Journal of Computational Dynamics
doi:10.3934/jcd.2016006

RIGOROUS BOUNDS FOR POLYNOMIAL JULIA SETS

Luiz Henrique de Figueiredo[1], Diego Nehab[1]
Jorge Stolfi[2] and João Batista S. de Oliveira[3]

[1]IMPA, Rio de Janeiro, Brazil
[2]Instituto de Computação, UNICAMP, Campinas, Brazil
[3]Faculdade de Informática, PUC-RS, Porto Alegre, Brazil

(Communicated by Bernd Krauskopf)

Abstract. We present an algorithm for computing images of polynomial Julia sets that are reliable in the sense that they carry mathematical guarantees against sampling artifacts and rounding errors in floating-point arithmetic. We combine cell mapping based on interval arithmetic with label propagation in graphs to avoid function iteration and rounding errors. As a result, our algorithm avoids point sampling and can reliably classify entire rectangles in the complex plane as being on either side of the Julia set. The union of the rectangles that cannot be so classified is guaranteed to contain the Julia set. Our algorithm computes a refinable quadtree decomposition of the complex plane adapted to the Julia set which can be used for rendering and for approximating geometric properties such as the area of the filled Julia set and the fractal dimension of the Julia set.

1. **Introduction.** We all have seen many beautiful images of Julia sets [33], but can we really trust these images? Images of Julia sets are typically generated by iterating a complex polynomial on a mesh of initial points in the complex plane. Points whose orbits do not go to infinity are in the filled Julia set, by definition. This popular rendering method, called the *escape time method*, raises doubts about how reliable these images are because we are left wondering whether the point sampling was representative, whether the grid was fine enough, whether the partial orbits were long enough, and whether rounding errors in floating-point arithmetic were relevant but ignored. Similar qualms apply to other rendering methods.

Therefore, for all we know, the images we have seen so far may simply be numerical artifacts. Of course, it is very unlikely that these images are actually false in any essential way, given that they have been computed independently many times and always *look* the same. However, as far as we know, no one has discussed the reliability of existing images of Julia sets, and no one has systematically produced images of Julia sets that carry mathematical guarantees. This is our goal here.

In the simplest setting, a reliable image is a black-and-white image whose pixels represent rectangles in the complex plane such that a pixel is black if and only if it contains a part of the filled Julia set. Such images are also called 1-*pixel approximations* because they show the set of points in the plane whose distance to the filled Julia set is at most the diameter of a pixel.

2010 *Mathematics Subject Classification.* 37-04, 37F50, 37F10, 37M99, 65P99.
Key words and phrases. Julia sets, adaptive refinement, cell mapping, robust rendering, fractals, computer-assisted proofs.

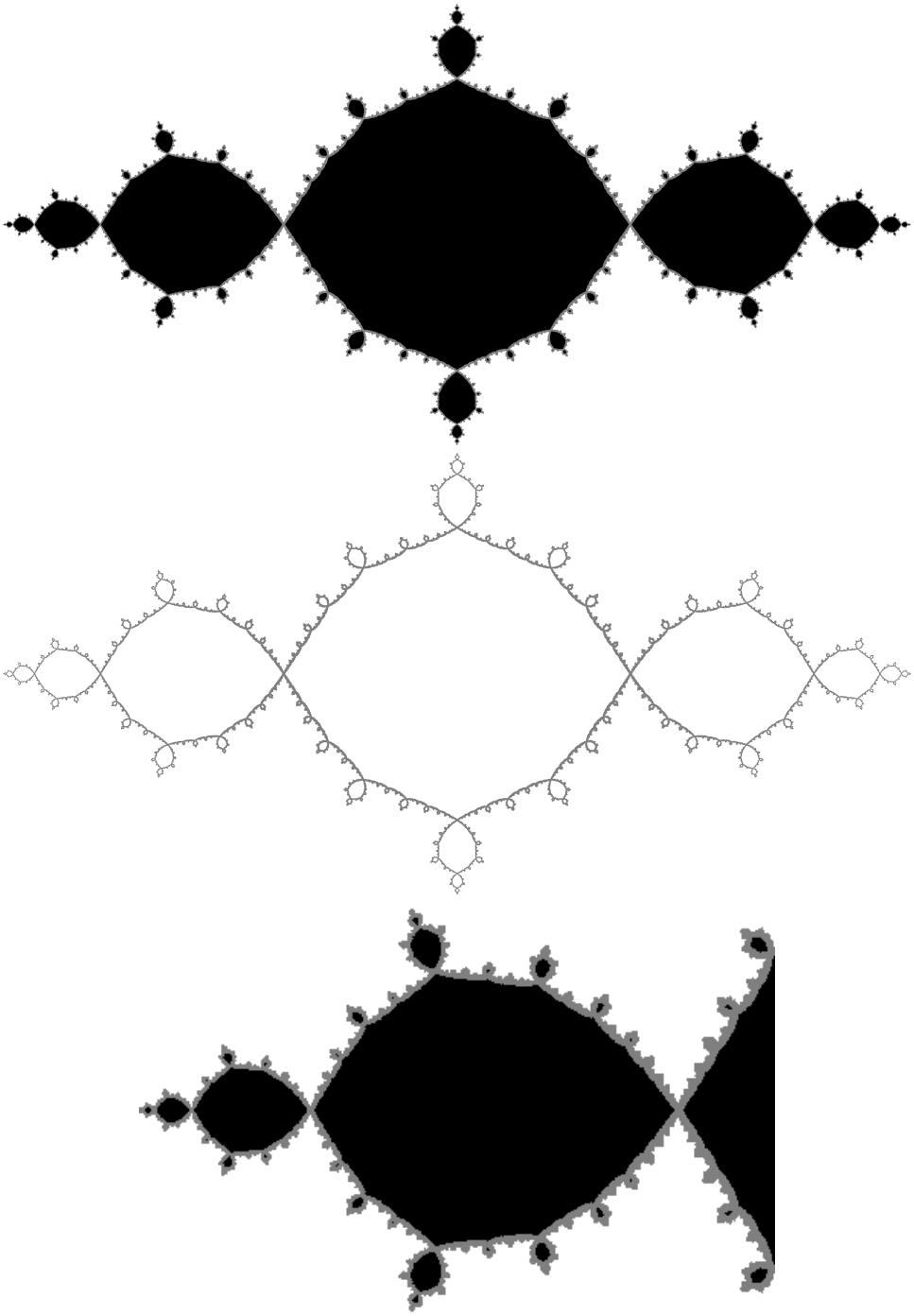

FIGURE 1. Image of the Julia set for $f(z) = z^2 - 1$ computed with our algorithm, cropped from a 4096×4096 image. Top: filled Julia set. Middle: Julia set. Bottom: zoom shows the gray region containing the Julia set. The white region is guaranteed to be outside the filled Julia set and the black region is guaranteed to be inside the filled Julia set.

In this paper, we describe how to compute images of Julia sets that are reliable in a weaker, but quite useful, sense. Our algorithm computes rigorous geometric bounds for a Julia set using a refinable quadtree decomposition [41] of the complex plane that is adapted to the Julia set. From this quadtree, we produce three-color images in which each pixel corresponds to a rectangle in the complex plane whose color comes with a mathematical guarantee: white pixels are guaranteed to be outside the filled Julia set, black pixels are guaranteed to be inside the filled Julia set, and the union of the gray pixels is guaranteed to contain the Julia set[1] (see Figure 1). Our images are not necessarily 1-pixel approximations but they are reliable because all pixels carry a mathematical guarantee for their color.

The main features of our algorithm that provide these guarantees are:

No point sampling. The algorithm uses cell mapping [18, 19] to reliably classify entire rectangles in the complex plane, not just a finite sample of points.

No partial orbits. The algorithm does not need to fix an arbitrary limit for the length of partial orbits or for the number of iterations needed to classify each pixel. In fact, the algorithm performs no function iteration at all. Instead, it tracks the fate of complete orbits by inspecting the directed graph induced by cell mapping.

No rounding errors. The algorithm uses interval arithmetic with directed rounding to handle all rounding errors robustly. In the classic quadratic case, it can use error-free fixed-point arithmetic on dyadic fractions whose range and precision depend only on the spatial resolution of the image. Standard double-precision floating-point arithmetic is sufficient to generate huge guaranteed images up to $10^6 \times 10^6$ pixels in size.

Although cell mapping methods have been proposed before for global analysis of dynamical systems via symbolic dynamics [18, 19, 8, 46, 20, 13, 9, 16, 21, 31, 1], as far as we know they have not been applied to systematically approximate Julia sets of complex polynomials. (The work of Hruska [16, 17] on Hypatia may be seen as a precursor to this but her goals were different.)

We start by defining precisely what we mean by a reliable image in §2 and by briefly recalling the main definitions and properties of Julia sets in §3. We then discuss the limitations of the main known algorithms for generating images of Julia sets in §4. We present cell mapping in §5 and our algorithm in §6. We then discuss its behavior and its limitations in §7, present some applications in §8 and some directions for future work in §9.

2. **Reliable images.**

A digital image of a compact region K inside a rectangular region Ω in the plane is obtained by first covering Ω with a uniform rectangular mesh whose cells correspond to pixels[2] and then assigning a color to each pixel. By painting each rectangle with the color of the corresponding pixel, we get a subset $\widetilde{K}$ of Ω that somehow approximates K. An image is *exact* when $\widetilde{K} = K$, but this is rarely the case: it can only happen when K has the right shape and position in Ω.

A two-color image of K tries to paint K black and the remainder of Ω white. Typically, a pixel will be black if its center lies in K, and white otherwise. More

[1] To quote Sherlock Holmes in *The Sign of Four*, "when you have eliminated the impossible, whatever remains, however improbable, must be the truth."

[2] For our purposes, pixels *are* squares, despite Alvy Ray Smith's Memo, "A pixel is not a little square" (Microsoft Technical Memo 6, July 17, 1995).

