].$$

However, these terms are highly redundant, so caching can reduce their evaluation to $O(R)$. For the top polynomial, we compute forward ($\mathbf{f}_i$) and reverse ($\mathbf{r}_i$) caches in $O(R)$ time,

$$\mathbf{f}_i = \prod_{j=1}^{i} \mathbf{t}^{\tau T_j} \qquad \mathbf{r}_i = \prod_{j=i}^{|\mathbb{T}|} \mathbf{t}^{\tau T_j}, \tag{9}$$

Using these, the product sequence evaluates in $O(R)$ time:

$$\frac{\partial \mathbf{T}(\mathbf{q})}{\partial \tau} = \sum_{i=1}^{|\mathbb{T}|} \mathbf{f}_i\, \tau \log \mathbf{t}_i\, \mathbf{r}_{i+1}. \tag{10}$$

These caches can be computed once per cell and then used for all the gradient terms. The τ and β gradient terms compute in $O(R)$, and the T_i and B_i terms compute in $O(1)$. Thus, these caches allow a cell's gradient to compute in $O(R)$ time.

Similar techniques can be used to accelerate Hessian computation, as the only new feature that appears is another log term. However, runs using the Hessian never converged faster than BFGS, and even with caching, take $O(R^2)$ to evaluate. Thus, we preferred the faster, gradient-only BFGS.

4.4. Computing an Initial Guess

Non-linear optimizers generally require a good initial guess [NW06], which for our problem means initial values for roots positions $\mathbf{t}_i$ and $\mathbf{b}_i$. Simple strategies were attempted, such as random placement in space, along the shape interior, and along the shape surface. Noise distributions guided by surface quantities such as curvature were also tried. In all cases, the results were unacceptable, and made it clear that a tailored solution was needed.

There is no known analytical method for optimal 3D root placement, but we leverage the intuition from 2D. The optimal 2D roots lie along the target curve [Lin14], and there is some evidence that these roots distribute according to the electrostatic potential of that curve [BDM13]. Extrapolating that these hold in 3D, we devised the following strategy.

Computing a Set of Monopoles: We use a *monopole approximation* to a shape's electrostatic potential as an initial guess. Similar methods previously have been used in graphics for acoustic simulations [JBP06]. Given the surface, Γ, of a target shape, we solve for the electrostatic potential p:

$$\nabla^2 P = 0 \tag{11}$$

$$P = 1 \quad \text{along } \Gamma \tag{12}$$

$$P = 0 \quad \text{at } \infty \text{ (Sommerfeld condition).} \tag{13}$$

Many excellent Boundary Element Method (BEM) libraries exist to solve this exterior radiation problem [SAB*14]. Note that this electrostatic potential P is distinct from the "potential" L in §4.1. Once P has been computed, we approximate it using a set of monopoles. Following the strategy of James et al. [JBP06] we define an offset surface Γ_+ as an isosurface displaced by a constant from the zero level set. We then locate monopoles $\mathbf{m}_i$ and weights w_i along Γ that approximate P along Γ_+:

$$P(\mathbf{x}) \approx \sum_{i=1}^{|\mathbb{M}|} \frac{w_i}{\|\mathbf{x} - \mathbf{m}_i\|} \qquad \forall \mathbf{x} \in \Gamma_+. \tag{14}$$

We use a greedy algorithm analogous to "'cubature fitting" [AKJ08] and "orthogonal matching pursuit" [TG07] to find these monopoles and weights. We want to find $|\mathbb{M}|$ monopoles that approximate the values of $P(\mathbf{x})$ at N points along Γ_+. We stack these N samples of P into a vector:

$$\mathbf{p} = [P(\mathbf{x}_1)\ P(\mathbf{x}_2)\ P(\mathbf{x}_3)\ \dots\ P(\mathbf{x}_N)]^\top \in \mathbb{R}^N. \tag{15}$$

For a single location $\mathbf{m}_i$, we can evaluate P at the N locations to form a "candidate" vector:

$$\mathbf{c}(\mathbf{m}_i) = \left[\frac{1}{\|\mathbf{x}_1 - \mathbf{m}_i\|} \quad \frac{1}{\|\mathbf{x}_2 - \mathbf{m}_i\|} \quad \cdots \quad \frac{1}{\|\mathbf{x}_N - \mathbf{m}_i\|} \right]^\top \in \mathbb{R}^N. \tag{16}$$

With $|\mathbb{M}|$ candidates, the vectors form a least squares problem:

$$[\mathbf{c}(\mathbf{m}_1)\ \dots\ \mathbf{c}(\mathbf{m}_{|\mathbb{M}|})][w_1\ \dots\ w_R]^\top = \mathbf{p} \tag{17}$$

We abbreviate this to $\mathbf{C}\mathbf{w} = \mathbf{p}$. While the residual $\mathbf{r} = \mathbf{C}\mathbf{w} - \mathbf{p}$ has a magnitude that exceeds a user-defined threshold ε, monopoles are added to the approximation and additional least squares problems are solved. Monopoles are added greedily. We generate blue noise candidates along the target surface and compute their columns (Eqn. 16). The candidate

with the largest absolute projection onto the residual is added to the approximation. The process is outlined in Alg. 1.

