Nonlinear Dyn (2017) 90:2457–2479
DOI 10.1007/s11071-017-3813-6

ORIGINAL PAPER

Fractal patterns from the dynamics of combined polynomial root finding methods

Krzysztof Gdawiec

Received: 14 December 2016 / Accepted: 14 September 2017 / Published online: 22 September 2017

Abstract Fractal patterns generated in the complex plane by root finding methods are well known in the literature. In the generation methods of these fractals, only one root finding method is used. In this paper, we propose the use of a combination of root finding methods in the generation of fractal patterns. We use three approaches to combine the methods: (1) the use of different combinations, e.g. affine and s-convex combination, (2) the use of iteration processes from fixed point theory, (3) multistep polynomiography. All the proposed approaches allow us to obtain new and diverse fractal patterns that can be used, for instance, as textile or ceramics patterns. Moreover, we study the proposed methods using five different measures: average number of iterations, convergence area index, generation time, fractal dimension and Wada measure. The computational experiments show that the dependence of the measures on the parameters used in the methods is in most cases a non-trivial, complex and non-monotonic function.

Keywords Fractal · Root finding · Iteration · Polynomiography

K. Gdawiec (✉)
Institute of Computer Science, University of Silesia,
Będzińska 39, 41-200 Sosnowiec, Poland
e-mail: kgdawiec@ux2.math.us.edu.pl

1 Introduction

Fractals, since their introduction, have been used in arts to generate very complex and beautiful patterns. While fractal patterns are very complex, only a small amount of information is needed to generate them, e.g. in the Iterated Function Systems only information about a finite number of contractive mappings is needed [29]. One of the fractal types widely used in arts is complex fractals, i.e. fractals generated in the complex plane. Mandelbrot and Julia sets together with their variations are examples of this type of fractals [30]. They are generated using different techniques, e.g. escape time algorithm [34] and layering technique [24].

Another example of complex fractal patterns is patterns obtained with the help of root finding methods (RFM) [18]. These patterns are used to obtain paintings, carpet or tapestry designs, sculptures [20] or even in animation [22]. For their generation, different methods are used. The most obvious method of obtaining various patterns of this type is the use of different root finding methods. The most popular root finding method used is the Newton method [14,35,37], but other methods are also widely used: Halley's method [17], the secant method [36], Aitken's method [38] or even whole families of root finding methods, such as Basic Family and Euler-Schröder Family [21]. Another popular method of generating fractal patterns from root finding methods is the use of different convergence tests [10] and different orbit traps [7,42]. In [6,34], it was shown that the colouring method of the patterns also plays a signif-

icant role in obtaining interesting patterns. Recently, new methods of obtaining fractal patterns from root finding methods were presented. In the first method, the authors used different iteration methods known from fixed point theory [13,23,32]; then, in [11], a perturbation mapping was added to the feedback process. Finally, in [12] we can find the use of different switching processes.

All the above-mentioned methods of fractal generation that use RFMs, except the ones that use the switching processes, use only a single root finding method in the generation process. Using different RFMs, we are able to obtain various patterns; thus, combining them together could further enrich the set of fractal patterns and lead to completely new ones, which we had not been able to obtain previously. In this paper, we present ways of combining several root finding methods to generate new fractal art patterns.

The paper is organised as follows. In Sect. 2, we briefly introduce methods of generating fractal patterns by a single root finding method. Next, in Sect. 3, we introduce three approaches on how to combine root finding methods to generate fractal patterns. The first two methods are based on notions from the literature, and the third one is a completely new method. Section 4 is devoted to remarks on the implementation of the algorithms on the GPU using shaders. Some graphical examples of fractal patterns obtained with the help of the proposed methods are presented in Sect. 5. In Sect. 6, numerical experiments regarding the generation times, average number of iteration, convergence area, fractal dimension and Wada measure of the generated polynomiographs are presented. Finally, in Sect. 7, we give some concluding remarks.

2 Patterns from a single root finding method

Patterns generated by a single polynomial root finding method have been known since the 1980s and gained much attention in the computer graphics community [28,38,42]. Around 2000, there appeared in the literature the term for the images generated with the help of root finding methods. The images were named polynomiographs, and the methods for their creation were collectively called polynomiography. These two names were introduced by Kalantari. The precise definition that he gave is the following [19]: polynomiography is the art and science of visualisation in approximation of the zeros of complex polynomials, via fractal and non-fractal images created using the mathematical convergence properties of iteration functions.

It is well known that any polynomial $p \in \mathbb{C}[Z]$ can be uniquely defined by its coefficients $\{a_n, a_{n-1}, \ldots, a_1, a_0\}$:

$$p(z) = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0 \tag{1}$$

or by its roots $\{r_1, r_2, \ldots, r_n\}$:

$$p(z) = (z - r_1)(z - r_2) \cdot \ldots \cdot (z - r_n). \tag{2}$$

When we talk about root finding methods, the polynomial is given by its coefficients, but when we want to generate a polynomiograph, the polynomial can be given in any of the two forms. The advantage of using the roots representation is that we are able to change the shape of the polynomiograph in a predictable way by changing the location of the roots.

In polynomiography, the main element of the generation algorithm is the root finding method. Many different root finding methods exist in the literature. Let us recall some of these methods, that later will be used in the examples presented in Sect. 5.

– The Newton method [21]

$$N(z) = z - \frac{p(z)}{p'(z)}. \tag{3}$$

– The Halley method [21]

$$H(z) = z - \frac{2p'(z)p(z)}{2p'(z)^2 - p''(z)p(z)}. \tag{4}$$

– The B_4 method—the fourth element of the Basic Family introduced by Kalantari [21]

$$B_4(z) = z - \frac{6p'(z)^2 p(z) - 3p''(z)p(z)^2}{p'''(z)p(z)^2 + 6p'(z)^3 - 6p''(z)p'(z)p(z)}. \tag{5}$$

– The E_3 method—the third element of the Euler-Schröder Family [21]

$$E_3(z) = N(z) + \left(\frac{p(z)}{p'(z)}\right)^2 \frac{p''(z)}{2p'(z)}. \tag{6}$$

– The Householder method [15]

$$H_h(z) = N(z) - \frac{p(z)^2 p''(z)}{2p'(z)^3}. \tag{7}$$

– The Euler–Chebyshev method [39]

$$E_C(z) = z - \frac{m(3-m)}{2}\frac{p(z)}{p'(z)} - \frac{m^2}{2}\frac{p(z)^2 p''(z)}{p'(z)^3}, \tag{8}$$

where $m \in \mathbb{R}$.

– The Ezzati–Saleki method [9]

$$E_S(z) = N(z) + p(N(z))\left(\frac{1}{p'(z)} - \frac{4}{p'(z)+p'(N(z))}\right). \tag{9}$$

In the standard polynomiograph generation algorithm, used for instance by Kalantari, we take some area of the complex plane $A \subset \mathbb{C}$. Then, for each point $z_0 \in A$ we use the feedback iteration using a chosen root finding method, i.e.

$$z_{n+1} = R(z_n), \tag{10}$$

where R is a root finding method, $n = 0, 1, \ldots, M$. Iteration (10) is often called the Picard iteration. After each iteration, we check if the root finding method has converged to a root using the following test:

$$|z_{n+1} - z_n| < \varepsilon, \tag{11}$$

where $\varepsilon > 0$ is the accuracy of the computations. If (11) is satisfied, then we stop the iteration process and colour z_0 using some colour map according to the iteration number at which we have stopped iterating, i.e. the iteration colouring. Another method of colouring polynomiographs is colouring based on the basins of attraction. In this method, each root of the polynomial gets a distinct colour. Then, after the iteration process we take the obtained approximation of the root and find the closest root of the polynomial. Finally, we colour the starting point with the colour that corresponds to the closest root.

In recent years, some modifications of the standard polynomiograph's generation algorithm were introduced. The first modification presented in [10] was based on the use of different convergence tests other than the standard test (11). The new tests were created using different metrics, adding weights in the metrics, using various functions that are not metrics, and even tests that are similar to the tests used in the generation of Mandelbrot and Julia sets were proposed. Examples of the tests are the following:

$$||z_{n+1}|^2 - |z_n|^2| < \varepsilon, \tag{12}$$

$$|0.01(z_{n+1} - z_n)| + |0.029|z_{n+1}|^2 - 0.03|z_n|^2| < \varepsilon, \tag{13}$$

$$|0.04\Re(z_{n+1} - z_n)| < \varepsilon \vee |0.05\Im(z_{n+1} - z_n)| < \varepsilon, \tag{14}$$

where $\Re(z)$, $\Im(z)$ denote the real and imaginary parts of z, respectively.

In [27], we can find the next modification which later was generalised in [13]. The modification is based on the use, instead of the Picard iteration, of the different iterations from fixed point theory. The authors have used ten different iterations, e.g.

– the Ishikawa iteration

$$\begin{aligned} z_{n+1} &= (1-\alpha)z_n + \alpha R(u_n), \\ u_n &= (1-\beta)z_n + \beta R(z_n), \end{aligned} \tag{15}$$

– the Noor iteration

$$\begin{aligned} z_{n+1} &= (1-\alpha)z_n + \alpha R(u_n), \\ u_n &= (1-\beta)z_n + \beta R(v_n), \\ v_n &= (1-\gamma)z_n + \gamma R(z_n), \end{aligned} \tag{16}$$

where $\alpha \in (0, 1]$, $\beta, \gamma \in [0, 1]$ and R is a root finding method. Moreover, in [13], iteration parameters (α, β, γ), which are real numbers, were replaced by complex numbers.

The last modification proposed in [11] is based on the use, during the iteration process, of the so-called perturbation mapping. In each iteration, the perturbation mapping alters the point from the previous iteration and this altered point is then used by the root finding method.

Putting the different modifications together, we obtain an algorithm that is presented as pseudocode in Algorithms 1 and 2. In the algorithms, I_v is an iteration method, Mann, Ishikawa, Noor, etc., that uses a chosen root finding method, with perturbation mapping added. The index v is a vector of parameters of the iteration method, i.e. $v \in \mathbb{C}^N$, where N is the number of parameters of the iteration. Similarly, C_u is a chosen

convergence test. The test takes two values: an element from the previous and the current iteration. Moreover, index u is a vector of parameters of the test and it contains, for instance, the calculation's accuracy $\varepsilon > 0$.

Algorithm 1: Rendering of a polynomiograph

Input: $p \in \mathbb{C}[Z]$, $\deg p \geq 2$ – polynomial, $A \subset \mathbb{C}$ – area, M – number of iterations, $I_v : \mathbb{C} \to \mathbb{C}$ – iteration, $C_u : \mathbb{C} \times \mathbb{C} \to \{true, false\}$ – convergence test, $colours[0..k]$ – colour map.
Output: Polynomiograph for the area A.

```
for z_0 ∈ A do
    [n, z] = ITERATEPOINT(z_0, p, I_v, C_u, M)
    Determine the colour for z_0 using n, z and the colour map colours
```

Algorithm 2: ITERATEPOINT

Input: $z_0 \in \mathbb{C}$ – point, $p \in \mathbb{C}[Z]$, $\deg p \geq 2$ – polynomial, $I_v : \mathbb{C} \to \mathbb{C}$ – iteration, $C_u : \mathbb{C} \times \mathbb{C} \to \{true, false\}$ – convergence test, M – number of iterations.
Output: The iteration number for which we stop the iteration process and the last calculated point.

```
ITERATEPOINT(z_0, p, I_v, C_u, M)
    n = 0
    while n < M do
        z_{n+1} = I_v(z_n)
        if C_u(z_n, z_{n+1}) = true then
            break
        n = n + 1
    return [n, z_{n+1}]
```

3 Combined root finding methods

Each polynomiograph is generated by a single root finding method. For a fixed polynomial, changing the root finding method changes the obtained pattern (Fig. 1). Similarly, when we fix the root finding method and change the polynomial, we obtain different patterns (Fig. 2). If we want to get new diverse patterns using the combination of patterns obtained with the different root finding methods or polynomials, then we need to do it manually using some graphics software, e.g. GIMP or Adobe Photoshop. This could be time consuming. So, it would be good to have a method that uses a combination of different properties of the individual root finding methods to obtain new and complex patterns. In this section, we introduce such methods.