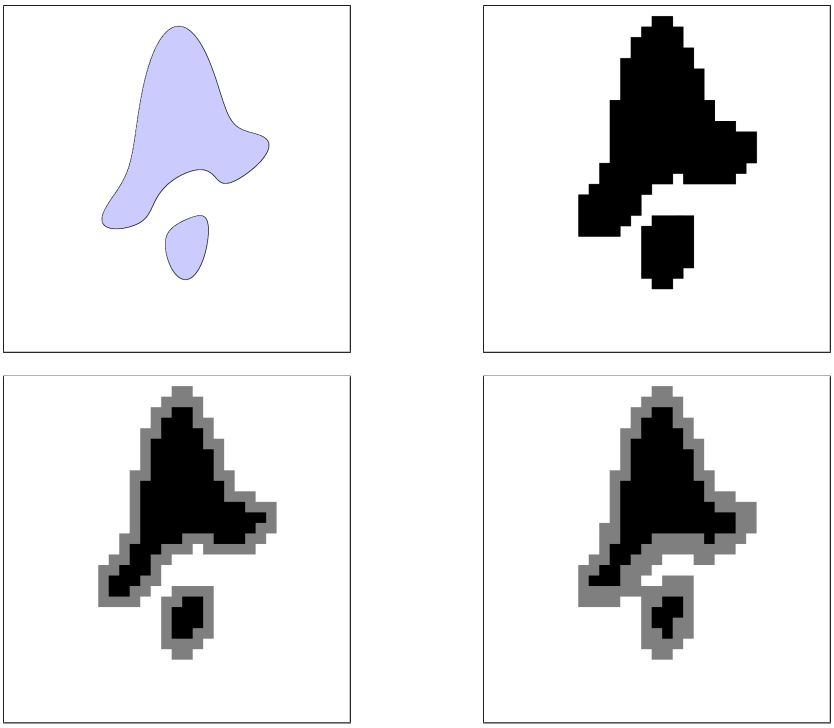

FIGURE 2. Top: A region in the plane and a two-color image of it. Bottom: Three-color reliable images. Interior cells are shown in black, exterior cells in white, border cells in gray. The left image is the best one for the given resolution. The right one is slightly less sharp, but remains reliable.

sophisticated sampling schemes exist and are frequently used. Nevertheless, $\widetilde{K}$ remains an approximation of K. In particular, $\widetilde{K}$ provides only an approximation of the membership function for K and we do not know how good this approximation is. Thus, we cannot really trust the information given by $\widetilde{K}$. Even regions given by explicit mathematical formulas require sophisticated methods for obtaining reliable images [47].

We say that an image of K is *reliable* if it is a three-color image in which the white region is guaranteed to be outside K, the black region is guaranteed to be inside K, and the gray region is guaranteed to contain the boundary of K. Such an image provides a reliable partial test for membership in K: if a point is in the white region, then it is not in K; if a point is in the black region, then it is in K. This test is not *complete*, because it does not say anything about points in the gray region, but the test is *reliable* because it never lies. A reliable image provides a precise measure of how well it approximates K: the quality of such an image is given by the size of the gray region. The *best* image for a given resolution is the one whose gray cells are exactly those intersected by the boundary of K. Best images are also called 1-*pixel approximations*. Figure 2 illustrates these concepts.

The images computed by our algorithm are reliable in the sense above, but they are not the best images in general. Nevertheless, the information that our images provide is reliable and can be used for a variety of tasks, as we shall see in §8.

3. **Julia sets.** Let f be a complex polynomial function of degree $d \geq 2$. In the classic quadratic case, $f(z) = z^2 + c$, where $c \in \mathbf{C}$. Julia sets arise naturally when we study the *dynamics* of f, that is, the behavior of the discrete dynamical system induced on $\mathbf{C}$ by the iterates of f: $f^1 = f$, $f^2 = f \circ f$, $f^3 = f \circ f \circ f$, etc. To study the long-term behavior of this dynamical system, we start with a point $z_0 \in \mathbf{C}$ and see what happens with the sequence $z_n = f^n(z_0)$, which is called the *orbit* of z_0 under f. The aim is to be able to predict the fate of each orbit.

The orbit of a point z_0 is either bounded or unbounded. It is well known that when the orbit is unbounded, it actually *escapes* away to infinity, in the sense that $|f^n(z_0)| \to \infty$ as $n \to \infty$. More precisely, there exists an *escape radius* R, which depends only on f, such that if $|z| > R$ then $|f^n(z)| \to \infty$ as $n \to \infty$. Therefore, the orbit of z_0 is unbounded iff it ever goes outside the *escape circle* of radius R centered at the origin.

In the quadratic case, we can take $R = \max(|c|, 2)$. In the general case, Douady [11] gives

$$R = \frac{1 + |a_d| + \cdots + |a_0|}{|a_d|}$$

as an escape radius for the polynomial $f(z) = a_d z^d + \cdots + a_0$. For $f(z) = z^2 + c$, this gives $2 + |c|$, which is just slightly greater than $\max(|c|, 2)$. Stroh [44] gives a smaller escape radius in the general case. However, the exact value of the escape radius does not matter much, because orbits converge exponentially to ∞ once they get beyond the escape radius.

Given that the orbit of z_0 must escape to ∞ when it is unbounded, we say in this case that z_0 is in the *attraction basin* of ∞, which is denoted by $A(\infty)$.[3] The complement $K = \mathbf{C} \setminus A(\infty)$ is thus the set of points having bounded orbits; it is called the *filled Julia set* of f. The *Julia set* J of f is the common boundary of $A(\infty)$ and K.

Even in the quadratic case, except for $c = 0$ (when it is a circle) and for $c = -2$ (when it is an interval), the Julia set J is a fractal set and so is elusive to draw. Pictures typically show the filled Julia set K when possible (that is, when K has an interior). As Devaney says [34, §3.3.3], images of filled Julia sets are somehow more appealing anyway.

The classic work of French mathematicians Pierre Fatou and Gaston Julia around 1918 highlighted the key role of the orbits of the critical points of f (i.e., the zeros of f') and established an important partial topological dichotomy: the Julia set is connected iff all critical points are in K, and the Julia set is a Cantor set if no critical points are in K.[4] In the quadratic case, because there is only one critical point, the dichotomy is complete: the Julia set is either connected or a Cantor set. This dichotomy leads to the famous Mandelbrot set, which represents in parameter space the set of $c \in \mathbf{C}$ such that J is connected. According to Fatou and Julia, this happens exactly when the critical point of f is in K, that is, when the orbit of 0 is bounded.

For details on the properties of Julia sets, see the surveys by Blanchard [2], Keen [22], Branner [3], and Milnor [26].

---

[3] When we extend f to a map of the Riemann sphere $\overline{\mathbf{C}} = \mathbf{C} \cup \{\infty\}$ by setting $f(\infty) = \infty$, we find that ∞ is an attracting fixed point of f.

[4] A Cantor set is totally disconnected: the only connected subsets are its points. The dichotomy is thus between two topological extremes: being one single piece and being essentially "fractal dust".

4. **Computing images of Julia sets.** Even in the quadratic case, the dynamics of f can be very complicated in the sense that it is very hard to decide whether a given point z_0 has a bounded orbit. (Some Julia sets are not even computable, though filled Julia sets always are [5].) Accordingly, Julia sets can be very complicated sets and many pictures have been made to try to convey their complexity [33].

There are several well-known algorithms for computing approximate images of Julia sets, such as the escape time method, the boundary scanning method, the inverse iteration method, and the distance estimation method [33, 34, 42]. These methods are easy to implement and have produced beautiful images that successfully exploit colors to convey the rich dynamics of complex maps. Nevertheless, as discussed below, these images are not really reliable because they come with no guarantees. For concreteness, we shall focus on the escape time method; similar remarks hold for the other methods.

Recall that the *escape time method* uses the definition of the filled Julia set K as the set of points whose orbit is bounded and the fact that K is contained in the escape circle. For each pixel in the image, the escape time method computes the orbit of the center of the pixel until it either exits the escape circle or a given maximum number of iterations is reached. In the first case, the point is classified as in $A(\infty)$ and the pixel is painted white. In the second case, the point is classified as in K and the pixel is painted black. By varying the color according to how long the orbit takes to exit the escape circle, beautiful images are produced.

The images produced by these methods are not reliable because they use point sampling, compute partial orbits to classify samples, and may be subject to rounding errors:

Point sampling. The images are produced by choosing a sample of points on a grid laid over a region of interest and classifying each sample point with respect to the Julia set. The methods tacitly assume that each sample point can be classified reliably and that the behavior between sample points is represented reliably by the behavior of the sample points nearby. Even if these assumptions were founded, one cannot choose the grid resolution reliably a priori. To mitigate this problem, one typically uses a finer grid to get more detail and gain more confidence on the images produced. However, for a fixed resolution, there are no guarantees that the images obtained by point sampling are correct.

Partial orbits. All these methods compute partial orbits for many points and so need to fix an arbitrary limit for the number of iterations performed. The methods compute at most N points in each orbit. In the escape time method, it may well happen that the first N points in an orbit are inside the escape circle but further points land outside it, and the orbit diverges. In this case, the starting point will be erroneously classified as in K instead of $A(\infty)$. Relying on partial orbits of fixed bounded length is a real problem in all methods: we cannot run the program forever and we cannot choose an N that is large enough for all points, because points in $A(\infty)$ that are very near J may take arbitrarily long to leave the escape circle. To mitigate this problem, one typically uses a large N to get more detail and gain more confidence on the images produced. However, in most cases there is a limit beyond which increasing N does not produce any visible improvement in the images, despite the much increased computation time. This may be naively taken as a sign that the image is correct, even though no such guarantees are made.

Rounding errors. To represent in a computer the exact value of a quadratic function at a point, one needs at least twice the number of bits used to represent the point. For a polynomial of degree d, one needs at least d times the number of bits. Thus, iterating a polynomial function using fixed-precision floating-point arithmetic may rapidly become inaccurate and so rounding errors during the iteration may influence the classification of a point, especially near the Julia set. In the escape time method, if rounding errors take an orbit outside the escape circle, then the starting point will be classified erroneously as in $A(\infty)$, even if the orbit in fact remains inside the circle forever. On the other hand, rounding errors may erroneously keep the orbit of a point inside the circle, thereby producing false points inside K. Some programs use multiple-precision floating-point arithmetic to avoid these problems, especially in deep zoom around the Julia set, at the cost of greatly increased computation times. The issue remains whether rounding errors during iteration actually do influence the classification of points. As far as we know, this issue has never been formally studied.[5]

In summary, the usual methods for producing pictures of Julia sets leave us wondering whether the point sampling was reliable, whether the grid was fine enough, whether the partial orbits were long enough, and whether rounding errors were relevant. Our algorithm does not suffer from these limitations.

The distance estimation method, described by Milnor [26, appendix H] and by Fischer [34, appendix D.3], approximates the filled Julia set by covering its exterior with an increasing family of disks. This is superficially similar to what we do with rectangles, but the computation of distance estimates requires function iteration and so is subject to the limitations discussed above.

Previous work on reliable algorithms for Julia sets has focused mainly on the computability of Julia sets and on the computational complexity of approximating Julia sets [37, 36, 4]. An excellent reference for these topics is the book by Braverman and Yampolsky [5]. This is important theoretical work that probes the limits of what in principle can be computed about Julia sets. In particular, that research focuses on 1-pixel approximations of the Julia set inside a rectangular region Ω.

