NUMA manuscript No.
(will be inserted by the editor)

Images of Julia sets that you can trust

Luiz Henrique de Figueiredo · **Diego Nehab** ·
Jorge Stolfi · **João Batista Oliveira**

Received: date / Accepted: date / Updated: January 25, 2013 at 10:50am

Abstract We present an algorithm for computing images of quadratic Julia sets that can be trusted in the sense that they contain numerical guarantees against sampling artifacts and rounding errors in floating-point arithmetic. We use cell mapping and color propagation in graphs to avoid function iteration and rounding errors. As a result, our algorithm avoids point sampling and can robustly classify entire rectangles in the complex plane as being on either side of the Julia set. The union of the regions 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.

Keywords Julia sets · adaptive refinement · cell mapping · computer-assisted proofs

Mathematics Subject Classification (2000) 37-04 · 37F50 · 37F10 · 37M99

1 Introduction

We all have seen many images of Julia sets [18]. Can these beautiful images really be trusted? A Julia set is the boundary between markedly different behaviors of the iteration of a nonlinear function on the complex plane. Julia sets are typically fractal, and near them rounding errors in the computation of the function may decide the iteration in favor of the wrong domain. Moreover, the number of iterations required to classify a point depends on the point and cannot be reliably fixed a priori. Therefore,

Luiz Henrique de Figueiredo* and Diego Nehab
IMPA–Instituto Nacional de Matemática Pura e Aplicada, Rio de Janeiro, Brazil
*Corresponding author: email: lhf@impa.br, voice: +55 21 2529-5142, fax: +55 21 2529-5129

Jorge Stolfi
Instituto de Computação, UNICAMP, Campinas, Brazil

João Batista Oliveira
Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brazil

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 systematically produced pictures that carry numerical guarantees.

In this paper, we describe how to compute images of Julia sets that can be trusted. Our algorithm computes a refinable quadtree decomposition 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 rectangular region 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 Fig. 1).

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

- *No point sampling.* The algorithm uses cell mapping [9,10] to reliably classify entire rectangles in the complex plane, not just a finite sample of points.
- *No orbits.* The algorithm does not need to fix an arbitrary limit for the number of iterations performed, that is, for the length of partial orbits computed. In fact, the algorithm performs no function iteration at all. Instead, it handles orbits by using color propagation in graphs induced by cell mapping.
- *No rounding errors.* The numbers processed by the algorithm are dyadic fractions that are restricted in range and precision and the algorithm uses error-free fixed-point arithmetic whose precision depends only on the spatial resolution of the image. Standard double-precision floating-point arithmetic is enough to generate huge guaranteed images up to $10^6 \times 10^6$ pixels over the square $[-2,2] \times [-2,2]$.

We start by briefly recalling the main definitions and properties of Julia sets in §2. We then discuss the limitations of the main known algorithms for generating images of Julia sets in §3. We present our algorithm in §4, discuss its behavior and its limitations in §5, present some applications in §6 and possible extensions in §7.

2 Julia sets

Consider the quadratic function $f: \mathbf{C} \to \mathbf{C}$ given by $f(z) = z^2 + c$, where c is a fixed complex number. (Most of what follows applies to all polynomials of degree at least 2, but we shall concentrate on the classic quadratic case.) Julia sets arise naturally when we study the *dynamics* of f, that is, the behavior of the discrete dynamical system induced by the iterates of f: $f^1 = f$, $f^2 = f \circ f$, $f^3 = f \circ f \circ f$, … To study the long-term behavior of this 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 orbit of a point z_0 is either bounded or unbounded. When it is unbounded, the orbit actually *escapes* away to infinity, in the sense that $|f^n(z_0)| \to \infty$ as $n \to \infty$. Here is a well-known elementary quantitative proof of this fact, which plays an important role in algorithms for drawing Julia sets:

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

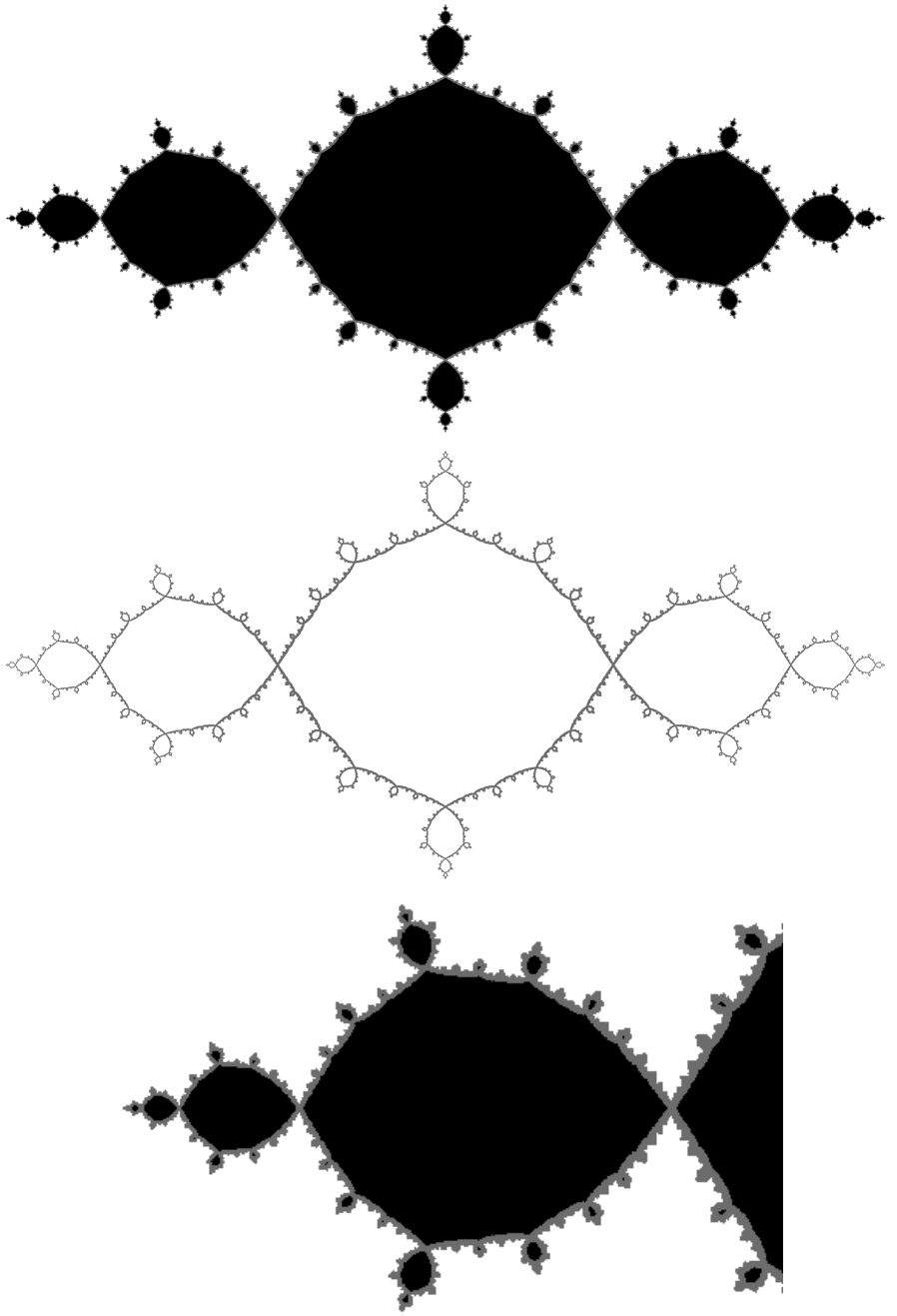

Fig. 1 Image of the Julia set for $c = -1$ computed with our algorithm, cropped from a 4096×4096 image. Top: filled Julia set. Middle: Julia set. Bottom: zoom shows the gray region containing the Julia set. The white region is guaranteed to be outside the filled Julia set and the black region is guaranteed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

Lemma 1 (escape radius) *Let $f(z) = z^2 + c$, where $c \in \mathbf{C}$, and let $R = \max(|c|, 2)$. If $z \in \mathbf{C}$ and $|z| > R$, then $|f^n(z)| \to \infty$ as $n \to \infty$.*

Proof The triangle inequality gives $|z^2| = |z^2 + c - c| \le |z^2 + c| + |c|$, and so $|f(z)| = |z^2 + c| \ge |z^2| - |c| = |z|^2 - |c| > |z|^2 - |z| = |z|(|z| - 1) > |z| > R$. Iterating this result, we get $|f^n(z)| > |z|(|z| - 1)^n \to \infty$ because $|z| - 1 > 1$. □

Therefore, if the orbit of z_0 is unbounded, then it must go beyond the *escape radius* R, in the sense that $|f^m(z_0)| > R$ for some $m \in \mathbf{N}$, and once it does that the lemma shows that the orbit of z_0 escapes to ∞. In this case, we say that z_0 is in the *attraction basin* of ∞, which is denoted by $A(\infty)$.[2] The complement $K = \mathbf{C} \setminus A(\infty)$ is thus the set of points having bounded orbits and is called the *filled Julia set* of f. The *Julia set* J of f is the common boundary of $A(\infty)$ and K.

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 elusive to draw, and so pictures typically show the filled Julia set K when possible (that is, when K has an interior). As Devaney says [19, §3.3.3], images of filled Julia sets are somehow more appealing anyway.

The classic work of Julia and Fatou in 1918–1919 highlighted the key role of the orbits of critical points and established an important topological dichotomy: the Julia set is either connected or a Cantor set. This dichotomy defines the famous Mandelbrot set, which represents in parameter space the set of $c \in \mathbf{C}$ such that J is connected. According to Julia and Fatou, 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 [1], Keen [11], Branner [2], and Milnor [14].

