Eurographics Symposium on Geometry Processing 2023
P. Memari and J. Solomon
(Guest Editors)

Volume 42 (2023), Number 5

A Shape Modulus for Fractal Geometry Generation

A.L. Schor and Theodore Kim

Yale University, U.S.A.

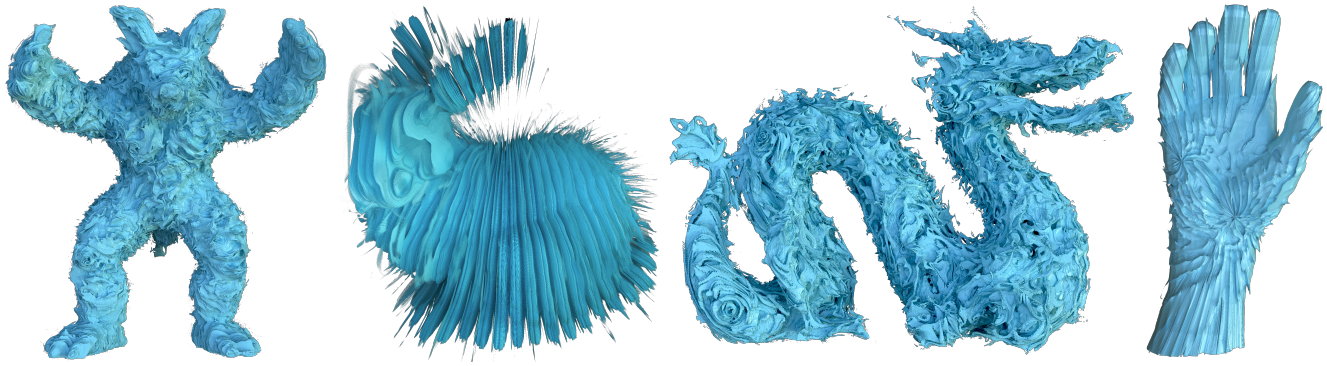

Figure 1: *Four quaternion Julia sets with different target shapes and fractal styles, generated using our method. Images are high-resolution; zoom in to see detail. The results were obtained orders of magnitude faster than with the state-of-the-art method [Kim15], with similar quality.*

Abstract

We present an efficient new method for computing Mandelbrot-like fractals (Julia sets) that approximate a user-defined shape. Our algorithm is orders of magnitude faster than previous methods, as it entirely sidesteps the need for a time-consuming numerical optimization. It is also more robust, succeeding on shapes where previous approaches failed. The key to our approach is a versor-modulus analysis of fractals that allows us to formulate a novel shape modulus *function that directly controls the broad shape of a Julia set, while keeping fine-grained fractal details intact. Our formulation contains flexible artistic controls that allow users to seamlessly add fractal detail to desired spatial regions, while transitioning back to the original shape in others. No previous approach allows Mandelbrot-like details to be "painted" onto meshes.*

CCS Concepts

• ***Computing methodologies*** → ***Computer graphics; Shape modeling;***

1. Introduction

Julia sets are a geometrically intricate family of fractals that continuously reveal new details under magnification. Their unique visual properties have been highlighted in many films over the last decade, including *Big Hero 6* [HRE*15], *Doctor Strange* [Sey16], and *Annihilation* [Fai18]. More broadly, the visual appeal of fractals has been leveraged in *Lucy* [KFB14], *Guardians of the Galaxy 2* [ESHG17] and production VR [SMB*18]. Fractal shapes produce unique visual structures at coarse and fine scales that are difficult to replicate with noise-based methods.

However, *controlling* these coarse- and fine-grained fractal details remains a significant challenge. Popular applications such as those for the Mandelbulb [Whi09; Mar19] take a time-consuming "explorer" approach where the user goes spelunking through fractal landscapes in the hopes of discovering new shapes. Film production pipelines hoping to incorporate fractal effects have had to adopt similar manual-exploration approaches [Gia17; Sey17] or resort to hand-sculpting generated fractal geometry [Sey16]. The difficulty of artistically directing of fractals are a well known challenge in VFX. As observed by one Weta supervisor: *The problem is you can't control fractals, and filmmaking requires control.* [Gia17]

In this paper, we present a fast and flexible new method for directly generating Julia sets corresponding to a user-desired shape. Using a new *versor-modulus* factorization, we show that fine-

grained fractal detail can be controlled through the versor component, and introduce a new *shape modulus* function to control the coarse, global shape. Our formulation allows fractal detail to be added in user-specified regions while seamlessly transitioning back to the original shape in others. To our knowledge, no prior formulation offers this level of control.

Previous work computed shape-conforming Julia sets using a time-consuming and numerically brittle optimization process that required extended precision floating point (see supplement §1, [Kim15]). Our method is orders of magnitude faster because it bypasses the optimization stage, is able to compute in standard precision, and immediately produces fractal versions of input shapes.

Our contributions are as follows:

- A versor-modulus analysis of quaternion Julia sets that reveals that the coarse-scale details are determined by the modulus, and the fine-scale details are set by the versor.
- A new *shape modulus* function that allows direct control of the overall shape of a Julia set while still maintaining its fractal details.
- Spatially varying controls that allows fractal detail to be added in user-specified regions, while seamlessly transitioning back to the original shape in others.

2. Background and Related Work

2.1. 3D Noise and Surface Texturing

An effective Julia set shape control method would produce geometry that resembles some target shape, so it could be broadly grouped with other surface texturing methods.

Many well-known texturing methods exist, such as hypertextures [PH89], 3D texels [KK89] or shell maps [PBFJ05], which can be applied to a surface using standard displacement mapping [Coo84]. However, Julia set-based surface texturing produces detailed structures across scales, such as the rosette on the dragon's tail in Figure 1 or the rings around the tail in Figure 11. Such features lie outside the characteristic looks generated by commonly used Perlin [Per85], Wavelet [CD05] or Gabor [LLD11] noise functions. In Figure 1, the fractal detail radically changes the connectivity and genus of the target shape, but the changes are localized to the high-frequency bands of detail, while the low-frequency characteristics of the overall shape are preserved. Equivalent looks would be difficult with existing noise methods.

2.2. Julia Set Preliminaries

Julia sets were first studied over a century ago [Jul18; Fat17], and became more widely-known when their fractal properties were investigated and popularized in the 1980s [Man80; DH85; Gle87].