Our algorithm delivers an approximation of the Julia set of a polynomial f inside a rectangular region Ω that is reliable in the sense of §2, but not necessarily is a 1-pixel approximation. Another limitation is that our algorithm needs Ω to contain the escape circle for f, and will process the entire Ω, even if the region of interest is a smaller subregion. This limits the amount of zoom that can be performed. This limitation is inherent to using cell mapping because f is transitive on J.[6]

5. **Cell mapping.** The core idea in *cell mapping* [18, 19, 8, 9, 31] is using a finite directed graph, called a *cell graph*, to represent the dynamics of the discrete dynamical system induced by a map $f\colon \mathbf{C} \to \mathbf{C}$. The motivation is to study the

[5] Milnor [26, page 49] says "The [inverse iteration] method is very insensitive to round-off errors, since f tends to be expanding on its Julia set, so that f^{-1} tends to be contracting." Steinmetz [43, page 175] says "Theorem 1 makes it plausible that in almost all cases rounding errors do not affect the computer graphics of Julia sets." Munafo [29] argues that rounding errors have nearly no influence on computed pictures of the Mandelbrot set. Similar arguments work for Julia sets. That is all we have found about this issue for polynomial Julia sets. For exponential Julia sets, Durkin [12] has proved that the escape time method produces correct images.

[6] Transitivity essentially means that the orbit of almost every point of J passes arbitrarily near every other point of J.

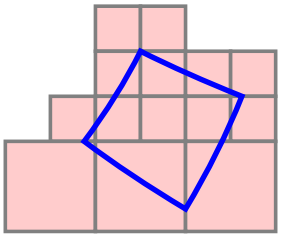

FIGURE 3. The cell graph has an edge $A \to B$ whenever $f(A)$ intersects B. This is illustrated here: $f(A)$ is shown in blue and the cells B that intersect $f(A)$ are shown in red.

fate of whole regions under the iteration of f instead of the fate of individual orbits. Cell mapping is one of the two main steps in our algorithm.

The cell graph. We start with a complete decomposition of $\mathbf{C}$ into a finite number of closed abutting cells. Our algorithm uses the cells of a quadtree decomposition of a rectangle Ω containing the escape circle for f, plus one cell representing the complement of Ω, which we call the *exterior*. Other special cells may appear later.

The vertices in the cell graph are exactly the cells in the given decomposition of $\mathbf{C}$. An edge $A \to B$ in the cell graph goes from vertex A to vertex B whenever $f(A)$ intersects B. Thus, $f(A)$ is contained in the union of all cells B that are the target of an edge $A \to B$ in the cell graph, as illustrated in Figure 3.

Dynamics in the cell graph. The cell graph captures the dynamics of f in the following sense: every orbit starting in a cell A can only visit the cells that can be reached from A by following a path in the cell graph. Therefore, the fate of such orbits is captured by the fate of such paths — and this can be found by graph traversals.

The cell graph is a conservative approximation for the dynamics of f. Indeed, edges $A \to B$ lead to cells that only partially overlap $f(A)$, and so in general $f(A)$ is properly contained in the union of the cells that intersect it (see Figure 3). Therefore, when we follow orbits starting in A by following paths starting at A, we will most probably visit cells that no longer intersect any orbits starting in A. Nevertheless, *all* orbits starting in A will be tracked by such graph traversals.

The dynamics of the cell graph are captured and summarized by its *condensed graph*, obtained by contracting each strongly connected component[7] of the cell graph to a single vertex. The condensed graph is *acyclic*, and its *sinks* play a key role in the dynamics. For brevity, a strongly connected component of the cell graph that is a sink in the condensed graph will be called simply a *sink component*.

The crucial observation is that an orbit is trapped once it enters a cell of a sink component, because edges starting at a cell of a sink component never leave that component. In this sense, sink components represent the possible fates for orbits. A sink component corresponds to an attracting region of f, that is, a neighborhood of an attracting fixed point or attracting periodic cycle. Figure 4 illustrates this. Note that the number of geometric connected components in a sink component is

[7] A subgraph H of a directed graph G is *strongly connected* if for each two vertices of H there is a path in H connecting them. A *strongly connected component* is a maximal strongly connected subgraph.

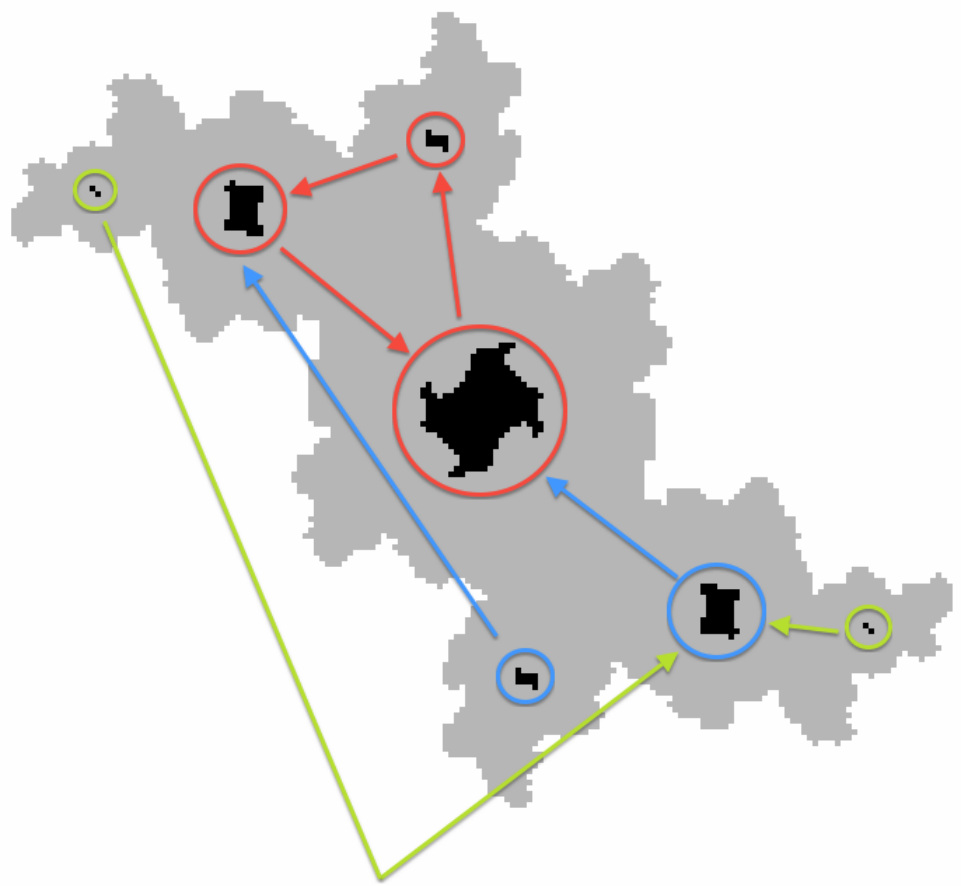

FIGURE 4. All edges from cells in the red group go to cells in the red group. All edges from cells in the blue group go to cells in the red group. All edges from cells in the green group go to cells in the blue group. We conclude that the cells in the red group form a sink component trapping a periodic cycle of period 3 and all seven groups are inside K.

the period of the cycle. The exterior is a sink component which we know a priori because ∞ is an attracting fixed point.

Our algorithm finds sink components and labels cells according to which sink component they are led to, if any. This is the other main step in the algorithm.

Finding the cell graph. The main difficulty in cell mapping is finding the edges in the cell graph, that is, given a cell A, deciding which cells intersect $f(A)$. We call this the *edge problem*.

For general nonlinear functions f, there is no simple geometric description for $f(A)$ on which to base an exact intersection test. The simplest solution for the edge problem is to use point sampling [8, 31]. While this solution gives good results in many cases, it is not guaranteed to find the complete cell graph: it may miss an

edge $A \to B$ simply because no sample point in A was mapped into B. Missing edges is fatal to our goal of computing rigorous geometric bounds.

Instead of using point sampling, we solve the edge problem by finding an *enclosure* $T \supseteq f(A)$ and adding edges $A \to B$ whenever T intersects B. The union of all such cells B contains T and so a fortiori contains $f(A)$. We use the simplest enclosure: a bounding box. The simple geometry of T greatly simplifies the intersection test. Figure 5 illustrates the technique.

The cell graph obtained by using T as a proxy for $f(A)$ contains more edges than the exact cell graph. (Compare the red cells in Figure 3 with the ones in Figure 5.) Using this larger graph makes cell mapping even more conservative than it already is by nature, but the technique remains simple and quite effective.

The natural computational tool for finding a bounding box T for $f(A)$ is *interval arithmetic* [27, 28]. Cell mapping using interval enclosures is the basis for reliable algorithms that reason about discrete dynamical systems [13, 16, 24, 32]. Several good interval libraries exist [23] and can be easily used to evaluate polynomial expressions on rectangles A in the plane, automatically giving a rectangle that contains $f(A)$. The evaluation is done robustly in standard floating-point arithmetic using directed rounding to guarantee reliable enclosures.

The enclosures computed with real interval arithmetic are always correct but rarely tight. For a complex polynomial f, better enclosures can be computed with an algorithm by Rokne [38]. Nevertheless, overestimation is not a very serious problem in our algorithm because the enclosures improve steadily as the cells decrease in size. Moreover, since our algorithm does not perform function iteration, it is not subject to the wrapping effect that plague some interval methods for dynamical systems.

In the quadratic case, $f(A)$ is a curvilinear quadrilateral whose boundary is formed by parabolic arcs and possibly one line segment (see Figure 5). In this case, an exact intersection test between $f(A)$ and a rectangle B can be devised in principle. However, this test is tedious to implement robustly. On the other hand, the bounding box computed with real interval arithmetic is tight in this case. This is a consequence of a more general result: when all variables in an expression appear exactly once, interval arithmetic computes the best bounds [28, Theorem 6.2]. For cells A in a quadtree decomposition of $\Omega = [-2, 2] \times [-2, 2]$, we do not even need interval arithmetic because a tight bounding box for $f(A)$ coincides with the bounding box of the images of the vertices of A under f.

6. Our algorithm.

Our algorithm computes a decomposition of the complex plane $\mathbf{C}$ into three reliable regions: a white region, which is contained in $A(\infty)$, a black region, which is contained in K, and a gray region, which contains J. Since K is contained in the escape circle of radius R centered at the origin, we can concentrate our attention on the square $\Omega = [-R, R] \times [-R, R]$, because then the complement $\mathbf{C} \setminus \Omega$ is contained in $A(\infty)$. However, our algorithm works on any larger rectangle Ω that contains the escape circle. In the quadratic case $f(z) = z^2 + c$, we get $R = 2$ and $\Omega = [-2, 2] \times [-2, 2]$ when $|c| \leq 2$, which is convenient for examples. (When $|c| > 2$, the Julia set is always a Cantor set.)