3 Computing images of Julia sets

Although f is a simple quadratic function, its dynamics 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 [4].) Accordingly, Julia sets can be very complicated sets and many pictures have been made to try to convey their complexity [18].

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 estimator method [18, 19, 24]. 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 come with no guarantees and cannot really be trusted. 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* of radius R centered at the origin. For each pixel in the image, the escape-time method computes the orbit of the center of the pixel until it exits the escape circle or a

[2] 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 attractive fixed point of f.

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 cannot be trusted because they rely on point sampling, compute partial orbits, 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 tacitly assuming that the behavior between sample points is represented reliably by the behavior of the sample points nearby. While both $A(\infty)$ and $K \setminus J$ are open, this assumption is not warranted near the Julia set J. To mitigate this problem, one typically uses a finer grid to get more detail and 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 the 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 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 there are no such guarantees.
- *Rounding errors.* To represent precisely the value of a quadratic function at a point in a computer one needs twice the number of bits used to represent the point. Thus, iterating a quadratic function in fixed-precision floating-point arithmetic rapidly loses precision 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, thereafter producing false points inside K. Some programs use multiple-precision floating-point arithmetic to avoid these problems, especially in deep zoom, 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 studied.

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. As mentioned in §1, our algorithm does not suffer from these limitations.

Previous work related to these issues has focused mainly on the computability of Julia sets and on the computational complexity of approximating Julia sets [21,20, 3]. A good reference is the book by Braverman and Yampolsky [4]. This is important theoretical work that probes the limits of what in principle can be computed about Julia sets. In particular, they imply that we can expect that hyperbolic Julia sets[3] are easy to draw but parabolic ones can be hard. However, as far as we know, the algorithms given in these papers have not been implemented in practice or have not been widely used. On the other hand, our algorithm does not promise to compute a 1-pixel approximation of the Julia set, a hopeless task according to the theoretical work just mentioned.

4 Our algorithm

Our algorithm computes a decomposition of the complex plane into three 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 $R = \max(|c|, 2)$ centered at the origin, we can concentrate our attention on the square region $\Omega = [-R,R] \times [-R,R]$, because $\mathbf{C} \setminus \Omega$ is contained in $A(\infty)$. However, our algorithm works on any rectangle Ω that contains the escape circle. For $|c| \leq 2$, we get $R = 2$ and $\Omega = [-2,2] \times [-2,2]$, which is very convenient.

Our algorithm uses an adaptive refinement method that computes a quadtree decomposition [23] of Ω into rectangular cells and assigns the appropriate color to each leaf cell in this tree (see Fig. 2). The steps performed by the algorithm are explained in detail below. Besides adaptive refinement (step 1), its main features are the use of *cell mapping* (step 2) and *color propagation* in graphs (steps 3 and 4).

0. [*init*] Start with a single-node quadtree containing a gray cell over Ω. Create a separate special white cell representing the complement of Ω, called the *exterior*.
1. [*refine*] Subdivide each gray leaf cell into four new gray subcells.
2. [*build cell graph*] For each gray leaf cell A, find a set of leaf cells such that $f(A)$ is contained in the union of these cells and the exterior. For each such cell B, add an edge $A \to B$ to the graph.
3. [*find new white cells*] Color white each gray leaf cell such that *all* paths from it reach white cells.
4. [*find new black cells*] Color black each gray leaf cell such that *no* path from it reaches a white cell.
5. [*prune*] Prune the quadtree from the bottom up, converting internal nodes whose children are all white or all black to a leaf of that color.
6. [*repeat*] If the desired quadtree resolution has not been reached, go to step 1.

When the algorithm ends, the resulting quadtree can be stored for future analysis or processed as desired. Typically, we generate a digital image from it by painting the leaves with their color. We shall discuss this and other applications of the quadtree in §6. We now give the details of the algorithm.

[3] A Julia set is *hyperbolic* when the critical point 0 converges to an attracting periodic orbit or to ∞ (which is an attracting fixed point).

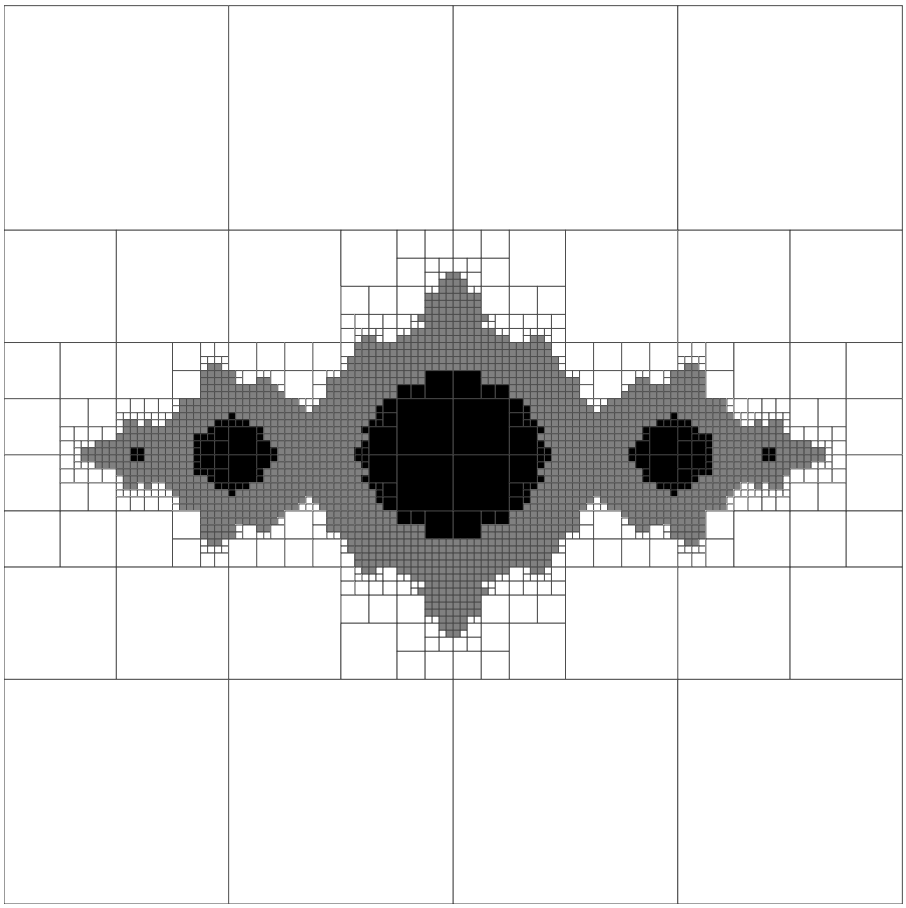

Fig. 2 Quadtree of refinement level 7 computed by our algorithm for $c = -1$ and $\Omega = [-2,2] \times [-2,2]$. The white region is contained in $A(\infty)$, the black region is contained in K, and the gray region contains J. The refinement process to level 14 is shown in Fig. 10.

Step 0 [*init*]. The single gray cell over Ω reflects our initial knowledge that J is completely contained in Ω. The white region is represented at this moment by the exterior cell and reflects our initial knowledge that the exterior is contained in $A(\infty)$. (Hence the requirement that Ω contains the escape circle.) No part of the interior of K is known at this moment and so the black region is empty. Together, these three facts establish the initial fundamental invariant of the algorithm: the white region is contained in $A(\infty)$, the black region is contained in K, and the gray region contains J.

Step 1 [*refine*]. This is the standard quadtree refinement step. Each gray leaf cell is subdivided into four equal child cells by bisecting its sides. The child cells are colored gray. This is an *adaptive* refinement because only gray cells are refined. White cells and black cells are not refined because their final color is known.

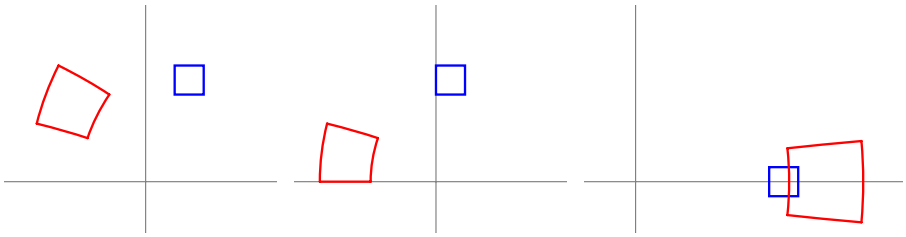

Fig. 3 The bounding box for the image of a quadtree cell is defined by the image of its vertices (left), even when the cell touches an axis (middle), but not when the cell straddles an axis (right). Here we took $c = 0$.

The aim of refining is to classify a subset of the new gray cells (typically those near the boundary of the gray region) as white or black, thus improving the approximation.

Step 2 [*build cell graph*]. In this step, we build a graph that represents the *cell mapping* [9,10,5] induced by f on $\mathbf{C}$, as decomposed into the leaf cells plus the exterior.

In cell mapping, one represents the dynamics of a discrete dynamical system induced by a map f using a directed *cell graph* whose vertices are cells that decompose the domain of f and whose edges $A \to B$ go from a cell A to a cell B whenever $f(A)$ intersects B. Thus, $f(A)$ is contained in the union U of all cells B that are the target of an edge $A \to B$ in the cell graph. Because $f(A)$ is usually properly contained in U, the cell graph is a conservative approximation for the dynamics of f. Nevertheless, cell mapping is a powerful tool. In particular, graph traversals replace function iteration with advantage because they can track orbits for whole sets of starting points. They do so very conservatively but robustly nevertheless.