Julia sets are produced by recursively iterating a dynamical map, and a *filled Julia set* is the set of spatial points whose magnitude does not approach infinity upon successive iterations. The Julia set itself is the boundary of this region. More formally, the filled Julia set $\mathbb{J}$ can be defined in terms of some map $\mathbf{f}$ on the complex numbers:

$$\mathbf{f}(\mathbf{x}) : \mathbb{C} \rightarrow \mathbb{C}$$
$$\mathbf{f}_1(\mathbf{x}) = \mathbf{f}(\mathbf{x}) \quad \mathbf{f}_2(\mathbf{x}) = \mathbf{f}(\mathbf{f}(\mathbf{x})) \quad \mathbf{f}_3(\mathbf{x}) = \mathbf{f}(\mathbf{f}(\mathbf{f}(\mathbf{x}))) \quad \ldots$$

Set membership of a complex point $\mathbf{x} \in \mathbb{C}$ is then determined by:

$$\mathbb{J}(\mathbf{f}) = \{\mathbf{x} : \lim_{n\rightarrow\infty} ||\mathbf{f}_n(\mathbf{x})|| \not\rightarrow \infty\}$$

In practice, some finite maximum iteration count $n_{\max} \in \mathbb{N}$ and radius $r_{\max} \in \mathbb{R}$ are selected. If $r_{\max}$ is exceeded at any point up to $n_{\max}$ iterations, the point is labelled as "escaped", otherwise it is considered to be within $\mathbb{J}$, i.e. $\mathbb{J}(\mathbf{f}) = \{\mathbf{x} | \neg\exists n < n_{\max} : ||\mathbf{f}_n(\mathbf{x})|| > r_{\max}\}$. The classic Julia set is computed by running such an iteration for every point on a grid in the complex plane and coloring pixels corresponding to set membership (see right).

While Julia sets were originally defined over the 2D complex plane, they were extended the computation to 3D by computing a Julia set over the 4D quaternions, and extracting a 3D slice [Nor82]. Similar to 2D, these sets can be computed using $\mathbf{f}(\mathbf{x}) : \mathbb{H} \rightarrow \mathbb{H}$, where $\mathbf{x}$ is now a quaternion. Custom methods were later developed for visualizing these sets [HSK89; HKS90], which then went on to explore whether they were simply or multiply connected [Har99].

However, the original Julia sets take on abstract forms that, while visually interesting, do not generally form recognizable shapes. For many applications, it would be desirable to control the macro-scale of the Julia set's shape while still somehow retaining its characteristic fractal appearance. This creates an inverse problem: *can we find a function whose Julia set resembles some target shape?*

2.3. Previous Julia Set Fitting Methods

Lindsey [Lin15] first investigated this question of inverse fitting in 2D, and presented an analytic method for creating rational polynomials whose Julia sets approximate any Jordan curve in the complex plane. However, the method relied heavily on results from complex analysis that do not generalize easily to 3D.

The QUIJIBO algorithm [Kim15] is a method for controlling Julia sets that is currently the only known method for solving the inverse problem in 3D. Our work builds on intuition gained from analyzing from results from this previous method. We summarize the method here to facilitate comparisons.

2.3.1. QUIJIBO Algorithm

QUIJIBO [Kim15] solves the inverse problem by performing nonlinear optimization on an very high-order rational function containing hundreds of roots:

$$\mathbf{f}(\mathbf{x}) = e^C \cdot \frac{(\mathbf{x}-\mathbf{t}_1)^{\tau T_1}(\mathbf{x}-\mathbf{t}_2)^{\tau T_2}\ldots(\mathbf{x}-\mathbf{t}_t)^{\tau T_t}}{(\mathbf{x}-\mathbf{b}_1)^{\beta B_1}(\mathbf{x}-\mathbf{b}_2)^{\beta B_2}\ldots(\mathbf{x}-\mathbf{b}_b)^{\beta B_b}}$$

To form an initial guess, the top polynomial's roots ($\mathbf{t}_1$ through $\mathbf{t}_t$) and the bottom polynomial's roots ($\mathbf{b}_1$ through $\mathbf{b}_b$) are placed according to a monopole approximation of the target shape's electrostatic potential. Previous results from 2D analysis ([BD13]) suggested that this could produce a close approximation.

Given the root positions from this monopole approximation, the root powers τ, β, $T_1 \dots T_t$ and $B_1 \dots B_b$ are sent to a non-linear optimizer that tries to match the signed distance field of the target shape to the potential field of the Julia set. This proved to be challenging optimization problem, as it intrinsically involves a highly non-linear rational function. Several coarsening and re-weighting strategies had to be devised to coax the optimization to converge.

This optimization could take hours to days (the maximum reported time was 44 hours), and would fail to converge for certain shapes. If the optimization converged, a final mesh was extracted by running Marching Cubes on the scalar field of $\log(||\mathbf{f}_n(\mathbf{x})||)$ for some finite n. This step was also computationally intensive, because $\mathbf{f}(\mathbf{x})$ often involved numerically sensitive rational functions that needed to be evaluated in 80-bit precision. However, this stage could be amortized over a parallel compute cluster, so the reliability of the non-linear optimization remained the main challenge.

While the QUIJIBO optimization converged for simpler shapes, it failed on more complicated geometry (see Figure 4). The extended (80-bit) precision requirement of the Marching Cubes stage was addressed in subsequent work [KD20], but the problems surrounding the core non-linear optimization remains. Even when the optimization succeeded, the rational function it outputs resists artistic direction. Small changes to the root's spatial positions or exponents could completely disintegrate the underlying fractal shape. We summarize the differences between our algorithm and QUIJIBO in Table 1.

We present a method that fits arbitrary shapes to Julia sets without any of these limitations. Our method *does not require a non-linear optimization* and instead produces a fractal shape approximation immediately. We show that our algorithm is more robust than previous methods, offers uniquely intuitive artistic controls, and succeeds where prior algorithms failed.