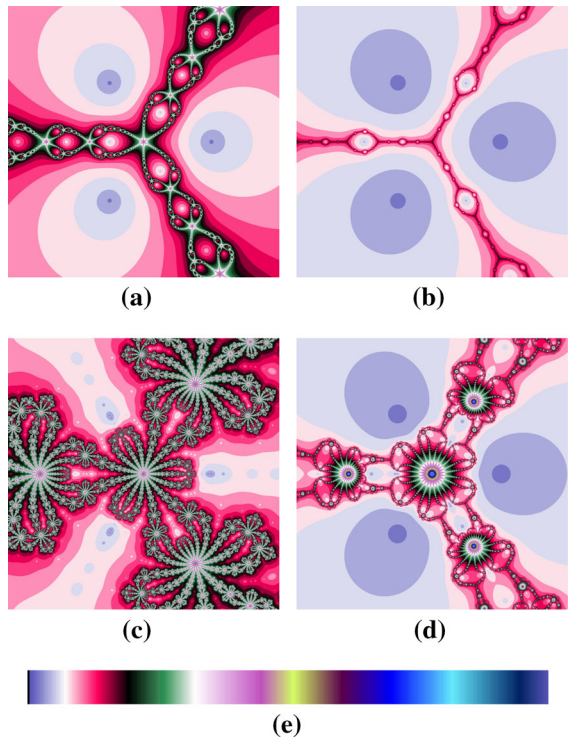


Fig. 1 Polynomiographs for $z^3 - 1$ generated using various root finding methods. **a** Newton, **b** Halley, **c** Euler–Chebyshev, **d** Ezzati–Saleki, **e** colour map

In the literature, we can find different methods of combining points, for instance:

– the affine combination [16]

$$\alpha_1 p_1 + \alpha_2 p_2 + \cdots + \alpha_n p_m = \sum_{i=1}^{m} \alpha_i p_i, \tag{17}$$

where $p_1, p_2, \ldots, p_m$ are points, $\alpha_1, \alpha_2, \ldots, \alpha_m \in \mathbb{R}$ and $\sum_{i=1}^{m} \alpha_i = 1$,

– the s-convex combination [31]

$$\alpha_1^s p_1 + \alpha_2^s p_2 + \cdots + \alpha_n^s p_m = \sum_{i=1}^{m} \alpha_i^s p_i, \tag{18}$$

where $p_1, p_2, \ldots, p_m$ are points, $\alpha_1, \alpha_2, \ldots, \alpha_m \geq 0$ are such that $\sum_{i=1}^{m} \alpha_i = 1$ and $s \in [0, 1]$.

We can use these combinations in the generation of polynomiographs. Let us say that we have m root finding methods and each method has its own iteration method I_{v_i} for $i = 1, 2, \ldots, m$. Further, for each iteration we fix α_i for $i = 1, 2, \ldots, m$, so that the sequence

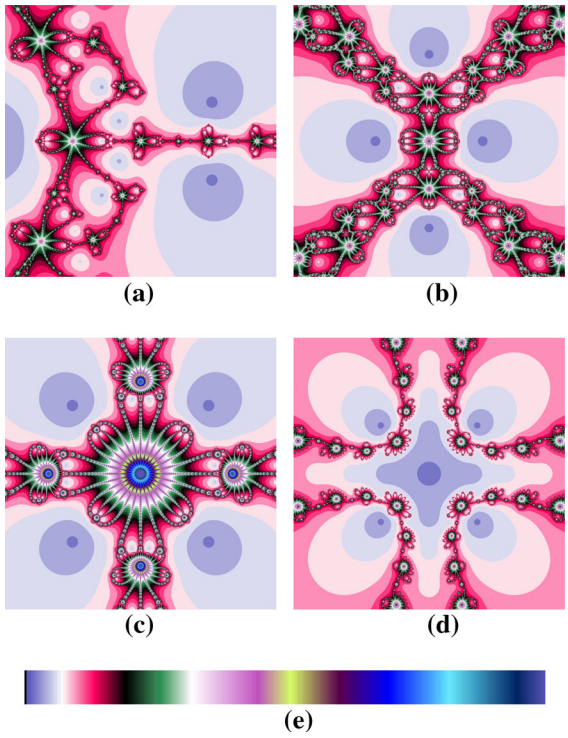


Fig. 2 Polynomiographs for different polynomials generated using the Householder method. **a** $z^3 - 3z + 3$, **b** $z^4 + z^2 - 1$, **c** $z^4 + 4$, **d** $z^5 + z$, **e** colour map

meets the conditions for the combination that we want to use, e.g. affine, s-convex. Then, we change the fourth line of Algorithm 2 in the following way:

– for the affine combination

$$z_{n+1} = \sum_{i=1}^{m} \alpha_i I_{v_i}(z_n), \tag{19}$$

– for the s-convex combination

$$z_{n+1} = \sum_{i=1}^{m} \alpha_i^s I_{v_i}(z_n). \tag{20}$$

In the case of the affine combination, we can extend this combination from real to complex parameters, so we take $\alpha_1, \alpha_2, \ldots, \alpha_m \in \mathbb{C}$ such that $\sum_{i=1}^{m} \alpha_i = 1$.

For a single root finding method in [13], Gdawiec et al. have used different iteration methods known from the fixed point theory. In fixed point theory, we can find iteration methods for more than one transformation, which are used to find the common fixed points of the transformations. We recall some of these iteration processes known from the literature. Let us assume that $T_1, T_2, \ldots, T_m : X \to X$ are transformations and $z_0 \in X$.

– The Das–Debata iteration [8]

$$\begin{aligned} z_{n+1} &= (1-\alpha_n)z_n + \alpha_n T_2(u_n), \\ u_n &= (1-\beta_n)z_n + \beta_n T_1(z_n), \end{aligned} \tag{21}$$

where $\alpha_n \in (0, 1]$, $\beta_n \in [0, 1]$ for $n = 0, 1, 2, \ldots$.

– The Khan–Domlo–Fukhar-ud-din iteration [25]

$$\begin{aligned} z_{n+1} &= (1-\alpha_n)z_n + \alpha_n T_m(u_{(m-1)n}), \\ u_{(m-1)n} &= (1-\beta_{(m-1)n})z_n + \beta_{(m-1)n} T_{m-1}(u_{(m-2)n}), \\ u_{(m-2)n} &= (1-\beta_{(m-2)n})z_n + \beta_{(m-2)n} T_{m-2}(u_{(m-3)n}), \\ &\vdots \\ u_{2n} &= (1-\beta_{2n})z_n + \beta_{2n} T_2(u_{1n}), \\ u_{1n} &= (1-\beta_{1n})z_n + \beta_{1n} T_1(z_n), \end{aligned} \tag{22}$$

where $\alpha_n \in (0, 1]$, $\beta_{in} \in [0, 1]$ for $n = 0, 1, 2, \ldots$ and $i = 1, 2, \ldots, m-1$.

– The Khan–Cho–Abbas iteration [26]

$$\begin{aligned} z_{n+1} &= (1-\alpha_n)T_1(z_n) + \alpha_n T_2(u_n), \\ u_n &= (1-\beta_n)z_n + \beta_n T_1(z_n), \end{aligned} \tag{23}$$

where $\alpha_n \in (0, 1]$, $\beta_n \in [0, 1]$ for $n = 0, 1, 2, \ldots$.

– The Yadav–Tripathi iteration [41]

$$\begin{aligned} z_{n+1} &= \alpha_n z_n + \beta_n T_1(z_n) + \gamma_n T_2(u_n), \\ u_n &= \alpha_n' z_n + \beta_n' T_2(z_n) + \gamma_n' T_1(z_n), \end{aligned} \tag{24}$$

where $\alpha_n, \beta_n, \gamma_n, \alpha_n', \beta_n', \gamma_n' \in [0, 1]$ and $\alpha_n + \beta_n + \gamma_n = \alpha_n' + \beta_n' + \gamma_n' = 1$ for $n = 0, 1, 2, \ldots$.

– The Yadav iteration [40]

$$\begin{aligned} z_{n+1} &= T_2(u_n), \\ u_n &= (1-\alpha_n)T_1(z_n) + \alpha_n T_2(z_n), \end{aligned} \tag{25}$$

where $\alpha_n \in [0, 1]$ for $n = 0, 1, 2, \ldots$.

Some of the iterations for m transformations when $T_1 = T_2 = \ldots = T_m$ reduce to the iterations for a single transformation, e.g. the Das–Debata iteration reduces to the Ishikawa iteration. Moreover, in the case of two mappings T_1, T_2, the Khan–Domlo–Fukhar-ud-din iteration reduces to the Das–Debata iteration.

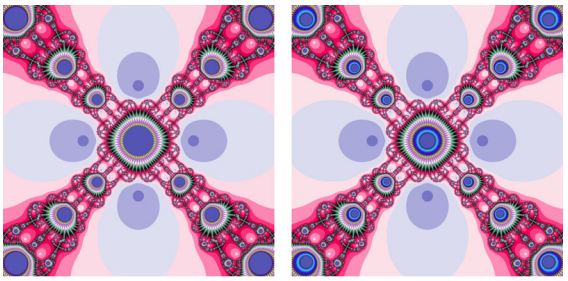
To combine root finding methods using the iterations for m transformation, we take m root finding methods as the transformations. For simplicity, we limit the sequences used in the iterations to constant sequences, i.e. $\alpha_n = \alpha$, $\beta_n = \beta$, $\gamma_n = \gamma$, $\alpha'_n = \alpha'$, $\beta'_n = \beta'$, $\gamma'_n = \gamma'$, $\beta_{(m-1)n} = \beta_{m-1}$, …, $\beta_{1n} = \beta_1$. Now, we replace the iteration used in Algorithm 2 by the new iteration.

In the two presented methods so far, we used a combination of different root finding methods for the same polynomial. The last proposed method generates polynomiographs in several steps and uses not only different root finding methods, but also various values of the other parameters in the consecutive steps. We call this type of polynomiography the multistep polynomiography. For each step, we take separate parameters that are used in standard polynomiography: polynomial, the maximal number of iterations, root finding method, iteration method, convergence test. Thus, we can use various values of the parameters in different steps, e.g. different polynomials. The area and colour map are defined only once for the multistep polynomiograph. In each step, we use the ITERATEPOINT function (Algorithm 2) and accumulate the number of iterations n that the function returns. Moreover, for each step we give a mapping $f : \mathbb{C} \to \mathbb{C}$ that transforms the difference between the last computed point in the iteration process for the step and the starting point of the step. We call this transformation the area transformation. The transformed point will then be used as a starting point for the next step. The pseudocode of the method is presented in Algorithm 3.

Algorithm 3: Rendering of a multistep polynomiograph

Input: $A \subset \mathbb{C}$ – area, $\{p_1, p_2, \ldots, p_N\}$ – polynomials, $\{I_{v_1}, I_{v_2}, \ldots, I_{v_N}\}$ – iterations, $\{M_1, M_2, \ldots, M_N\}$ – number of iterations, $\{C_{u_1}, C_{u_2}, \ldots, C_{u_N}\}$ – convergence tests, $\{f_1, f_2, \ldots, f_N\}$ – area transformations, $colours[0..k]$ – colour map.
Output: Multistep polynomiograph for the area A.

```
for z0 ∈ A do
    m = 0
    z = z0
    for i = 1, 2, ..., N do
        [n, u] = IteratePoint(z, p_i, I_{v_i}, C_{u_i}, M_i)
        m = m + n
        z = f_i(u − z)
    Determine the colour for z0 using m and the colour map colours
```



(a) **(b)**

Fig. 3 Polynomiographs obtained for the same parameters but with different precision of the calculations. **a** `float`, **b** `double`

4 GPU implementation

In the generation algorithm of complex fractals, each point is calculated independently from the others. This makes the algorithm easy to parallelise on the GPU. In the literature, we can find implementations of the algorithms for the generation of Mandelbrot [4] and Julia sets [33] using the OpenGL Shading Language (GLSL). The idea of implementation using GLSL of the generation algorithm for the fractals obtained with the help of RFMs is very similar. Calculations for each point are made in the fragment shader, but the number of parameters that are needed in the calculations is larger than in the case of the Mandelbrot or Julia sets.

In the GLSL implementation of the Mandelbrot and Julia sets, the `float` type is usually used in the calculations, which is sufficient if we do not zoom into the sets. But when we zoom in, the precision of the `float` type is insufficient and we lose details of the fractal. In the case of the RFM fractals, we have the same issue even in the macro scale, i.e. without zooming in. Figure 3 presents an example of RFM fractal pattern in $[-2.5, 2.5]^2$ obtained using the same parameters, but with different number types used in the calculations: (a) `float`, (b) `double`. In the central part of Fig. 3a and in the branches of the pattern, we see circular areas of uniform colour. When we look at the same areas in Fig. 3b, we see that more details appeared. Thus, in our implementation we used the `double` type, which was introduced in version 4.0 of the GLSL.

To represent a point, we can use two separate variables: one for the real part and the other for the imaginary part, but in GLSL it is better to use the `dvec2`

type. In this way, we need to implement only the multiplication and division of complex numbers, because for the addition and subtraction we can use the arithmetic operators $+, -$ defined for the `dvec2` type.

In every presented generation algorithm, we need to pass input parameters to the shader. Some of the parameters are passed as single uniform variables, but some of them need to be passed as a sequence, e.g. coefficients for the affine combination. Because in GLSL we do not have dynamic arrays, we need to overcome this issue using an array of fixed size and passing to the shader the actual number of elements that will be used in the algorithm. Of course, the number should be less than the size of the array. Nevertheless, we cannot use this technique when it comes to the coefficients of the polynomial and its derivatives, because the degree of the polynomial can be large, e.g. in [21], Kalantari used polynomials of degree 36 to generate some interesting polynomiographs. To pass polynomial and its derivatives, we used a two-dimensional floating point texture. Each row in this texture stores coefficients of one polynomial (polynomial, derivative). In multistep polynomiography, we need to pass several polynomials and their derivatives. In this case, the number of columns in the texture is determined by the largest degree of the polynomials and we store the coefficients in sequence, i.e. the coefficients of the first step polynomial and its derivatives, the coefficients of the second step polynomial and its derivatives, etc. Each of the input parameters (root finding method, iteration, convergence test, area transformation) cannot be written as a single function with some parameters, and we do not want to recompile the shader every time we change one of these input parameters. Thus, to change these parameters in a running application we used GLSL's subroutines and their arrays.