Our algorithm uses an adaptive refinement method that computes a quadtree decomposition [41] of Ω into rectangular cells and assigns an appropriate *label* to each leaf cell in this tree. At the end, we generate an image from the quadtree by mapping labels to actual colors (see Figure 6).

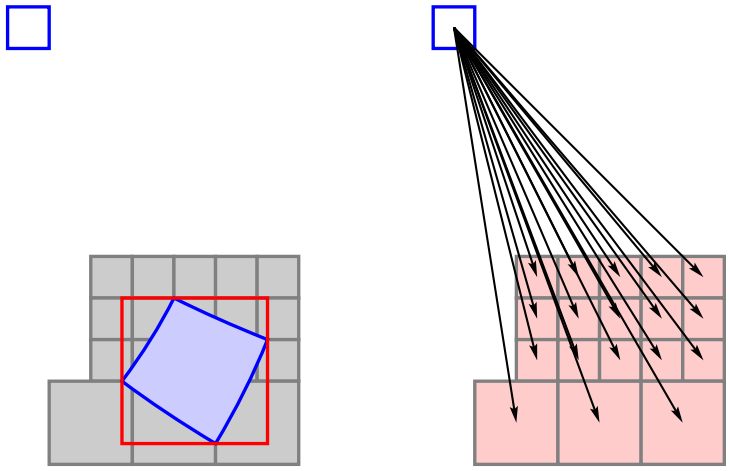

FIGURE 5. Image of the cell $[-1.25, -1] \times [0.5, 0.75]$ under $f(z) = z^2 - 1$: exact image (blue), bounding box (red), target cells (gray), and the edges in the cell graph starting at the original cell. Compare with Figure 3.

All the action happens on leaf cells labeled *active*: they are yet to be classified with respect to J but their union contains J. Other leaf cells are labeled according to the particular sink component that traps all orbits emanating from them. We say that such cells are *labeled by* the corresponding sink. Leaf cells labeled by the exterior sink component are known to be outside K. Leaf cells labeled by other sink components, if any, are known to be inside K.

The steps performed by the algorithm are explained briefly here and in more detail below. Besides adaptive refinement (step 1), the main features of our algorithm are the use of *cell mapping* (step 2) and *label propagation* in graphs (step 3):

Julia(f, Ω):

0. [*init*] Start with a single-node quadtree containing an active cell over Ω. Create a special sink cell representing the complement of Ω, i.e., the exterior sink component.
1. [*refine*] Subdivide each active cell in the quadtree into four new active subcells.
2. [*build cell graph*] Add a vertex to the graph for each active cell and for each known sink cell. For each active cell A, compute an enclosure $T \supseteq f(A)$. For each leaf cell B that intersects T, add an edge $A \to B$ to the graph. Add an edge from each sink cell to itself.
3. [*propagate labels*] Find the sink components in the condensed cell graph and create sink cells and labels for each sink found. Propagate the label of each sink component to the cells that are trapped by it.
4. [*prune*] Prune the quadtree from the bottom up. Each internal node whose children all have a common label, different from active, becomes a leaf with that label.
5. [*repeat*] If the desired quadtree resolution has not been reached, go back to step 1.

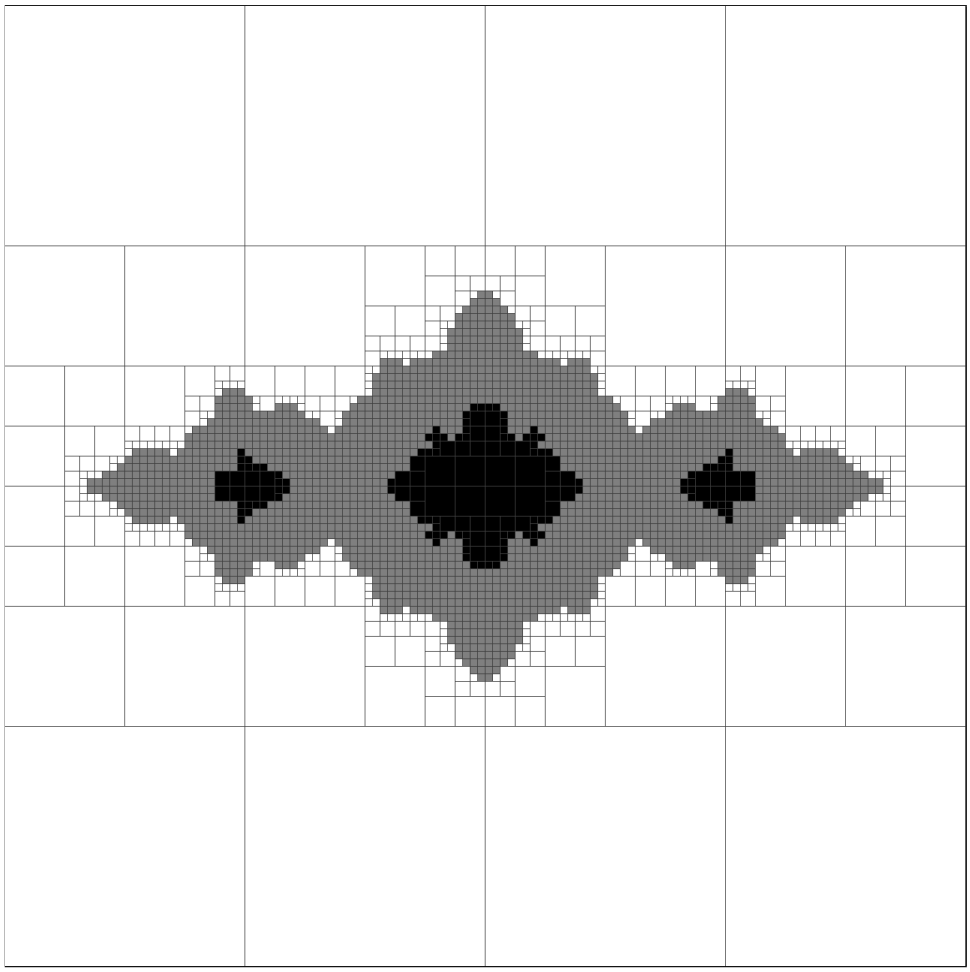

FIGURE 6. Quadtree of refinement level 7 computed by our algorithm for $f(z) = z^2 - 1$ and $\Omega = [-2, 2] \times [-2, 2]$. The white region is contained in $A(\infty)$, the black region is contained in K, and the gray region contains J.

When the algorithm ends, the resulting quadtree can be stored for analysis (such as the estimation of area or fractal dimension), for further refinement by restarting the algorithm at step 1, or for rendering by painting the leaves with colors assigned to each label. We shall discuss these and other applications for the quadtree in §8. We now give the details of the algorithm.

Step 0 [*init*]. The single active cell over Ω reflects our initial knowledge that J is completely contained in Ω. The exterior sink cell reflects our initial knowledge that the exterior is contained in $A(\infty)$. (Hence the requirement that Ω contains the escape circle.) There are no other sink cells and therefore no cells with other labels. Together, these three facts establish the initial fundamental invariant of the algorithm: the cells labeled by the exterior are contained in $A(\infty)$, the cells labeled by other sinks (none at this moment) are contained in K, and the active cells contain J.

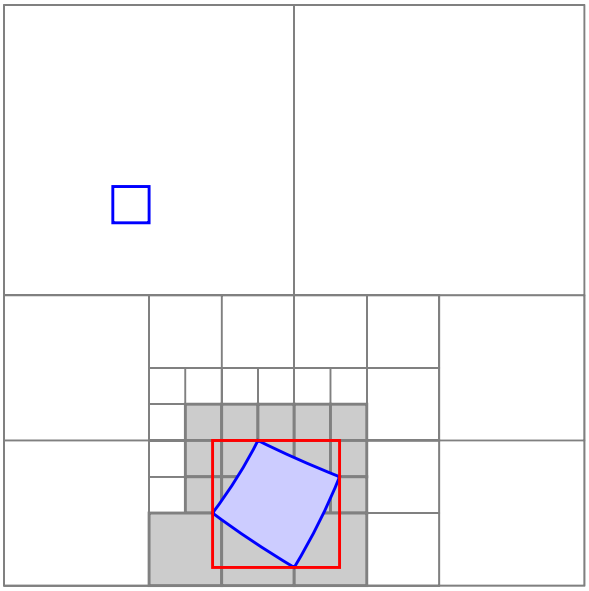

FIGURE 7. Image of the cell $[-1.25, -1] \times [0.5, 0.75]$ under $f(z) = z^2 - 1$: actual image (blue), bounding box (red), quadtree traversal for locating the leaves that intersect the bounding box.

Step 1 [*refine*]. This is the standard quadtree refinement step. Each active cell is subdivided into four equal child cells by bisecting its sides. The child cells are labeled *active* and their parent cell loses its label, since it is no longer a leaf. This is an *adaptive* refinement because only active cells are refined. Cells labeled by sink cells are not refined because their fate is already known.

The aim of refining is to classify a subset of the new active cells (typically those near the boundary of the *active* region) with a sink label, thus improving the approximation. The expectation is that by reducing the size of cells we get better enclosures and a more precise cell graph.

Step 2 [*build cell graph*]. In this step, we build a directed graph that represents the cell mapping induced by f on $\mathbf{C}$. Each active cell becomes a vertex in the cell graph. We also include in the graph a special *sink cell* for each sink component that we already have found. We know a priori that the exterior is a sink component because it leads only to itself. Therefore, the exterior is a sink cell. Like the exterior, all sink cells have self-loop edges only. All other edges in the cell graph emanate from the active cells because the fate of inactive leaves is already known.

Recall that we find the edges emanating from a cell A by finding a bounding box T for $f(A)$ and adding edges $A \to B$ whenever T intersects B. We traverse the quadtree to locate the leaves that intersect T, thus avoiding testing T against all leaves in the quadtree (see Figure 7). The simple geometry of T simplifies both the traversal and the intersection test. When B is not active, we change the edge $A \to B$ to $A \to S$, where S is the sink cell corresponding to B's label. This helps to reduce the size of the cell graph by having as vertices only active cells and sink cells.

After this step, we have a directed graph whose vertices are the active leaves plus all the sinks and whose edges emanate from all the active leaves. This graph

is needed only for the label propagation in step 3; a new graph is built after every refinement step because it is based on a different set of leaves.