Algorithm 1: computeMonopoleApproximation(ε)

Data: ε = desired monopole accuracy.
begin
  $\mathbf{r} = \mathbf{p}$
  The initial set of monopoles $\mathbb{M} = \varnothing$
  while $||\boldsymbol{r}||_2 > \varepsilon$ **do**
    $\mathbb{C}$ = 100 random monopole candidates
    $\mathbf{m}_{\text{new}} = \arg\max_{\mathbf{m}_i \in \mathbb{C}} |\mathbf{c}(\mathbf{m}_i) \cdot \mathbf{r}|$
    Add $\mathbf{m}_{\text{new}}$ to $\mathbb{M}$
    Build $\mathbf{C}\mathbf{w} = \mathbf{p}$ from $\mathbb{M}$
    Solve for least-squares weights $\mathbf{w}$, update $\mathbf{r} = \mathbf{p} - \mathbf{C}\mathbf{w}$
  return M
end

Building the initial guess: In order to convert the monopole approximation into an initial guess, we use Properties 1 and 2 from §3.2. The locations of the positive and negative poles found by the fit have straightforward interpretations as roots of $\mathbf{R}$. The repulsive roots in $\mathbb{B}$ "hollow out" the concave regions of the target shape, so the monopoles $\mathbf{m}_i$ with negatives weights W_i become the roots in $\mathbb{B}$. The corresponding B_i are set to $|W_i|$. The attractive roots of $\mathbb{T}$ pull $\mathbb{J}$ towards thin, convex features, so the monopoles with positive weights become the roots in $\mathbb{T}$, and the weights becomes the initial values of T_i. This intuition is validated when the roots are visualized (Fig. 8).

4.5. Optimization Strategies

With an energy, its gradient, and an initial guess strategy defined, we can now run an optimization. We tested a variety of optimizers, and found that the limited-memory, variable-metric solver in TAO [MSW*12] yielded the best convergence. As negative values would erroneously flip roots to the other side of the rational, we added non-negativity constraints to $\tau, \beta, [T_1 \ \ldots \ T_T]^\top$ and $[B_1 \ \ldots \ B_B]^\top$.

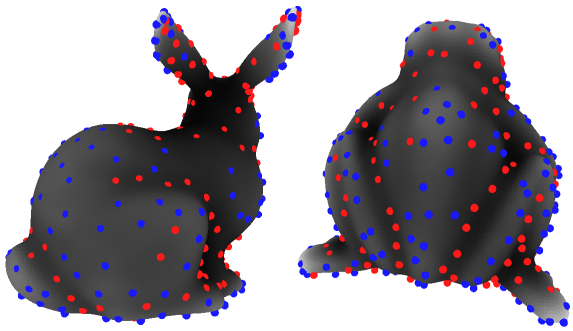


Figure 8: *Roots found by Alg. 1. Surface potential is shown in grayscale, positive roots are blue and negative roots are red. Negative roots tend towards dark, concave regions, while positive roots appear in lighter, convex regions.*

Computing the energy function: Evaluating the energy function ψ and its derivatives is the most time-consuming part of the optimization. From §4.3, $\psi(\mathbf{R}, \phi, \mathbf{q})$ is the energy at a single spatial point. Discretizing Eqn. 5 iterates over all N^3 grid cells in the SDF,

$$\psi(\mathbf{R}, \phi) = \sum_{X=1}^{N} \sum_{Y=1}^{N} \sum_{Z=1}^{N} \psi(\mathbf{R}, \phi, [\mathbf{x}(X,Y,Z)\ 0]^\top), \qquad (18)$$