A known issue in the generation of complex fractals is aliasing. To deal with this problem, we implemented the supersampling method, which is widely used in many programs for generating fractal patterns, e.g. Fractint, ChaosPro, Ultra Fractal.

5 Graphical examples

In this section, we present some graphical examples of the fractal art patterns obtained using modifications proposed in Sect. 3. In all the examples, we used supersampling anti-aliasing with a factor of 4. In the polynomiographs with the basins of attraction, the distinct colours indicate distinct basins of attractions.

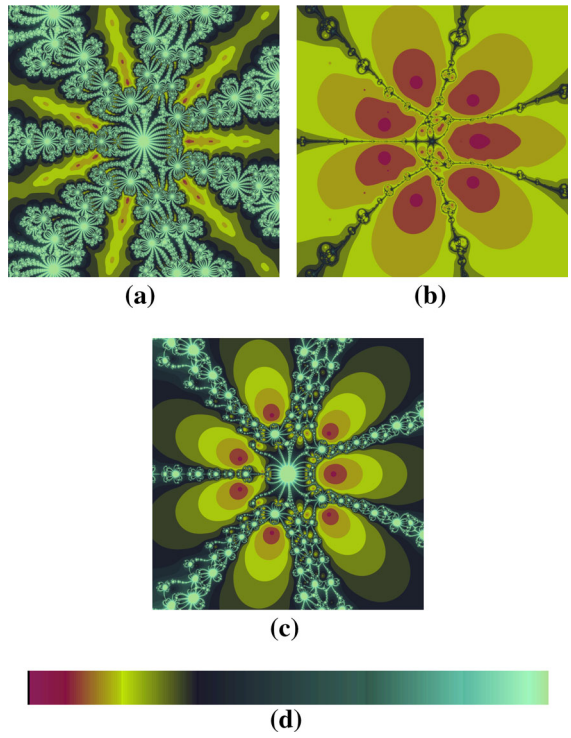


Fig. 4 Polynomiographs for $z^7 + z^2 - 1$ generated using various root finding methods. **a** Euler–Chebyshev, **b** B_4, **c** Householder, **d** colour map

We start with an example presenting the use of an affine combination of the root finding methods. In the example, we use three root finding methods: Euler–Chebyshev (with $m = 1.7$), B_4 and Householder. The other parameters used were the following: $p(z) = z^7 + z^2 - 1$, $A = [-2.5, 2.5]^2$, $M = 20$, Picard iteration for all three root finding methods and convergence test (11) with $\varepsilon = 0.001$. Figure 4 presents the polynomiographs obtained using the standard polynomiograph generation algorithm for the three root finding methods, and Fig. 5 shows their basins of attraction.

Examples of fractal patterns obtained with the help of an affine combination of the root finding methods are presented in Fig. 6, and their basins of attraction in Fig. 7. Comparing the images from Fig. 4 with the images from Fig. 6, we see that the obtained patterns differ from the original ones, forming new patterns. Moreover, we see that the new patterns possess some features of the polynomiographs obtained with the sin-

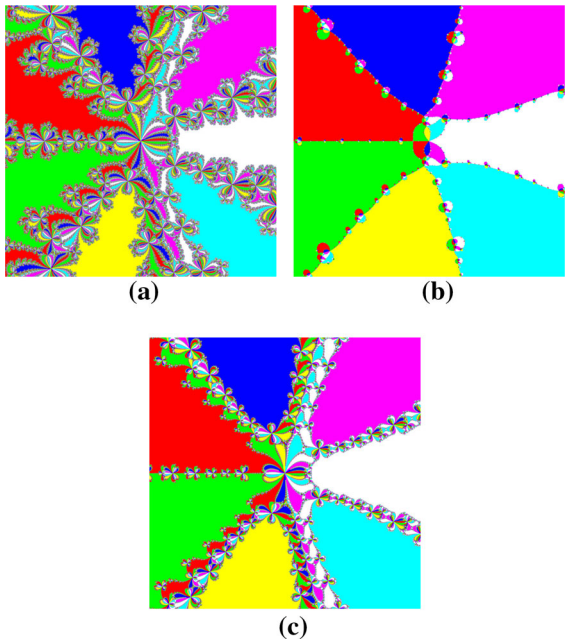


Fig. 5 Basins of attraction for $z^7 + z^2 - 1$ generated using various root finding methods. **a** Euler–Chebyshev, **b** B_4, **c** Householder

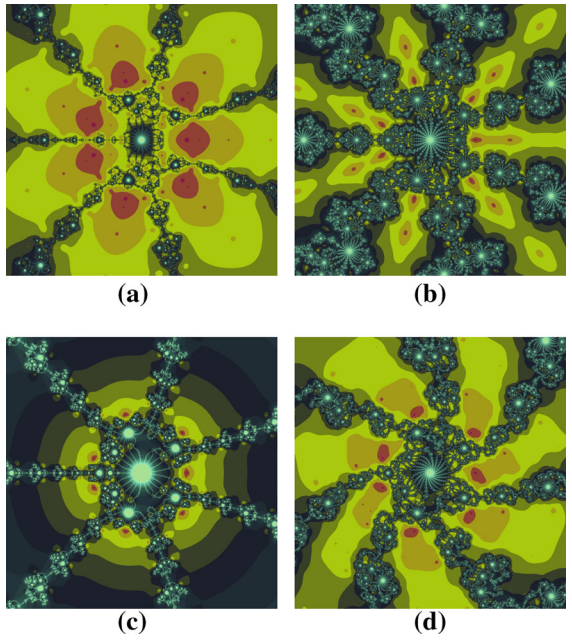


Fig. 6 Fractal patterns obtained with an affine combination of the root finding methods from Fig. 4. **a** $\alpha_1 = 0.1$, $\alpha_2 = 0.8$, $\alpha_3 = 0.1$, **b** $\alpha_1 = 0.7$, $\alpha_2 = 1.0$, $\alpha_3 = -0.7$, **c** $\alpha_1 = -0.8$, $\alpha_2 = 0.9$, $\alpha_3 = 0.9$, **d** $\alpha_1 = -0.3 - 0.2\mathbf{i}$, $\alpha_2 = 1.6 - 0.7\mathbf{i}$, $\alpha_3 = -0.3 + 0.9\mathbf{i}$

gle root finding methods. We also observe that the use of complex coefficients with a nonzero imaginary part introduces some twists in the patterns, which were not present in any of the original patterns. The dependence of the pattern's shape on the parameters of the affine combination is a non-trivial function. The non-triviality of this function is shown in Sect. 6.4, where the fractal dimension of patterns obtained with the affine combination is studied.

In the second example, we use various iteration processes of several root finding methods. Figures 8 and 9 present polynomiographs and basins of attractions obtained with the single root finding methods (Ezzati–Saleki, Halley) used in the iteration processes. The other parameters used to generate the patterns were the following: $p(z) = z^4 + 4$, $A = [-2, 2]^2$, $M = 25$, Picard iteration and convergence test (11) with $\varepsilon = 0.001$.

Figure 10 presents polynomiographs obtained with the help of the different iteration processes of the two considered root finding methods ($T_1 = E_S$, $T_2 = H$), and Fig. 11 shows their basins of attraction. The parameters for the iteration processes were the following:

(a) Das–Debata $\alpha = 0.3$, $\beta = 0.9$,
(b) Khan–Cho–Abbas $\alpha = 0.5$, $\beta = 0.5$,

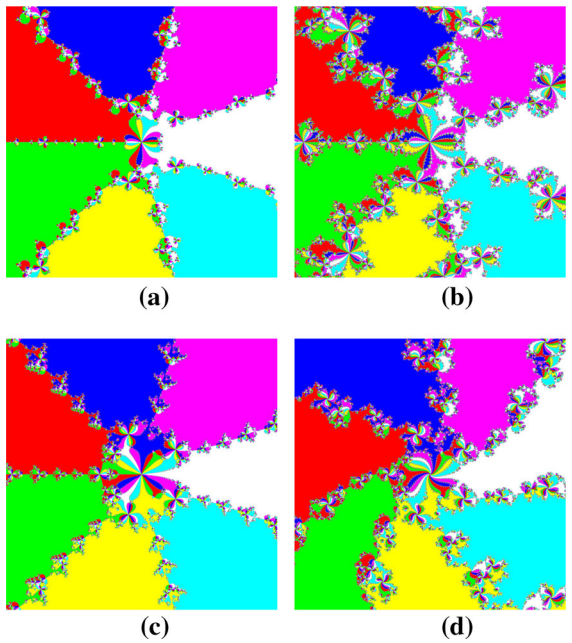


Fig. 7 Basins of attraction obtained with an affine combination of the root finding methods from Fig. 5. **a** $\alpha_1 = 0.1$, $\alpha_2 = 0.8$, $\alpha_3 = 0.1$, **b** $\alpha_1 = 0.7$, $\alpha_2 = 1.0$, $\alpha_3 = -0.7$, **c** $\alpha_1 = -0.8$, $\alpha_2 = 0.9$, $\alpha_3 = 0.9$, **d** $\alpha_1 = -0.3 - 0.2\mathbf{i}$, $\alpha_2 = 1.6 - 0.7\mathbf{i}$, $\alpha_3 = -0.3 + 0.9\mathbf{i}$

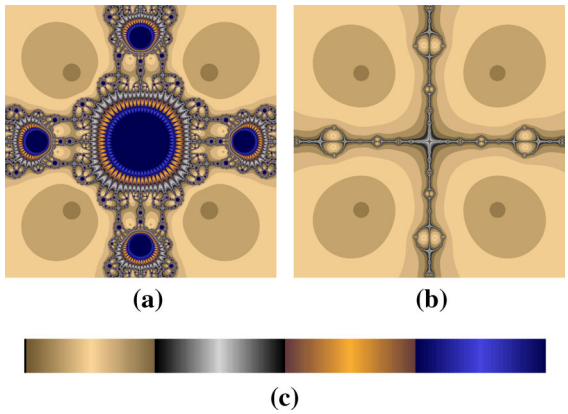


Fig. 8 Fractal patterns used in the example with the iteration processes. **a** Ezzati–Saleki, **b** Halley, **c** colour map

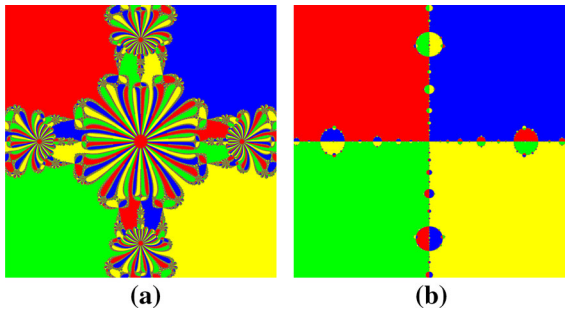


Fig. 9 Basins of attraction used in the example with the iteration processes. **a** Ezzati–Saleki, **b** Halley

(c) Yadav–Tripathi $\alpha = 0.3$, $\beta = 0.3$, $\gamma = 0.4$, $\alpha' = 0.8$, $\beta' = 0.1$, $\gamma' = 0.1$,
(d) Yadav $\alpha = 0.9$.

One can observe that the obtained patterns have properties of the two original patterns generated using the methods used in the iteration processes. For instance, the central part of the patterns reminds the central part of the pattern obtained with the Ezzati–Saleki method, and the four branches are thinner than the ones in the case of the Ezzati–Saleki method reminding branches obtained with the Halley method, e.g. Fig. 10d. In general, the obtained patterns differ from the original ones in a significant way. The dependence of the pattern's shape on the parameters used in the iteration processes, similar to the case of an affine combination, is a non-trivial function. A detailed discussion on this dependence is made in Sect. 6.4.