Step 3 [*label cells*]. Recall that in cell mapping we follow orbits by following paths in the cell graph and that the sink components play a key role because they trap orbits. In this step, we label each active cell according to where the orbits starting in that cell go. If *all* such orbits go to the *same* sink component, the cell gets the label of that component and so becomes inactive. Otherwise, the cell remains active.

Our strategy for this *label propagation* is: First, we find the sink components of the cell graph, giving a different label to each one. Then, each strongly connected component that goes to a single sink component gets the label of that sink. This process is repeated for each strongly connected component that only goes to components of the same label; the component (and their cells) then get that label. The cells in components that do not go to a single sink naturally remain active.

The exterior cell is a sink component of the cell graph because it leads only to itself. All cells trapped by the exterior cell must then be inside $A(\infty)$. Any other sink component must be inside K, because the orbits trapped by that sink component are bounded forever and can never escape to infinity. All cells trapped by an interior sink component must then also be inside K.

In the quadratic case, there are at most two sink components. Higher-degree polynomials can have multiple attracting regions.[8] It is useful to give different labels to different interior sinks and be able to tell different attracting regions apart. This opens opportunities for rendering nicer, more informative images. Figure 8 shows an example of a Julia set with two sinks and their basins of attraction.

Except for the exterior, which is known a priori, the identification of attracting regions via sink components depends on the geometric resolution of the cells. It may take several refinement steps until an internal sink component appears. Different sink components may appear at different levels. Smaller attracting regions appear later. Figure 8 also illustrates this.

We use Tarjan's classic algorithm [45] to find the strongly connected components of the cell graph, and implicitly its condensed graph. An important feature of Tarjan's algorithm is that it finds strongly connected components in reverse topological order. In particular, it finds the sink components first. We give a different label to each sink component when we find it. We propagate these labels to the other strongly connected components as soon as these components are found by Tarjan's algorithm: we just have to check that each newly found strongly connected component only points to strongly connected components of the same label, in which case it gets that label. This label propagation does not alter the linear-time complexity of Tarjan's algorithm.

This step propagates sink labels to previously active cells. Cells newly labeled by the exterior sink component increase the area we are certain belongs to $A(\infty)$. Cells newly labeled by other sink components increase the area we are certain belongs to K. The union of all cells that remain labeled active still contains J. The fundamental invariant of the algorithm is thus preserved.

Step 4 [*prune*]. In this step, we consolidate the new label information found in step 3. Whenever all four child nodes of a cell have the same label and this label is not *active*, their parent node becomes a leaf with that label and the child nodes

[8] According to Fatou and Julia, each attracting region in $\mathbf{C}$ contains a critical point of f, and so there are at most $d-1$ attracting regions if f has degree d.

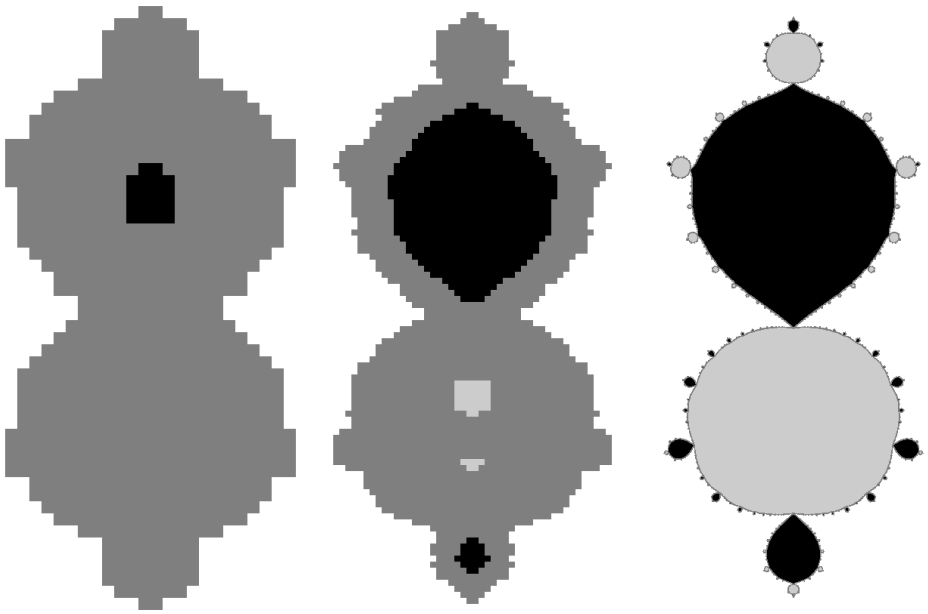

FIGURE 8. A cubic Julia set having two attracting fixed points, shown at level 14 (right). Their basins of attraction are shown in black and light gray. The corresponding interior sinks are found at level 6 (left) and at level 7 (middle). Note how we can tell from the final image which bulbs go to which of the two large bulbs. Here, $f(z) = z^3 + az + b$ with $a = 1.25$ and $b = 0.025\,i$.

are removed from the quadtree. This process is repeated recursively upwards the quadtree.

This pruning guarantees that the quadtree is as shallow as possible or, equivalently, that its leaves are as large as possible, within the geometric constraints of the quadtree decomposition. Although this step is not strictly necessary, it is simple and quite useful because it improves the traversals in step 2 and simplifies the traversals used for image generation and other geometric applications discussed in §8.

7. **Discussion.** Figure 9 shows the refinement process performed by our algorithm to compute the Julia set for $f(z) = z^2 + c$ with $c = -0.12 + 0.74\,i$. The evolution of the approximation shown there is typical: As soon as some white cells are found (at level 2 here), the approximation of the exterior improves steadily after each refinement. Similarly, as soon as some interior cells are found (at level 8 here), the next few refinements find most of the interior quickly.

Our algorithm works for different types of filled Julia sets: connected sets with non-empty interior (Figure 9), connected sets with empty interior (Figure 10), and totally disconnected Cantor sets (Figure 11). The supplemental material contains animations of the refinement process for a few Julia sets. We have computed several thousand pictures of quadratic Julia sets. They are available as an interactive panorama of the Mandelbrot set.[9]

[9] http://monge.visgraf.impa.br/panorama/viewer/index.html?img=../julia-256GP/julia.xml

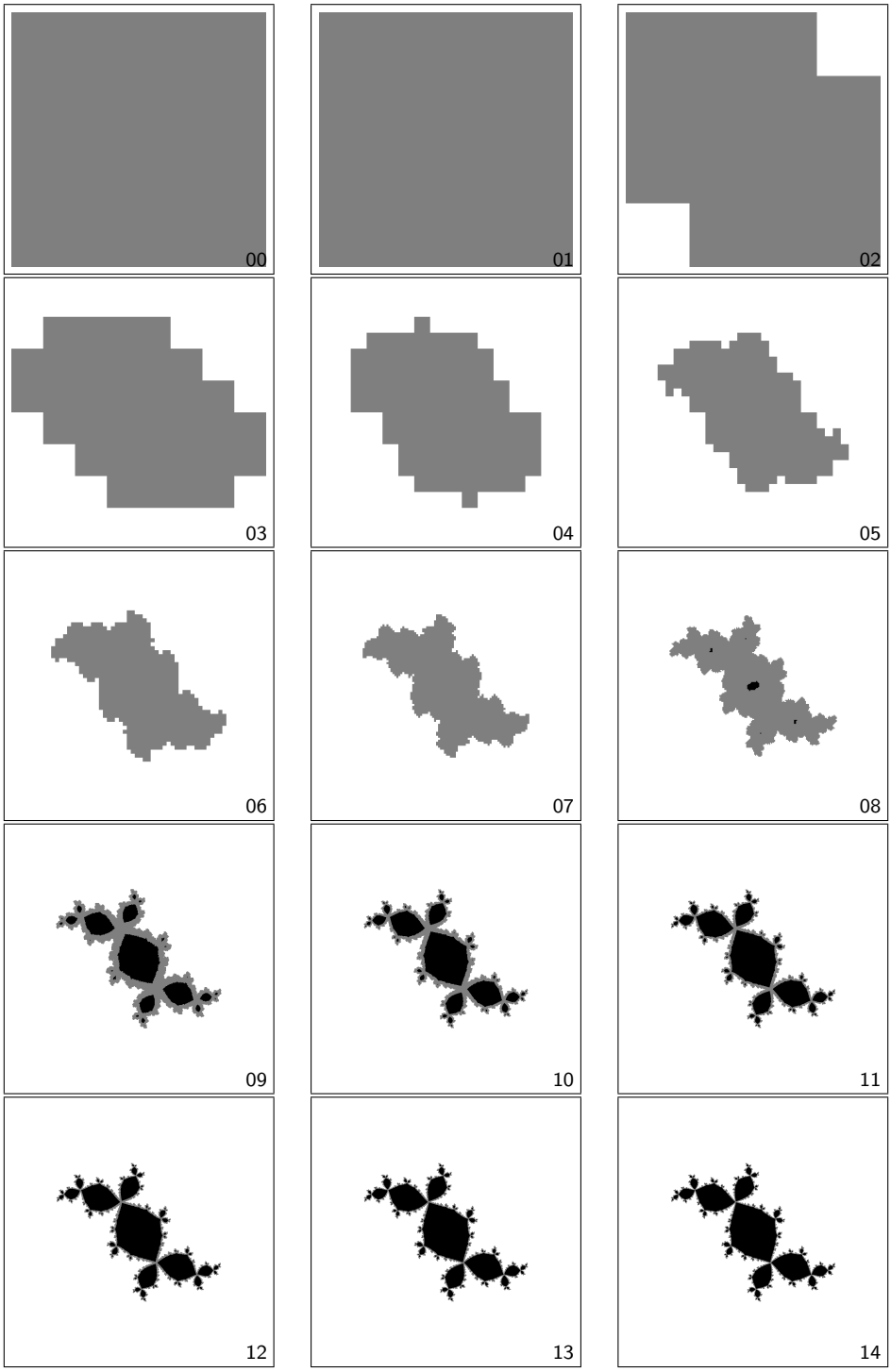

FIGURE 9. Levels 0 to 14 of adaptive approximation of the Julia set for $f(z) = z^2 + c$ with $c = -0.12 + 0.74\,i$.