The main difficulty in cell mapping is finding the edges in the cell graph, that is, deciding which cells $f(A)$ intersects. 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, and the simplest solution for the edge problem is to use point sampling: for each cell A and for each point p in a finite set of samples chosen in A (perhaps sampling A more finely on its boundary [5]), we identify the cells B that contain $f(p)$ and add an edge $A \to B$ to the cell graph. While this solution is simple to implement and gives good results in many cases, it is not guaranteed to find the complete cell graph: we may miss an edge $A \to B$ simply because no sample point in A is mapped into B. Missing edges is fatal to an algorithm that aims to provide robust computational proofs such as the guaranteed images we seek.

Under the quadratic function $f(z) = z^2 + c$, a rectangular cell A is mapped to a curvilinear quadrilateral $f(A)$ whose boundary is formed by parabolic arcs and possibly one line segment (see Fig. 3), and an exact intersection test between $f(A)$ and rectangles B can be devised in principle. Nevertheless, because this test is tedious to implement, we have chosen a more conservative but much simpler test that uses a bounding box T for $f(A)$, as explained below. Although using bounding boxes for finding edges in the cell graph makes cell mapping even more conservative than it already is by nature, we have found that it is simple, efficient, and quite effective.

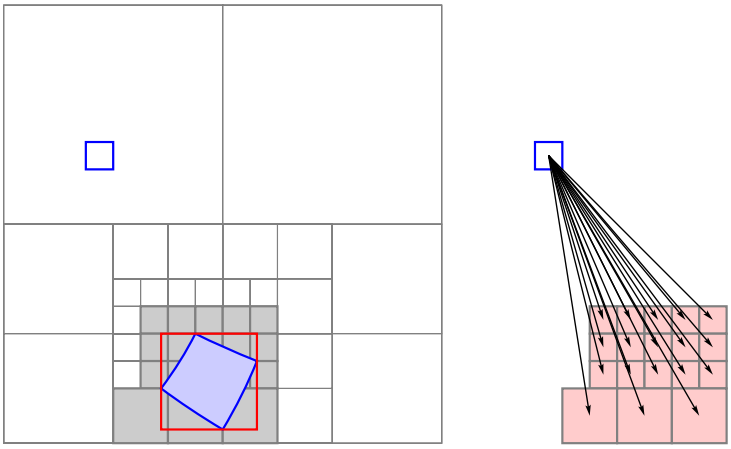

Fig. 4 Image of the cell $[-1.25,-1]\times[0.5,0.75]$ for $c=-1$: actual image (blue), bounding box (red), quadtree traversal for locating the bounding box, and the edges in the cell graph starting at the original cell.

For cells A in a quadtree decomposition of $\Omega=[-R,R]\times[-R,R]$, a tight bounding box T for $f(A)$ is easy to compute because it coincides with the bounding box of the images of the vertices of A under f. Indeed, writing $c=a+b\,i$, $z=x+y\,i$, and $f(z)=(x^2-y^2+a)+(2xy+b)\,i=u+v\,i$, we see that f sends horizontal lines and vertical lines to parabolas of the form $u=g(v)$ having vertex in $v=b$, except that f sends the coordinate axes to the line $v=b$. Therefore, the boundary of $f(A)$ is formed by parabolic arcs that are monotonic in both the u and the v directions, except around the vertex, when $v=b$, that is, when $xy=0$. Since no quadtree cell A straddles a coordinate axis, the parabolic arcs that define $f(A)$ are always monotonic. When the boundary of A touches a coordinate axis, $f(A)$ has a horizontal segment that does not affect the bounding box (see Fig. 3, middle). When Ω is not centered at the origin, some quadtree cells A will straddle a coordinate axis and the bounding box T has to take into account the image of the points where the boundary of A crosses that axis. In that case, we need to compute the image of six points in A, not just four, to find T.

Having found a bounding box T for $f(A)$, we add edges $A\to B$ for each leaf cell B that intersects T, including the exterior cell. The union of all such cells B contains T and so a fortiori contains $f(A)$. We traverse the quadtree to locate the leaves that intersect T, thus avoiding testing T against all leaves in the quadtree (see Fig. 4). The simple geometry of T simplifies both the traversal and the intersection test.

After this step, we have a directed graph whose vertices are the leaves plus the exterior and whose edges emanate from all the gray leaves. This graph is needed only for the color propagation done in Steps 3 and 4; a new graph is built after every refinement step because it is based on a different set of leaves.

Step 3 [*find new white cells*]. In this step, we identify gray leaf cells that are *guaranteed* to be white based on the following crucial observation: if *all* paths in the cell graph starting at a gray cell A eventually reach white cells, then the orbits of *all* points in A reach $A(\infty)$. By definition, A must then be inside $A(\infty)$ and we can color A white.

We have used two different methods to find new white cells: *front propagation* and *reverse sweep*. Although their results are the same, each method has its advantages and disadvantages, as discussed below:

- *Front propagation.* This method repeatedly loops over all gray cells in the cell graph. Throughout each loop, every gray cell A for which $f(A)$ is contained in $A(\infty)$ is promptly transformed into a new white cell. These are the cells whose edges all point to white cells. Since each new white cell increases the region currently assigned to $A(\infty)$, the process is repeated until an entire loop completes during which no new white cells are found. Thus, at iteration k this method finds the gray leaf cells from which the longest path leading to white cells has length k.
- *Reverse sweep.* The same goal can be achieved with a single traversal of the *reverse cell graph*, starting from each gray cell. For each visited cell A, we check whether $f(A)$ is contained in $A(\infty)$. (This test uses the original graph.) If so, A is marked white and the traversal proceeds recursively, visiting all cells B for which $f(B) \cap A \neq \varnothing$. (This test uses the reverse graph.) Note that a gray cell A that will eventually be marked white may not be so marked the first time it is inspected, because A may point to another gray cell B that has not yet been marked white either. However, as soon as the last such cell B has been marked white, A will be visited again and at that moment it will be finally marked white as well. By assumption, all paths emanating from A eventually terminate in white cells, and so A cannot be part of a cycle.

The front propagation method is easier to understand and implement, but it can be inefficient because it can go repeatedly over all gray leaf cells in the cell graph. The reverse sweep methods is efficient (runs in linear time), but it needs almost twice the amount of memory, since it needs to build and maintain the reverse cell graph.

Regardless of the method used, after this step we have propagated white to some gray cells, increasing the white region and reducing the external part of the gray region, thus finding a better approximation for $A(\infty)$ and a better enclosure for J.

Step 4 [*find new black cells*]. In this step, we identify gray leaf cells that can be *proven* to be black based on the following crucial observation: if *no* paths in the cell graph starting at a gray cell A ever reach a white cell, then no orbit starting in A ever reaches $A(\infty)$. By definition, A must then be inside K because all orbits starting in A are bounded. We can therefore color A black.

To find new black cells, we mark all gray leaf cells that *do* have paths to white cells. Once all these cells have been marked, the new black cells are simply those gray cells that have *not* been marked. To mark the gray leaf cells that have paths to white cells, we use two methods very similar to those used in Step 3:

- *Front propagation.* Repeatedly loop over unmarked gray cells in the cell graph. In each loop, mark all gray cells A for which $f(A) \cap A(\infty) \neq \varnothing$. When a loop completes without a single new cell being marked, the propagation stops.
- *Reverse sweep.* Traverse the reverse cell graph starting from each gray cell A for which $f(A) \cap A(\infty) \neq \varnothing$. For each visited cell, mark and recursively traverse all cells B for which $f(B) \cap A \neq \varnothing$.

Again, the front propagation method is simpler but can be inefficient; the reverse sweep method is efficient but needs twice the memory. Of course, if reverse sweep is used in Step 3, then the reverse graph does not need to be rebuilt in Step 4.

Regardless of the method used, after this step we have propagated black to some gray cells or even found some black cells for the first time, increasing the black region and reducing the internal part of the gray region, thus finding a better approximation for K and a better enclosure for J. As mentioned in §5, once the algorithm has found some black cells, it finds most of them quickly in the next iterations. This step may be skipped when K is known to have empty interior; for instance, when $|c| > 2$.

Step 5 [*prune*]. In this step, we consolidate the new color information found in Steps 3 and 4. Whenever all four child nodes of a cell have the same color, their parent node becomes a leaf painted with that color and the child nodes are removed from the quadtree. This process is repeated recursively upwards in the quadtree.

This step guarantees that the quadtree is as shallow as possible or, equivalently, that it has leaves that are as large as possible, within the geometric constraints of the quadtree decomposition. Although this step is not strictly necessary, it significantly improves the traversals in Step 2 and it simplifies the traversals used for image generation and other geometric applications discussed in §6.

5 Discussion

Fig. 5 shows some of Julia sets computed with our algorithm.[4] The corresponding refinement processes to level 14 are shown in Figs. 10–15.

Note how the algorithms works for different types of Julia sets: connected sets with non-empty interior (Figs. 10–13), connected sets with empty interior (Fig. 14), and totally disconnected Cantor sets (Fig. 15). In all cases, the approximation of the exterior improves steadily after each refinement. For the 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. The speed of convergence naturally depends on the value of 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 outside but near the boundary of the Mandelbrot set, the algorithm may need to perform several refinements steps before the Julia set emerges as disconnected (see Fig. 15). Once it does, we have a computational proof that c is not in the Mandelbrot set. We cannot hope to prove with our algorithm that a given c is on the boundary of the Mandelbrot set because an interior region will never appear and the approximation will remain connected (see Fig. 14).

In agreement with the theoretical results mentioned in §3, we have found it easier to approximate the Julia set for c in the interior of the Mandelbrot set, but harder to approximate the Julia set for c near the boundary of the Mandelbrot set, especially

[4] For full resolution pictures, see http://www.impa.br/~lhf/julia. We have computed several thousand pictures of Julia sets. They are available as an interactive panorama of the Mandelbrot set at http://monge.visgraf.impa.br/panorama/viewer/index.html?img=../julia-256GP/julia.xml.

