Eurographics Conference on Visualization (EuroVis) 2013
B. Preim, P. Rheingans, and H. Theisel
(Guest Editors)

Volume 32 (2013), Number 3

Visual Explanation of the Complexity in Julia Sets

Okke Schrijvers† and Jarke J. van Wijk

Dept. Math. and Computer Science, Eindhoven University of Technology, The Netherlands

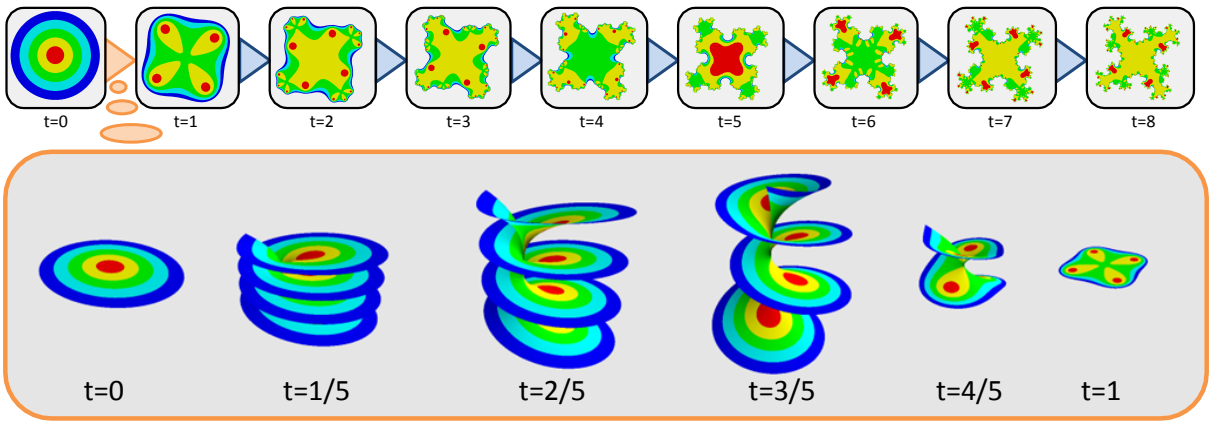


Figure 1: Schematic depiction of the construction of Julia sets to provide a visual explanation for their complexity. In a number of steps (top row), complexity builds up; a smooth animation (bottom row) shows how the shape is distorted per step.

Abstract

Julia sets based on quadratic polynomials have a very simple definition, yet a highly intricate shape. Our contribution is to provide a visual explanation for this complexity. To this end we show the construction of Julia sets as a dynamic process, in contrast to showing just a static image of the set itself. Our method is based on the Inverse Iteration Method (IIM). We start with a disk, which is successively distorted. The crucial step is to show an animation of the effect of taking a root of a subset of the complex plane. We present four different approaches for this, using a Riemann surface, a corkscrew, a fan, and disks as metaphors. We packaged our results in an interactive tool with a simple interface, such that everybody can view and inspect these for different Julia sets. The results are useful for teaching complex analysis, promoting mathematics, entertainment, and, above all, as a visual explanation for the complexity of Julia sets.

Categories and Subject Descriptors (according to ACM CCS): I.3.8 [Computer Graphics]: Applications—

1. Introduction

Julia sets are well-known and fascinating mathematical objects, whose complex shapes have amazed and inspired both academics as well as the general public. They are a perfect example of the beauty of mathematics: a concise definition yields spectacular results. An intriguing question is where this complexity comes from. How can we understand why a simple definition gives rise to such highly complex, intricate shapes? Many have produced images of Julia sets, but these static images do not answer this question. In this paper we provide a visual answer by showing how Julia sets can be constructed using a dynamic process, and show how the complexity gradually increases.

Our inspiration here comes from the mundane and well-

† Okke Schrijvers is currently at Stanford University.

© 2013 The Author(s)
Computer Graphics Forum © 2013 The Eurographics Association and Blackwell Publishing Ltd. Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

known puff pastry metaphor, which is often used in chaos theory to explain the basics of sensitivity to initial conditions. Puff pastry is made in a number of steps, where in each step the dough is stretched and folded. Many layers result, and points that were initially close are wide apart. This metaphor is easy to understand, because each iteration is not a jump from one state to the next, but a smooth and easy to understand deformation of the object. We show how this metaphor can be used to construct Julia sets, using a sequence of animated steps, where in each step the surface is smoothly distorted, and show how this process gives a visual explanation of the complexity of Julia sets. Julia sets emerge as the result of an iterative process of complex functions. Our method is based on the observation that complex functions have firm roots in geometric transformations and can be understood visually [Nee99].

In Section 2 we discuss related work. For our purposes, the Inverse Iteration Method is most useful. We start with a disk, which we iteratively distort into more complex shapes. A model for this is presented in Section 3. One key problem is to visualize taking the n-th root of a subset of the complex plane. We present four different solutions, based on different metaphors and strategies for cutting and animating a subset of the complex plane, and thereby provide novel solutions for the visualization of these multivalued complex functions, based on homotopic mapping. We have integrated our approach in a standalone demonstration tool, which we present in Section 4, along with a number of examples. We aimed here at providing users immediate access to the results as well as enabling them to experiment with a variety of options. Our colleagues from the Mathematics department confirmed that such a tool can be very useful to engage, motivate, and teach students about complex functions in general and fractal sets in particular. It can also be used as a visual example for chaos theory, as it shows how iterations and non-linearity lead to complexity. Finally, we discuss the work and identify future challenges in Section 5.

2. Background

In the early twentieth century Pierre Fatou and Gaston Julia initiated the study of the dynamics of complex polynomial and rational maps. After 1925 interest vanished, but it revived in the seventies thanks to Mandelbrot's discovery of the set that bears his name and the spectacular images that were made using computer graphics [Man77, Man82, PR87, Pe88], as well as by the strongly increasing interest in chaos theory [Gle87]. Visualization is used by mathematicians and scientists who are interested in the inherent structure and complexity of the objects, but also, their fascinating shapes have intrigued a large audience, and images of Julia sets have found their way to popular culture.