3. A Versor-Modulus Analysis of Julia Sets

We will begin by first analyzing the shapes obtained by the QUIJIBO algorithm [Kim15] in order to motivate our design. We can decompose the output of any quaternion function $\mathbf{f}(\mathbf{x}) : \mathbb{H} \to \mathbb{H}$, into its *versor* function $\mathbf{d}(\mathbf{x})$ and its *modulus* function $r(\mathbf{x})$ [Nee97]:

$$r(\mathbf{x}) = \|\mathbf{f}(\mathbf{x})\| \qquad \mathbf{d}(\mathbf{x}) = \frac{\mathbf{f}(\mathbf{x})}{\|\mathbf{f}(\mathbf{x})\|}$$

Intuitively, we are factoring each quaternion into its magnitude (modulus) and direction (versor). We have discovered that this factorization separates the *structure* (modulus) from the *texture* (versor) of the underlying fractal.

Figure 2 shows a 2D slice from a shape obtained using the QUIJIBO algorithm. The center plot shows the log of the modulus $r(\mathbf{x})$: the radius of the 4-sphere onto which $\mathbf{x}$ will be projected at the next iteration. The outline of the Julia set itself is drawn in white. The plot on the right is an RGB map of the versor field $\mathbf{d}(\mathbf{x})$. The overall structure of the Julia set is *already visible* in $r(\mathbf{x})$.

The basin of attraction around $\|\mathbf{x}\| = 0$ and towards $\|\mathbf{x}\| = \infty$ are also visible, as illustrated on the left of Figure 3. The white circle lies in a region around the origin that forms a basin of attraction towards the *interior* of the Julia set. Once an $\mathbf{f}_i(\mathbf{x})$ iterate enters this zone, it can never escape, and will spiral closer to the origin for all future iterations, guaranteeing membership in $\mathbb{J}$. Similarly, the black circle lies in a region sufficiently far from the origin that it forms a basin of attraction *away* from the Julia set. Iterates will amplify towards infinity in subsequent iterations, and by definition not become members of $\mathbb{J}$.

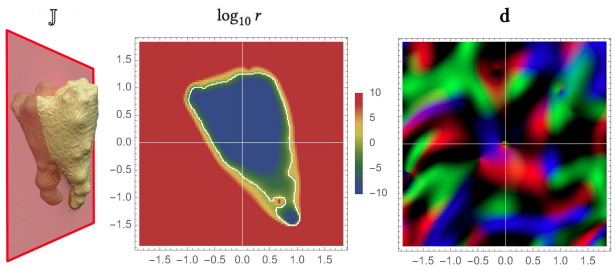

Figure 2: ***Left:*** *An optimization result from the QUIJIBO code release [Kim17], using a "tooth" as a target shape.* ***Middle:*** *Modulus field $r(\mathbf{x})$ as defined in section 3, plotted on a truncated log scale along the slice shown by the 2D plane on the left. The Julia set boundary is outlined in white.* ***Right:*** *Versor field $\mathbf{d}(\mathbf{x})$ plotted as RGB along the same slice. While $\mathbf{d}(\mathbf{x})$ is a quaternion, it lies along the $||\mathbf{d}(\mathbf{x})|| = 1$ simplex, so three colors suffice for visualization; here color encodes the three imaginary components of the quaternion.*

The dynamics near these basins of attraction are *almost entirely independent* of $\mathbf{d}(\mathbf{x})$ (Figure 2, right). If $\mathbf{r}(\mathbf{x})$ is sufficiently small or large, it dominates the fate of the iterate, and the angular direction is of little consequence. The cases where $\mathbf{d}(\mathbf{x})$ *does* influence membership in $\mathbb{J}$ is illustrated by the cyan circle on the right of Figure 3. In this case, portions of the circle lie in different super-attracting regions, and $\mathbf{d}(\mathbf{x})$ selects between the two.

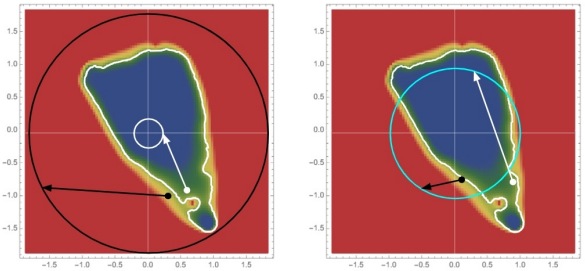

Figure 3: *Two plots showing r(***x***) from Figure 2, with arrows from* $\mathbf{x}$ *to* $\mathbf{f}(\mathbf{x})$*. Each plot shows one* $\mathbf{x}$ *which escapes outside* $\mathbb{J}$ *(black) and one* $\mathbf{x}$ *which does not (white).* ***Left:*** *two values of* $\mathbf{x}$ *project to radii where* $\mathbf{d}$ *becomes irrelevant. The white circle lies entirely in the basin of attraction around* $\|\mathbf{x}\| = 0$*, and the black circle lies entirely in the basin for* $\|\mathbf{x}\| = \infty$*.* ***Right:*** *Two values of* $\mathbf{x}$ *project to the same* $\mathbf{r}$ *but different* $\mathbf{d}$*, determining membership in* $\mathbb{J}$*. The cyan circle does not lie entirely within either basin at* 0 *or* ∞*.*

Method	Iterated Function	Parameters
QUIJIBO	$\mathbf{p}(\mathbf{x}) = e^C \cdot \frac{(\mathbf{x}-\mathbf{t}_1)^{\tau T_1}(\mathbf{x}-\mathbf{t}_2)^{\tau T_2}\ldots(\mathbf{x}-\mathbf{t}_t)^{\tau T_t}}{(\mathbf{x}-\mathbf{b}_1)^{\beta B_1}(\mathbf{x}-\mathbf{b}_2)^{\beta B_2}\ldots(\mathbf{x}-\mathbf{b}_b)^{\beta B_b}}$	• $\mathbf{t}_1 \ldots \mathbf{t}_n$ and $\mathbf{b}_1 \ldots \mathbf{b}_n$, placed to approximate a target shape's electrostatic potential, where $n \approx 300$. • τ, β, $T_1 \ldots T_n$ and $B_1 \ldots B_n$ are optimized over hours or days to minimize a surface-based objective function. • Changes to any of the $n \approx 300$ parameters can completely dissolve the overall shape
Shape Modulus (Ours)	$\mathbf{p}(\mathbf{x}) = e^{\alpha \cdot (\phi(\mathbf{x})+\beta)} \cdot \mathbf{d}(\mathbf{x})$	• $\mathbf{d}(\mathbf{x})$ can be any quaternion function; it does not influence the overall shape of the Julia set • α and β provide an artistically intuitive 2D parameter space, yielding a family of target-fitting Julia sets.