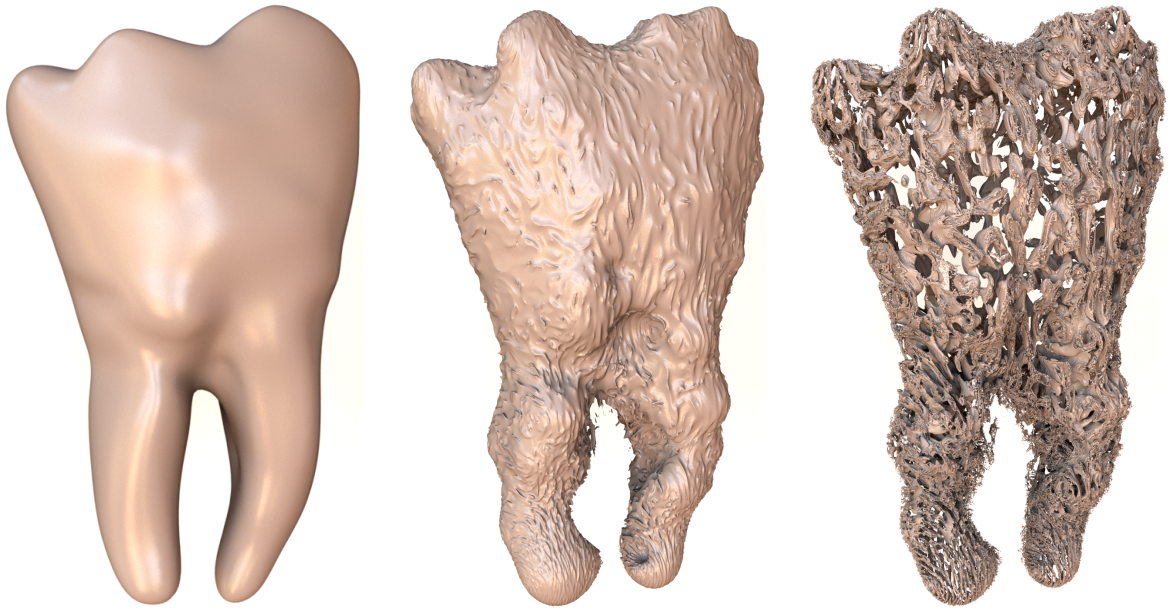
where $\mathbf{x}(X,Y,Z)$ is the cell center at grid index (X,Y,Z). Evaluating the energy and derivatives for a grid cell takes $O(R)$ time, so a single optimization iteration takes $O(RN^3)$. In general, $R \approx 300$ while $N^3 \approx 100^3$, so N is the clear limiter. We will describe several strategies for addressing this complexity, and then combine into a unified solve.

Geometric Coarsening: In the spirit of geometric multigrid methods [BHM00], one natural strategy is to compute the optimization over a hierarchy of coarsened SDFs. None of the terms in our energy function depend on a specific grid spacing, so implementation is straightforward. The normalization constant, Eqn. 7, needs to be recomputed at each level, but this computation is negligible.

Variable Coarsening: Another common strategy is to construct a coarse representation that is tailored to the specific problem, and perform alternating coarse- and fine-scale optimizations on two discrete levels. We introduced the τ and β variables in §4.2 to enable this strategy. The variable-coarsened optimization then takes place over the three global variables C, τ and β instead of all $R+3$ variables This strategy is complementary to geometric coarsening, as the $\frac{\partial \psi}{\partial \tau}$ and $\frac{\partial \psi}{\partial \beta}$ terms still require $O(RN^3)$ time to compute, so coarsening the SDF accelerates their update as well.

Reweighting $W(\phi, \mathbf{q})$: While the two strategies described above make useful optimization progress, we encountered situations where the filled Julia set $\mathbb{J}$ poorly approximated specific geometric features of the target shape. We encouraged the optimization to expend more effort matching specific geometric features by reweighting the $W(\phi, \mathbf{q})$ term in the energy function.

We computed the $\phi_{\mathbb{J}}$ of the current fractal using the fast sweeping method [Zha05], whose running time was negligible compared to the rest of the optimization. Cells that were on the interior of both ϕ and $\phi_{\mathbb{J}}$ were tagged as "well-captured", while those inside ϕ but outside of $\phi_{\mathbb{J}}$ were tagged as "poorly-captured". The values of $W(\phi, \mathbf{q})$ were scaled so the sum of the energies in the well- and poorly- captured regions were equal. This encouraged the optimizer to expend 50% of its effort "filling in" poorly fit regions.



(a) (b) (c)

Figure 9: *(a) The original Tooth. (b) Julia set of the 108-root rational found by our algorithm. (c) Highly intricate surface obtained by translating the roots by 0.54 in the z direction. Image is very high-resolution; please zoom in to see details.*

Optimization Schedule: It is unclear what order to apply the strategies in, i.e. what their *schedule* should be. We found that some minor experimenting was needed to find the best schedule for a shape. This to roughly analogous to choosing between V, W and Full cycles in multigrid.