The filled Julia set K of a complex function $f : \mathbb{C} \to \mathbb{C}$ is defined as the set of points $z \in \mathbb{C}$ that do not escape to infinity under the iteration sequence $z_{j+1} = f(z_j)$ with $z_0 = z$, i.e.,

$$K = \{z \in \mathbb{C} \mid \forall j \in \mathbb{N} : |z_j| \leq C\} \quad (1)$$

for some constant $C \in \mathbb{R}$. The Julia set J is the boundary of the filled Julia set. An intensively studied type is the family of quadratic Julia sets J_c (and filled in versions K_c), where for f a quadratic polynomial $f(z) = z^2 + c$ is used with parameter $c \in \mathbb{C}$, yielding the iteration sequence

$$z_{j+1} = z_j^2 + c.$$

Many have developed algorithms and applications to visualize Julia sets. The simplest approach is to use forward iteration, directly based on the definition of Julia sets. First, associate each pixel of a raster display with a point z in the complex plane. Next, for each pixel, assign z to z_0 and iterate a number of times. If $|z_j| > C$, the pixel is outside and colored white; otherwise if $j > N$, where N is a fixed number of iterations, the pixel is assumed to be inside the filled Julia set and is colored black. Typically $C = 2$ is used, as this is proven to be sufficient for connected quadratic Julia sets [PR87]. This algorithm yields a simple black on white image. More insight in the structure can be obtained by coloring the outside of the filled Julia set by how many iterations it takes for a point to escape beyond the constant C, known as encirclement [PJS03]. The resulting images suggest that the Julia set is generated by a circle that is locally deformed to create a more complex curve, similar to a Koch curve. This is not correct, as is discussed in Section 4. Also, instead of the number of iterations, a continuous value (the potential) can be shown via a color scale or height that indicates the distance to the Julia set, yielding spectacular images. Other options for annotation are the use of field lines and equipotential lines.

Insight in the internal structure of the filled Julia set can be obtained via coloring by basin of attraction. Points that do not escape are usually attracted to a cycle of one or more points $a_i, i = 1, \ldots, m$, the so called attractor. The interior of the filled Julia set is the basin of attraction of the attractor. We can also consider the basin of attraction per point of the attractor a_i, i.e., the set of points that converge to a_i after repeated cycles of m iterations. Insight in the internal structure of the filled Julia set is obtained by assigning a different color to each such basin.

These approaches based on forward iteration produce static images of Julia sets. They show the complexity, but do not give any explanation for the cause of this complexity. We argue that the latter can be achieved by showing an animation where Julia sets are constructed incrementally. To this end, we consider the Inverse Iteration Method (IIM), which is an alternative approach to generate images of Julia sets. Given a point z_0 that belongs to a quadratic Julia set, its two preimages (also belonging to the Julia set) are given by

$$z_1 = \pm\sqrt{z_0 - c}.$$

Recursive application of this recipe leads to an exponentially

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.

growing set of points of the Julia set, which will quickly exhaust memory. An alternative is to use an approach often used for Iterated Function Systems, known as the Chaos Game: at each iteration, randomly pick the positive or negative root to be used as next point.

Images of Iterate Function Systems can also be generated with an image based approach [vWS04]. Starting with an image of a shape, for instance a circle or a square, multiple scaled copies of this image are rendered and blended. Next, this step is done again for the resulting image, which yields a more complex image. Repeating this step quickly leads to an image of an Iterated Function System. Recently, de Smit et al. [dSMP*12] used a similar approach to produce images of Julia sets.

We do not aim at producing novel visualizations of Julia sets, but instead we focus on providing insight in the generating process. The image based approach provides a good starting point, as a sequence of discrete images is produced with increasing complexity. The next challenge is to smoothly animate each step in this process. This implies that complex mappings have to be visualized and animated smoothly. A complex function can be interpreted as a mapping of a plane to itself, which is conceptually simple, but would require four dimensions to show as a direct analog of the display of real functions as a graph. One approach is to show mapped isoparameter lines, yielding a distorted grid. Color can be used to give more insight in the fate of the domain after mapping. To visualize multi-valued functions the third dimension can be used. Here, the grid is typically shown on a Riemann surface, such that different branches (such as positive and negative roots) are assigned to different parts of the surface. In combination with lifted domain coloring [PP09] insightful images are obtained.

To show the mapping induced by complex functions, animation can be used. Animation is certainly not a panacea [TMB02], but we do consider it to be highly useful here. We use puff pastry as metaphor, and hence the use of animation meets the congruence principle of Tversky and her colleagues. Furthermore, it gives a vivid display well suited for educational purposes. Finally, we could not think of a more effective method, and found it difficult to explain what is going on via static images.

On the web many examples can be found of animations in 2D and 3D of mathematical concepts. Some inspiring examples are the work of Wegenkittl et al. on visualizing dynamical systems [WGP97], the work of Arnold and Rogness [AR08] on the visualization of Möbius transformations; tutorial examples of Arnold for various complex maps using homotopic mappings [Arn]; the package $f(z)$ of Lascaux Software [Las]; a playful version of complex mappings has been provided by Kalantari [Kal04]; the very clear animations of Jason Ross show complex functions and multivaluedness [Ros]. Parallel with the work described here, Steven Wittens [Wit13] has developed animations to show the construction of Julia fractals. Several ideas are shared, such as the use of inverse iteration. In our terminology, he uses the Disks metaphor. However, per step the fate of a single disk is shown, followed by copying the result afterwards, instead of showing multiple disks simultaneously; some other diffences are that we show other metaphors and use color to highlight deformations, the attractor and basins of attraction.