Table 1: *Summary of our method compared to QUIJIBO [Kim15].*

Our key observation is that while $r(\mathbf{x})$ does not entirely determine every detail of the Julia set, it does define its overall shape. We leverage this intuition to design a version of $r(\mathbf{x})$ that has knowledge of the user-defined target shape.

3.1. A Shape-Modulus Function

Any quaternion polynomial $\mathbf{p}(\mathbf{x})$ can be expressed as a versor and modulus:

$$\mathbf{p}(\mathbf{x}) = r(\mathbf{x}) \cdot \mathbf{d}(\mathbf{x}) = \|\mathbf{p}(\mathbf{x})\| \cdot \frac{\mathbf{p}(\mathbf{x})}{\|\mathbf{p}(\mathbf{x})\|}. \tag{1}$$

We instead define $\hat{r}(\mathbf{x})$, a new *shape modulus* function that contains shape information, while leaving the versor function the same:

$$\hat{r}(\mathbf{x}) = e^{\alpha \cdot (\phi(\mathbf{x})+\beta)} \qquad \mathbf{d}(x) = \frac{\mathbf{p}(\mathbf{x})}{\|\mathbf{p}(\mathbf{x})\|}. \tag{2}$$

Our new dynamical map then becomes:

$$\mathbf{f}(\mathbf{x}) = \hat{r}(\mathbf{x}) \cdot \mathbf{d}(\mathbf{x}) \tag{3}$$

Here, $\mathbf{p}(\mathbf{x})$ is some rational quaternion function, $\phi(\mathbf{x})$ is a signed distance field (SDF) of the target surface, and α and β are control variables. The iterations occur in $\mathbb{H}$, while $\phi(\mathbf{x})$ is only defined over $\mathbb{R}^3$, so we discard one quaternion component when indexing into ϕ, effectively extruding the SDF along the discarded axis. For the examples presented here, the third complex coordinate (k) was discarded, but choosing any other coordinate yielded qualitatively similar results. To avoid this extrusion becoming visible in the final result, a 3D slice transverse to the discarded axis is selected for meshing.

We designed $\hat{r}(\mathbf{x})$ to place the $\|\mathbf{x}\| = 0$ super-attracting basin within the target shape, the $\|\mathbf{x}\| = \infty$ basin outside the target shape, and to smoothly transition between the two so that the versor field $\mathbf{d}(\mathbf{x})$ remains relevant. The target shape then overlaps the region where fractal details appear.

The prior formulation [Kim15] had to carefully select the root positions of the rational function $\mathbf{p}(\mathbf{x})$ and then run a non-linear optimization over the course of hours or days. Our new formulation requires *neither* of these steps. We have found that almost *any* function $\mathbf{p}(\mathbf{x})$ suffices to produce a fractal shape approximation (see §3.3.2 for further discussion). Despite this apparent simplicity, Figure 4 shows our formulation succeeding on inputs where the prior algorithm failed. Our algorithm found an armadillo fractal in 2.9 seconds, whereas QUIJIBO optimized for 4.5 hours before failing. The dragon was found in 5.1 seconds, while QUIJIBO failed after 3.5 hours. From this perspective, our algorithm is respectively over 5600× and 2500× faster than the previous approach.

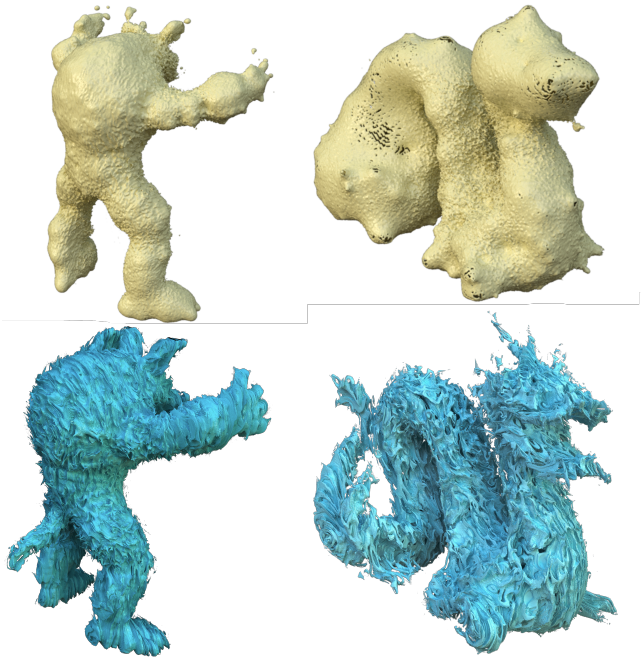

Figure 4: ***Top:*** *Failed Julia set fits included in the QUIJIBO code release [Kim17].* ***Bottom:*** *Our method successfully fits both shapes, and completed up to* 5600× *faster.*

Figure 5: *A bunny-shaped Julia set with a thick shell of fractal detail. The origin of the dynamical system has been translated, causing the interior of the set to erode.*

3.2. Control Parameters

The α and β in Equation 2 allow for fine-grained control of the Julia set (Figure 6). They respectively control the steepness and position of $\hat{r}(\mathbf{x})$ with respect to the shape $\phi(x)$, and enable high-level control of the *shell* around the target shape where fractal details can appear (Figure 8). We call α the *thinness* parameter, as larger α yields progressively thinner shells of fractal detail around the target shape. We call β the *offset* parameter, as it controls the distance from the target surface that fractal details will appear, effectively expanding or contracting the centerline of the shell.

These parameters do not *solely* define the thickness and position of the shell of detail, as the geometry of the target shape and the scale and origin of the coordinate system are also influential factors, but they can be used to *tune* the thickness and position of the shell. Increasing α always yields a thinner shell, and increasing β always expands the shell outwards.

We have found that these two controls are highly effective for designing custom Julia sets. They allow a user to decide which details from the target shape can be eroded away by fractal detail, and provide a wide variety of overall fractal appearances. Since α and β create continuous changes in the output shape, they can be spatially varied to smoothly "dial in" fractal detail on different parts of the target shape without creating visual discontinuities (Figures 7 and 13). This results in intuitive artistic controls: more fractal detail can be dialed in to a specific region by assigning it a small α, while thin features can be preserved in other regions by setting β to be large. Prior methods [Kim15; Lin15] do not provide controls of this kind; any adjustments to the results cause wildly unpredictable and counter-intuitive changes in overall shape.