We will list the optimization schedule for each example as follows. A **F**ull pass involving all $R+3$ variables will be denoted 100(**F**), where the 100 indicates a 100^3 SDF. A **V**ariable-coarsened pass is denoted 100(**V**), and a **R**eweighted pass 100(**R**). Strategies are applied sequentially, so letters can be concatenated. E.g. 25(**VFR**) 50(**VFR**), indicates that a variable-coarsened, full, and reweighted pass took place at 25^3, and then again at 50^3. The schedules for each example are listed in §5.1.

5. Results and Discussion

All computations were run on a 12-core, 2.66 Ghz Mac Pro, and OpenMP was used extensively. As described in §4.5, TAO 2.1 [MSW*12] was used as the optimizer.

Triangles were obtained by using Marching Cubes on the $\log(\|\mathbf{R}_n(\mathbf{q})\|)$ field, with a bisection search performed at each edge. The geometry was rendered in Mitsuba [Jak10]. Rendering methods that rely on a "distance estimators" [HSK89] were unfortunately not directly applicable, because they were formulated for quadratic polynomials, not the arbitrary rationals we use here. Even in the quadratic case, these estimators can yield inferior values if the ray origin is not already close to the surface (§3.4 of Hart et al. [HSK89]). Developing better estimators for arbitrary rationals remains a direction for future work.

Shape Transformation Parameters: For each computed Julia set, we found that a variety of operations produce dramatic, non-linear transformations.

- The shape is not translation invariant. The shape disintegrates as distance from the origin increases.
- The shape is not linear under scaling. Scaling the roots by more than one tears the shape apart, while scaling by less than one forces the shape into a sphere.
- Increasing the slice parameter S, i.e. $\mathbf{q}_\mathbf{x} = [\mathbf{x}\ S]^\top$, causes the shape to gradually disintegrate.

5.1. Algorithm Performance

Monopole Performance: In order to encourage the optimization to focus on broad features, we smoothed each shape using Poisson surface reconstruction [KBH06]. This also yielded more regularly shaped elements for the BEM solve. Algorithm 1 was always run with $\varepsilon = 4\%$, and as seen in Table 1, the running time of this stage never exceeded 5% of the total time.

Optimization Performance: For each shape, we ran the following optimization schedules:

- Tooth (Fig. 9): 25(**VFRF**) 50(**VFRF**) 97(**VFR**)
- Bunny (Fig. 1): 25(**VFRF**) 50(**VFRF**) 97(**VF**)

Example	$\lvert\mathbb{T}\rvert$	$\lvert\mathbb{B}\rvert$	R	BEM solve	Monopole	Optimization	Iterations	Total Time
Tooth	65	43	108	00:45:35 (3%)	00:00:22 (< 1%)	22:44:11 (96%)	7312	23:30:08
Bunny	178	153	331	01:47:08 (4%)	00:04:29 (< 1%)	42:59:10 (96%)	8224	44:50:47
Frog	194	170	364	00:28:25 (41%)	00:05:05 (7%)	00:35:27 (51%)	1379	01:08:57
Armadillo	268	278	546	00:42:05 (15%)	00:07:13 (2%)	03:44:10 (82%)	2675	04:33:28
Dragon	352	356	708	01:15:19 (35%)	00:11:18 (5%)	02:10:59 (60%)	3014	03:37:36

Table 1: *Example running times and complexities, in increasing order. All timings are in* hours:minutes:seconds*. In all cases, the monopole approximation was computed with* $\varepsilon = 4\%$ *in Algorithm 1. The Tooth and Bunny examples were allowed to run for a long time, but the overall shape is captured early on. The Armadillo and Dragon optimizations did not run as long because they both terminated early due to insufficient exponent bits.*

- Frog (Fig. 10): 25(**VFR**) 50(**VFR**) 97(**V**)
- Armadillo (see supplement): 25(**VFR**) 50(**V**)
- Dragon (see supplement): 25(**VFRF**) 50(**V**)

Detailed statistics for each example are listed in Table 1. All computations were run in 80-bit extended precision, and when $R \gtrapprox 350$, they began to run out of bits in the exponent. For all of the optimizations, the overall shape was captured early on, so a recognizable result was obtained if the exponent precision was exceeded. The fitting error per iteration is plotted in Fig. 11. Schedule transitions are visible as the jumps in the plots. The most important finding was that the **V** stage should be run first. Most of the rapid convergence regions correspond to this stage. In many cases, e.g. around step 6000 for the Bunny example, switching strategies clearly defeated a local minimum. In others, e.g. between steps 2000 and 4000 in the Tooth example, switching strategies only resulted in a temporary detour.