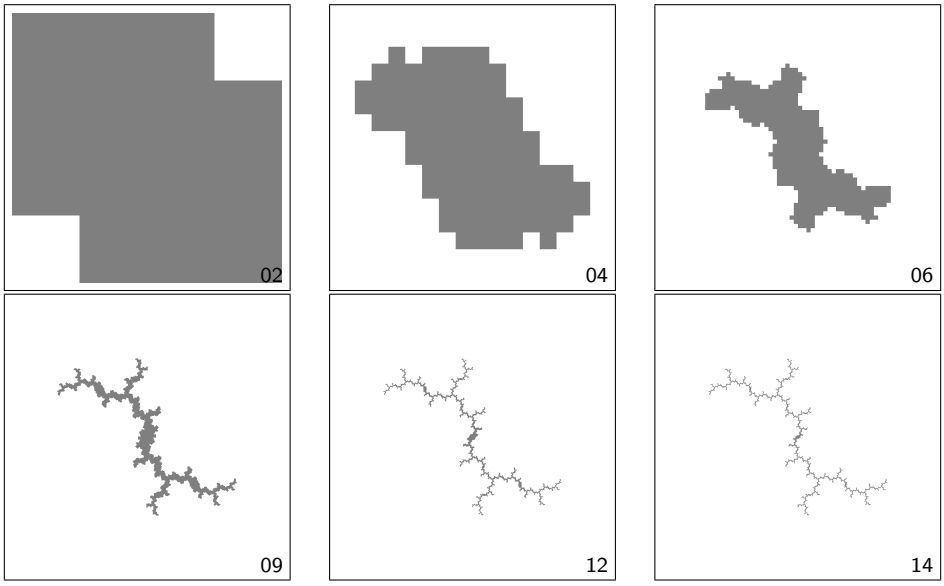

FIGURE 10. Approximation of quadratic Julia set with $c = 0 + 1\,i$.

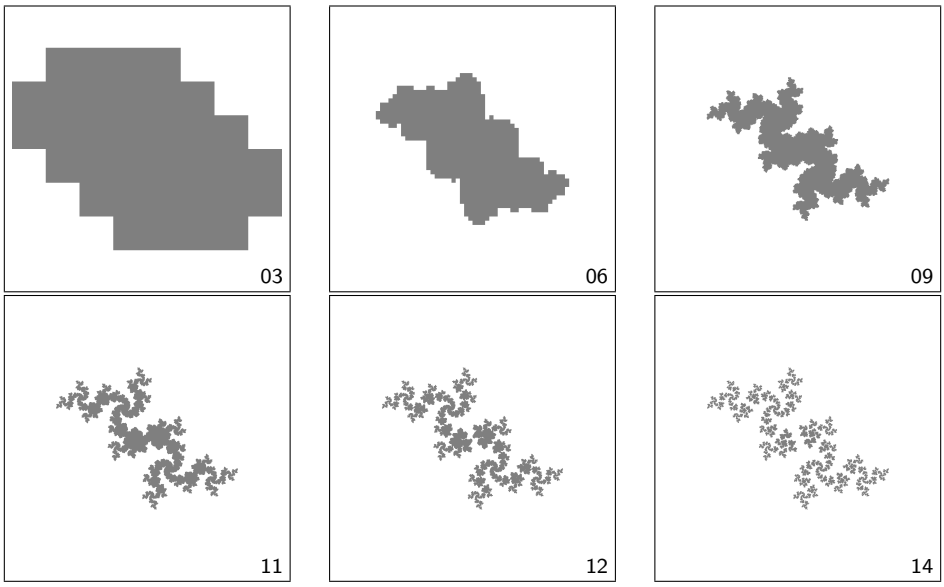

FIGURE 11. Approximation of quadratic Julia set with $c = -0.25 + 0.74\,i$.

For the filled Julia sets with non-empty interior, it may take several refinements steps until the interior first appears, and then further refinements seem to find most of the interior quickly. See Figure 12 for a striking example. The algorithm discovers interior cells when it finds a sink component that is not the exterior. The cells in this component form a neighborhood of the attracting fixed point or periodic cycle

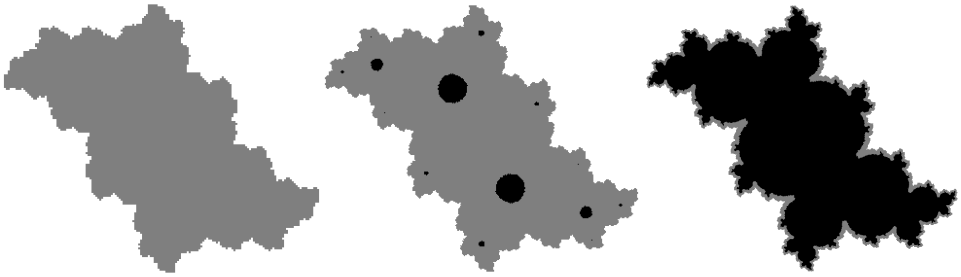

FIGURE 12. Interior cells appearing suddenly and quickly filling the interior. At level 8 (left), no interior cells were found. At level 9 (middle), a small sink is found. At level 10 (right), most of the interior appears. Here, $f(z) = z^2 + c$ with $c = -0.12 + 0.6\,i$.

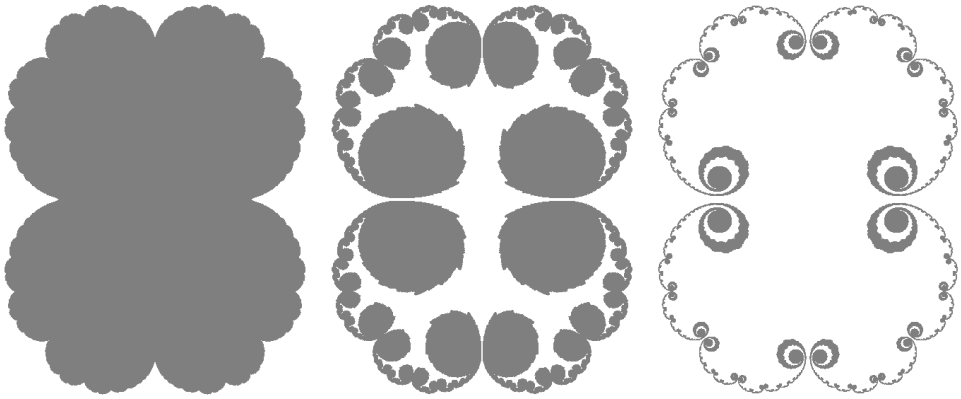

FIGURE 13. Julia set suddenly breaking. At level 10 (left), it seems connected. At level 11 (middle), it breaks. At level 14 (right), more pieces are apparent. In the limit, the Julia set is a Cantor set. Here, $f(z) = z^2 + c$ with $c = 0.26$.

that is mapped to itself under f. The size of this neighborhood depends on the magnitude of the derivative of f at the attractor, which naturally depends on f, that is, on c. Once the interior appears, we have a computational proof that c is in the Mandelbrot set.

When the Julia set is a Cantor set, especially when c is near the boundary of the Mandelbrot set, the algorithm may need to perform several refinements steps before the Julia set emerges as disconnected (see Figure 13). Once it does, we have a computational proof that c is not in the Mandelbrot set (see also Figure 11).

The parameter $c = 0.25$ is on the boundary of the main cardioid of the Mandelbrot set. The corresponding filled Julia set has an interior and looks very much like the one in Figure 13 (left), except that the interior should be painted black. However, our algorithm cannot find any point in the interior because the attracting fixed point for $c = 0.25$ is on the boundary of the filled Julia set. Therefore, every cell containing the fixed point must have an edge to the exterior and so stays active forever, regardless of the refinement level. Julia sets for such bifurcation points are notoriously difficult to render reliably.

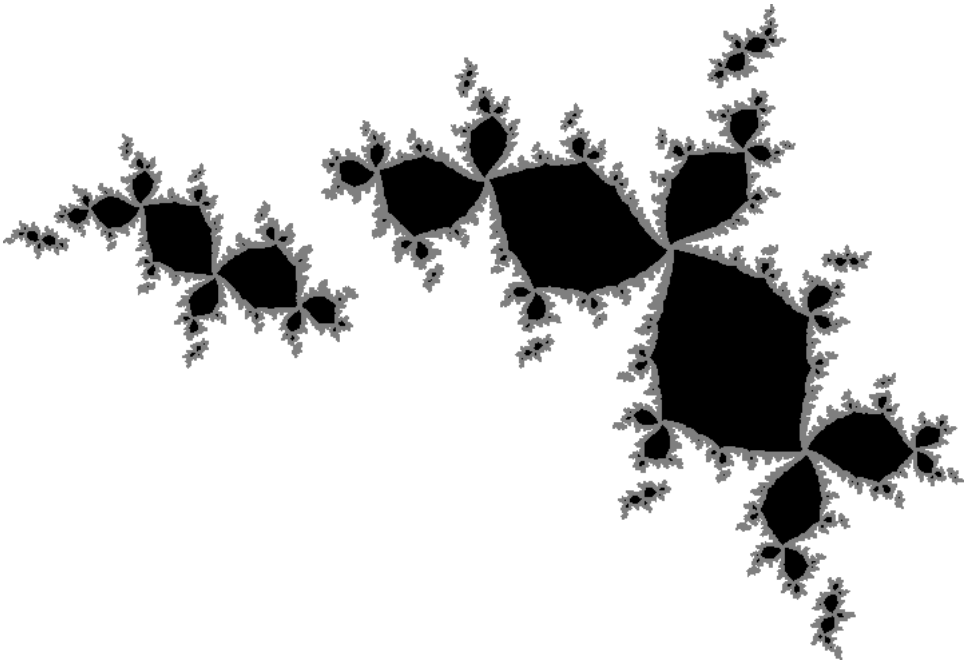

FIGURE 14. A disconnected cubic Julia set with interior, shown at level 14. The interior appears at level 10. The Julia set breaks at level 11. Here, $f(z) = z^3 - 3a^2z + b$ with $a = -0.5769525 + 0.175\,i$ and $b = 0.8$.

Our algorithm works for polynomials of higher degree and produces images of Julia sets that are not as well known as the quadratic ones. Figure 8 and 14 show two cubic Julia sets. Unlike quadratic Julia sets, cubic ones can simultaneously have an interior and be disconnected (Figure 14).

Performance. The performance of the algorithm seems quite uniform, in the following sense: In all our tests, using a prototype implementation in Lua, step 2 took about 80% of the total time, step 3 about 15%, and steps 1 and 4 less than 2% each. The algorithm took from 10 seconds to 5 minutes of CPU time and used from 175M to 12G of memory (see Table 1). Actual times and memory usage depended on the polynomial f. For $f(z) = z^2 + c$, the time depends on where c is in the Mandelbrot set, and Table 1 reflects this. The algorithm tends to be faster when c is near the center of a hyperbolic component and slower when c is near the boundary. Similar remarks apply to cubic polynomials, but the parameter space in this case has two complex parameters and is much more complicated [25].