One desirable feature of the QUIJIBO approach is that translating the optimized polynomial roots "erodes" the Julia set from the inside outward and creates intricate new details. Our formulation produces a similar effect (see Figure 5) when translating the origin of the dynamical system. This additional control allows for further exploration of possible textures and looks.

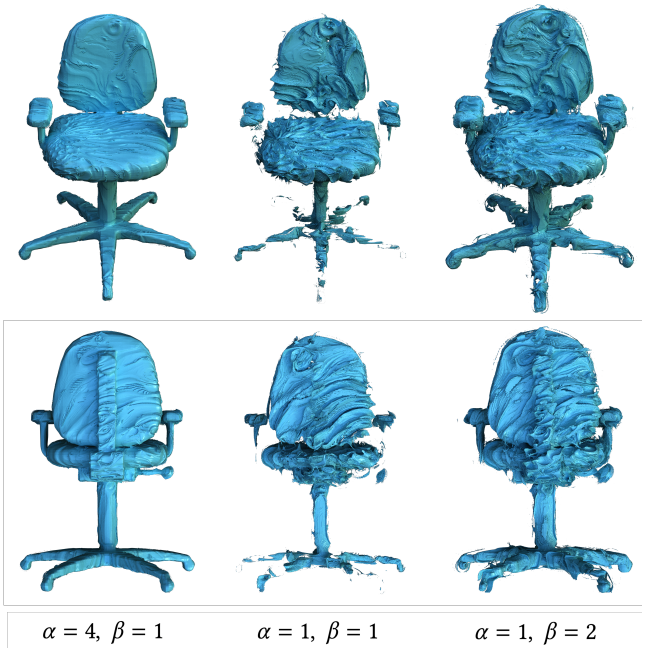

Figure 6: *Two views of the "*Breakfaster *office chair" [Cro22] Julia sets generated with the same ϕ and $\mathbf{d}$, but different α and β. The α parameter controls the thickness of the shell where fractal detail can appear, and β controls the position of that shell relative to the target surface.* ***Left:*** *Initial values for α and β yield a chair with a thin shell of fractal detail around the surface.* ***Middle:*** *Decreasing α makes the shell thicker, adding more fractal detail, but eroding the legs and armrests.* ***Right:*** *Increasing β moves the shell outwards, ensuring that the thin legs, armrests and adjustment handle are fully captured.*

3.3. Discussion

3.3.1. Modulus Function

To better understand the effect of our new modulus function $\hat{r}(\mathbf{x})$, we can examine its behavior under iteration while making no assumptions about the versor function. Instead, we can treat $\mathbf{d}(\mathbf{x})$ as an unknown function that returns a unit vector in any direction. In this case, we can interpret $\hat{r}(\mathbf{x})$ as altering the *probability* that an iterate will fall into the $\|\mathbf{x}\| = 0$ or $\|\mathbf{x}\| = \infty$ basins of attraction.

If we assume that α is sufficiently large, and that the origin lies inside the target shape, our $\hat{r}(\mathbf{x})$ ensures that points far enough inside the target object, with a sufficiently negative $\phi(\mathbf{x})$, will be captured by the $\|\mathbf{x}\| = 0$ basin. Conversely, it ensures that points far enough outside the target object, with a sufficiently positive $\phi(\mathbf{x})$, will be captured by the $\|\mathbf{x}\| = \infty$ basin. For points between these

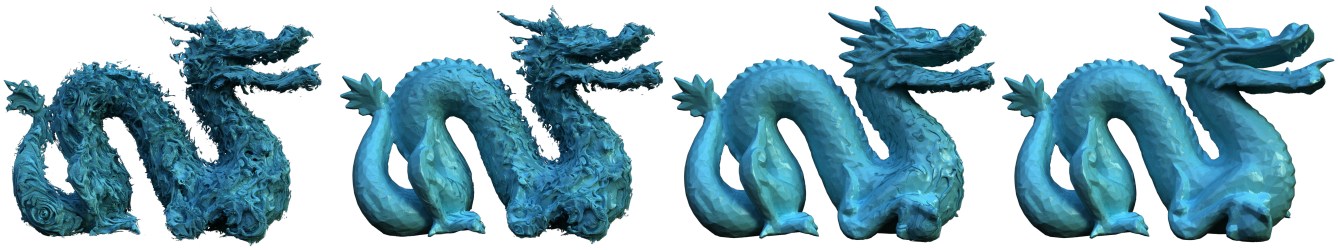

Figure 7: *Four dragon-shaped Julia sets with spatially-varying* α*. The leftmost has a constant* $\alpha = 10$*, the rightmost set has a constant* $\alpha = 300$*, and the two in the middle show a spatially varying* α *that transitions between them. The effect of* α *is non-linear, so we obtained a smooth transition by using a cubic interpolant.*

two extremes, where $\hat{r}(\mathbf{x})$ guarantees neither inclusion nor exclusion, $\phi(\mathbf{x})$ can be used to estimate the *likelihood* of set membership. Specifically, for a given target shape's $\phi(\mathbf{x})$, it is possible to compute the relationship between the radius of an origin-centered sphere and the average $\phi(\mathbf{x})$ along surface of that sphere. We show a 2D case in Figure 8.

Due to the structure of $\hat{r}(\mathbf{x})$, and again assuming that the origin lies inside the target shape, smaller values of $\hat{r}(\mathbf{x})$ mean that the iterate is more likely to fall into the $\|\mathbf{x}\| = 0$ basin, while larger values are biased towards the $\|\mathbf{x}\| = \infty$ basin. The overall shape of the distance-radius curve in Figure 8 determines the spatial distribution of fractal detail in the final Julia set.