3. Model

We now present our model for the iterative construction of the Julia set, using moving and distorting surfaces. We start with some basic notions, followed by a generic model. Our approach is based on the Inverse Iteration Method, but instead of applying it to single points, we apply it to a set of points simultaneously, and aim to show the effect of iterations by animation of this set. We present four different solutions for the details of the animation. We aim to give a complete, technical description, such that our approach can be reproduced easily; examples of the resulting animations can be found in the accompanying video.

3.1. Basics

We consider a slightly generalized version of the quadratic Julia sets J_c, instead of using a power of 2, we use a power of n, where n is an integer larger than 1. In forward iteration, the basic step is then

$$z_{j+1} = z_j^n + c. \tag{2}$$

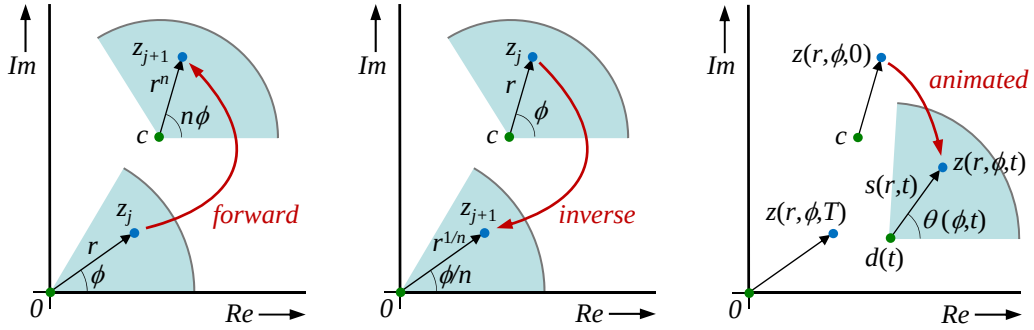
Using polar coordinates r and ϕ, this step can easily be interpreted geometrically, as is explained in any basic course on complex functions. Using $z_j = r(\cos\phi + i\sin\phi) = re^{i\phi}$ we get $z_{j+1} = r^n e^{i\phi n} + c$, in other words, the angle ϕ is multiplied by n, the radius r is replaced by its $n-$th power, and finally the point is translated over c, see Figure 2a. Also, a disk $\{z \in \mathbb{C} \mid |z| \leq A\}$ is transformed into a disk $\{z \in \mathbb{C} \mid |z-c| \leq A^n\}$, centered around c with a radius which is the n-th power of the original radius A. Each sector of angle $2\pi/n$ maps to a full disk.

To show how the Julia set emerges as a sequence of iterations, we base our approach on the Inverse Iteration Method. We start with a disk with radius A, centered around c, repeatedly apply the inverse step

$$z_{j+1} = (z_j - c)^{1/n} \tag{3}$$

and show this via a smooth animation, shown schematically in Figure 2b and c. The challenge here is that this mapping is multi-valued; each point $z_j \neq c$ is mapped to n points z_{j+1}. As a result, the complex plane is covered by multiple sheets called *branches*. The center of the disk is called a branch point, and branches meet at branch cuts. We must decide how to make the different branches visible as well as how many branch cuts to use and where to position these.

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.



(a) Forward mapping. (b) Inverse mapping. (c) Animated inverse mapping.

Figure 2: Mapping $f(z) = z^n + c$. Forward and inverse mapping are shown, as well as the notation used in the animated model.

We first define a generic model. We describe the motion in the complex plane as

$$z(r,\phi,t) = s(r,t)e^{i\theta(\phi,t)} + d(t), \tag{4}$$

with $r \in [0,A]$, the initial distance to the center c of the disk; $\phi \in [\phi_0, \phi_0 + 2\pi n)$, the initial angle; and with $t \in [0,T]$ a time parameter from start time 0 to end time T. Note that the width of the interval for ϕ has to be $2\pi n$, such that multi-valuedness can be dealt with, but the lower bound ϕ_0 of this interval is not fixed. The functions s, θ, and d model the variation of the radius, angle, and displacement over time, and are subject to the following constraints:

$$\begin{array}{lll} s(r,0) = r, & \theta(\phi,0) = \phi, & d(0) = c, \\ s(r,T) = r^{1/n}, & \theta(\phi,T) = \phi/n, & d(T) = 0. \end{array}$$

Substitution shows that these give the desired result:

$$\begin{aligned} z(r,\phi,0) &= re^{i\phi} + c, \\ z(r,\phi,T) &= (z(r,\phi,0) - c)^{1/n}. \end{aligned}$$

To handle multiple branches, we embed the complex plane in 3D, and model this moving surface as

$$\mathbf{p}(r,\phi,t) = \begin{pmatrix} Re(z(r,\phi,t)) \\ Im(z(r,\phi,t)) \\ h(r,\phi,t) \end{pmatrix}$$

where h denotes the offset from the plane, with the constraints $h(r,\phi,0) = h(r,\phi,T) = 0$.

Next, we consider the animation: the definition of $s(r,t)$, $\theta(\phi,t)$, $d(t)$, and $h(r,\phi,t)$. Three different aspects have to be addressed. First, the fact that there are n roots for z_j has to be dealt with by splitting the original disk; second, the exponentiation has to be shown; and third, the disk has to be moved to the origin. We denote these aspects by S (split), E (exponentiation), and M (move). We use two auxiliary functions to describe transitions for animation purposes. Linear interpolation of a value v from v_1 to v_2 is described by

$$L_X(v_1,v_2,t) = \begin{cases} v_1 & \text{if } t \le t_{X1} \\ v_1 + \frac{t-t_{X1}}{t_{X2}-t_{X1}}(v_2 - v_1) & \text{if } t_{X1} < t < t_{X2} \\ v_2 & \text{if } t \ge t_{X2} \end{cases}$$

where the label X is S, E, or M, and where $[t_{X1}, t_{X2}]$ denotes the interval where the transition for aspect X is made. To describe intermediate changes (from 0 to v and back to 0) we define a ramp function

$$I_X(v,t) = L_{X_a}(0,v,t) - L_{X_b}(0,v,t),$$

by combining two linear interpolations. Here X_a refers to the start of the ramp $[t_{X1}, t_{X2}]$ and X_b to the end $[t_{X3}, t_{X4}]$. We use simple linear transitions here, we found this to be satisfactory for our purposes, but obviously smoother transitions can be used.