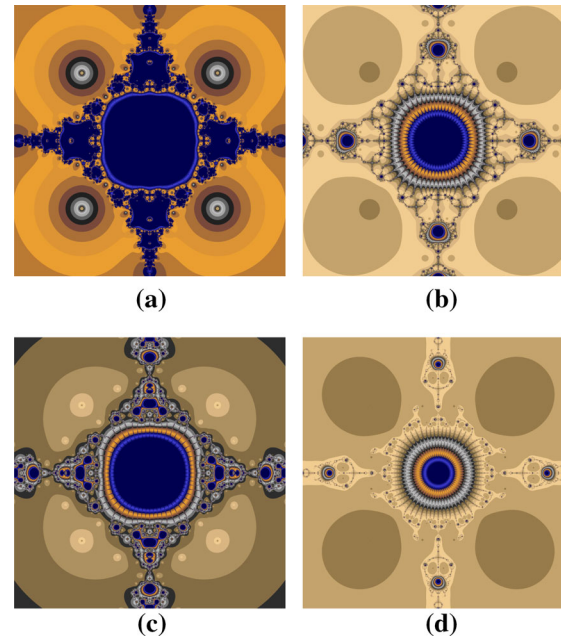


Fig. 10 Fractal patterns obtained with the help of different iteration processes. **a** Das–Debata, **b** Khan–Cho–Abbas, **c** Yadav–Tripathi, **d** Yadav

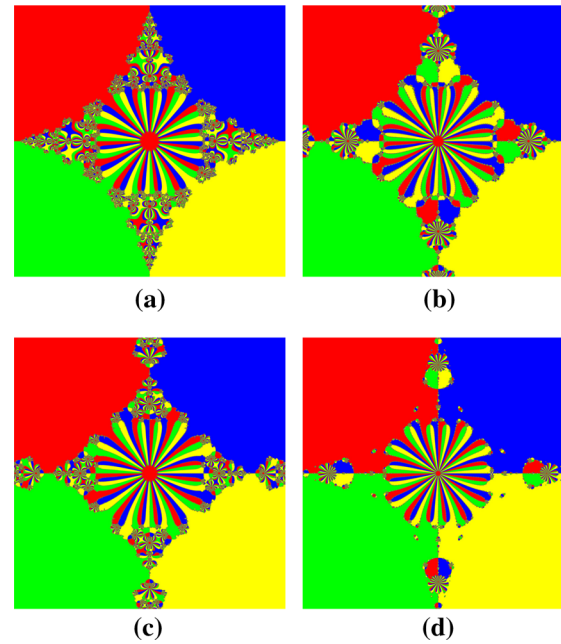


Fig. 11 Basins of attraction obtained with the help of different iteration processes. **a** Das–Debata, **c** Khan–Cho–Abbas, **d** Yadav–Tripathi, **e** Yadav

Table 1 Parameters used to generate the patterns from Figs. 12 and 13

Step	RFM	$p(z) =$	M	I_v	C_u	$f(z) =$
1	N	$z^3 - 1$	25	Picard	(11)	$p(z)$
2	H	$z^4 + 1$	10	Picard	(13)	$p(z)$
3	E_3	$z^4 + 1$	35	Picard	(11)	$p(z)$

In the last examples, we present some patterns obtained with the help of multistep polynomiography. We start with an example showing how the fractal pattern changes from step to step. In this example, we use three root finding methods with different values of the parameters. The parameters used in the generation algorithm are presented in Table 1 and for all the methods $A = [-2.5, 2.5]^2$ and $\varepsilon = 0.001$. Figure 12 presents the polynomiographs obtained using the root finding methods and the parameters from Table 1.

The successive steps in the generation of the multistep polynomiograph are presented in Fig. 13. After the first step, we obtain the pattern that was created with the first root finding method. Now, when we add the second step, the pattern becomes more complex and differs from the two original patterns used to generate it. The addition of the third step further changes the pattern, which significantly differs from the patterns of previous steps. Moreover, we can observe that the main shape (division of the plane into three parts) of the resulting pattern is determined by the polynomial used in the first step. The polynomials from the other steps add some further details to the pattern, mainly

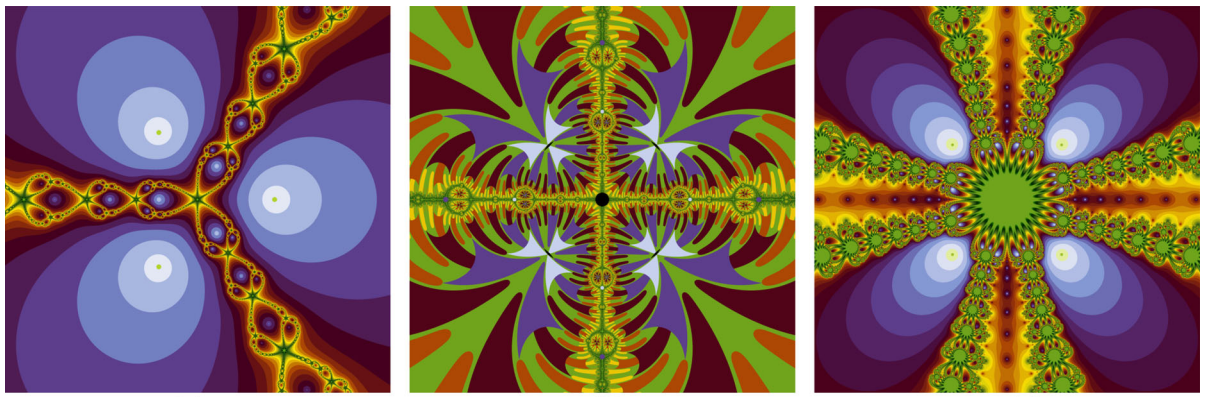


Fig. 12 Polynomiographs obtained using the parameters from Table 1. **a** N, **b** H, **c** E_3

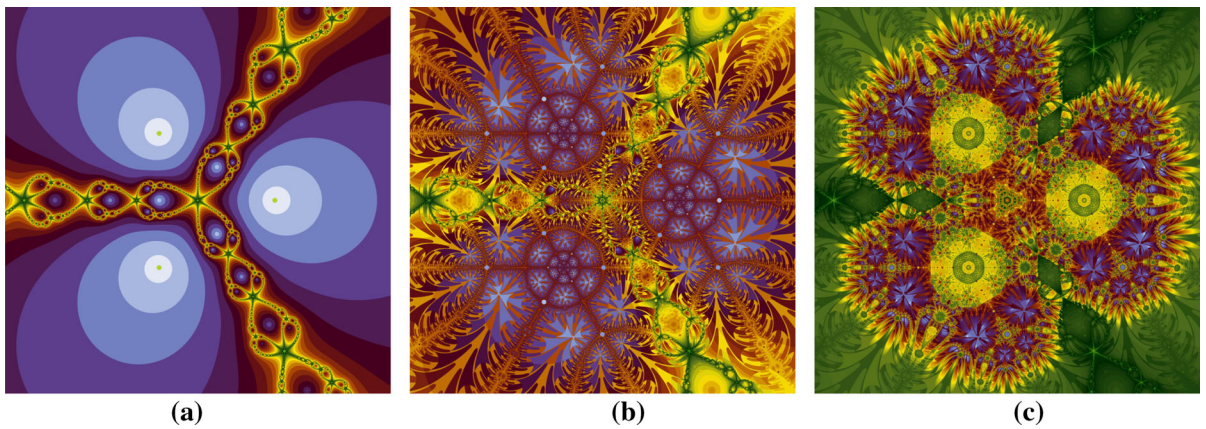


Fig. 13 Successive steps in the generation of a multistep polynomiograph for the parameters from Table 1. **a** 1st step, **b** 2nd step, **c** 3rd step

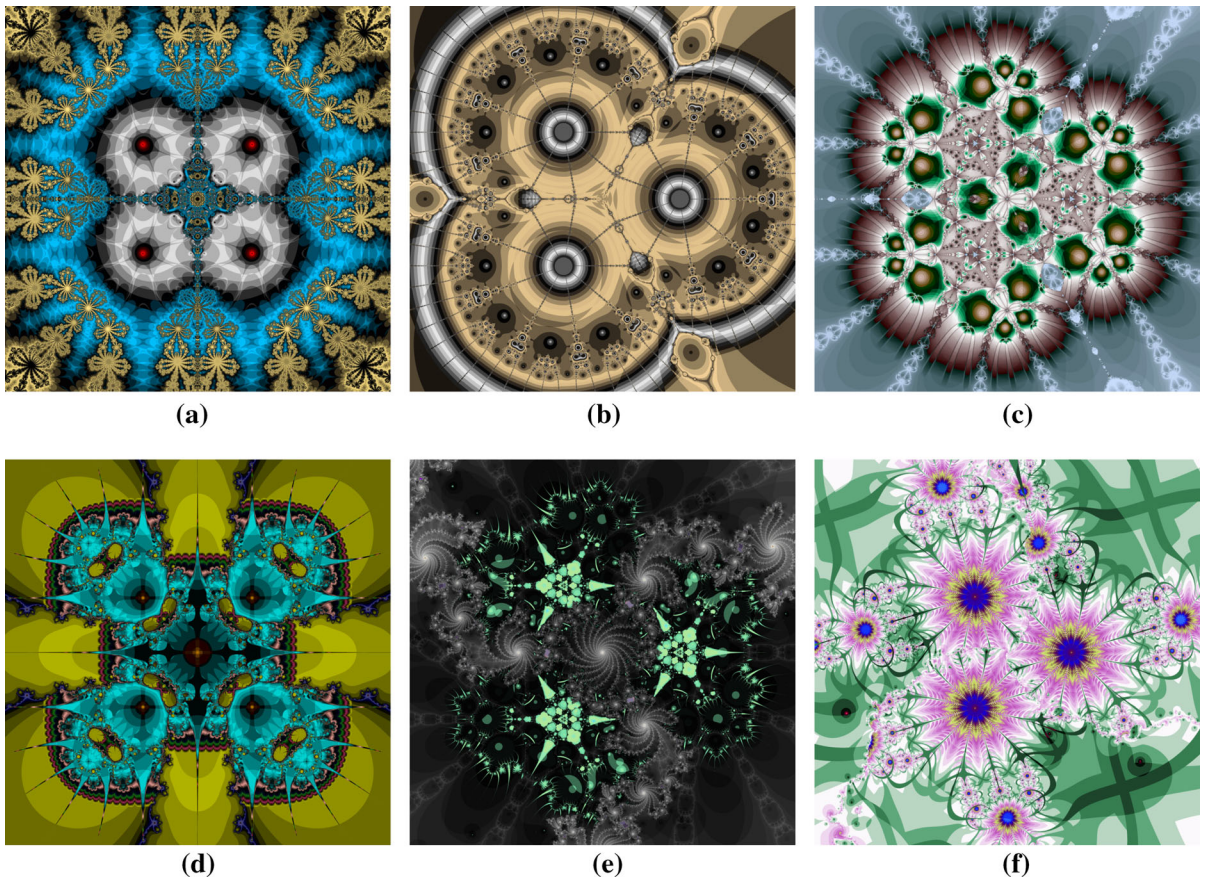


Fig. 14 Examples of fractal patterns obtained with the help of multistep polynomiography

in the regions where the first root finding method has converged fast to the roots.

The last example presents various fractal patterns obtained with the help of multistep polynomiography. The patterns are presented in Fig. 14 and the parameters used to generate them are gathered in Table 2. The ε parameter in all the examples was equal to 0.001. From the images, we see that using different combinations of the parameters, e.g. root finding methods, iteration processes, we are able to obtain very diverse and interesting fractal patterns.

6 Numerical examples

In this section, we present numerical examples and a comparison of the proposed methods of combining the root finding methods. We divide the examples according to the measures: average number of iterations [1], convergence area index [2], generation time [13], fractal dimension [44] and Wada measure [44].

In all the examples, the same polynomial is used, namely $p(z) = z^3 - 1$. Other common parameters used in the examples are the following: $A = [-1, 1]^2$, $M = 30$, convergence test (11) with $\varepsilon = 0.001$, polynomiograph resolution 800×800 pixels.

In each method of combining the root finding methods, we have parameters that belong to intervals. In the experiments, we divide each of the intervals independently into 201 equally spaced values and for each of the values we calculate the given measure. In the example with the affine combination, we use two combinations. In the first combination, we use three root finding methods, so we have two parameters, because the third parameter is determined by the condition that the parameters must sum to one. In the second example, we use two root finding methods, but this time with the complex coefficients. In this case, we have only one

Table 2 Parameters used to generate the patterns from Fig. 14

	Area	Step	RFM	$p(z) =$	M	I_v	C_u	$f(z) =$
(a)	$[-2.5, 2.5]^2$	1	N	$z^4 + 1$	25	Noor $\alpha = 0.3$, $\beta = 0.2$, $\gamma = 0.2$	(11)	$p(z)$
		2	E_C $m = 1.7$	$z^3 - 1$	15	Picard	(12)	$p(z)$
(b)	$[-2.5, 2.5]^2$	1	H	$z^3 - 1$	10	Picard	(11)	$1/z^3$
		2	H_h	$z^3 - 1$	25	Noor $\alpha = 0.5$ $\beta = 0.5$, $\gamma = 0.5$	(11)	$1/z^3$
(c)	$[-3, 3]^2$	1	B_4	$z^3 - 1$	5	Picard	(11)	$p(z)$
		2	N	$z^4 + 1$	25	Picard	(11)	$p(z)$
		3	E_3	$z^3 - 1$	55	Ishikawa $\alpha = 0.6$, $\beta = 0.2$	(11)	$p(z)$
(d)	$[-2.5, 2.5]^2$	1	N	$z^5 + z$	25	Picard	(11)	$p(z)$
		2	E_3	$z^5 + z$	5	Picard	(14)	$p(z)$
		3	E_3	$z^5 + z$	55	Ishikawa $\alpha = 0.6$, $\beta = 0.2$	(11)	$p(z)$
(e)	$[-2.5, 2.5]^2$	1	E_C $m = 1.7$	$z^3 - 1$	55	Ishikawa $\alpha = 0.7 + 0.4\mathbf{i}$, $\beta = 0.6$	(11)	$p(z)$
		2	B_4	$z^5 + z$	15	Picard	(11)	$p(z)$
		3	E_3	$z^4 + 1$	5	Picard	(14)	$p(z)$
(f)	$[-3, 3]^2$	1	H	$z^3 - 1$	20	Noor $\alpha = 0.8 - 0.4\mathbf{i}$ $\beta = 0.2$, $\gamma = 0.9 - 0.5\mathbf{i}$	(11)	$z/(0.75 + 2.5\mathbf{i})$
		2	N	$z^4 + 1$	20	Ishikawa $\alpha = 0.7$, $\beta = 0.7$	(14)	$z/(0.75 + 2.5\mathbf{i})$

parameter, because the second one, similar to the previous example, is determined using affine's combination condition. The only parameter in this case is separated into a real and imaginary part, and each of the parts is then divided independently into 201 values. In the example with the different iteration processes, we use two root finding methods, so the number of parameters in each case is the following: 2 for the Das–Debata iteration, 2 for the Khan–Cho–Abbas, 1 for the Yadav iteration and 6 for the Yadav–Tripathi one. In the case of the Yadav–Tripathi iteration, we can reduce the number of parameters to four, because $\alpha + \beta + \gamma = 1$ and $\alpha' + \beta' + \gamma' = 1$. Even for four parameters, the analysis of the results is difficult, so we reduced the number of parameters to two by fixing the other two parameters. In this way, we obtained two examples. In the first example, we have fixed $\alpha' = 0.2$, $\beta' = 0.5$, $\gamma' = 0.3$ and varying α, β, γ, and in the second we have fixed $\alpha = 0.2$, $\beta = 0.5$, $\gamma = 0.3$ and varying α', β', γ'.