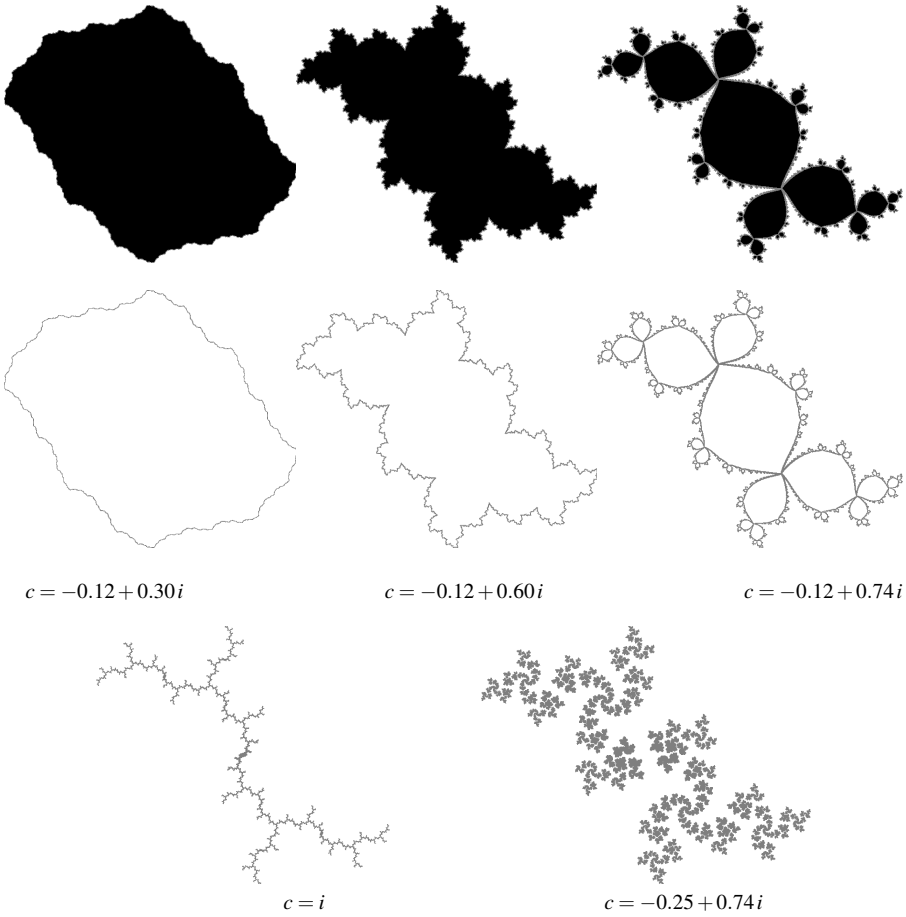

Fig. 5 Examples of Julia sets and filled Julia sets computed with our algorithm.

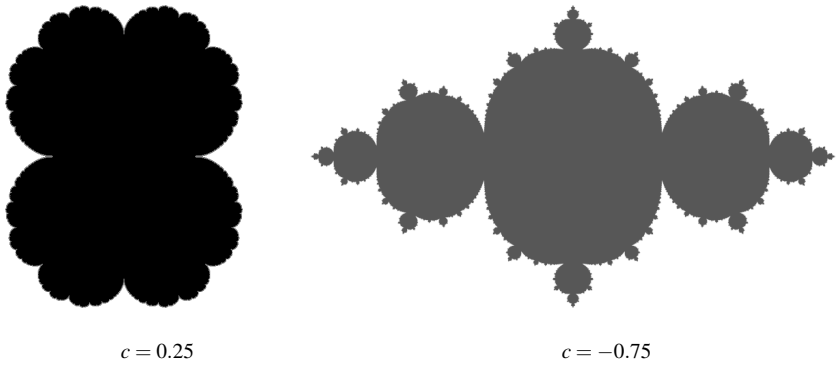

Fig. 6 Parabolic Julia sets.

near bifurcation points. For instance, our algorithm finds a very good approximation for the interior of K when $c = 1/4$ (a well-known bifurcation point) but it did not find any part of the interior of K when $c = -3/4$ (another well-known bifurcation point). The difference is that for $c = 1/4$ the algorithm successfully finds quadtree cells containing the parabolic point which are invariant under f, and so can paint them black; on the other hand, no such cells exist for $c = -3/4$ and no interior appears and all interior cells remain gray forever (see Fig. 6).

Limitations. Although our algorithm produces guaranteed images of Julia sets, it has some limitations:

- The resolution of the quadtree and the size of cell graph are limited by the available memory. In current machines, one cannot go beyond 20 levels, 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).
- Our algorithm needs to compute a quadtree for $\Omega = [-R,R] \times [-R,R]$ 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.

About floating-point computations. Our algorithm works with standard floating-point arithmetic but is not subject to errors due to rounding or truncation. The numerical part of the algorithm is concentrated in Step 2, where we compute a bounding box for the image of a quadtree cell under f. For simplicity, we only discuss the case $|c| \le 2$, when $R = 2$ and $\Omega = [-2,2] \times [-2,2]$. (In the general case, replace R with a power of 2 greater than it and adapt the discussion below.)

Recall that we need to compute $x^2 - y^2 + a$ and $2xy + b$ for $x + y\,i$ a vertex of a quadtree cell and $c = a + b\,i$. At level L, the coordinates of the vertices of all quadtree cells are dyadic fractions of the form $t/2^{L-2}$, where t is an integer with $|t/2^{L-2}| \le 2$ or $|t| \le 2^{L-1}$. This means that we need L bits to represent these coordinates: 2 bits for the integer part and $L-2$ bits for the fractional part.

Now, $x, y, a, b \in [-2,2]$ implies $x^2, y^2 \in [0,4]$, $x^2 - y^2, xy \in [-4,4]$, and so $x^2 - y^2 + a \in [-6,6]$, and $2xy + b \in [-10,10]$. This means that the integer part of the results needs 4 bits. For representing the fractional part of the results, note that squares and products need twice as many bits as their inputs and each addition needs one more bit to account for a potential carry. ($x^2 - y^2$ does not need an additional bit.) Thus, if the fractional part of a and b is represented with F bits, then the fractional part of the results can be represented with $N = \max(2(L-2), F) + 1$ bits.

Therefore, the entire computation can be done exactly in fixed-point arithmetic having 4 bits for the integer part and N bits for the fractional part. We do not really need fixed-point arithmetic because if the computer represents floating-points numbers with M mantissa bits, then we can compute these expressions with no rounding errors as long as $N + 4 \le M$. In standard double-precision floating-point arithmetic, we have $M = 53$ and so error-free computation is guaranteed for $L \le 26$ and $K \le 48$.

We represent each of a and b by either a single floating-point number or an interval of floating-point numbers having width 2^{-48}. For instance, $0.12 \in [0.12^-, 0.12^+]$ and

$0.30 \in [0.30^-, 0.30^+]$, where

$$0.12^- = 0.11999999999999744204615126363933086395263671875 0$$
$$0.12^+ = 0.12000000000000099475983006414026021957397460937 5$$
$$0.30^- = 0.29999999999999715782905695959925651550292968750 0$$
$$0.30^+ = 0.30000000000000071054273576010018587112426757812 5$$

This allows c to be represented with high precision, but has two consequences: First, the bounding box computed in Step 2 has to be slightly enlarged to handle the representation of a and b as tiny intervals. Second, this implies that the Julia sets we show are actually the combination of all Julia sets for c in a tiny rectangle: 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 c is much higher than the spatial precision of the quadtree, no differences can be seen, unless c is very near the boundary of the Mandelbrot set [7].

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

Take $z \in A(\infty)$. Let $U_0 = \mathbf{C} \setminus \Omega$. Then U_0 is an open set contained in $A(\infty)$. If $z \in U_0$, then the algorithm establishes that $z \in A(\infty)$ with no effort since $|z| > R$. Otherwise, let $U_1 = f^{-1}(U_0)$. Then U_1 is an open set, because U_0 is open and f is continuous. If $z \in U_1$, then $f(z) \in U_0$ and there is a small box B_0 around $f(z)$ totally contained in U_0. Since U_0 is open and f is continuous, there is a small box B_1 around z totally contained in U_1 such that $f(B_1) \subseteq B_0$. When the algorithm reaches a spatial resolution below the sizes of B_1 and B_0, it will discover that $f(B_1) \subset U_0$, which proves that $z \in A(\infty)$. In the general case, let $U_n = f^{-1}(U_{n-1})$. Then $z \in U_n$ for some n since by hypothesis $z \in A(\infty)$. By induction, the algorithm certifies that every point in U_{n-1} is in $A(\infty)$. In particular, there is a small box B_{n-1} around $f(z)$ totally contained in U_{n-1}. Since U_{n-1} is open and f is continuous, there is a small box B_n around z totally contained in U_n such that $f(B_n) \subseteq B_{n-1}$. When the algorithm reaches a spatial resolution below the sizes of B_n and B_{n-1}, it will prove that $z \in A(\infty)$.

The case $z \in K$ is harder. If K has no interior, then $J = K = \mathbf{C} \setminus A(\infty)$, and so the algorithm correctly classifies $z \in K$ by failing to prove that $z \in A(\infty)$. When K has an interior, we need an analog of U_0 to bootstrap the classification of points in the interior of K. For this, we assume that there is an attracting period orbit in K. Then there is a set of small open discs around the points in this orbit that is invariant under f. Using this set follows a continuity argument quite similar to the one given above. This argument fails when J is parabolic for instance and we may end up with a gray K (see Fig. 6).