3.2. Animation Types

We now describe various types of animations we defined to show the iteration step, which we called the Corkscrew, Fan, Riemann surface, and Disk model. Figure 3 provides an overview of these, and the accompanying video shows them in motion. We use C, F, R, and D as subscripts to denote differences between these where needed.

Corkscrew. The simple Corkscrew model is defined by

$$\begin{aligned} \phi_0 &= -\pi n, \\ s(r,t) &= r^{L_E(1,1/n,t)}, \\ \theta(\phi,t) &= L_E(\phi, \phi/n, t), \\ d(t) &= L_M(c,0,t), \text{and} \\ h_C(\phi,t) &= I_S((H/2\pi n)\cdot\theta(\phi,t),t), \end{aligned}$$

where H is a scale parameter for the height. Most parameters are interpolated in the obvious way. The height depends here on the current angle θ, such that the net result is that the disk is first unfolded like a corkscrew or spiral stairs, and next rolled up again. For ϕ_0 we use $-\pi n$ rather than 0, to obtain a more symmetric result and to minimize rotation of the branch cuts for ϕ_0 and $\phi_0 + 2\pi n$.

Fan. In the Corkscrew model the branch point, i.e., the center of the moving disk, is intermediately mapped to a line.

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.

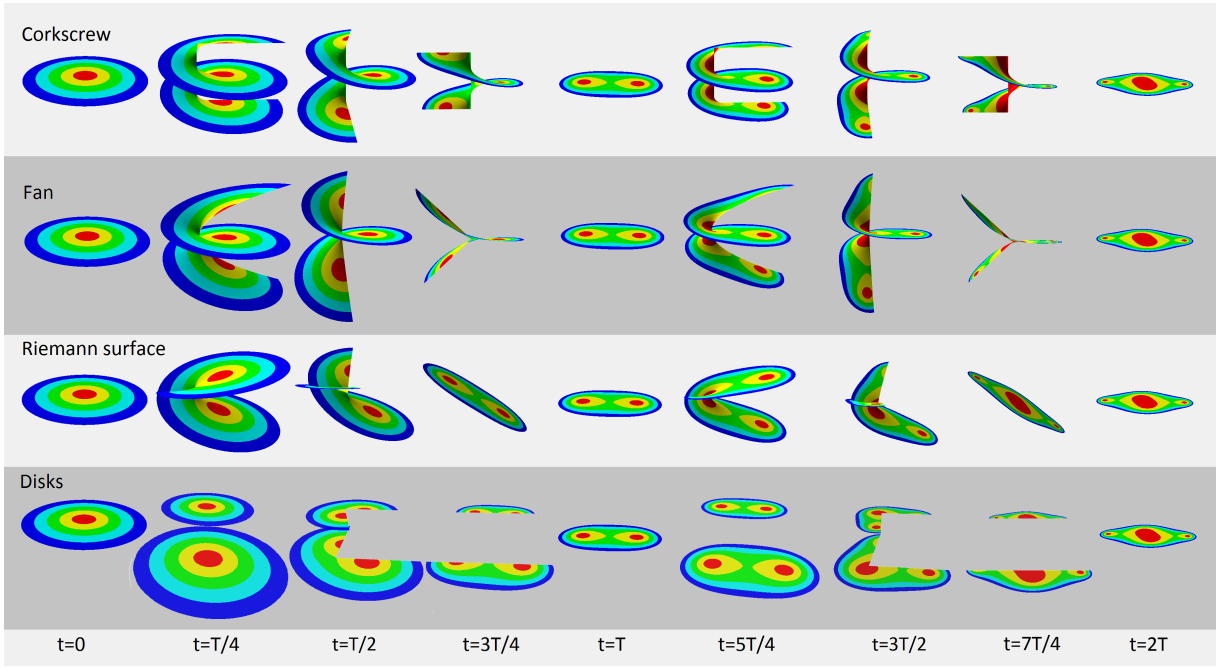


Figure 3: Overview of animation types. Each row shows an animation type (from top to bottom Corkscrew, Fan, Riemann surface, and Disks), from left to right frames from two complete cycles are shown.

In the Fan model we use

$$h_F(r,\phi,t) = r^{1/n} h_C(\phi,t).$$

such that the center of the disk remains a point. The initial branch cuts are shown as curves in the shape of graphs of root functions, as a result of using the final radius $r^{1/n}$ rather than the instantaneous radius $s(r,t)$ for scaling the height.

Riemann surface. In both previous models the surface is interrupted after the multivaluedness is dealt with: the curves for $\phi = \phi_0$ and for $\phi = \phi_0 + 2\pi n$, the opposite sides of the single branch cut used, are offset from the plane in opposite directions. Technically, this is not correct: topologically the multifolded disk is one uninterrupted surface, i.e., a Riemann surface. Based on the standard way to depict $z^{1/n}$ as a Riemann surface, we get

$$h_R(r,\phi,t) = I_S(H \cdot Re(z(r,\phi,T)),t),$$

similar to the videos of Ross [Ros]. A disadvantage of this approach is that such a surface is self-intersecting, which is unavoidable using embedding in 3D.