Here are some brief comments on the tests in Table 1:

- q1 is in the center of the period-2 bulb and its Julia set is easy to approximate.
- q2 is near the center of the main cardioid, which is the period-1 bulb, and its Julia set is very easy to approximate.
- q3 is in the main cardioid but is close to a period-3 bulb and its Julia set is harder to approximate.

name	figure	time	mem	c
q1	1	47.52	971	$-1 + 0\,i$
q2	–	9.91	175	$-0.12 + 0.3\,i$
q3	12	38.77	722	$-0.12 + 0.6\,i$
q4	9	122.33	2350	$-0.12 + 0.74\,i$
q5	10	55.81	1040	$0 + 1\,i$
q6	11	371.51	7193	$-0.25 + 0.74\,i$
q7	–	17.19	324	$0.2499 + 0\,i$
q8	–	808.96	12318	$0.25 + 0\,i$
q10	13	316.39	5078	$0.26 + 0\,i$
qp31	–	28.47	545	$-1.7548776662467 + 0\,i$
qp41	–	111.41	2283	$-1.3107026413368 + 0\,i$
qp42	–	86.77	1774	$-0.15652016683376 + 1.0322471089228\,i$
qp43	–	206.12	4033	$0.28227139076691 + 0.53006061757853\,i$
qp51	–	274.66	5050	$-0.50434017544624 + 0.56276576145298\,i$
qp52	–	305.14	6421	$0.37951358801592 + 0.3349323055975\,i$

TABLE 1. Time (secs) and memory (Mb) used to compute some quadratic Julia sets for $f(z) = z^2 + c$ to level 14.

- q4 is near the center of that period-3 bulb and its Julia set is even harder to approximate.
- q5 is at the tip of an antenna and its Julia set is connected but has no interior.
- q6 is somewhat near q4 but is outside the Mandelbrot set. Its Julia set is a Cantor set.
- q7, q8, q10 show a transition from the main cardioid to the exterior. q7 is in the main cardioid and its Julia set is very easy to approximate, despite q7 being very near to q8, which is at the boundary of the main cardioid. The interior of the Julia set of q8 will never appear, as discussed earlier. q10 is near q8 but outside the Mandelbrot set. Its Julia set is a Cantor set.
- qp31 is at the center of a period-3 bulb. Its Julia set is easy to approximate even though its interior first appears at level 11. It is easier than the Julkia set of q4.
- qp41, qp42, qp43 are period-4 points and qp51, qp52 are period-5 points. Their Julia sets have different geometries, which make them different to approximate.

Limitations. All images of Julia sets produced by our algorithm are reliable. Nevertheless, our algorithm has some limitations.

In many cases, our algorithm seems to converge as the resolution of the quadtree goes to infinity in the sense that if there were no limits on time, memory, spatial resolution, and arithmetic precision, then the algorithm would correctly classify every point in $A(\infty)$ and in K. In practice, the algorithm is limited by these finite resources.

The main limitation is memory, a limitation shared by all cell mapping methods [31]. The resolution of the quadtree and the size of cell graph are limited by the available memory. In current machines, we cannot go beyond level 20, which translates to a spatial resolution of a little less than 4×10^{-6}, even if it allows the generation of huge images (up to $2^{20} \times 2^{20} \approx 10^6 \times 10^6$ pixels).

The other limitation is that our algorithm is *global* in the sense that it needs to compute a quadtree for the entire square $\Omega = [-R, R] \times [-R, R]$, even if the region

of interest is a smaller subrectangle. This requirement limits the amount of zoom that can be performed. This limitation is inherent to using cell mapping for Julia sets because f is transitive on J. Indeed, transitivity implies that the active cells around the Julia set form a strongly connected component[10]; once we include such a cell in the graph, we need to include all of them.

About floating-point computations. Our algorithm works with standard double-precision floating-point arithmetic but is not subject to errors due to rounding or truncation because it uses interval arithmetic with outward rounding. This means that, if necessary, lower limits of intervals are rounded down, towards $-\infty$, to the previous floating-point number, and upper limits of intervals are rounded up, towards $+\infty$, to the next floating-point number. In particular, the escape radius R is rounded up and the coefficients of the polynomial f are represented either by a single floating-point number when they are exact or by a tiny interval of consecutive floating-point numbers.

As a consequence, the Julia sets that we show are actually the combination of the Julia sets for all such tiny perturbations of the coefficients of f: the white region is the intersection of all white regions, the black region is the intersection of all black regions, and the gray region is the union of all gray regions. Since the precision of the coefficients is much higher than the spatial precision of the quadtree, no differences can be seen, except when f is on or very near a transition or bifurcation. In the quadratic case $f(z) = z^2 + c$, this happens when c is on the boundary of the Mandelbrot set [11].

8. Applications.

Reliable images. The prime application for the quadtree that our algorithm computes is generating digital images of Julia sets that we can trust, like the ones in §7. A quadtree computed down to level L naturally gives a $2^L \times 2^L$ digital image over Ω simply by painting the leaves of the quadtree with colors corresponding to their labels. However, the resolution of the image need not be the same as the resolution of the quadtree. When the resolution of the image is greater than the resolution of the quadtree, we get "blocky" images, like two of the images shown in Figure 8. When the resolution of the image is smaller than the resolution of the quadtree, we get subpixel information that can be used in two ways: paint the pixel gray if at least one quadtree leaf below the image resolution is gray, or paint the pixel with the average color of the quadtree leaves below it. The first method was used in the images shown in this paper. The second method gives sharper anti-aliased images which are shown in the supplemental material and were used in the panorama mentioned in footnote 9. See Figure 15 for a comparison between the two kind of images. Both kinds of images are reliable in the sense that the white region is contained in $A(\infty)$, the black region is contained in K, and the gray region contains J. For a finer classification, when a pixel has *both* white and black subpixels, we can certify that the pixel contains a part of J.

We can generate a digital image of the Julia set in any rectangular region of interest Q of the complex plane by traversing the quadtree and visiting only the nodes that intersect Q. While this allows limited zoom, we still need to compute the quadtree over the whole of Ω, even if Q is smaller than Ω.

[10] This strongly connected component is by far the largest one; virtually all others have *one* cell.

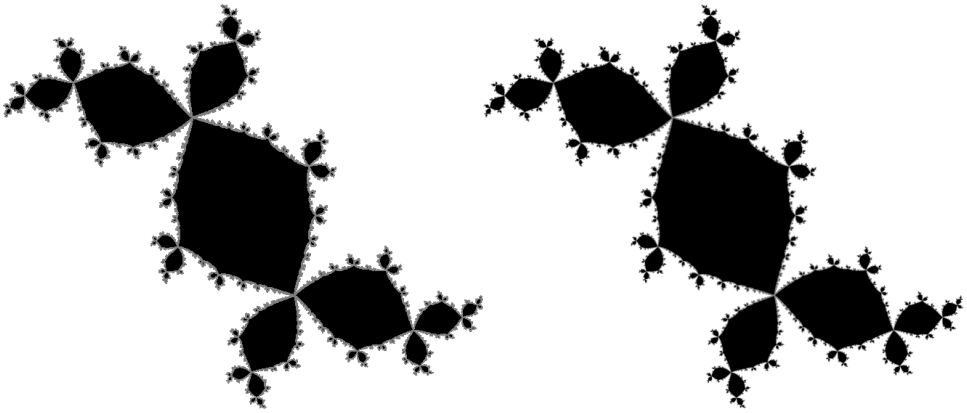

FIGURE 15. Image without anti-aliasing (left) and with anti-aliasing (right). Here, $f(z) = z^2 + c$ with $c = -0.12 + 0.74\,i$.

Point and box classification. A simple quadtree traversal reliably and efficiently classifies a point $z \in \mathbf{C}$ as in $A(\infty)$ or in K, unless it lands in the gray region. If a point z lands in a gray cell near the border of the gray region, one can frequently classify it correctly by locating $f(z)$ with an additional quadtree traversal. This classification by quadtree traversal is easily extended to entire boxes. In particular, by using outward rounding, it reliably classifies tiny boxes around points with floating-point coordinates. Naturally, large boxes will not be classified at all if they straddle the gray region.

Point classification can be used to generate high-quality images using wide filters and sample sharing [30]. In this context, the quadtree serves as a piecewise constant function that can be sampled at low cost at any point in the plane.

Area of filled Julia sets. Following Milnor [26, appendix A], the area of a quadratic filled Julia set K can in principle be computed using an infinite series given by Gronwall's area theorem [15]:

$$\pi(1 - |a_2|^2 - 3|a_4|^2 - 5|a_6|^2 - \cdots)$$

where the coefficients are given recursively by

$$a_2 = -\frac{c}{2},\ a_{2n} = \frac{1}{2}(a_n - a_n^2) - \sum_{\substack{2 \le j < n \\ j \text{ even}}} a_j a_{2n-j},\ a_{2n+1} = 0$$

This series converges slowly, but truncating it gives upper bounds for the area of K.

We can use our quadtree to find both lower and upper estimates for the area of K: The area of the black region gives a lower estimate and the combined area of the black and gray regions gives an upper estimate. Figure 16 shows a graph of these estimates for $-1.25 \le c \le 0.25$ using quadtrees of level 19. The graph also shows the upper bounds computed with 100000 terms of the series, following Milnor. One can barely see the difference between the lower bound and the upper bound computed from the quadtrees, but one can see that our upper bounds are better than the ones obtained from the series, especially for $c < -0.75$.

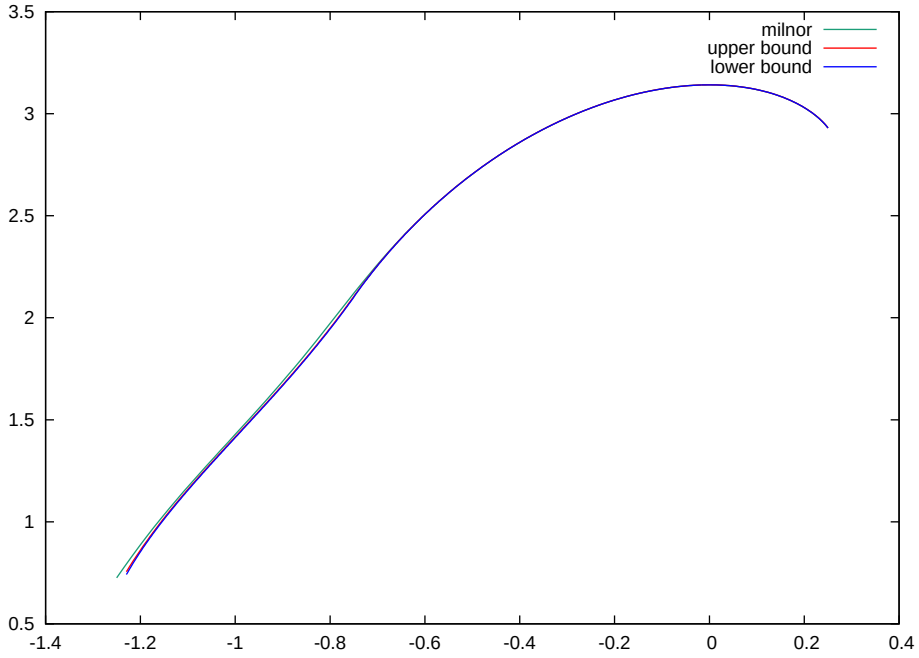

FIGURE 16. Area of the filled Julia set for $-1.25 \leq c \leq 0.25$ computed with our algorithm: lower and upper bounds compared with Milnor's estimate.