5.2. Limitations and Future Work

Limitations: Our main limitation is that 80-bit precision is needed to resolve the large rational division. Even then, the optimization can run out exponent bits. While this is the first work that *can* find a Mandelbrot-like set shaped like a bunny, further work is needed to make it more robust. Many promising avenues exist in this respect, such as rationals in Lagrange form,

$$\mathbf{R}(\mathbf{q}) = e^C \cdot \frac{(\mathbf{q}-\mathbf{t}_1)^{T_1}}{(\mathbf{q}-\mathbf{b}_1)^{B_1}} \frac{(\mathbf{q}-\mathbf{t}_2)^{T_2}}{(\mathbf{q}-\mathbf{b}_2)^{B_2}} \cdots, \tag{19}$$

but investigating these possibilities and their effects on the optimization is beyond our scope.

Our energy function uses an Eulerian SDF, so it predictably has trouble capturing thin features. Many of the stages of in the optimization schedule could clearly have been terminated earlier, e.g. iterations 400 to 7000 for the Tooth did not improve the shape dramatically. Each stage was run until the gradient vanished or the line search failed. Other termination criteria were also attempted, but no reliable results were obtained. More robust criteria could reduce the number of unnecessary iterations.

Future Work: Many design choices were made at each stage of the algorithm, and while they were the ones we found to be most effective, more efficient alternatives are likely to exist. More easily optimized energies, other initial guess strategies, direct analytic methods, and faster rendering methods are all directions for future work.

Some of the shapes exhibit divots at root locations, and these features would reduce if more roots were introduced. Increasing the number of roots increases the precision issues, so finding a rational representation with better conditioning would again be a significant step. Finally, we have only scratched the surface regarding parameters that can be used to manipulate these shapes.

Acknowledgements

A special thanks goes to Dr. Lindsey for answering numerous questions during the initial stages of this work, and for generously sharing her code for the 2D case. The author would also like to thank the reviewers for recognizing the potential of this speculative work, and their detailed feedback. This work was supported by a UCSB Regents Junior Faculty Fellowship, and National Science Foundation CAREER award (IIS-1253948). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. I acknowledge support from the Center for Scientific Computing from the CNSI, MRL: an NSF MRSEC (DMR-1121053) and Hewlett Packard.

Appendix A: The Energy Gradient

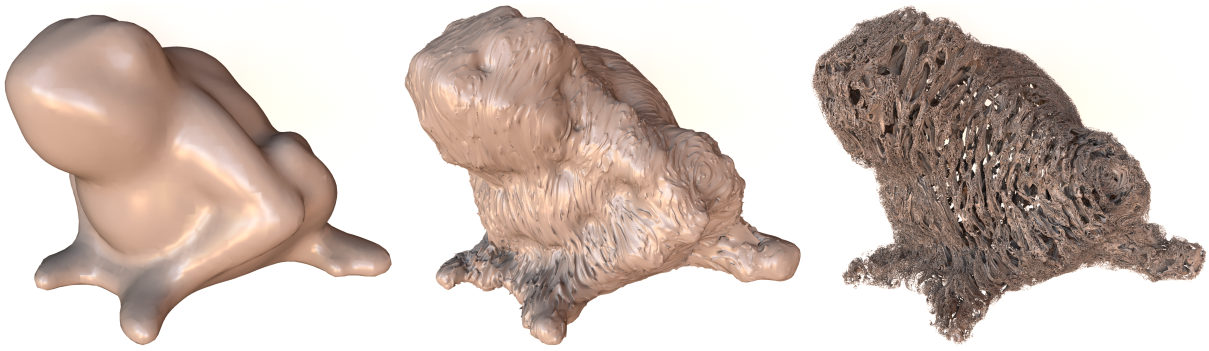
When we have obtained $\frac{\partial \psi}{\partial c}$, $\frac{\partial \psi}{\partial \tau}$, $\frac{\partial \psi}{\partial \beta}$, $\frac{\partial \psi}{\partial T_i}$ and $\frac{\partial \psi}{\partial B_i}$, we are done. For parsimony, we will assume $\mathbf{q} = \mathbf{q}_\mathbf{x}$ in what follows. We start by isolating the log term and its derivative:

$$L = \log(\|\mathbf{R}(\mathbf{q})\|) \tag{20}$$

$$\frac{\partial \psi}{\partial L} = \alpha \cdot W(\phi, \mathbf{q}) \cdot \operatorname{sech}^2(\alpha \cdot L). \tag{21}$$