Different root finding methods are used in the different methods of combining them. In the first example with the affine combination, we use three root finding methods: Ezzati–Saleki, Halley and Newton, and in the second example, we use Euler–Chebyshev (with $m = 1.7$) and Newton. The Picard iteration is used for all root finding methods in both examples with the affine combination. For the example with the iterations using several root finding methods, we use the Ezzati–Saleki and Halley methods.

In multistep polynomiography, we can use a different polynomial in each step. Thus, in this case, we cannot compute the considered measures—except for generation time—because in each step we have a different set of roots, so we do not know if after performing all the steps the method has converged to a root or not. Because of this, in the case of this method we will measure only generation time. The times will be measured

for two implementations, namely an implementation using CPU and another using GPU.

All the algorithms for polynomiograph generation were implemented using GLSL shaders (GPU implementation) and, in the case of multistep polynomiography, the CPU implementation was made in a Java-based programming language named Processing. The experiments were performed on a computer with the following specifications: Intel i5-4570 (@3.2 GHz) processor, 16 GB DDR3 RAM, AMD Radeon HD7750 with 1 GB GDDR5, and Microsoft Windows 10 (64-bit).

6.1 Average number of iterations

The average number of iterations (ANI) [1,2] is computed from the polynomiograph obtained using the iteration colouring. Each point in this polynomiograph corresponds to the number of iterations needed to find a root or to the maximal number of iterations when the root finding method has not converged to any root. The ANI measure is computed as the mean number of iterations in the given polynomiograph.

Figure 15 presents the average number of iterations in the parameters' space for the affine combination of three root finding methods. From the plot, we see that ANI is a non-trivial and non-monotonic function of the parameters. The minimal value of ANI, equal to 2.908, is attained at $\alpha_1 = 0.0$, $\alpha_2 = 0.89$, $\alpha_3 = 0.11$ and the maximal value 20.422 is attained at $\alpha_1 = -1.0$, $\alpha_2 = -1.0$, $\alpha_3 = 3.0$, so the dispersion of the values is wide. The values of ANI for the methods used in the combination are the following: 5.926 for the Ezzati–Saleki method, 3.155 for the Halley method and 5.082 for the Newton method. Comparing these values with the minimal value obtained with the affine combination, we see that using the affine combination we are able to obtain a lower value of ANI than in the case of the methods used in the combination.

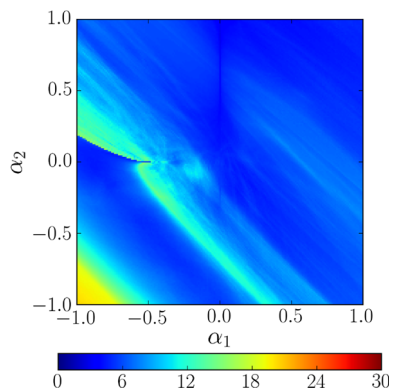


Fig. 15 The average number of iterations in the parameters' space ($\mathbb{R}^2$) for the affine combination of three root finding methods

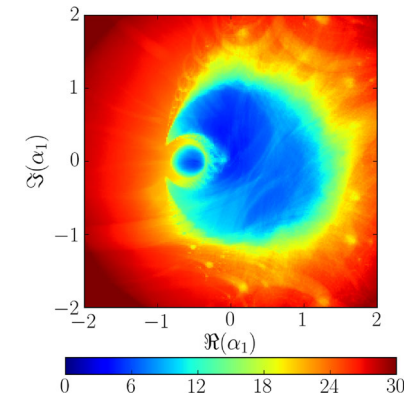


Fig. 16 The average number of iterations in the parameters' space ($\mathbb{C}$) for the affine combination of two root finding methods

The plot of ANI in the parameters' space for the affine combination of two methods is presented in Fig. 16. Looking at the plot, we see that the ANI is a very complex function of the parameters. We do not see any monotonicity of this function. Moreover, we see that the use of the complex parameters has a great impact on the value of ANI. The low values of ANI are visible in the central part of the plot, attaining the minimum 4.222 at $\alpha_1 = 0.04 + 0.12\mathbf{i}$, $\alpha_2 = 0.96 - 0.12\mathbf{i}$. The maximal value of ANI (30) is attained at $\alpha_1 = -2 - 2\mathbf{i}$, $\alpha_2 = 3 + 2\mathbf{i}$ (the lower-left corner of the considered area). Thus, dispersion in this case is also wide. The values of ANI for the two methods used in the combination are the following: 7.846 for the Euler–Chebyshev method and 5.082 for the Newton method. Thus, also in the case of using the complex parameters in the affine combination we are able to obtain a lower value of ANI than in the case of the component methods of the combination.

Figure 17 presents the results of computing the ANI measure obtained for different iteration processes. The values of ANI for the root finding methods (with the Picard iteration) that were used in these iteration processes are the following: 5.926 for the Ezzati–Saleki method and 3.155 for the Halley method. For the Das–Debata iteration (Fig. 17a), we see that the α parameter has the biggest impact on the ANI, i.e. the greater the

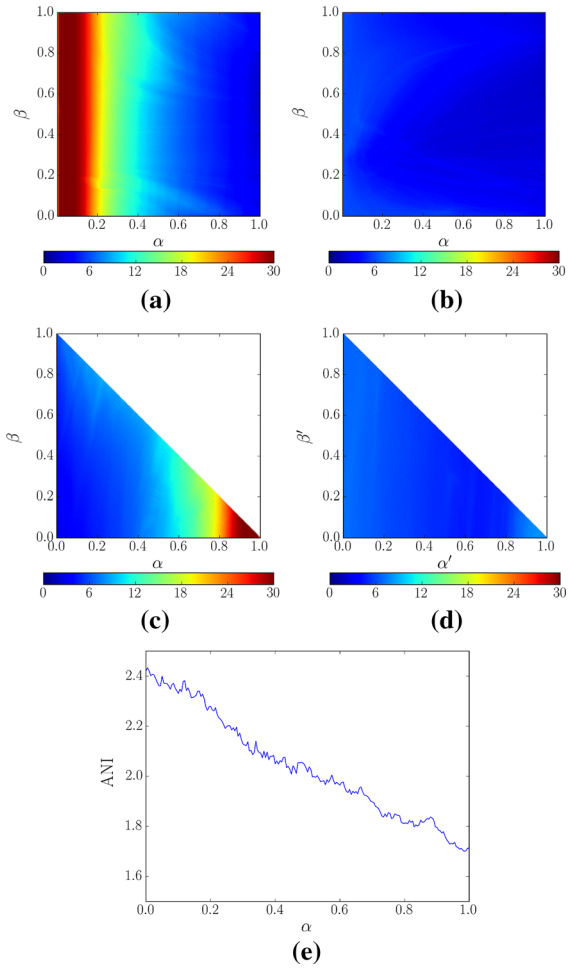


Fig. 17 The average number of iterations in the parameters' space ($[0, 1]^2$ and $[0, 1]$) for various iteration processes. **a** Das–Debata, **b** Khan–Cho–Abbas, **c** Yadav–Tripathi (fixed α', β', γ') **d** Yadav–Tripathi (fixed α, β, γ), **e** Yadav

value of α the lower the value of ANI. The minimum value 1.738, that is significantly lower in the case of the Ezzati–Saleki and Halley methods, is attained at $\alpha = 1.0$, $\beta = 0.665$. In Fig. 17b, we see the plot for the Khan–Cho–Abbas iteration. This plot seems to have a uniform colour. This is due to the fact that for this iteration we have small dispersion of the values, and the obtained values of ANI are small compared to the maximum number of 30 iterations (minimum 1.738, maximum 6.238). In order to show better the variation of the values, a plot with the colours limited to the available values is presented in Fig. 18a. We see that ANI is a complex function of the parameters used in the iteration process. The results for the Yadav–Tripathi iteration are

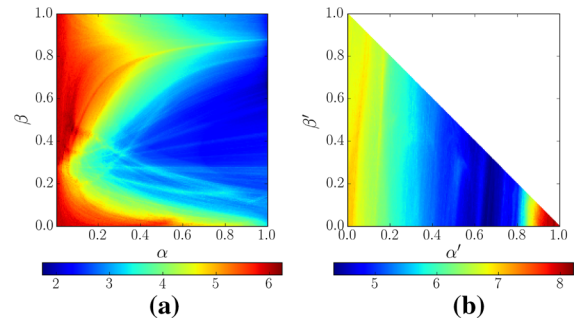


Fig. 18 The average number of iterations in the parameters' space ($[0, 1]^2$) for various iteration processes (the colours are limited to the available numbers of performed iterations). **a** Khan–Cho–Abbas, **b** Yadav–Tripathi (fixed α, β, γ)

presented in Fig. 17c and d. In the case of fixed α', β' and γ' (Fig. 17c), we see that the higher the value of α is, the higher the value of ANI. The minimal value (2.132) is attained at $\alpha = 0$, $\beta = 0$, $\gamma = 1$ ($\alpha' = 0.2$, $\beta' = 0.5$, $\gamma' = 0.3$ are fixed). In the other case (Fig. 17d), similarly to the Khan–Cho–Abbas iteration, we see a plot with an almost uniform colour. The dispersion of the values in this case is between 4.33 and 8.216. In Fig. 18b, a plot with better visibility of the values' variation is presented. From this plot, we see that the dependency of ANI on the parameters is more complex than in the case of fixed α', β', γ'. This time, the minimum is attained at $\alpha' = 0.675$, $\beta' = 0.02$, $\gamma' = 0.305$ ($\alpha = 0.2$, $\beta = 0.5$, $\gamma = 0.3$ are fixed). The last plot in Fig. 17 presents the dependency of ANI on the α parameter in the Yadav iteration. The dispersion of the values is smaller than in the case of the Khan–Cho–Abbas iteration and ranges from 1.7 to 2.433. Thus, for all values of α we obtain better results than in the case of methods used in the iteration. Moreover, we see that the higher the value of α is, the lower the value of ANI.

6.2 Convergence area index

The convergence area index (CAI) [1,2] is computed from the polynomiograph obtained using the iteration colouring. Using the polynomiograph, we count the number of converging points n_c, i.e. points whose value in the polynomiograph is less than the maximum number of iterations. Next, we divide n_c by the total number of points n in the polynomiograph, i.e.

$$\mathrm{CAI} = \frac{n_c}{n}. \tag{26}$$

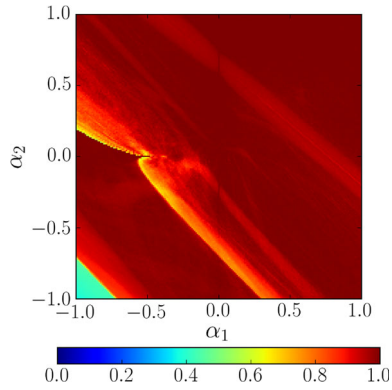


Fig. 19 The convergence area index in the parameters' space ($\mathbb{R}^2$) for the affine combination of three root finding methods

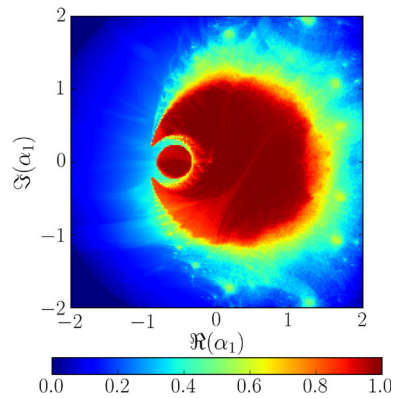


Fig. 20 The convergence area index in the parameters' space ($\mathbb{C}$) for the affine combination of two root finding methods

The value of CAI is between 0 and 1 and tells us how much of the polynomiograph's area has converged to roots. The higher the value is, the larger area has converged (0—no point has converged, 1—all points have converged).

The CAI measure in the parameters' space for the affine combination of three root finding methods is presented in Fig. 19. Similarly to ANI, the function of the parameters is a non-trivial one. Many points in the parameters' space have a high value of CAI. The maximal value (1.0) is, for instance, attained at $\alpha_1 = -1.0$, $\alpha_2 = -0.07$, $\alpha_3 = 2.07$. The minimal value of 0.38, which indicates a poor convergence of the combination, is attained at $\alpha_1 = -1.0$, $\alpha_2 = -1.0$, $\alpha_3 = 3.0$ (the lower-left corner of the considered area). All the methods (Ezzati–Saleki, Halley and Newton) used in the combination obtained the same value of CAI, namely 1.0, so all points have converged to the roots. Thus, only the points in the parameters' space that obtained the same value of CAI are comparable with the original methods. For instance, the point with the lowest ANI has a value of CAI equal to 1.0; thus, using a combination with those parameters gives the same convergence, but less iterations are needed to converge to the root.