6 Applications

Images of Julia sets that you can trust. The prime application for the quadtree that our algorithm computes is generating digital images of Julia sets that you can trust, like the

ones in Fig. 5. A quadtree computed down to level L naturally gives a $2^L \times 2^L$ digital image over $\Omega = [-R,R] \times [-R,R]$ simply by painting the leaves of the quadtree with their color. 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 some of the images shown in Figs. 10–15. 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 and was used in the panorama mentioned in footnote 4. Both kinds of images can trusted 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 using Q as a filter, that is, by visiting only the nodes that intersect Q. However, even if Q is smaller than Ω, we still need to compute the quadtree over the whole of Ω.

Point and box classification. A simple quadtree traversal can robustly classify 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. Classification by quadtree traversal is easily extended to entire boxes, though naturally large boxes will not be classified at all if they straddle the gray region.

Area of filled Julia sets. Following Milnor [14, App. A], the area of the filled Julia set K can in principle be computed using Gronwall's theorem [8] applied to the inverse Böttcher map ψ, which satisfies

$$\psi(w^2) = \psi(w)^2 + c$$

Because of this conjugation, the Laurent series of ψ near ∞ has the form

$$\psi(w) = w\left(1 + \frac{a_2}{w^2} + \frac{a_4}{w^4} + \frac{a_6}{w^6} + \cdots\right)$$

where the coefficients are given recursively by

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

Gronwall's area theorem says that the area of K is then

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

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

When K has an interior and so has non-zero area, 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. Fig. 7 (top) 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 bonds are better than the ones obtained from the series, especially for $c < -0.75$, that is, in the period-2 circular bulb. Fig. 7 (bottom) shows the absolute error bounds between these estimates. They are low in the middle of the main cardioid, reach a local peak at the bifurcation points $c = 0.25$, $c = -0.75$, and $c = -1.25$, and are again locally low in the middle of the period-2 circular bulb.

7 Conclusion

Extension to higher-degree polynomials. It is easy to extend our algorithm to handle complex polynomials of any degree $d \ge 2$: we just need an explicit escape radius for it and a reliable way to compute a bounding box for the image of a rectangular cell. The rest of the algorithm remains intact.

Douady [7] gives

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

as an escape radius for the polynomial $a_d z^d + \cdots + a_0$, when $d \ge 2$. (This result is easily proved along the same lines as in §2.) For the classic quadratic polynomials $z^2 + c$, this gives $2 + |c|$, which is just slightly greater than $\max(|c|, 2)$ given in §2. Stroh [25] gives 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 go beyond the escape radius. Fig. 8 shows two examples of Julia sets for the cubic map $f(z) = z^3 + c$ computed with our algorithm. For these c, $R = 2.5$ suffices. As Fig. 9 shows, in general cubic Julia sets can have an interior and be disconnected.

The natural computational tool for computing a bounding box for the image $f(A)$ of a cell A is interval arithmetic [15, 16]. Cell mapping using interval enclosures is the basis for robust algorithms for reasoning about discrete dynamical systems [13, 17].

Several good interval libraries exist [12] 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 floating-point arithmetic with directed rounding to guarantee reliable enclosures. This avoids the need for an error analysis like the one given in §5, which is fortunate because, for polynomials of degree d, we would be limited to about $53/d$ levels if we insisted on error-free computations with standard double-precision floating-point arithmetic.

The enclosures computed with interval arithmetic are always correct but rarely tight. This is not a problem for our algorithm because the enclosures decrease linearly as the cells decrease in size and we do not perform function iteration, and so are not subject to the wrapping effect that plague some interval methods. For the quadratic map, $f(z) = z^2 + c$, the bounding box computed with interval arithmetic is tight and can replace the ad hoc procedure given in §4, Step 2. This is a consequence of a

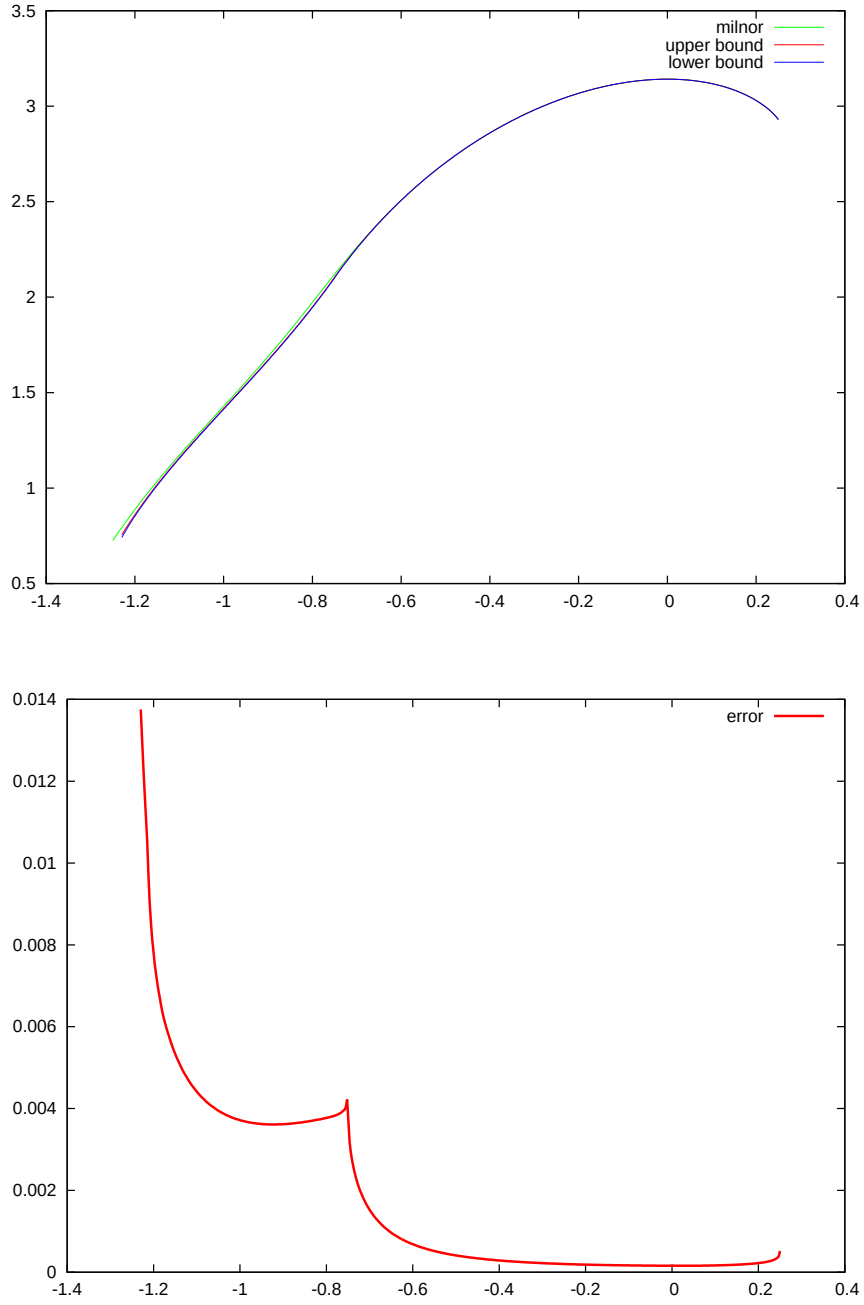

Fig. 7 Area of the filled Julia set for $-1.25 \leq c \leq 0.25$ computed with our algorithm: lower and upper bounds (top), absolute error bounds (bottom).

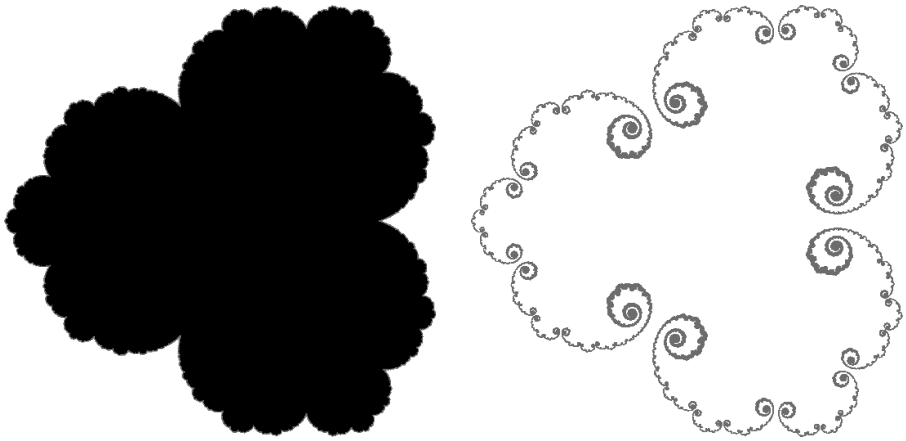

Fig. 8 Cubic Julia sets for $f(z) = z^3 + c$ when $c = 0.38$ (left) and $c = 0.41$ (right).

more general result: when all variables in an expression appear exactly once, interval arithmetic computes the best bounds [16, Theorem 6.2].

Future work. From the gray region in the quadtree it is simple to extract lower and upper estimates for the diameter and for the box dimension of Julia sets. It would be interesting to compare these estimates with the upper bound given by Ruelle [22]:

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

and also with the numerical results of Saupe [24].

We intend to adapt our algorithm for approximating Julia sets of rational functions, especially of 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 and no region on which to bootstrap the method. We hope to be able to find trapping zones around the zeros directly from the cell graph, instead of having to explicitly find and give those regions to the algorithm. The strongly connected components of the cell graph will probably play a key role here [13, 17].

Acknowledgements Preliminary results of this research were presented at SCAN 2002 and at the Dagstuhl-Seminar on Algebraic and Numerical Algorithms and Computer-assisted Proofs in 2005. Closer to the present form, this research was presented at the First Palis-Balzan International Symposium on Dynamical Systems in 2012. This research started when J. B. Oliveira was visiting the Visgraf laboratory at IMPA during its summer post-doctoral program in 2002. Visgraf is sponsored by CNPq, FAPERJ, FINEP, and IBM Brasil. The authors are partially supported by CNPq research grants.