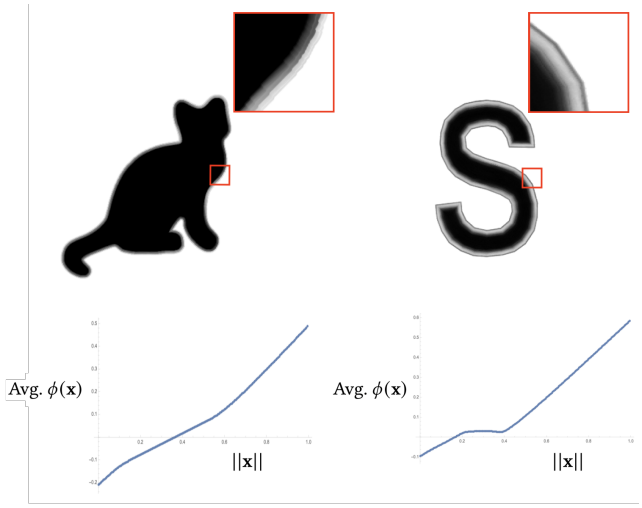

Figure 8: ***Top:*** *Average of 1000 2D Julia sets generated with our technique for two target shapes. Each used randomly-generated* $\mathbf{d}(\mathbf{x})$*. Darker regions correspond to points more likely to be inside* $\mathbb{J}$*. Black areas are inside* $\mathbb{J}$*, and white areas are outside.* ***Bottom:*** *With the origin inside the target shape, a plot of average* $\phi(\mathbf{x})$ *along a circle's circumference over its radius. Sufficiently small or large values of* $\phi(\mathbf{x})$ *will dictate set membership, so* $\mathbb{J}$ *will match the target shape.*

3.3.2. Versor Function

Though $\hat{r}(\mathbf{x})$ determines the overall shape of the Julia set, $\mathbf{d}(x)$ determines the small-scale character of the fractal detail. As specified in Equation 2, all of our Julia sets were computed by normalizing the output of a quaternion polynomial $\mathbf{p}(\mathbf{x})$. This was done to facilitate comparisons with previous methods, but we have found that for our purposes any $\mathbf{d}(\mathbf{x})$ will suffice. The frequency of details in the versor function (Figs. 2, 9, 10) directly determines the frequency of fractal detail in the final Julia set.

This flexibility in $\mathbf{d}(\mathbf{x})$ allows a wide variety of fractal appearances to be generated for a single target shape (Figure 12). Different polynomials introduce different features, such as spiky ridges, swirls, and fibrous textures, among others. These bunny fractals were found in 55.1 seconds, while QUIJIBO algorithm took 43 hours, yielding a 2800$\times$ speedup.

Any $\mathbf{d}(\mathbf{x})$ that produces continuous and turbulent values will produce interesting fractal detail. For most shapes, laying down polynomial roots randomly inside a bounding box of the target shape produced an acceptable $\mathbf{d}(\mathbf{x})$. For each of our examples, 3 to 5 random polynomials were generated and the most visually interesting result was selected.

The space of possible versor functions and corresponding textures is difficult to characterize completely, but we have observed some tendencies. For Figs. 1 center left, 9, and 11, the polynomial roots were clustered tightly inside the target shape. In other examples with more regular, turbulent textures, the roots were more evenly distributed in space. Altering the positions of the polynomial roots in the versor function seems to smoothly deform the texture of the fractal while the large-scale shape remains constant. Thus, similar roots in $\mathbf{d}(\mathbf{x})$ yield similar fractal textures.

Additionally, using the same $\mathbf{d}(\mathbf{x})$ for two different target shapes allows a user to transfer the qualitative fractal texture from one Julia set result to another, such that similar fractal details appear in similar spatial locations. Even if the target shapes are entirely disjoint, they still produce qualitatively similar fractal appearances (Figure 11). This property allows a user to try out many texture by generating many $\mathbf{d}(\mathbf{x})$. If a desirable texture is found, it can be applied across multiple shapes.

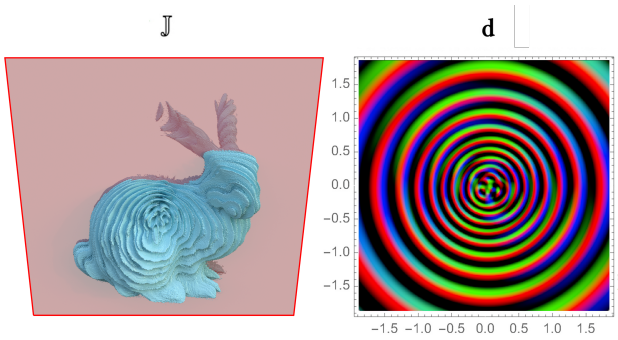

Figure 9: ***Left:*** *Bunny-shaped Julia set generated using our method,* ***Right:*** *As in Fig. 2, RGB plot of versor function* $\mathbf{d}(\mathbf{x})$ *along the slice shown at left. Circular rings are visible both in* $\mathbf{d}(\mathbf{x})$ *and the final Julia set.*

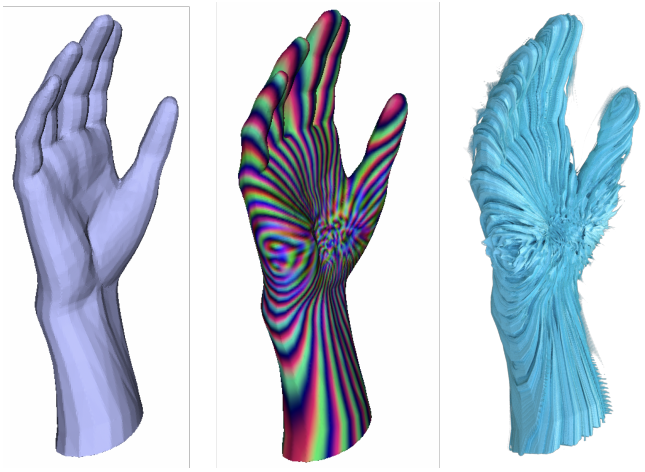

Figure 10: ***Left:*** *Target hand shape.* ***Middle :*** *As in Figure 2, the versor function* $\mathbf{d}(\mathbf{x})$ *is plotted using RGB along the surface of the target mesh. The overall texture of the Julia set is already visible in* $\mathbf{d}(\mathbf{x})$. ***Right:*** *A Julia set generated using these* $\phi(x)$ *and* $\mathbf{d}(\mathbf{x})$.