We use this term in all of the following expressions. Since $\frac{\partial L}{\partial C} = 1$, the chain rule yields our first expression:

$$\frac{\partial \psi}{\partial C} = \frac{\partial \psi}{\partial L} \cdot \frac{\partial L}{\partial C} = \frac{\partial \psi}{\partial L}. \tag{22}$$



(a) (b) (c)

Figure 10: *(a) The original Frog. (b) Julia set of the 364-root rational found by our algorithm. (c) Highly intricate surface obtained by translating the roots by 1.07 in the z direction. Image is very high-resolution; please zoom in to see details.*

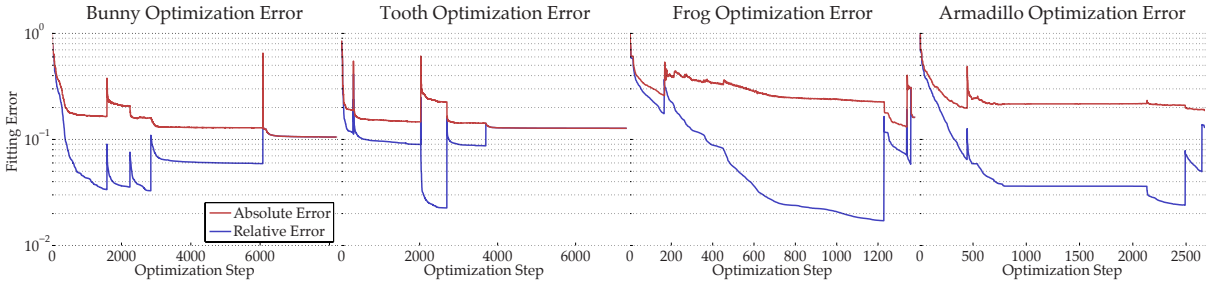


Figure 11: *Fitting energy $\psi(\mathbf{R},\phi)$, per iteration. The values have been normalized from $[-1,1]$ to $[0,1]$ to enable a log scale. The* absolute *error, is computed over all grid cells in ϕ, using the weight function from Eqn. 6. The* relative *error is the myopic energy that the optimizer uses directly, where ϕ may have been coarsened or reweighted. The jumps in energy correspond to switches between stages in the optimization schedule.*

The derivatives with respect to T_i then take the form:

$$\begin{aligned}\frac{\partial\psi}{\partial T_i} &= \frac{\partial\psi}{\partial L}\cdot\frac{\left(\mathbf{R}(\mathbf{q})\cdot e^{C\,\frac{\partial\mathbf{T}(\mathbf{q})}{\partial T_i}\,\mathbf{B}(\mathbf{q})}\right)}{\mathbf{R}(\mathbf{q})\cdot\mathbf{R}(\mathbf{q})}\\ \frac{\partial\mathbf{T}(\mathbf{q})}{\partial T_i} &= \prod_{j=1}^{i-1}\mathbf{t}_j^{\tau T_j}\;\tau\log\mathbf{t}_i\prod_{j=i}^{|\mathbb{T}|}\mathbf{t}_j^{\tau T_j}.\end{aligned}\tag{23}$$

The derivatives with respect to B_i are slightly more subtle:

$$\begin{aligned}\frac{\partial\psi}{\partial B_i} &= \frac{\partial\psi}{\partial L}\cdot\frac{\left(\mathbf{R}(\mathbf{q})\cdot\left(-\mathbf{I}\,e^{C\,\frac{\mathbf{T}(\mathbf{q})}{\mathbf{DB}(\mathbf{q})}}\right)\right)}{\mathbf{R}(\mathbf{q})\cdot\mathbf{R}(\mathbf{q})}\\ \mathbf{DB}(\mathbf{q}) &= \prod_{j=1}^{i-1}\mathbf{b}_j^{\beta B_j}\;(\beta\log\mathbf{b}_i)^{-1}\prod_{j=i}^{|\mathbb{B}|}\mathbf{b}_j^{\beta B_j},\end{aligned}\tag{24}$$

where $-\mathbf{I}=[-1\;0\;0\;0]^\top$. The $\mathbf{DB}(\mathbf{q})$ is almost identical to $\frac{\partial\mathbf{T}(\mathbf{q})}{\partial T_i}$ save for the log inverse. Usually this term would flip to the numerator, but multiplication order must be preserved for quaternions, so we keep it in the denominator.