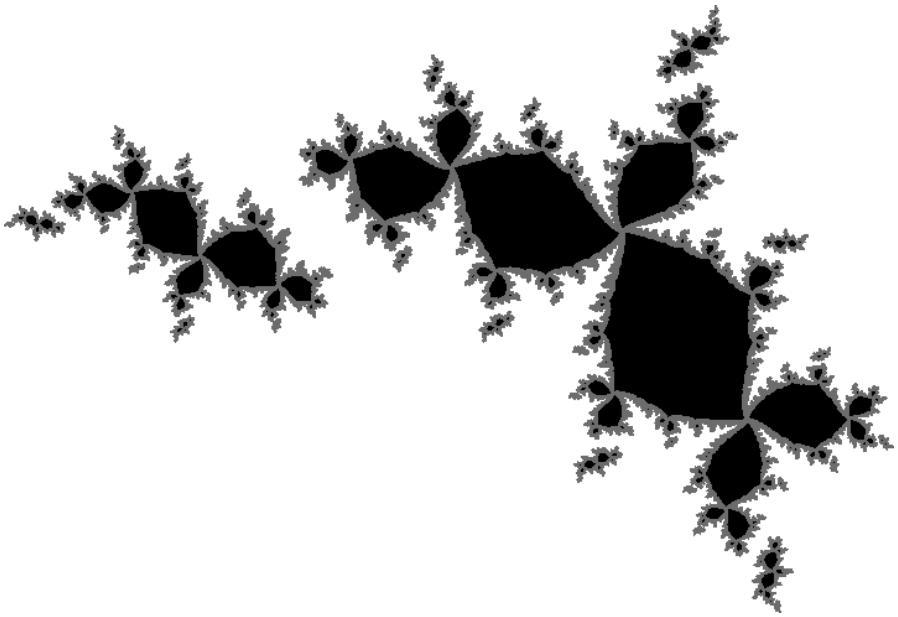

Fig. 9 A disconnected cubic Julia set with interior: $f(z) = z^3 + az + b$.

References

1. Blanchard, P.: Complex analytic dynamics on the Riemann sphere. Bulletin of the American Mathematical Society **11**(1), 85–141 (1984)
2. Branner, B.: The Mandelbrot set. In: [6], pp. 75–105
3. Braverman, M.: Hyperbolic Julia sets are poly-time computable. Electronic Notes in Theoretical Computer Science **120**, 17–30 (2005)
4. Braverman, M., Yampolsky, M.: Computability of Julia sets, *Algorithms and Computation in Mathematics*, vol. 23. Springer-Verlag, Berlin (2009)
5. Dellnitz, M., Hohmann, A.: A subdivision algorithm for the computation of unstable manifolds and global attractors. Numerische Mathematik **75**(3), 293–317 (1997)
6. Devaney, R.L., Keen, L. (eds.): Chaos and Fractals: The Mathematics behind the Computer Graphics. Proceedings of Symposia in Applied Mathematics 39. AMS (1989)
7. Douady, A.: Does a Julia set depend continuously on the polynomial? In: R.L. Devaney (ed.) Complex Dynamical Systems: The Mathematics Behind the Mandelbrot and Julia Sets, Proceedings of Symposia in Applied Mathematics 49, pp. 91–138. AMS (1994)
8. Gronwall, T.H.: Some remarks on conformal representation. Annals of Mathematics **16**(1-4), 72–76 (1914/15)
9. Hsu, C.S.: Cell-to-cell mapping: A method of global analysis for nonlinear systems. Springer-Verlag (1987)
10. Hsu, C.S.: Global analysis by cell mapping. International Journal of Bifurcations and Chaos **2**(4), 727–771 (1992)
11. Keen, L.: Julia sets. In: [6], pp. 57–74
12. Kreinovich, V.: Interval software. `http://cs.utep.edu/interval-comp/intsoft.html`
13. Michelucci, D., Foufou, S.: Interval-based tracing of strange attractors. International Journal of Computational Geometry & Applications **16**(1), 27–39 (2006)
14. Milnor, J.: Dynamics in one complex variable, *Annals of Mathematics Studies*, vol. 160, third edn. Princeton University Press (2006)
15. Moore, R.E.: Interval Analysis. Prentice-Hall (1966)
16. Moore, R.E., Kearfott, R.B., Cloud, M.J.: Introduction to interval analysis. SIAM (2009)

17. Paiva, A., de Figueiredo, L.H., Stolfi, J.: Robust visualization of strange attractors using affine arithmetic. Computers & Graphics **30**(6), 1020–1026 (2006)
18. Peitgen, H.O., Richter, P.H.: The Beauty of Fractals: Images of complex dynamical systems. Springer-Verlag (1986)
19. Peitgen, H.O., Saupe, D. (eds.): The Science of Fractal Images. Springer-Verlag (1988)
20. Rettinger, R.: A fast algorithm for Julia sets of hyperbolic rational functions. Electronic Notes in Theoretical Computer Science **120**, 145–157 (2005)
21. Rettinger, R., Weihrauch, K.: The computational complexity of some Julia sets. In: Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC 2003), pp. 177–185. ACM (2003)
22. Ruelle, D.: Repellers for real analytic maps. Ergodic Theory and Dynamical Systems **2**(1), 99–107 (1982)
23. Samet, H.: The quadtree and related hierarchical data structures. Computing Surveys **16**(2), 187–260 (1984)
24. Saupe, D.: Efficient computation of Julia sets and their fractal dimension. Physica D **28**(3), 358–370 (1987)
25. Stroh, C.M.: Julia sets of complex polynomials and their implementation on the computer. Master's thesis, University of Linz (1997)

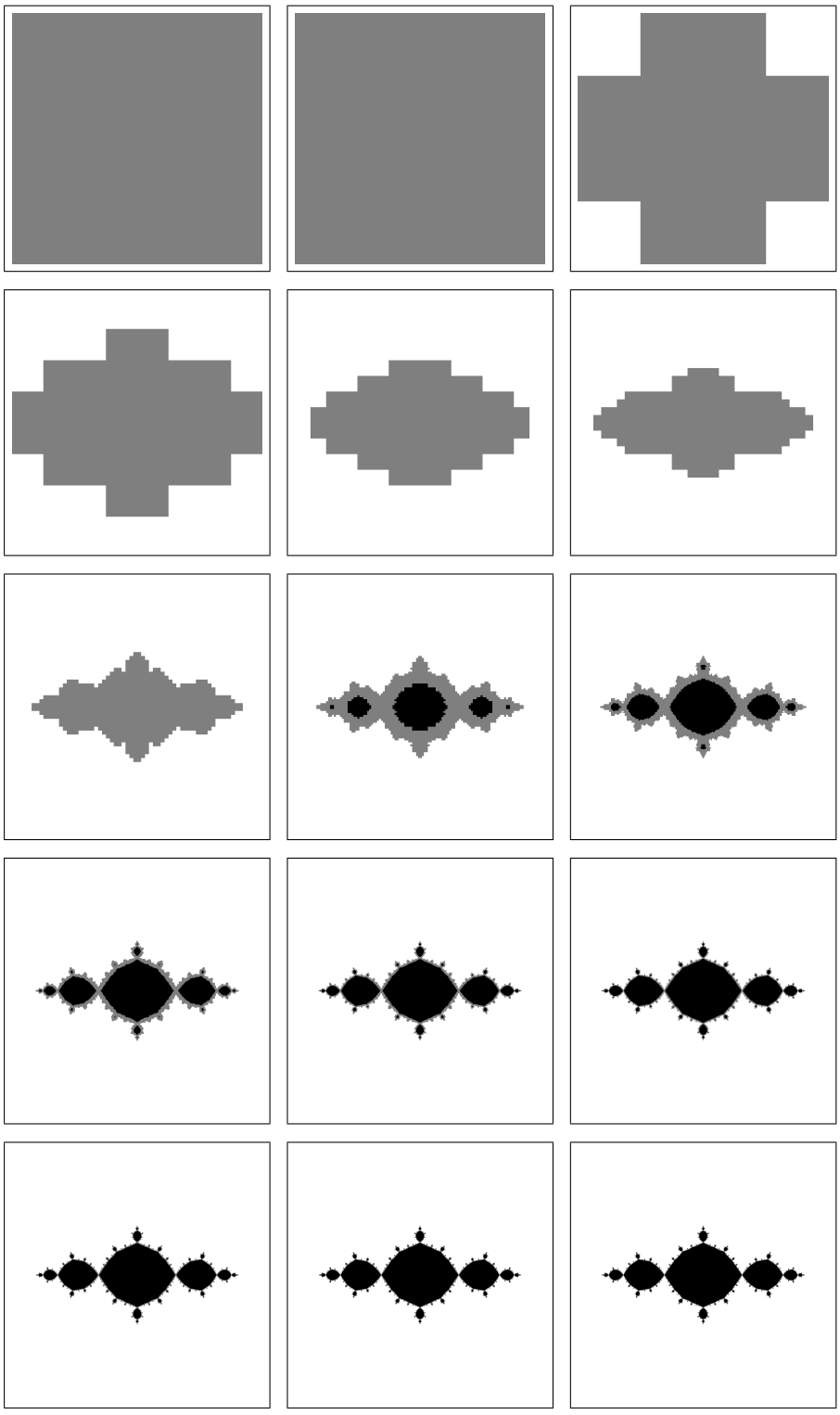

Fig. 10 Levels 0 to 14 of adaptive approximation of the Julia set for $c = -1$.