Disks. Another variation is to split up the multi-valued surface into n branches, using n branch cuts, shown as separate disks, moving in the complex plane. Initially, the angles of each disk $k, k = 0,\ldots,n-1$, have a range

$$[\alpha_{k1},\alpha_{k2}] = [\phi_0 + 2\pi k, \phi_0 + 2\pi(k+1)],$$

finally they should have a range

$$[\beta_{k1},\beta_{k2}] = [\alpha_{k1}/n + 2\pi m_k, \alpha_{k2}/n + 2\pi m_k].$$

The terms $2\pi m_k$, with $m_k \in \mathbb{Z}$, are added to provide additional freedom here, while still satisfying the constraint that the final disk is uniformly covered by pie-shaped segments. Using $m_k = 0$ gives strong rotations for some disks. Selection of m_k such that

$$\max(|\beta_{k1} - \alpha_{k1}|, |\beta_{k2} - \alpha_{k2}|)$$

is minimized gives better results. Figure 4 shows this schematically. As an example, if $m_k = 0$ would be used for the pink disk, the purple branch-cut would have to rotate a full circle; by adding 2π it can remain in place.

To separate the disks, we temporarily translate them in the complex plane in a star-pattern, i.e.,

$$d_D(k,t) = d(t) + I_S(\, D(\cos\gamma_k + i\sin\gamma_k), t),$$

where D is the maximum offset used and $\gamma_k = \phi_0 + 2\pi(k + 1/2)/n$ is the center angle of the sector of disk k. The positions and branch cuts are therefore dependent on our choice for ϕ_0, however this does not influence the final result of the iteration. Finally, we use a small offset such that the disks can be shown sorted for depth order: $h_D(k,t) = k\varepsilon$.

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.

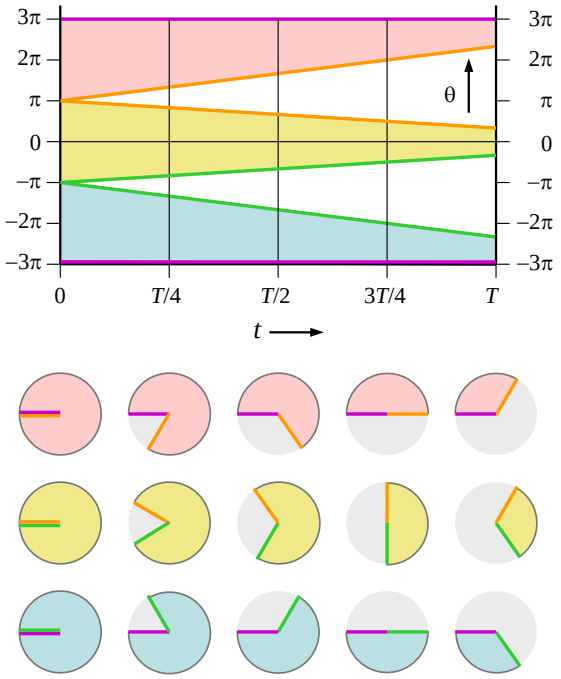


Figure 4: When the branches are shown as separate disks, here for $n = 3$, their rotation can be reduced by adding offsets $2\pi m_k$, here 2π for the pink disk and -2π for the blue disk. The purple, orange, and green lines represent the two sides of the three branch cuts.

4. JuliaInMotion

We have implemented the model for animating the construction of Julia sets in an interactive application, which we called JuliaInMotion (JIM). We first present an overview of the application, including a discussion of design choices and features. Next, we give examples how JIM can be used to obtain insight in Julia sets, followed by a report of reactions of prospective users.

4.1. Design

Implementation. We use an RGBA texture to represent the filled Julia set, where the opacity channel A is used for set membership. The method for generating new iterations is based on Image-Based Iterated Function Systems [vWS04]. We take the texture of iteration j, apply the transformation given in Equation 3 for n branches, and render the results to a new texture for iteration $j+1$. Since the transformation is non-linear, we cannot simply use a transformed quadrilateral. Therefore, we map the texture to a polar grid mesh where each vertex is individually updated according to Equation 3. We found that using a grid with 60 sectors and 30 rings was sufficient. Only the coordinates of one branch have to be computed here, the other branches are identical modulo a rotation around the origin. The new texture is rendered using an off-screen framebuffer object and since at any point only the results of two iterations are used, we can ping-pong between two textures. Various other solutions are possible here, such as recalculating the texture anew per pixel after one or more time steps, but we found the simple approach used here to be sufficient.

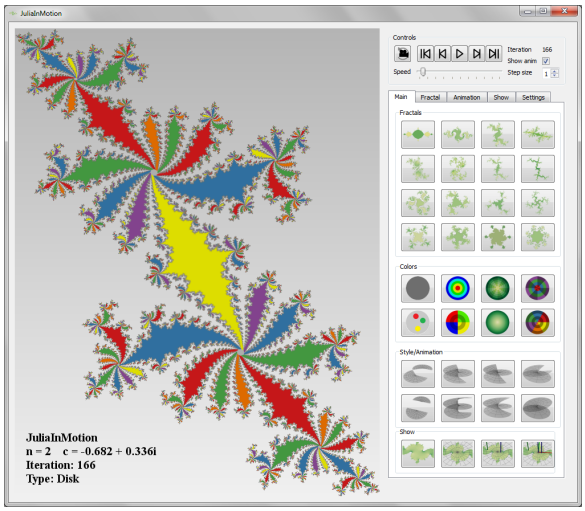


Figure 5: The main window of JuliaInMotion.

For the animations that show the transition between the results of two iterations, we use the same polar grid and update each vertex according to Equation 4. It is possible to do this using shaders, but we found that generating vertex positions using the CPU was efficient enough.

User Interface. The main window of JIM, shown in Figure 5, was designed to give the user quick and easy access to all the functionality, without requiring detailed knowledge of Julia sets or complex functions. The left side shows the animated Julia set. Users can pan, zoom, and change their viewpoint at any moment by dragging the mouse. The right side holds a control panel. The control panel contains a panel with playback functionality —such as play, pause, fast forward, reset and animation speed— that is always visible. In addition, there are a number of tabs that give access to the more advanced functionality of the application. The main tab contains sets of buttons that trigger presets for the type of Julia set, coloring scheme, the animation style and timing, and additional visualization options. The user can arbitrarily click on any of the buttons to explore the options and always obtain nice results. The other tabs give direct access to application parameters, such that everything can be tweaked if desired. Examples of these parameters are the values of n and c that define the Julia set; the colors used and settings for the timing of the animation. Also, options are provided to export images and image sequences. In the following paragraphs we discuss various aspects that can be controlled through the main tab of the user interface.