Figure 20 presents the convergence area index in the parameters' space for the affine combination of two root finding methods. The CAI measure for the methods used in the combination was the following: 0.991 for the Euler–Chebyshev method and 1.0 for the Newton method. From the plot in Fig. 20, we see that the dependence of CAI on the parameters is very similar to the one obtained in the case of ANI—function with a very complex shape. The highest values of CAI are obtained in the central part of the plot—the maximum is, for instance, attained at $\alpha_1 = 0.04 + 0.12\mathbf{i}$, $\alpha_2 = 0.96 - 0.12\mathbf{i}$ (the point with the minimal value of ANI). To the contrary, the minimal value (0.0—no point has converged) is attained at the lower-left corner of the area. Thus, using the affine combination with the complex parameters can worsen in a significant way the convergence of the method compared to the methods used in the combination.

The results of computing the CAI measure in the parameters' space for various iteration processes are presented in Fig. 21. The CAI measure for both the root finding methods (Ezzati–Saleki, Halley) that were used in these iteration processes is equal to 1.0. For the Das–Debata iteration (Fig. 21a), we see that, similar to the case of ANI, the biggest impact on the CAI is due to the α parameter. For low values of α the CAI is also low, and from about 0.179 the value of CAI becomes high. The value of 1.0 has about 40% of the points in the parameters' space. Compared to the Das–Debata iteration, the Khan–Cho–Abbas iteration (Fig. 21b) has lower dispersion of the values – from 0.962 to 1.0. In order to see better the variation in the values, a plot with the colours limited to the available values is presented in Fig. 22a. From the plot, we see that CAI is a complex function of the parameters. Moreover, about 60% of the points has the highest value (1.0). For the Yadav–Tripathi iteration (Fig. 21c, d), we see a very similar dependency like the one for ANI. For the first case (fixed α', β', γ'), we see that the biggest impact on the value of CAI is due to the α parameter and that for high values of α the CAI measure has low values. Moreover, only about 33% of the points in the parame-

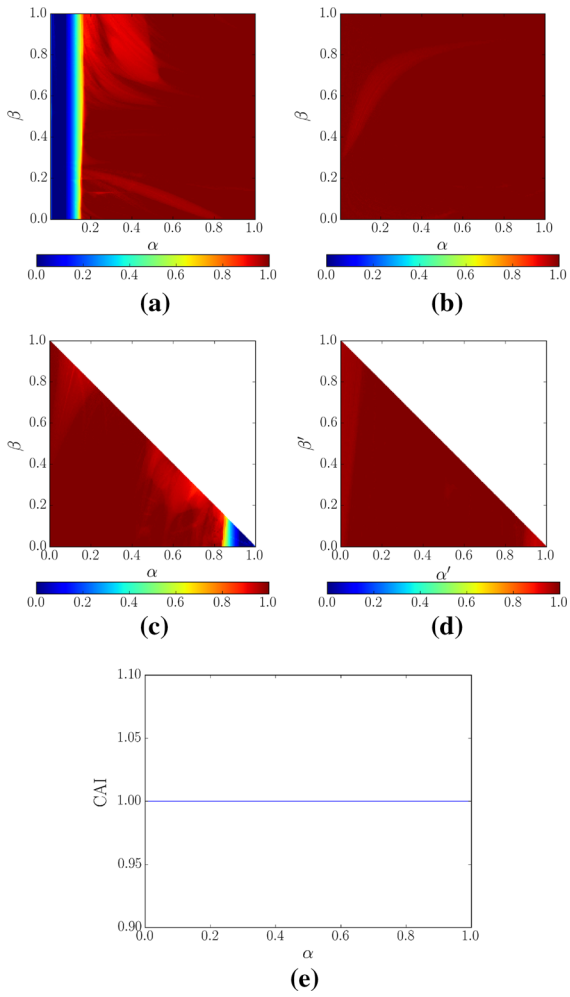


Fig. 21 The convergence area index in the parameters' space ($[0, 1]^2$ and $[0, 1]$) for various iteration processes. **a** Das–Debata, **b** Khan–Cho–Abbas, **c** Yadav–Tripathi (fixed α', β', γ') **d** Yadav–Tripathi (fixed α, β, γ), **e** Yadav

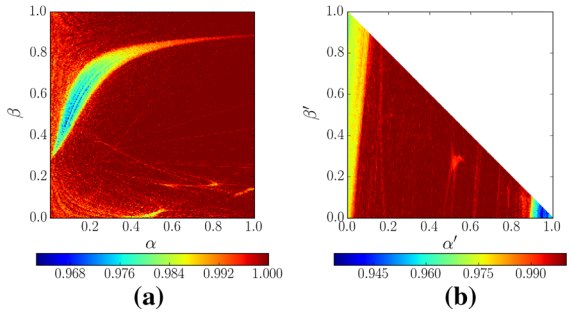


Fig. 22 The convergence area index in the parameters' space ($[0, 1]^2$) for various iteration processes (the colours are limited to the available values of CAI). **a** Khan–Cho–Abbas, **b** Yadav–Tripathi (fixed α, β, γ)

ters' space has a value of 1.0. In the second case (fixed α, β, γ), dispersion is low—values from 0.933 to 1.0. The variation of the CAI values is a non-monotonic function of the parameters (Fig. 22b) and the value of 1.0 has about 45% of the points. For the last iteration, the Yadav iteration, the dependency on the parameter α of the CAI measure is a very simple function (Fig. 21e). The function is constantly equal to 1.0, so for all values of α all the points in the considered area have converged to the roots.

6.3 Polynomiograph generation time

Generation time [13] is the time it takes to generate the polynomiograph. This time gives us information about the real time of computations, as distinct from the ANI, where we have information about the number of iterations without taking into account the cost of computations in a single iteration.

In all examples, we are searching for the roots of the same polynomial. In order to be able to compare visually the plots in this section, the same interval of time values was used to assign the colours. The ends of the interval were selected as the minimum (0.035) and the maximum (0.42) value of time taken from all the examples. All times are in seconds.

Figure 23 presents the generation times in the parameters' space for the affine combination of three root finding methods. From the plot, we see that there is no simple relationship between the parameters and time. The times vary between 0.048 and 0.320 s. The shortest time is attained at $\alpha_1 = 0.0, \alpha_2 = 0.89, \alpha_3 = 0.11$.

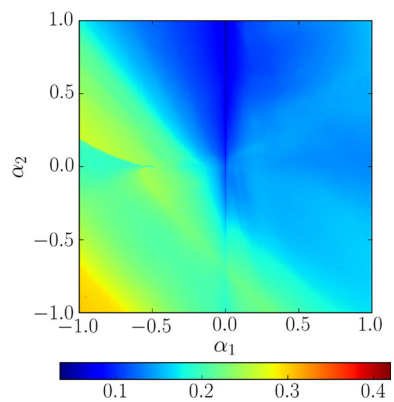


Fig. 23 Generation time (in seconds) in the parameters' space ($\mathbb{R}^2$) for the affine combination of three root finding methods

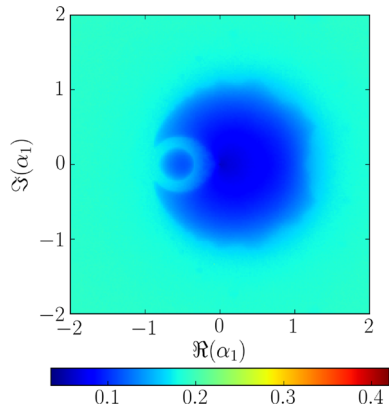


Fig. 24 Generation time (in seconds) in the parameters' space ($\mathbb{C}$) for the affine combination of two root finding methods

This is the same point as the point with the lowest value of ANI. Comparing the shortest time with the times measured for the root finding methods used in the combination, i.e. 0.069 s for the Ezzati–Saleki method, 0.015 s for the Halley method and 0.026 s for the Newton Method, we see that the best time was obtained from the combination of the Ezzati–Saleki method, while the other two methods obtained a shorter time.

The obtained generation times for the affine combination of two root finding methods are presented in Fig. 24. The overall shape of the obtained function is very similar to the functions obtained for the other two measures (ANI, CAI). The shortest time (0.057 s) is attained at $\alpha_1 = 0$, $\alpha_2 = 1$, which corresponds to the second method used in the combination, i.e. the Newton method. Thus, we see that the point with the shortest time does not correspond to the point with the lowest value of ANI, i.e. $\alpha_1 = 0.04 + 0.12\mathbf{i}$, $\alpha_2 = 0.96 - 0.12\mathbf{i}$. The time for the point with the lowest ANI is 0.070 s.

The results obtained for the various iteration processes are presented in Fig. 25. Generation times for the root finding methods used in the iteration processes were the following: 0.069 s for the Ezzati–Saleki method and 0.015 s for the Halley method. The plot for the Das–Debata iteration (Fig. 25a) is very similar to the plots of ANI and CAI. The longest times are obtained for low values of α, whereas the shortest for the high values. The shortest time (0.042) is attained at $\alpha = 1$, $\beta = 0$. At this point, the ANI measure is equal to3.155, so it does not correspond to the point with the lowest value of ANI (1.738). For the Khan–Cho–Abbas iteration (Fig. 25b), again we see a plot

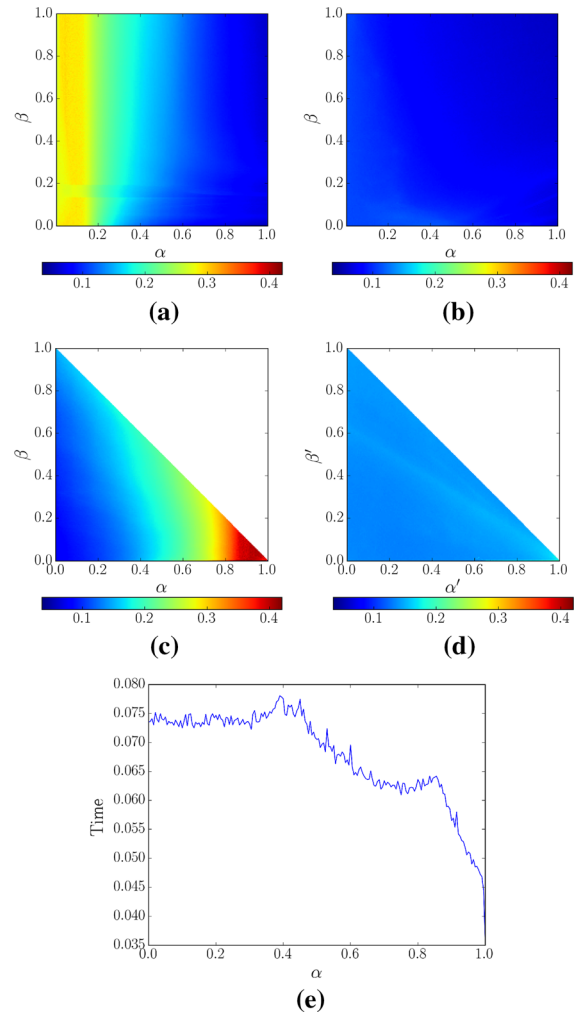


Fig. 25 Generation time (in seconds) in the parameters' space ($[0, 1]^2$ and $[0, 1]$) for various iteration processes. **a** Das–Debata, **b** Khan–Cho–Abbas, **c** Yadav–Tripathi (fixed α', β', γ') **d** Yadav–Tripathi (fixed α, β, γ), **e** Yadav

of almost uniform colour. The variations in the values are shown in Fig. 26a. From the plot, we see that the dependency is different than in the case of ANI and CAI and that it is a very complex function. The shortest time (0.041) is attained at $\alpha = 1$, $\beta = 0$. This point also does not correspond to the lowest value of ANI, because at this point we have ANI equal to 3.155 and the lowest value is equal to 1.738. In the first case of the Yadav–Tripathi iteration (Fig. 25c), the dependency of time on the parameters looks very similar to the one for ANI, but in the second case (Figs. 25d, 26b) we see a different dependency. In Fig. 26b, we see that the variation in the values of time is small, and the variation

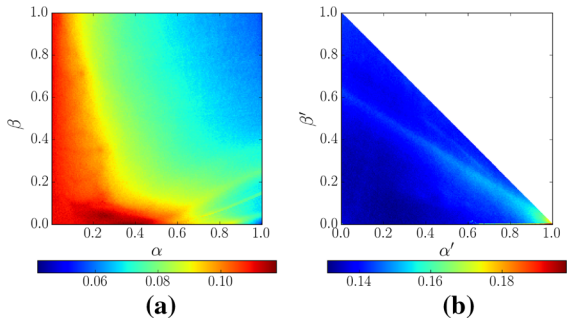


Fig. 26 Generation time (in seconds) in the parameters' space ($[0, 1]^2$) for various iteration processes (the colours are limited to the available values of time). **a** Khan–Cho–Abbas, **b** Yadav–Tripathi (fixed α, β, γ)

Table 3 Generation times (in seconds) of multistep polynomiographs using GPU and CPU implementations

Figures	GPU	CPU	CPU/GPU
13a	0.256	10.235	39.980
13b	0.765	29.499	38.560
13c	3.547	125.474	35.374
14a	3.297	167.778	50.888
14b	4.532	288.217	63.595
14c	8.222	250.494	30.466
14d	11.358	367.619	32.366
14e	3.269	194.876	59.613
14f	2.053	83.311	40.580

is bigger only near the lower-right corner. The shortest times are the following: 0.07 s (at $\alpha = 0$, $\beta = 0.005$, $\gamma = 0.995$) for the first case and 0.131 s (at $\alpha' = 0.22$, $\beta' = 0.045$, $\gamma' = 0.735$) for the second case. Similar to the other iterations, these points do not overlap with the points with the lowest value of ANI. The last plot in Fig. 25 presents the dependence of generation time on the value of parameters α for the Yadav iteration. We clearly see that the value of α has a great impact on time and that the dependency has other characteristics than in the case of ANI. The time varies between 0.035 and 0.078 s, so the shortest time is better than the one obtained for the Ezzati–Saleki method, but worse than the time obtained for the Halley method.