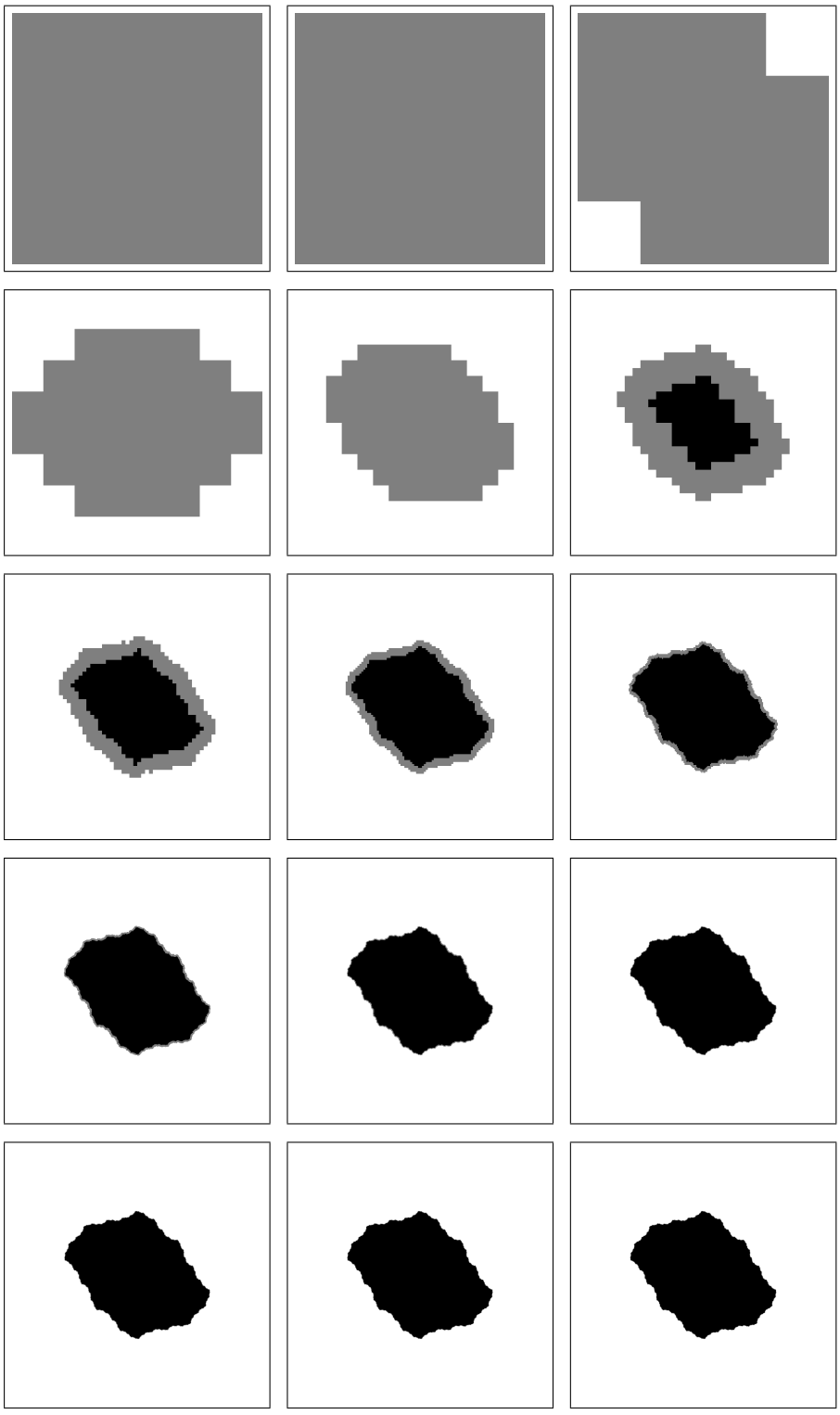

Fig. 11 Levels 0 to 14 of adaptive approximation of the Julia set for $c = -0.12 + 0.30\,i$.

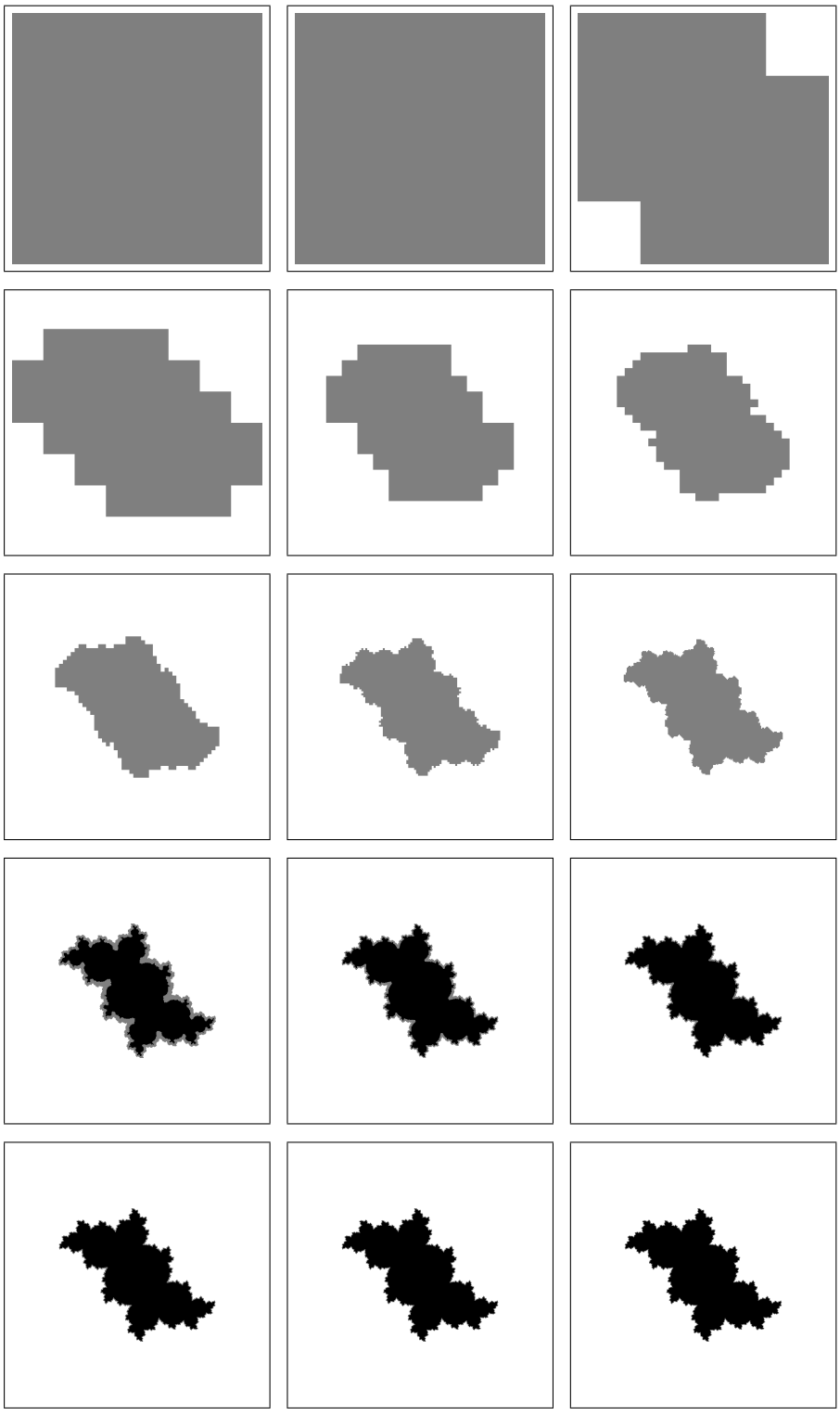

Fig. 12 Levels 0 to 14 of adaptive approximation of the Julia set for $c = -0.12 + 0.60\,i$.