For the coarse level variable τ, a sum of products appears,

$$\begin{aligned}\frac{\partial\psi}{\partial\tau} &= \frac{\partial\psi}{\partial L}\cdot\frac{\left(\mathbf{R}(\mathbf{q})\cdot e^{C\,\frac{\partial\mathbf{T}(\mathbf{q})}{\partial\tau}\,\mathbf{B}(\mathbf{q})}\right)}{\mathbf{R}(\mathbf{q})\cdot\mathbf{R}(\mathbf{q})}\\ \frac{\partial\mathbf{T}(\mathbf{q})}{\partial\tau} &= \sum_{i=1}^{|\mathbb{T}|}\left[\prod_{j=1}^{i-1}\mathbf{t}_j^{\tau T_j}\;T_i\log\mathbf{t}_i\prod_{j=i}^{|\mathbb{T}|}\mathbf{t}_j^{\tau T_j}\right].\end{aligned}\tag{25}$$

and an analogous sum appears for β,

$$\begin{aligned}\frac{\partial\psi}{\partial\beta} &= \frac{\partial\psi}{\partial L}\cdot\frac{\left(\mathbf{R}(\mathbf{q})\cdot\left(-\mathbf{I}\,e^{C\,\frac{\mathbf{T}(\mathbf{q})}{\mathbf{DB}_\beta(\mathbf{q})}}\right)\right)}{\mathbf{R}(\mathbf{q})\cdot\mathbf{R}(\mathbf{q})}\\ \mathbf{DB}_\beta(\mathbf{q}) &= \sum_{i=1}^{|\mathbb{B}|}\left[\prod_{j=1}^{i-1}\mathbf{b}_j^{\beta B_j}\;(B_i\log\mathbf{b}_i)^{-1}\prod_{j=i}^{|\mathbb{B}|}\mathbf{b}_j^{\beta B_j}\right].\end{aligned}\tag{26}$$

Care must be taken with $\frac{\partial\psi}{\partial B_i}$ and $\frac{\partial\psi}{\partial\beta}$ to avoid dividing by zero. If β or B_i equals zero, the inverse generates a NaN. If we instead think of $\frac{\mathbf{T}(\mathbf{q})}{\mathbf{DB}(\mathbf{q})}$ as $\mathbf{T}(\mathbf{q})\mathbf{DB}(\mathbf{q})^{-1}$, the ordering

reverses, and the inverse is pushed to the exponents:

$$\mathbf{DB}(\mathbf{q})^{-1} = \prod_{j=|\mathbb{B}|}^{i} \mathbf{b}_j^{-\beta B_j}\, \beta \log \mathbf{b}_i \prod_{j=i-1}^{1} \mathbf{b}_j^{-\beta B_j}. \qquad (27)$$

The β now has no inverse, so when $\beta = 0$, it is clear that the product sequence should merely evaluate to zero.

References

[AKJ08] AN S. S., KIM T., JAMES D. L.: Optimizing cubature for efficient integration of subspace deformations. *ACM Trans. on Graphics 27*, 5 (Dec. 2008), 165.

[Bar88] BARNSLEY M. F.: *Fractals Everywhere*. Academic Press, 1988.

[BDM13] BAKER M., DE MARCO L.: Special curves and post-critically finite polynomials. *Forum of Mathematics, Pi 1* (1 2013).

[BHM00] BRIGGS W. L., HENSON V. E., MCCORMICK S. F.: *A Multigrid Tutorial*. SIAM, 2000.

[BY07] BRAVERMAN M., YAMPOLSKY M.: Constructing non-computable Julia sets. In *ACM Symposium on Theory of Computing* (New York, NY, USA, 2007), ACM, pp. 709–716.

[CC02] CHEN Y., CHENG P.: Heat transfer and pressure drop in fractal tree-like microchannel nets. *International Journal of Heat and Mass Transfer 45*, 13 (2002), 2643 – 2648.

[Cra05] CRANE K.: Ray tracing quaternion Julia sets on the gpu, 2005. [http://bit.ly/1mWj764; accessed 28-July-2014].

[Dev92] DEVANEY R. L.: *A First Course in Chaotic Dynamical Systems*. Westview Press, 1992.

[DH82] DOUADY A., HUBBARD J. H.: Itération des polynômes quadratiques complexes. *CR Acad. Sci. Paris 294* (1982), 123–126.

[EMP*02] EBERT D. S., MUSGRAVE F. K., PEACHEY D., PERLIN K., WORLEY S.: *Texturing and Modeling: A Procedural Approach*, 3rd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[Fat17] FATOU P.: Sur les substitutions rationnelles. *Comp. Rend. heb. S. Acad, Sci 164* (1917), 806–808.