3.4. Performance

Our method bypasses the lengthy non-linear optimization stage of previous approaches [Kim15], and only needs to run Marching Cubes to compute the final triangles of $\mathbf{f}(\mathbf{x})$ from Equation 3. This is trivially parallelizable, allowing for short wall-clock generation times if enough parallel computation power is available. All geometry computation was run in parallel on a compute cluster using one core per job, split across several nodes. The cores used were Intel Xeon 8268, 6240, and E5-2660 v3 and v4 CPUs, with clock speeds ranging from 2.0-2.9 GHz. Parameters and timings are in Table 2.

Similar to QUIJIBO [Kim15], Marching Cubes was performed on the $\log(||\mathbf{f}_n(\mathbf{x})||)$ field and a bisection search was performed along each grid edge to refine the final mesh. The examples here were computed on the CPU, but because this method only requires standard precision it would be well-suited for running on a GPU.

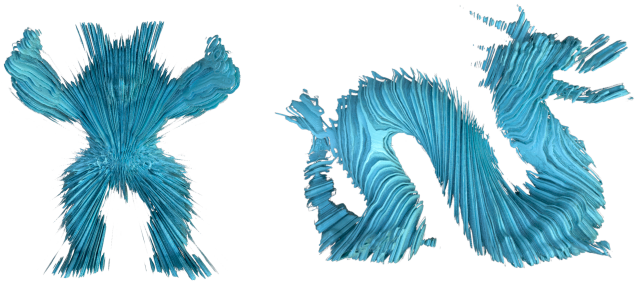

Figure 11: *Two Julia sets generated with different target shapes but using the same polynomial in in* $\mathbf{d}(\mathbf{x})$. *The two sets have very different overall shapes, but share a similar ridged appearance.*

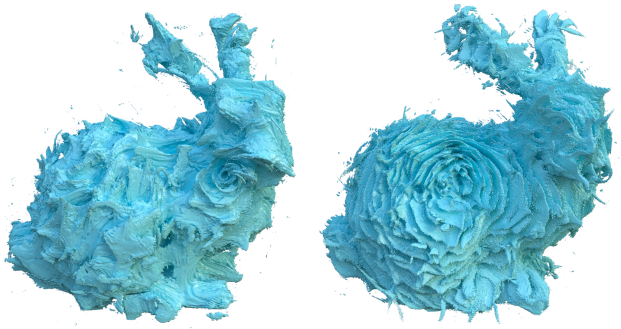

Figure 12: *Julia sets generated with the same* $r(\mathbf{x})$ *but different* $\mathbf{d}(\mathbf{x})$. *The overall shape remains constant, but the fractal detail differs. These fractals were found in* 55.1 *seconds, over* $2800\times$ *faster than with previous methods.*

For example, previous CUDA implementations [KD20] could easily be adapted to our formulation.

4. Limitations and Future Work

We have presented a versor-modulus analysis and shape modulus function for generating Julia sets to fit arbitrary target shapes. It is both faster and more robust than previous methods. This opens the door to efficiently exploring this style of geometry, and there are many avenues for improvement and future research.

Many design decisions made during the development of this algorithm likely have effective alternatives. While we have had success generating $\mathbf{d}(\mathbf{x})$ using quaternion polynomials, in principle any map or vector field could be used. Especially desirable would be a means of choosing a $\mathbf{d}(\mathbf{x})$ that yields a specific type of fractal detail. Investigating the specific dynamics of $\mathbf{d}(\mathbf{x})$ fields could yield even greater control over the fractal character of the generated shapes.

When setting control parameters α and β, it is relatively straightforward to guess good initial values and then fine-tune to achieve a desired visual quality. However, it should also be possible to analyze the target shape and directly determine parameters to achieve shell thicknesses and positions to achieve a specific look. Under this formulation, the shape is highly predetermined before computation

Table 2: *Marching Cubes performance. Triangles were computed on a grid resolution of* 1600^3 *for all examples. Each target shape used a different quaternion polynomial generated by placing roots with random powers inside the target shape's bounding box. Jobs were divided spatially, so each had varying amounts of geometry to compute. Thus, wall-clock time is not exactly equal to the total time divided by the number of cores. All times are in hour:minutes:seconds.*

Fig.	Example	$\mathbf{p}(x)$ degree	α	β	Triangles	Total cores	CPU time	Wall-clock time
1	Dragon (gradient α, avg. frame)	40	10-300	0	26.5M	64	10:05:03	1:21:11
7	Dragon (constant α)	40	10	0	53.5M	64	41:58:28	2:01:57
1	Armadillo	40	40	0.05	160.3M	64	70:40:48	4:20:46
6	Chair (varying α, β, avg. frame)	10	1-4	1-2	4.23M	512	19:41:33	0:05:37
1	Hand	40	90	0.02	12.7M	512	17:00:23	0:05:27
1	Bunny	10	6	0	355.8M	512	62:40:51	1:01:45

Figure 13: *Julia set of a bunny with a thin shell of fractal detail only along the body via a spatially-varying* α*. As in Fig. 7, a cubic interpolant is used on* α *to create a smooth visual transition.*

even begins, so there may be a more optimal strategy for generating the final triangles than vanilla Marching Cubes. Each iteration could also be optimized further by detecting additional basins of attraction, and terminating computation when an iterate enters these regions.

Acknowledgements

We thank the reviewers for their helpful comments. This work was supported by Adobe and the Teng and Han Family Fund.

References

[BD13] BAKER, MATTHEW and DE MARCO, LAURA. "Special curves and postcritically-finite polynomials". *Forum of Mathematics, Pi* 1 (2013). ISSN: 2050-5086 2.

[CD05] COOK, ROBERT L. and DEROSE, TONY. "Wavelet Noise". *ACM Trans. Graph.* 24.3 (July 2005), 803–811. ISSN: 0730-0301 2.

[Coo84] COOK, ROBERT L. "Shade Trees". *Proceedings of SIGGRAPH*. New York, NY, USA: Association for Computing Machinery, 1984, 223–231. ISBN: 0897911385 2.

[Cro22] CRONENBERG, DAVID. *Crimes of the Future*. 2022 5.

[DH85] DOUADY, ADRIEN and HUBBARD, JOHN HAMAL. "On the dynamics of polynomial-like mappings". *Annales scientifiques de l'École normale supérieure*. Vol. 18. 2. 1985, 287–343 2.

[ESHG17] EBB, MATT, SUTHERLAND, RICHARD, HECKENBERG, DANIEL, and GREEN, MILES. "Building detailed fractal sets for "Guardians of the Galaxy Vol. 2"". *ACM SIGGRAPH Talks*. 2017, 1–2 1.