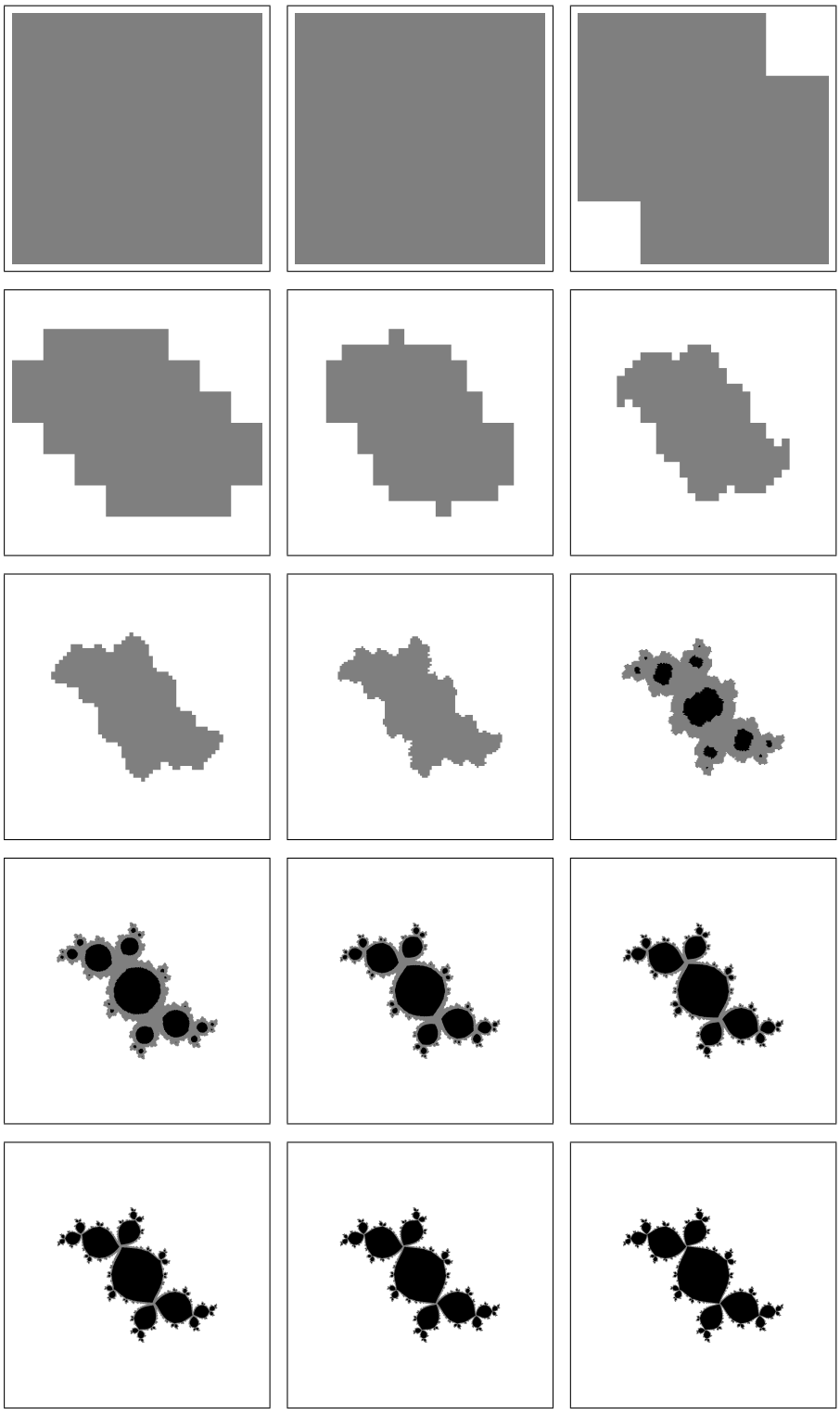

Fig. 13 Levels 0 to 14 of adaptive approximation of the Julia set for $c = -0.12 + 0.74\,i$.

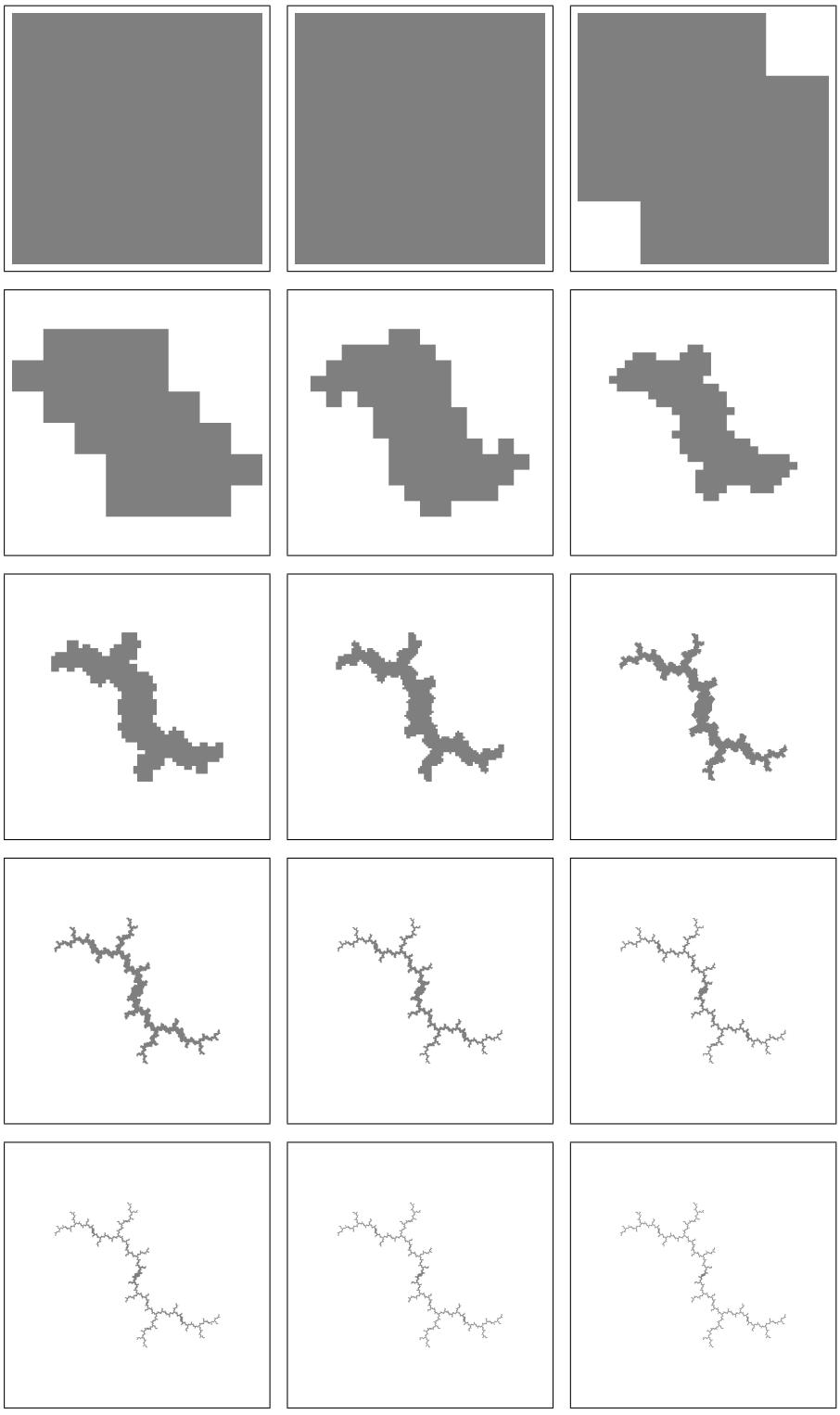

Fig. 14 Levels 0 to 14 of adaptive approximation of the Julia set for $c = i$.

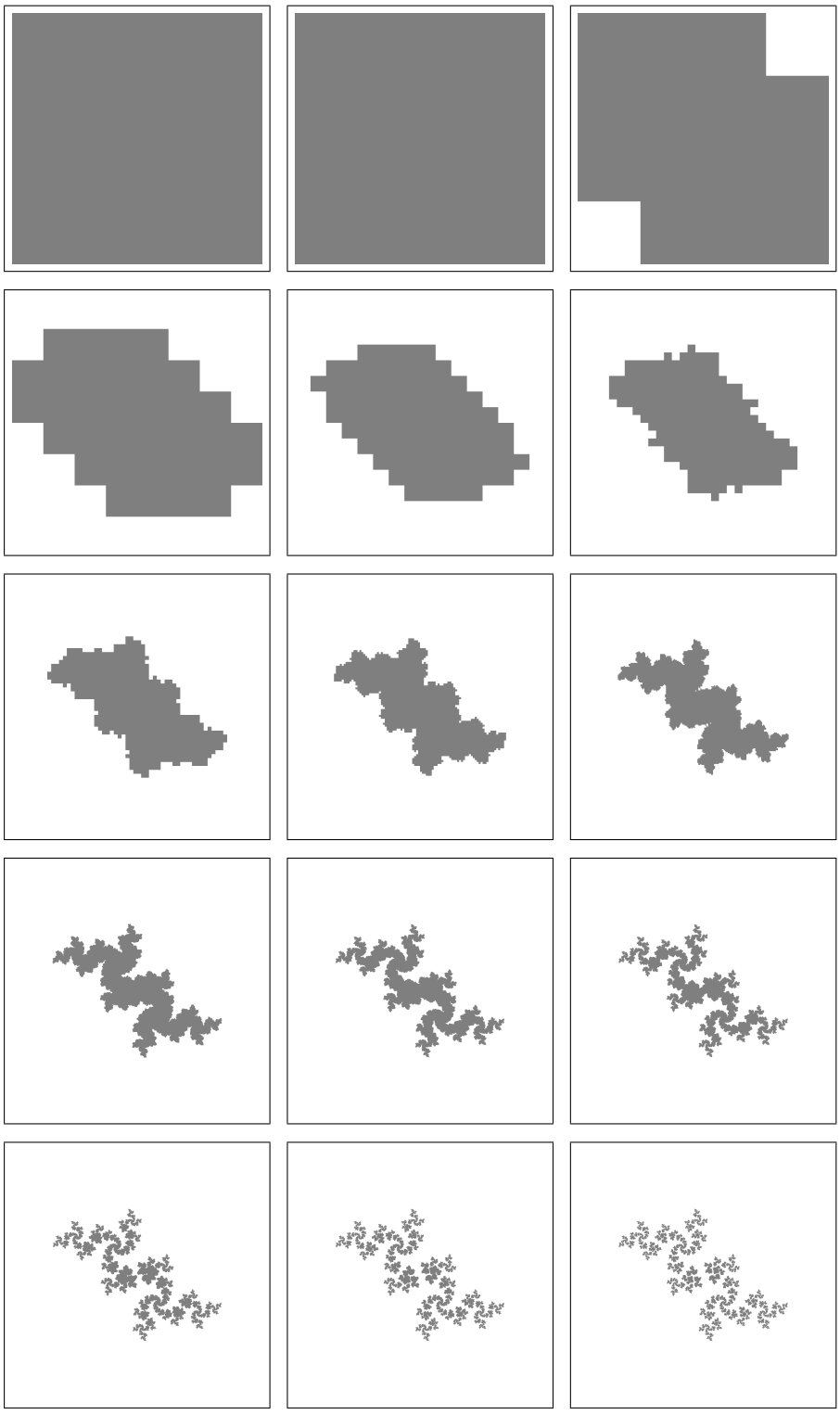

Fig. 15 Levels 0 to 14 of adaptive approximation of the Julia set for $c = -0.25 + 0.74\,i$.