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.

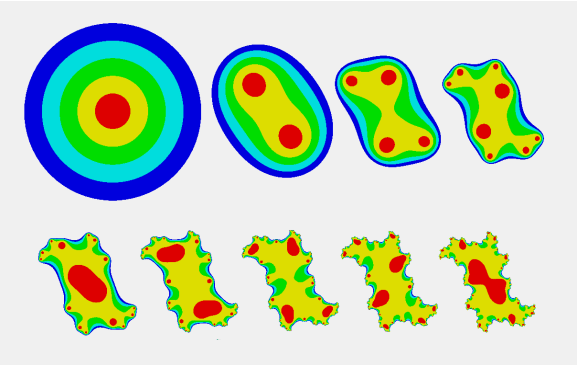


Figure 6: Results for a sequence of iterations. The blue and green rings vanish quickly here.

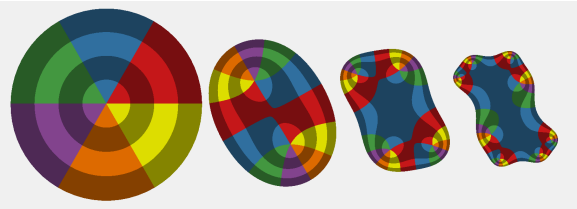


Figure 7: Alternative color fill, which shows that equipotential lines remain perpendicular to field lines.

Julia sets. We hand-picked sixteen examples of interesting Julia sets, and made these easily selectable via buttons. The buttons show a miniature version of the end results. We varied the complexity, starting with examples of quadratic Julia sets (first two rows), followed by examples with a larger value for n.

Coloring. We also predefined a number of color schemes. If we are solely interested in finding the filled Julia set, we can start with a uniformly colored disk. However, using a pattern for the disk reveals much clearer how one shape is iteratively transformed into the next.

A simple but very effective color scheme is to use concentric circles. As can be seen in many of the figures presented, the duplication of the central red dot reveals how complexity builds up and enables the viewer to trace the distortion. Furthermore, concentric circles give insight in how close to the origin points are trapped. For instance, in Figure 6 we show the first 8 iterations for $c = 0.2 + 0.55i$. The blue and cyan rings quickly become smaller to the point where they are just barely visible, indicating that all points are trapped in the inner three disks. This also implies that if we start with a larger or different shape, e.g., a square with side length 4, that this will result in the same Julia set, since the excess area is compressed within a few iterations.

More insight can be gained by accentuating radial lines in addition to concentric circles. This polar grid serves a similar

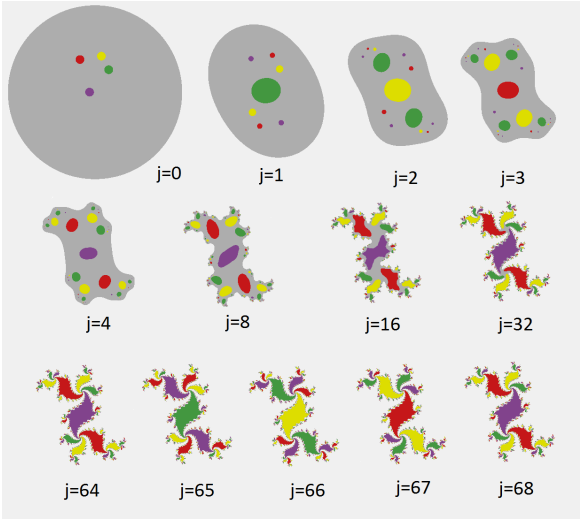


Figure 8: Coloring attractor and basins of attraction.

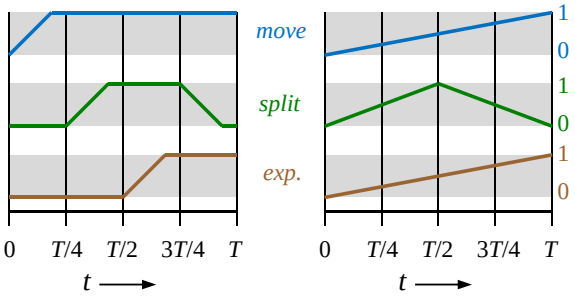
purpose as equipotential and field lines that are traditionally used to show what happens on the exterior of the Julia set. Figure 7 shows that perpendicular crossings remain perpendicular, thereby visualizing that the mapping is conformal.

For most color maps, color patterns tend to repeat themselves. See for example Figure 6, where the center is first red, then yellow for three iterations, after which the pattern repeats itself. This is caused by a so called attractor: a set of m points in the complex plane, specific to the choice for c, to which the iteration sequence in Equation 2 converges. Each point of the attractor has an associated basin of attraction, which in the end is colored uniformly. These basins are cyclically visited, such that the colors for iterations j and $j+m$ are the same, while the iterations in between show a cyclic permutation of the colors.

Based on this, we can emphasize the points of the attractor in the starting image and use them to highlight the basins of attractions. The attractor can be found using a simple algorithm: start a forward iteration sequence, beginning with 0; when a cycle is found, each point in the cycle is an element of the attractor. We start from 0, because this point must be in the basin of attraction, if there is one. It is a well known theorem that if there is an attractor, then its basin of attraction must include a critical point, and here 0 is the only critical point. We mark the points in the attractor initially with differently colored small disks. Figure 8 shows the effect. In the initial iterations, the shapes are cut open starting from these disks, where the color permutes per iteration. During the animations, the disks expand and distort, until finally only colored areas remain. The Julia set shown stabilizes, but the coloring and animation show that at each iteration the different lobes cycle around their meeting points.