[FS12] FLOCK T., STRICHARTZ R.: Laplacians on a family of quadratic Julia sets I. *Transactions of the American Mathematical Society 364*, 8 (2012), 3915–3965.

[Gol04] GOLDMAN R.: The fractal nature of bézier curves. In *Geometric Modeling and Processing* (2004).

[HSK89] HART J. C., SANDIN D. J., KAUFFMAN L. H.: Ray tracing deterministic 3-d fractals. In *Proceedings of SIGGRAPH* (New York, NY, USA, 1989), ACM, pp. 289–296.

[Jak10] JAKOB W.: Mitsuba renderer, 2010. http://www.mitsuba-renderer.org.

[JBP06] JAMES D. L., BARBIČ J., PAI D. K.: Precomputed acoustic transfer: Output-sensitive, accurate sound generation for geometrically complex vibration sources. *ACM Trans. Graph. 25*, 3 (July 2006), 987–995.

[Jul18] JULIA G.: Memoire sur l'iteration des fonctions rationnelles. *Journal de mathématiques pures et appliquées* (1918), 47–246.

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson surface reconstruction. In *Proceedings of the Eurographics Symposium on Geometry Processing* (2006), pp. 61–70.

[Lin13] LINDSEY K.:, 2013. (personal communication).

[Lin14] LINDSEY K.: Shapes of polynomial Julia sets. *Ergodic Theory & Dynamical Systems (in press)* (2014).

[LLC*10] LAGAE A., LEFEBVRE S., COOK R., DEROSE T., DRETTAKIS G., EBERT D., LEWIS J., PERLIN K., ZWICKER M.: State of the art in procedural noise functions. In *Eurographics STAR Reports* (May 2010), Hauser H., Reinhard E., (Eds.), Eurographics Association.

[Man80] MANDELBROT B. B.: Fractal aspects of the iteration of $z \to \lambda z$ (1-z) for complex λ and z. *Annals of the New York Academy of Sciences 357*, 1 (1980), 249–259.

[Man83] MANDELBROT B. B.: *The fractal geometry of nature*. W. H. Freeman, New York, 1983.

[MSW*12] MUNSON T., SARICH J., WILD S., BENSON S., MCINNES L. C.: *TAO 2.0 Users Manual*. Tech. Rep. ANL/MCS-TM-322, Mathematics and Computer Science Division, Argonne National Laboratory, 2012. http://www.mcs.anl.gov/tao.

[Nor82] NORTON A.: Generation and display of geometric fractals in 3-d. In *Proceedings of SIGGRAPH* (1982), pp. 61–67.

[NW06] NOCEDAL J., WRIGHT S.: *Numerical Optimization*, 2nd ed. Springer, 2006.

[PM12] PHARR M., MARK W. R.: ispc: a SPMD compiler for high-performance cpu programming. In *Innovative Parallel Computing Conf.* (May 2012).

[PR86] PEITGEN H.-O., RICHTER P. H.: *The Beauty of Fractals*. Springer-Verlag, 1986.

[SAB*14] ŚMIGAJ W., ARRIDGE S., BETCKE T., PHILLIPS J., SCHWEIGER M.: Solving boundary integral problems with BEM++. *ACM Trans. on Math. Software (to appear)* (2014).

[SK10] SANDERS J., KANDROT E.: *CUDA by Example: An Introduction to General-Purpose GPU Programming*, 1st ed. Addison-Wesley Professional, 2010.

[SLG05] SCHAEFER S., LEVIN D., GOLDMAN R.: Subdivision schemes and attractors. In *Proceedings of the Eurographics Symposium on Geometry Processing* (2005).

[TG07] TROPP J. A., GILBERT A. C.: Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory 53*, 12 (2007), 4655–4666.

[Vas13] VASS J.: *On the Geometry of IFS Fractals and its Applications*. PhD thesis, University of Waterloo, Waterloo, ON, 2013.

[WG03] WERNER D., GANGULY S.: An overview of fractal antenna engineering research. *IEEE Antennas and Propagation Magazine 45*, 1 (2003), 38–57.

[Whi09] WHITE D.: The unravelling of the real 3d Mandelbulb, 2009. [http://www.skytopia.com/project/fractal/mandelbulb.html; accessed 28-July-2014].

[Zha05] ZHAO H.: A fast sweeping method for eikonal equations. *Mathematics of Computation 74*, 250 (2005), 603–627.