In Table 3, generation times (in seconds) of multistep polynomiographs from Figs. 13 and 14 are gathered. The times for the GPU implementations vary between 0.256 and 11.358 s, whereas for the CPU implementation between 10.235 and 367.619 s. The speed-up obtained with the GPU implementation varies between 30.466 and 63.595 times. Similar speed-up was observed also for the other methods of combining root finding methods (affine combination, various iteration processes).

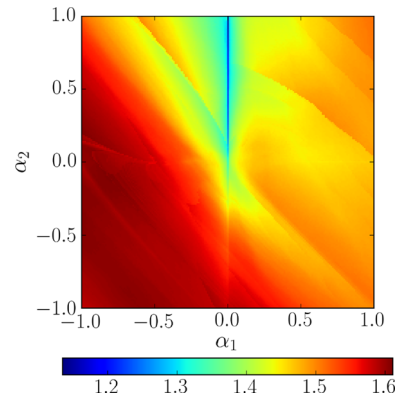


Fig. 27 The fractal dimension in the parameters' space ($\mathbb{R}^2$) for the affine combination of three root finding methods

6.4 Fractal dimension

The fractal dimension (FD) of a set provides an objective means to compare different sets in terms of their complexity [44]. Using FD, we can compare how details of a pattern change with the scale at which it is being measured. In this experiment, we compute the fractal dimension of boundaries of the basins of attraction of the polynomiograph. In order to estimate fractal dimension, we use the standard box-counting method [5].

Fractal dimension in the parameters' space for the affine combination of three root finding methods is presented in Fig. 27. The fractal dimension of boundaries of the basins of attraction of the original methods used in the combination was the following: 1.464 for the Ezzati–Saleki method, 1.135 for the Halley method and 1.324 for the Newton method. From the plot in Fig. 27, we see that the FD is a non-trivial, complex function of the parameters. Thus, we are not able to predict the complexity of the pattern easily. The dispersion of the values is between 1.135 and 1.611, so we obtain patterns with different complexity. The minimal value of FD is attained at $\alpha_1 = 0, \alpha_2 = 1, \alpha_3 = 0$. This point in the parameters' space corresponds to the second root finding method, i.e. the Halley method. The maximal value is attained at $\alpha_1 = -0.85, \alpha_2 = -0.07$,

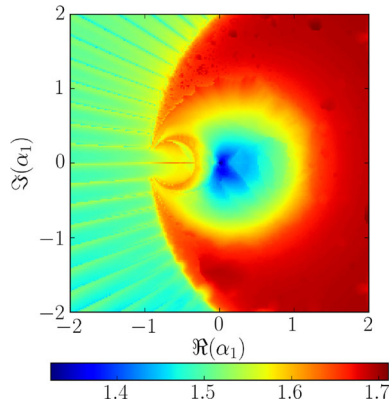


Fig. 28 The fractal dimension in the parameters' space ($\mathbb{C}$) for the affine combination of two root finding methods

$\alpha_3 = 1.92$. Comparing the values of FD obtained for the affine combination to the values of FD of the original methods, we see that with the use of the combination we are able to obtain patterns of higher complexity than that of the original patterns.

The results of computing FD depending on the parameters of the affine combination of two root finding methods are presented in Fig. 28. The obtained approximation of the dependence function is very complex and non-monotonic. Moreover, it is almost symmetrical with respect to the line $\Im(\alpha_1) = 0$. The lowest values of FD are visible in the central part of the plot, attaining the minimum at $\alpha_1 = 0$, $\alpha_2 = 1$. The point with the minimum value of FD corresponds to the Newton method, that was used as the second method in the example. The FD for the Newton method is equal to 1.324, and for the other method used in the combination—the Euler–Chebyshev method—1.562. The maximal value of FD 1.711, that is higher than the values of FD for the methods used in the combination, is attained at $\alpha_1 = 1.3 + 1.8\mathbf{i}$, $\alpha_2 = -0.3 - 1.8\mathbf{i}$. Moreover, from the plot we see that for $\Re(\alpha_1) > 0$ the obtained patterns for most of the points in the parameters' space have a higher complexity than in the case of $\Re(\alpha_1) < 0$.

Figure 29 presents plots of fractal dimension in the parameters' space for various iteration processes. The fractal dimension for the root finding methods that were used in the iteration processes is the following: 1.464 for the Ezzati–Saleki method and 1.135 for the Halley method. For the Das–Debata iteration (Fig. 29a), we see a different relationship than in the case of the previous measures. This time the biggest impact on the

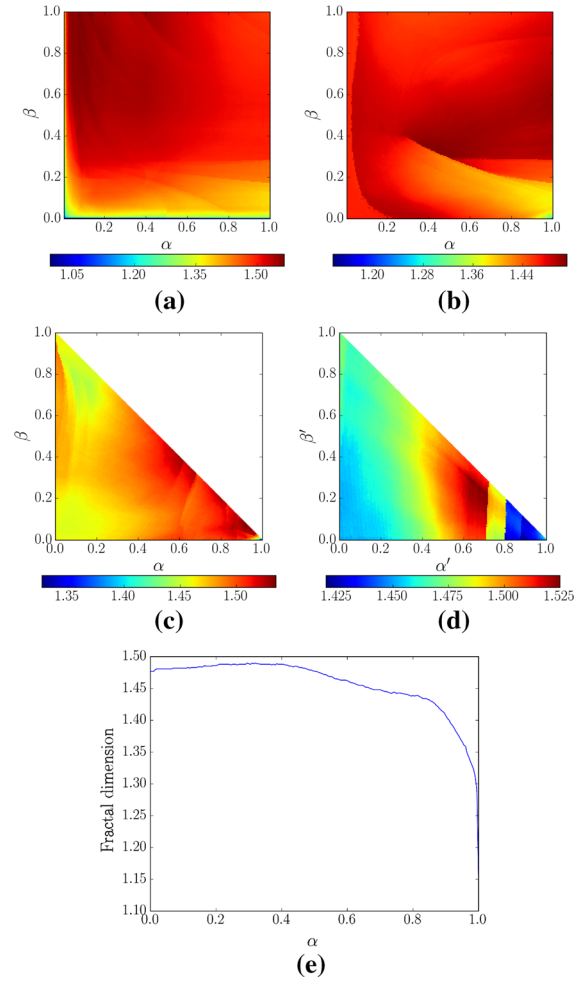


Fig. 29 The fractal dimension in the parameters' space ($[0, 1]^2$ and $[0, 1]$) for various iteration processes. **a** Das–Debata, **b** Khan–Cho–Abbas, **c** Yadav–Tripathi (fixed α', β', γ') **d** Yadav–Tripathi (fixed α, β, γ), **e** Yadav

measure—fractal dimension—is due to the β parameter. For very low values of β, a low value of FD is visible. The minimal value 0.944 is attained at $\alpha = 0.004$, $\beta = 0$. This value of FD is less than the value of FD for both methods used in the iteration. To the contrary, the maximal value 1.566 (attained at $\alpha = 0.079$, $\beta = 0.945$) is greater than the value of FD for the original methods. Thus, using the Das–Debata iteration we are able to obtain patterns with both lower and higher complexity compared to the patterns obtained with the original methods. In Fig. 29b, the plot for the Khan–Cho–Abbas iteration is presented. The dispersion of values of FD is between 1.135 and 1.511. The minimal value of FD is attained at $\alpha = 1$, $\beta = 0$

(lower-right corner of the parameters' space), whereas the maximal value at $\alpha = 0.412, \beta = 0.330$. Similar to the Das–Debata iteration, the relationship between the parameters and the measure is different than in the case of other measures. The function is non-monotonic. In the case of the Yadav–Tripathi iteration (Fig. 29c and d), the dispersion of FD values is different. For the first case, the values vary between 1.327 and 1.535, and for the second case between 1.42 and 1.525. In both cases, the values of FD are greater than for the Halley method (1.135). Moreover, the variations of the values in the parameters' space are different in both cases. In the first case (Fig. 29c), we see low variation and obtain patterns with similar complexity, whereas in the second case (Fig. 29d) we see higher variation which results in obtaining patterns with various complexities. The plot of FD for the last iteration process, the Yadav iteration, is presented in Fig. 29e. From this plot, we can observe that when we want to obtain patterns with higher complexity we must take lower values of α. The minimal value of FD, equal to 1.135, is attained at $\alpha = 1$, whereas the maximal value 1.49 at $\alpha = 0.295$.

6.5 Wada measure

A point P on the basin of attraction boundary is a Wada point if every open neighbourhood of P has a non-empty intersection with at least three different basins [43]. If every boundary point is a Wada point, then the set has a Wada property. In order to check if the set has the Wada property, one can calculate a Wada measure. A Wada measure W for a compact non-empty set $F \subset \mathbb{C}$ is defined as

$$W = \lim_{\varepsilon \to 0} \frac{N_3(\varepsilon)}{N(\varepsilon)}, \tag{27}$$

where $N(\varepsilon)$ is the number of ε-sized boxes that cover F, and $N_3(\varepsilon)$ is the number of ε-sized boxes that cover F and intersect at least 3 different basins of attraction.

In the experiment to approximate the Wada measure of the basins of attraction, we use an algorithm introduced in [44]. This measure indicates what percentage of boundary points is neighbouring more than two basins of attraction simultaneously. This takes values from 0 (region does not resemble a Wada situation) to 1 (region has full Wada property), where 0 corresponds to 0% and 1 to 100%.

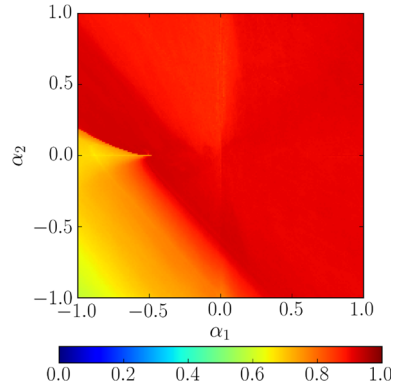


Fig. 30 The Wada measure in the parameters' space ($\mathbb{R}^2$) for the affine combination of three root finding methods

Figure 30 presents the plot of a Wada measure in the parameters' space for the affine combination of three root finding methods. The Wada measure for the methods used in the combination has a high value, i.e. 0.909 for the Ezzati–Saleki method, 0.847 for the Halley method and 0.886 for the Newton method. From the plot, we can observe that using the affine combination we can obtain basins of attraction that have a lower Wada measure than the methods used in the combination. The values of the Wada measure for the affine combination vary between 0.585 and 0.932. Thus, using the affine combination we cannot only decrease the value of the Wada measure, but also increase it. The minimal value of the Wada measure is attained at $\alpha_1 = -0.99, \alpha_2 = -1, \alpha_3 = 2.99$, whereas the maximal value at $\alpha_1 = -0.57, \alpha_2 = 0.01, \alpha_3 = 1.56$.

The Wada measure in the parameters' space for the affine combination of two root finding methods is presented in Fig. 31. From the plot, we see that the Wada measure is a highly non-trivial and non-monotonic function of the parameters. Moreover, we can observe that the obtained function is symmetric with respect to the line $\Im(\alpha_1) = 0$. For $\Re(\alpha_1) > 0$, the Wada measure has a high value with the maximum of 0.965 located at $\alpha_1 = 1.42 - 1.56\mathbf{i}, \alpha_2 = -0.42 + 1.56\mathbf{i}$. The low values are located in the left semi-plane, i.e. for $\Im(\alpha_1) < 0$, and the minimum 0.32 is attained at $\alpha_1 = -2, \alpha_2 = 3$. Comparing the values with the values of the Wada measure for the original methods used in the combination (0.916 for the Euler–Chebyshev method and 0.886 for the Newton method), we see that using the affine combination with complex parameters we are able to reduce

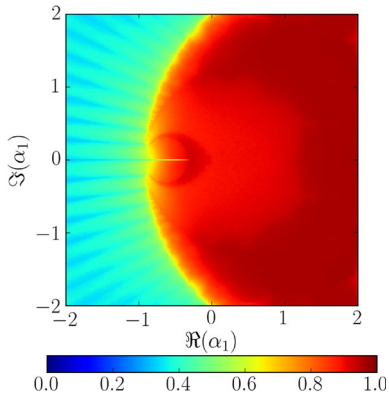


Fig. 31 The Wada measure in the parameters' space ($\mathbb{C}$) for the affine combination of two root finding methods

the Wada property of the basins of attraction significantly.