Animation style and timing. The next two rows of buttons

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.



(a) Sequential (b) Parallel

Figure 9: Animation timing styles. Per style the variation of three parameters between 0 and 1 over time is shown.

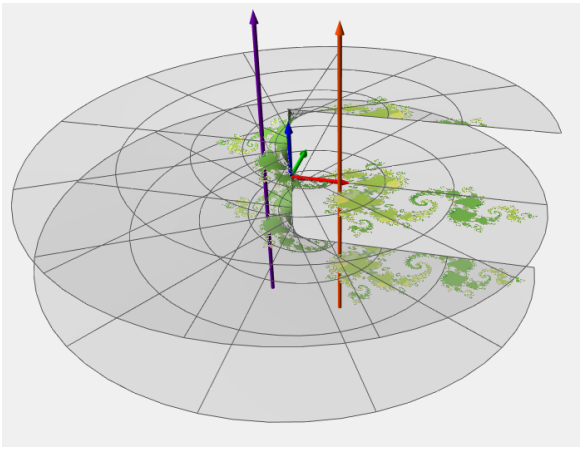


Figure 10: Visual annotation to aid in understanding the math behind Julia sets.

enable the user to select an animation style, from left to right Disk, Corkscrew, Fan, and Riemann; the two rows denote different animation timing styles. The animations defined in Section 3 expose several parameters that can be used to generate different animation styles. While parameters such as the height of a corkscrew can easily be tweaked to a visually optimal value, finding an optimal animation timing turned out to be more involved. In the end, we found two different schemes at extreme opposites to be most useful and offer these as presets. The sequential scheme aims at showing each step in the animation as clearly as possible. Only one aspect (translation, splitting, exponentiation, merging) is shown at a time, with a small pause in between. Besides giving insight in the different steps, this timing gives a rhythmic, factory-like feeling. Also, we offer a parallel scheme, where the different steps are combined into a perpetually moving scene. The timelines of these schemes are given in Figure 9, the effect on a cubic Julia set is shown in Figure 11.

Annotation. Besides the visualization of the Julia sets themselves, we have implemented several visual annotations to aid in understanding the math behind the animation, see Figure 10. First of all, we have implemented a polar grid of the complex plane. It helps to understand that the filled Julia set is a subset of the complex plane and it clarifies the structure of the transformation. Additionally, we have added markers for the coordinate axes, where red denotes the real axis, green the imaginary axis, and blue the axis perpendicular to the complex plane. Finally, we provide two markers that denote the origin of the complex plane and the position of c. It is between these two points that the shape is translated. Different combinations in increasing complexity can be selected via the buttons in the bottom row.

4.2. Examples

JuliaInMotion was developed to give insight in the construction of Julia sets. Here we enumerate a number of insights that can be obtained from the animations, and give some more examples.

Complexity. Our primary aim is to find a visual explanation for the complexity of Julia sets. Standard images suggest that the boundary is continuously deformed; our animations give a better answer. Especially the disk animations clearly reveal how Julia sets (i.e., the boundaries of filled Julia sets) are constructed. In each iteration, the boundary is cut open, n copies of the boundary are made, which are distorted, and finally assembled head to tail to form the new boundary. Repeating this step gives a boundary with a quickly increasing complexity.

Self-similarity and symmetry. This also explains the self-similarity of Julia sets: Each new shape is constructed from copies of one original shape, hence similar features and sub-shapes emerge. Rotational symmetry comes from the fact that all branches undergo the same transformation and are rotated over different angles, such that an n-fold rotational symmetry results.

The animations also show that the filled Julia set is a fixed set under $f(z)$, i.e., $K_c = f(K_c)$. Figure 8 shows that after a number of iterations the overall shape converges, and that each new shape is the same after being split up, distorted, and merged. However, the moving basins show that the interior is rearranged.

Attractors. We showed how the attractor and basins of attraction per point of the attractor can be visualized explicitly. Because we use inverse iteration, points of the attractor appears as repellers. At each iteration step, the area around c is slightly expanded until basins of attraction are formed. The coloring helps to understand this process. In addition, it shows that the number of symmetric lobes around meeting points is equal to the length of the cycle of points of the attractor.

Complex functions. Taking the square or higher order root of a single complex number is understandable for most stu-

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.

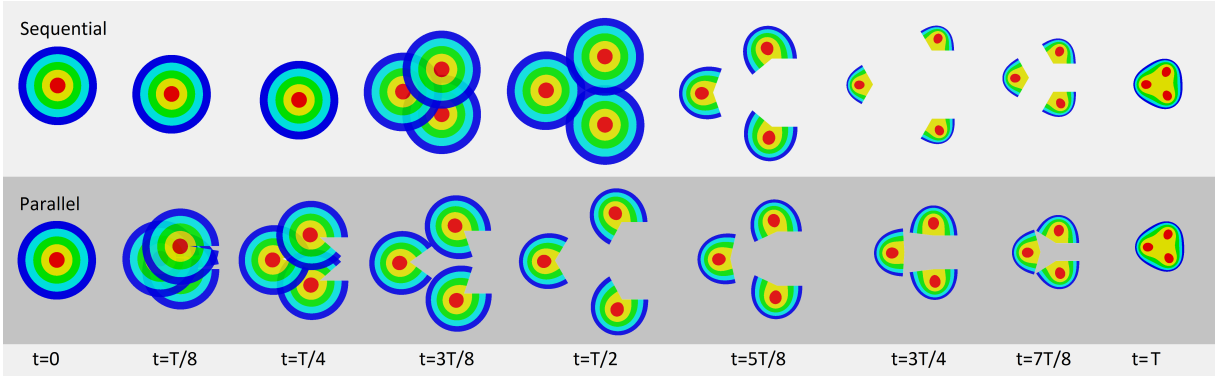


Figure 11: Sequential (top) and parallel (bottom) animation timing styles, shown for one iteration of a cubic Julia set.

dents; understanding what happens to the complex plane is much harder. Our animations show the root function in a lively way, where a variety of styles can be used.