9. **Conclusion.** One direction for future work is improving efficiency and this mainly means trying to reduce the size of the cell graph. Reducing the number of vertices is probably not easy. Reducing the number of edges can be done by finding better enclosures for the image of a cell under f and also by implementing better tests for deciding whether $f(A)$ intersects B. There are two techniques for finding better enclosures that would need to be tested and their costs weighted against potential improvements in the cell graph. One is using circular interval arithmetic [14, 35] or affine arithmetic [7, 40]. The other is an algorithm by Rokne [38] which is based on degree elevation of the Bernstein representation of f on the boundary of a rectangular cell and that provides a convex polygon that contains $f(A)$. This technique could reduce the number of edges modestly at the expense of a more complicated intersection test. It would also need careful outward rounding of polygon coordinates to ensure reliability.

Another direction for future work is in applications. From the evolution of the size of the gray region in the quadtree, it is should be simple to extract lower and upper estimates for the box dimension of Julia sets. It would be interesting to compare these estimates in the quadratic case with the upper bound given by Ruelle [39]:

$$\dim_H(J) = 1 + \frac{|c|^2}{4 \log 2} + \text{higher-order terms}$$

and also with the numerical results of Saupe [42].

Still another direction for future work is in extending the algorithm from polynomials to more general functions. In particular, we intend to study how to adapt our

algorithm to approximate Julia sets of rational functions, especially those obtained from Newton's method for solving polynomial equations. The main difference is that ∞ is no longer an attracting fixed point and so there is no exterior. We will probably need to extend the quadtree representation to the whole Riemann sphere [6].

Acknowledgments. Preliminary results of this research were presented at SCAN 2002, at the Dagstuhl-Seminar on Algebraic and Numerical Algorithms and Computer-assisted Proofs in 2005, and, closer to the present form, at the First Palis–Balzan International Symposium on Dynamical Systems in 2012. This research was performed in the Visgraf Computer Graphics laboratory at IMPA. Visgraf is supported by the funding agencies FINEP, CNPq, and FAPERJ, and also by gifts from IBM Brasil, Microsoft, NVIDIA, and other companies. The authors are partially supported by CNPq research grants.

REFERENCES

[1] Z. Arai, On hyperbolic plateaus of the Hénon map, *Experimental Mathematics*, **16** (2007), 181–188.
[2] P. Blanchard, Complex analytic dynamics on the Riemann sphere, *Bulletin of the American Mathematical Society*, **11** (1984), 85–141.
[3] B. Branner, The Mandelbrot set, in *Amer. Math. Soc.*, **39** (1989), 75–105.
[4] M. Braverman, Hyperbolic Julia sets are poly-time computable, *Electronic Notes in Theoretical Computer Science*, **120** (2005), 17–30.
[5] M. Braverman and M. Yampolsky, *Computability of Julia Sets*, Springer-Verlag, 2009.
[6] R. Carniel, A quasi-cell mapping approach to the global dynamical analysis of Newton's root-finding algorithm, *Applied Numerical Mathematics*, **15** (1994), 133–152.
[7] L. H. de Figueiredo and J. Stolfi, Affine arithmetic: concepts and applications, *Numerical Algorithms*, **37** (2004), 147–158.
[8] M. Dellnitz and A. Hohmann, A subdivision algorithm for the computation of unstable manifolds and global attractors, *Numerische Mathematik*, **75** (1997), 293–317.
[9] M. Dellnitz and O. Junge, Set oriented numerical methods for dynamical systems, in *Handbook of dynamical systems*, North-Holland, **2** (2002), 221–264.
[10] R. L. Devaney and L. Keen (eds.), *Chaos and Fractals: The Mathematics behind the Computer Graphics*, Proceedings of Symposia in Applied Mathematics 39, AMS, 1989.
[11] A. Douady, Does a Julia set depend continuously on the polynomial?, in *Complex Dynamical Systems: The Mathematics Behind the Mandelbrot and Julia Sets* (ed. R. L. Devaney), Proceedings of Symposia in Applied Mathematics, AMS, **49** (1994), 91–138.
[12] M. B. Durkin, The accuracy of computer algorithms in dynamical systems, *International Journal of Bifurcation and Chaos in Applied Sciences and Engineering*, **1** (1991), 625–639.
[13] Z. Galias, Rigorous investigation of the Ikeda map by means of interval arithmetic, *Nonlinearity*, **15** (2002), 1759–1779.
[14] E. Grassmann and J. Rokne, The range of values of a circular complex polynomial over a circular complex interval, *Computing*, **23** (1979), 139–169.
[15] T. H. Gronwall, Some remarks on conformal representation, *Annals of Mathematics*, **16** (1914/15), 72–76.
[16] S. L. Hruska, Constructing an expanding metric for dynamical systems in one complex variable, *Nonlinearity*, **18** (2005), 81–100.
[17] S. L. Hruska, Rigorous numerical studies of the dynamics of polynomial skew products of $\mathbf{C}^2$, in *Complex dynamics*, vol. 396 of Contemp. Math., AMS, 2006, 85–100.
[18] C. S. Hsu, *Cell-to-cell Mapping: A Method of Global Analysis for Nonlinear Systems*, Springer-Verlag, 1987.
[19] C. S. Hsu, Global analysis by cell mapping, *International Journal of Bifurcations and Chaos*, **2** (1992), 727–771.
[20] O. Junge, Rigorous discretization of subdivision techniques, in *International Conference on Differential Equations, Vol. 1, 2 (Berlin, 1999)*, World Sci. Publ., 2000, 916–918.
[21] W. D. Kalies, K. Mischaikow and R. C. A. M. VanderVorst, An algorithmic approach to chain recurrence, *Foundations of Computational Mathematics*, **5** (2005), 409–449.

[22] L. Keen, Julia sets, in *Proc. Sympos. Appl. Math.*, **39** (1989), 57–74.
[23] V. Kreinovich, Interval software, `http://cs.utep.edu/interval-comp/intsoft.html`.
[24] D. Michelucci and S. Foufou, Interval-based tracing of strange attractors, *International Journal of Computational Geometry & Applications*, **16** (2006), 27–39.
[25] J. Milnor, Remarks on iterated cubic maps, *Experimental Mathematics*, **1** (1992), 5–24.
[26] J. Milnor, *Dynamics in One Complex Variable*, vol. 160 of Annals of Mathematics Studies, 3rd edition, Princeton University Press, 2006.
[27] R. E. Moore, *Interval Analysis*, Prentice-Hall, 1966.
[28] R. E. Moore, R. B. Kearfott and M. J. Cloud, *Introduction to Interval Analysis*, SIAM, 2009.
[29] R. P. Munafo, Roundoff error, `http://mrob.com/pub/muency/roundofferror.html`, 1996, Accessed: 2015-12-09.
[30] D. Nehab and H. Hoppe, A fresh look at generalized sampling, *Foundations and Trends in Computer Graphics and Vision*, **8** (2014), 1–84.
[31] G. Osipenko, *Dynamical Systems, Graphs, and Algorithms*, vol. 1889 of Lecture Notes in Mathematics, Springer-Verlag, 2007.
[32] A. Paiva, L. H. de Figueiredo and J. Stolfi, Robust visualization of strange attractors using affine arithmetic, *Computers & Graphics*, **30** (2006), 1020–1026.
[33] H.-O. Peitgen and P. H. Richter, *The Beauty of Fractals: Images of Complex Dynamical Systems*, Springer-Verlag, 1986.
[34] H.-O. Peitgen and D. Saupe (eds.), *The Science of Fractal Images*, Springer-Verlag, 1988.
[35] M. S. Petković and L. D. Petković, *Complex Interval Arithmetic and Its Applications*, Wiley-VCH Verlag, 1998.
[36] R. Rettinger, A fast algorithm for Julia sets of hyperbolic rational functions, *Electronic Notes in Theoretical Computer Science*, **120** (2005), 145–157.
[37] R. Rettinger and K. Weihrauch, The computational complexity of some Julia sets, in *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC 2003)*, ACM, 2003, 177–185.
[38] J. Rokne, The range of values of a complex polynomial over a complex interval, *Computing*, **22** (1979), 153–169.
[39] D. Ruelle, Repellers for real analytic maps, *Ergodic Theory and Dynamical Systems*, **2** (1982), 99–107.
[40] S. M. Rump and M. Kashiwagi, Implementation and improvements of affine arithmetic, *Nonlinear Theory and Its Applications, IEICE*, **6** (2015), 341–359.
[41] H. Samet, The quadtree and related hierarchical data structures, *Computing Surveys*, **16** (1984), 187–260.
[42] D. Saupe, Efficient computation of Julia sets and their fractal dimension, *Physica D*, **28** (1987), 358–370.
[43] N. Steinmetz, *Rational Iteration: Complex Analytic Dynamical Systems*, Walter de Gruyter & Co., 1993.
[44] C. M. Stroh, *Julia Sets of Complex Polynomials and Their Implementation on the Computer*, Master's thesis, University of Linz, 1997.
[45] R. Tarjan, Depth-first search and linear graph algorithms, *SIAM Journal on Computing*, **1** (1972), 146–160.
[46] W. Tucker, The Lorenz attractor exists, *C. R. Acad. Sci. Paris Sér. I Math.*, **328** (1999), 1197–1202.
[47] J. Tupper, Reliable two-dimensional graphing methods for mathematical formulae with two free variables, in *Proceedings of SIGGRAPH '01*, ACM, 2001, 77–86.

Received March 2016; revised August 2016.

E-mail address: `lhf@impa.br`
E-mail address: `diego@impa.br`
E-mail address: `stolfi@ic.unicamp.br`
E-mail address: `oliveira@inf.pucrs.br`