[Fai18] FAILES, IAN. *Mandelbulbs, Mutations and Motion Capture: The Visual Effects of Annihilation*. 2018. URL: https://vfxblog.com/2018/03/12/mandelbulbs-mutations-and-motion-capture-the-visual-effects-of-annihilation/ (visited on 10/17/2022) 1.

[Fat17] FATOU, PIERRE. "Sur les substitutions rationnelles". *Comptes Rendus de l'Académie des Sciences de Paris* 164 (1917), 806–808 2.

[Gia17] GIARDINA, CAROLYN. *'Guardians of the Galaxy Vol. 2': A Digital Kurt Russell and Other VFX Tricks Revealed*. en-US. May 2017. URL: https://www.hollywoodreporter.com/movies/movie-news/guardians-galaxy-vol-2-vfx-tricks-revealed-1001266/ (visited on 04/11/2023) 1.

[Gle87] GLEICK, JAMES. *Chaos: making a new science*. Penguin Books, 1987 2.

[Har99] HART, JOHN C. "Computational topology for shape modeling". *Proceedings of Shape Modeling International*. IEEE. 1999, 36–43 2.

[HKS90] HART, JOHN C, KAUFFMAN, LOUIS H, and SANDIM, DANIEL J. "Interactive visualization of quaternion Julia sets". *Proceedings of the IEEE Conference on Visualization*. IEEE. 1990, 209–218 2.

[HRE*15] HUTCHINS, DAVID, RILEY, OLUN, ERICKSON, JESSE, et al. "Big Hero 6: into the portal". *ACM SIGGRAPH Talks*. 2015, 1–1 1.

[HSK89] HART, JOHN C, SANDIN, DANIEL J, and KAUFFMAN, LOUIS H. "Ray tracing deterministic 3-D fractals". *Proceedings of SIGGRAPH*. 1989, 289–296 2.

[Jul18] JULIA, GASTON. "Mémoire sur l'itération des fonctions rationnelles". *J. Math. Pures Appl.* 8 (1918), 47–245 2.

[KD20] KIM, THEODORE and DUFF, TOM. "A Massive Fractal in Days, Not Years". *Journal of Computer Graphics Techniques (JCGT)* 9.2 (2020), 26–36. ISSN: 2331-7418 3, 7.

[KFB14] KIM, ALEX, FERREIRA, DANIEL P, and BEVINS, STEPHEN. "Capturing the infinite universe in" Lucy": fractal rendering in film production." *SIGGRAPH Talks*. 2014, 1–1 1.

[Kim15] KIM, THEODORE. "Quaternion Julia Set Shape Optimization". en. *Computer Graphics Forum* 34.5 (Aug. 2015), 167–176. ISSN: 01677055 1–5, 7.

[Kim17] KIM, THEODORE. *Quijibo: Source Code for Quaternion Julia Set Shape Optimization*. 2017. URL: https://github.com/theodorekim/QUIJIBO 3, 4.

[KK89] KAJIYA, J. T. and KAY, T. L. "Rendering Fur with Three Dimensional Textures". *Proceedings of SIGGRAPH*. New York, NY, USA: Association for Computing Machinery, 1989, 271–280. ISBN: 0897913124 2.

[Lin15] LINDSEY, KATHRYN A. "Shapes of polynomial Julia sets". en. *Ergodic Theory and Dynamical Systems* 35.6 (Sept. 2015), 1913–1924. ISSN: 0143-3857, 1469-4417 2, 5.

[LLD11] LAGAE, ARES, LEFEBVRE, SYLVAIN, and DUTRE, PHILIP. "Improving Gabor Noise". *IEEE Transactions on Visualization and Computer Graphics* 17.8 (2011), 1096–1107 2.

[Man80] MANDELBROT, BENOIT B. "Fractal aspects of the iteration of z→ Λz (1-z) for complex Λ and z". *Annals of the New York Academy of Sciences* 357.1 (1980), 249–259 2.

[Mar19] MARCZAK, KRZYSZTOF. *Mandelbulber: 3D Fractal Explorer*. 2019. URL: https://sites.google.com/site/mandelbulber/home (visited on 10/17/2022) 1.

[Nee97] NEEDHAM, TRISTAN. *Visual complex analysis*. Oxford University Press, 1997 3.

[Nor82] NORTON, ALAN. "Generation and display of geometric fractals in 3-D". *ACM SIGGRAPH Computer Graphics* 16.3 (July 1982), 61–67. ISSN: 0097-8930 2.

[PBFJ05] PORUMBESCU, SERBAN D., BUDGE, BRIAN, FENG, LOUIS, and JOY, KENNETH I. "Shell Maps". *Proceedings of SIGGRAPH*. Los Angeles, California: Association for Computing Machinery, 2005, 626–633. ISBN: 9781450378253 2.

[Per85] PERLIN, KEN. "An Image Synthesizer". *Proceedings of SIGGRAPH*. New York, NY, USA: Association for Computing Machinery, 1985, 287–296. ISBN: 0897911660 2.

[PH89] PERLIN, K. and HOFFERT, E. M. "Hypertexture". *Proceedings of SIGGRAPH*. New York, NY, USA: Association for Computing Machinery, 1989, 253–262. ISBN: 0897913124 2.

[Sey16] SEYMOUR, MIKE. *Doctor Strange's Magical Mystery Tour In Time*. 2016. URL: https://www.fxguide.com/fxfeatured/dr-stranges-magical-mystery-tour-in-time/ (visited on 10/17/2022) 1.

[Sey17] SEYMOUR, MIKE. *The fractal nature of Guardians of the Galaxy Vol. 2*. 2017. URL: https://www.fxguide.com/fxfeatured/the-fractal-nature-of-guardians-of-the-galaxy-vol-2/ (visited on 04/11/2023) 1.

[SMB*18] SAAM, JOHANNES, MERCHANTE, MARIANO, BEAVERS, PATRICK, et al. "Fractal multiverses in VR". *ACM SIGGRAPH Talks*. 2018, 1–2 1.

[Whi09] WHITE, DANIEL. *The Unravelling of the Real 3D Mandelbulb*. 2009. URL: https://www.skytopia.com/project/fractal/mandelbulb.html (visited on 10/17/2022) 1.