Higher exponents. For the understanding of the underlying principles of the construction of Julia sets, looking at quadratic Julia sets is most illuminating. However, our method generalizes to larger exponents and can also be used to understand these more complex Julia sets. Figure 11 shows a cubic example, more examples can be found in the accompanying video.

4.3. Feedback

We have demonstrated JuliaInMotion to six colleagues from mathematics, with a varying age and experience, ranging from a PhD student to an emeritus professor. Their experience included mathematics research, teaching complex analysis, and external communication, particularly to high school students.

We started our interviews by asking if they could explain the reason for the complexity of Julia sets. We typically got no answer to this, which convinced us that this is indeed an open problem. We then showed various features of JuliaInMotion, stepwise increasing the complexity. All our respondents reacted very enthusiastically: they were intrigued by the animations, but could quickly understand how they were constructed and what they showed. Three of them took over the mouse and started to experiment themselves. They saw a variety of applications for JuliaInMotion: for master classes of high school students; to explain fractals and complexity to lay audiences; to generate intriguing movie-clips; but also as an appetizer and educational aid for teaching complex functions at university level. Some specific quotes: “Escher was born too soon”, “I didn’t know that this was possible” and “[JuliaInMotion] is several steps beyond any visualization [of complex mappings] that I’ve seen”. We were delighted by their enthusiasm. They also gave us good advice, for instance to show the definition of the fractal textually on screen, to make clear that there is solid math behind this.

In line with our expectations, our colleagues did not have a strong preference for one of the animation styles. The disks give a clean and compact view on the process, but using 3D scenes was found to be more spectacular. The Corkscrew and Fan are easier to understand than the Riemann surface, but the latter is mathematically a better model, as it initially has no branch cuts.

5. Conclusions

We have presented a method to visualize the construction of Julia sets as a continuous process. The use of smooth animations to show the iterative distortion of space strongly helps to understand the generating process, and provides a visual explanation for the complexity of Julia sets. We have presented four different styles for visualizing complex root functions; each with their own benefits. Furthermore, we have presented JuliaInMotion, an interactive application that enables a lay audience to play with and learn about Julia sets. We think our work can be applied for presentation, communication, and teaching purposes, and the first reactions of prospective users strongly confirmed this.

Our current animation model and application can visualize the construction of Julia sets for mappings of the form $f(z) = z^n + c$. This leads to the question which other mappings would lend themselves to our visualization method, and how these can be visualized using a similar approach as used here. An initially promising possibility seems the Mandelbrot set, for which the mapping differs only slightly: $f(z) = z^2 + z_0$, where z_0 is the initial value of the considered point in the complex plane. For forward iteration this is not a problem as z_0 is known from the first iteration onwards. However, our Inverse Iteration Method approach works the other way around, hence z_0 is not available, and we have not yet found out if this can be dealt with.

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.

References

[AR08] ARNOLD D. N., ROGNESS J.: Möbius transformations revealed. *Notices of the AMS 55*, 10 (Nov 2008), 1226–1231. 3

[Arn] ARNOLD D. N.: Graphics for complex analysis. http://www.ima.umn.edu/~arnold/complex.html, Last visited March 5, 2013. 3

[dSMP*12] DE SMIT B., MCCLURE M., PALENSTIJN W. J., SPARLING E. I., WAGON S.: Through the looking glass, and what the quadratic camera found there. *The mathematical Intelligencer 34*, 3 (2012), 30–34. 3

[Gle87] GLEICK J.: *Chaos - Making a New Science*. Viking Penguin, 1987. 2

[Kal04] KALANTARI B.: Polynomiography and applications in art, education, and science. *Computers & Graphics 28*, 3 (2004), 417–430. 3

[Las] LASCAUX SOFTWARE: f(z) - the complex variables program. http://www.lascauxsoftware.com/, Last visited March 5, 2013. 3

[Man77] MANDELBROT B.: *Fractals: Form, Chance and Dimension*. Addison-Wesley, 1977. 2

[Man82] MANDELBROT B.: *The Fractal Geometry of Nature*. W.H. Freeman, 1982. 2

[Nee99] NEEDHAM T.: *Visual Complex Analysis*. Oxford University Press, 1999. 2

[Pe88] PEITGEN H.-O., (EDS.) D. S.: *The Science of Fractal Images*. Springer-Verlag, 1988. 2

[PJS03] PEITGEN H.-O., JÜRGENS H., SAUPE D.: *Chaos and Fractals*, 2 ed. Springer-Verlag, 2003. 2

[PP09] POELKE K., POLTHIER K.: Lifted domain coloring. *Computer Graphics Forum 28*, 3 (2009), 735–742. 3

[PR87] PEITGEN H.-O., RICHTER P.: *The Beauty of Fractals: Images of Complex Dynamics*. Springer-Verlag, 1987. 2

[Ros] ROSS J.: Multivalued functions i. http://science.larouchepac.com/riemann/page/22,Last visited March 5, 2013. 3, 5

[TMB02] TVERSKY B., MORRISON J. B., BÉTRANCOURT M.: Animation: can it facilitate? *Int. J. Hum.-Comput. Stud. 57*, 4 (2002), 247–262. 3

[vWS04] VAN WIJK J. J., SAUPE D.: Image based rendering of iterated function systems. *Computers & Graphics 28*, 6 (2004), 937–943. 3, 6

[WGP97] WEGENKITTL R., GRÖLLER E., PURGATHOFER W.: Visualizing the dynamical behavior of wonderland. *IEEE Computer Graphics and Applications 17*, 6 (1997), 71–79. 3

[Wit13] WITTENS S.: How to fold a Julia fractal: A tale of numbers that like to turn, January 2013. http://acko.net/blog/how-to-fold-a-julia-fractal, Last visited March 5, 2013. 3

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.