The results of computing the Wada measure depending on the parameters of various iteration processes are presented in Fig. 32. The values of the Wada measure computed for the methods used in the iterations were the following: 0.909 for the Ezzati–Saleki method and 0.847 for the Halley method. Looking at the plot for the Das–Debata iteration (Fig. 32a), we see that the lowest values of the Wada measure are obtained for very low values of α, attaining the minimum 0.013 (no Wada property) at $\alpha = 0.004$, $\beta = 0$. The value of the measure increases as the value of α increases obtaining the maximum value of 0.902 at $\alpha = 0.975$, $\beta = 0.68$. In the case of the Khan–Cho–Abbas iteration (Fig. 32b), we can observe that for all the values of the parameters the Wada measure is high. The dispersion of the values is between 0.847 and 0.915. The variation of the values is very complex (Fig. 33a) and looks like a noisy image. The minimal value is attained at $\alpha = 1$, $\beta = 0$ (the lower-right corner of the parameters' space), whereas the maximal value at $\alpha = 0.407$, $\beta = 0.02$. For the first case of the Yadav–Tripathi iteration (Fig. 32c), the Wada measure has low values near the lower-right vertex of the parameters' space. The minimal value 0.417, which is significantly less than in the case of the original methods used in the iteration, is attained at $\alpha = 0.99$, $\beta = 0$, $\gamma = 0.01$ ($\alpha' = 0.2$, $\beta' = 0.5$, $\gamma' = 0.3$ are fixed). The high values of the Wada measure are obtained for low values of α, attaining the maximum of 0.911 at $\alpha = 0$, $\beta = 0.985$, $\gamma = 0.015$ ($\alpha' = 0.2$, $\beta' = 0.5$, $\gamma' = 0.3$ are fixed). In the second case of the Yadav–Tripathi iteration (Fig. 32d), the dispersion

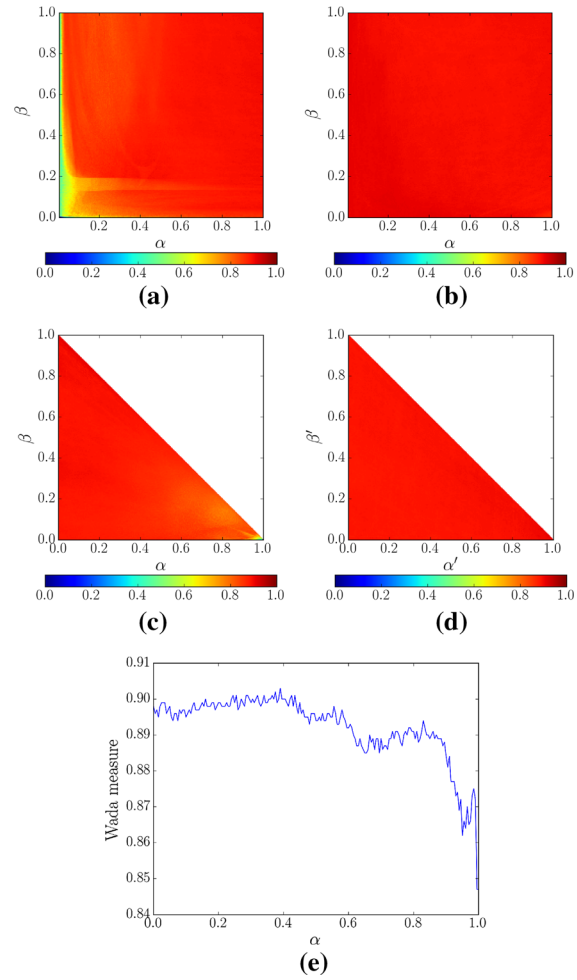


Fig. 32 The Wada measure in the parameters' space ($[0, 1]^2$ and $[0, 1]$) for various iteration processes. **a** Das–Debata, **b** Khan–Cho–Abbas, **c** Yadav–Tripathi (fixed α', β', γ') **d** Yadav–Tripathi (fixed α, β, γ), **e** Yadav

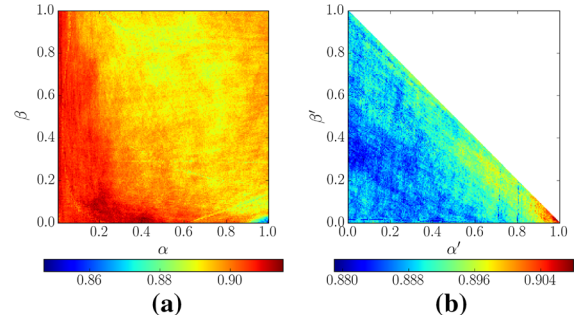


Fig. 33 The Wada measure in the parameters' space ($[0, 1]^2$) for various iteration processes (colours are limited to the available numbers of performed iterations). **a** Khan–Cho–Abbas, **b** Yadav–Tripathi (fixed α, β, γ)

of the values is low. The values are between 0.879 and 0.908, so they lie between the values of the Wada measure obtained for the methods used in the iteration. The variation of the values, similar to the case of the Khan–Cho–Abbas iteration, is a complex function (Fig. 33b). The plot for the Yadav iteration (Fig. 32e) shows that the Wada measure significantly depends on the α parameter. The values of the measure (minimum 0.847, maximum 0.903) lie between the values obtained for the methods used in the iteration.

7 Conclusions

In this paper, we presented the concept of the use of combinations of different polynomial root finding methods in the generation of fractal patterns. We used three different combination methods: (1) affine, s-convex combination, (2) iteration processes from fixed point theory, (3) multistep polynomiography. The use of combined root finding methods gives us more possibilities to obtain new and very interesting fractal patterns. Moreover, we numerically studied the proposed methods using five different measures. The obtained results showed that in most cases the dependence of the considered measure on the methods' parameters is a non-trivial, non-monotonic function. The results also showed that using the combined root finding methods we are able to obtain faster convergence, measured in iterations, than using the methods of the combination separately.

When we search for an interesting fractal pattern using polynomiography, we must make the right choice of polynomial, iteration process, convergence test etc., and using trial and error, we must find an interesting area [21]. Adding the possibility of using different combination methods of the root finding methods makes the choice even more difficult, so there is a need for an automatic or semi-automatic method that will find interesting fractal patterns. The notion of an interesting pattern is very difficult to define, because of the subjectivity of what is interesting. Nevertheless, there are some attempts to estimate this notion. For instance, in [3] Ashlock and Jamieson introduced a method of exploring Mandelbrot and Julia sets using evolutionary algorithms with different fitness functions. In future research, we will attempt to develop a method similar to Ashlock's that will search for interesting patterns in polynomiography.

Acknowledgements The author would like to thank the anonymous referees for their careful reading of the manuscript and their fruitful comments and suggestions that helped improve this paper.

References

1. Ardelean, G., Balog, L.: A qualitative study of Agarwal et al. iteration. Creat. Math. Inform. **25**(2), 135–139 (2016)
2. Ardelean, G., Cosma, O., Balog, L.: A comparison of some fixed point iteration procedures by using the basins of attraction. Carpathian J. Math. **32**(3), 277–284 (2016)
3. Ashlock, D., Jamieson, B.: Evolutionary exploration of complex fractals. In: Hingston, P., Barone, L., Michalewicz, Z. (eds.) Design by Evolution, Natural Computing Series, pp. 121–143. Springer, Berlin (2008). doi:10.1007/978-3-540-74111-4_8
4. Bailey, M., Cunningham, S.: Graphics Shaders: Theory and Practice, 2nd edn. CRC Press, Boca Raton (2012)
5. Barnsley, M.: Fractals Everywhere, 2nd edn. Academic Press, Boston (1993)
6. Barrallo, J., Sanchez, S.: Fractals and multi-layer coloring algorithm. In: Sarhangi, R., Jablan, S. (eds.) Bridges: Mathematical Connections in Art, Music, and Science, pp. 89–94. Bridges Conference, Southwestern College, Winfield (2001)
7. Carlson, P.: Two artistic orbit trap rendering methods for Newton M-set fractals. Comput. Graph. **23**(6), 925–931 (1999). doi:10.1016/S0097-8493(99)00123-5
8. Das, G., Debata, J.: Fixed points of quasinonexpansive mappings. Indian J. Pure Appl. Math. **17**(11), 1263–1269 (1986)
9. Ezzati, R., Saleki, F.: On the construction of new iterative methods with fourth-order convergence by combining previous methods. Int. Math. Forum **6**(27), 1319–1326 (2011)
10. Gdawiec, K.: Polynomiography and various convergence tests. In: Skala, V. (ed.) WSCG 2013 Communication Papers Proceedings, pp. 15–20. Vaclav Skala—Union Agency, Plzen, Czech Republic (2013)
11. Gdawiec, K.: Perturbation mappings in polynomiography. In: Gruca, A., Brachman, A., Kozielski, S., Czachórski, T. (eds.) Man–Machine Interactions 4, Advances in Intelligent Systems and Computing, vol. 391, pp. 499–506. Springer, Berlin (2016). doi:10.1007/978-3-319-23437-3_42
12. Gdawiec, K.: Switching processes in polynomiography. Nonlinear Dyn. **87**(4), 2235–2249 (2017). doi:10.1007/s11071-016-3186-2
13. Gdawiec, K., Kotarski, W., Lisowska, A. Polynomiography based on the non-standard Newton-like root finding methods. Abstr. Appl. Anal. (2015). doi:10.1155/2015/797594
14. Gilbert, W.: Newton's method for multiple roots. Comput. Graph. **18**(2), 227–229 (1994). doi:10.1016/0097-8493(94)90097-3

15. Householder, A.: The Numerical Treatment of a Single Nonlinear Equation. McGraw-Hill, New York (1970)
16. Janke, S.: Mathematical Structures for Computer Graphics. Wiley, Hoboken (2015)
17. Kalantari, B.: Polynomiography and applications in art, education, and science. In: Proceeding of ACM SIGGRAPH 2003 Educators Program, pp. 1–5 (2003). doi:10.1145/965106.965108
18. Kalantari, B.: A new visual art medium: polynomiography. ACM SIGGRAPH Comput. Graph. Q. **38**(3), 21–23 (2004). doi:10.1145/1015999.1016002
19. Kalantari, B.: Polynomiography and applications in art, education and science. Comput. Graph. **28**(3), 417–430 (2004). doi:10.1016/j.cag.2004.03.009
20. Kalantari, B.: Two and three-dimensional art inspired by polynomiography. In: Sarhangi, R., Moody, R. (eds.) Renaissance Banff: Mathematics, Music, Art, Culture, pp. 321–328. Bridges Conference, Southwestern College, Winfield (2005)
21. Kalantari, B.: Polynomial Root-Finding and Polynomiography. World Scientific, Singapore (2009)
22. Kalantari, B., Kalantari, I., Andreev, F.: Animation of mathematical concepts using polynomiography. In: Proceeding of ACM SIGGRAPH 2004 Educators Program, p. 27 (2004). doi:10.1145/1186107.1186138
23. Kang, S., Alsulami, H., Rafiq, A., Shahid, A.: *S*-iteration scheme and polynomiography. J. Nonlinear Sci. Appl. **8**(5), 617–627 (2015)
24. Kelley, A.: Layering techniques in fractal art. Comput. Graph. **24**(4), 611–616 (2000). doi:10.1016/S0097-8493(00)00062-5
25. Khan, A., Domlo, A.A., Fukhar-ud din, H.: Common fixed points Noor iteration for a finite family of asymptotically quasi-nonexpansive mappings in Banach spaces. J. Math. Anal. Appl. **341**(1), 1–11 (2008). doi:10.1016/j.jmaa.2007.06.051
26. Khan, S., Cho, Y., Abbas, M.: Convergence to common fixed points by a modified iteration process. J. Appl. Math. Comput. **35**(1), 607–616 (2011). doi:10.1007/s12190-010-0381-z
27. Kotarski, W., Gdawiec, K., Lisowska, A.: Polynomiography via Ishikawa and Mann iterations. In: Bebis, G., Boyle, R., Parvin, B., Koracin, D., Fowlkes, C., Wang, S., Choi, M.H., Mantler, S., Schulze, J., Acevedo, D., Mueller, K., Papka, M. (eds.) Advances in Visual Computing. Lecture Notes in Computer Science, vol. 7431, pp. 305–313. Springer, Berlin (2012). doi:10.1007/978-3-642-33179-4_30
28. Liu, X.D., Zhang, J.H., Li, Z.J., Zhang, J.X.: Generalized secant methods and their fractal patterns. Fractals **17**(2), 211–215 (2009). doi:10.1142/S0218348X09004387
29. Nikiel, S.: Iterated Function Systems for Real-Time Image Synthesis. Springer, London (2007)
30. Pickover, C. (ed.): Chaos and Fractals: A Computer Graphical Journey. Elsevier, Amsterdam (1998)
31. Pinheiro, M.: *s*-convexity—foundations for analysis. Differ. Geom. Dyn. Syst. **10**, 257–262 (2008)
32. Rafiq, A., Tanveer, M., Nazeer, W., Kang, S.: Polynomiography via modified Jungck, modified Jungck Mann and modified Jungck Ishikawa iteration scheme. PanAm. Math. J. **24**(4), 66–95 (2014)
33. Sellers, G., Wright Jr., R., Haemel, N.: OpenGL SuperBible, 7th edn. Addison-Wesley, Boston (2015)
34. Sisson, P.: Fractal art using variations on escape time algorithm in the complex plane. J. Math. Arts **1**(1), 41–45 (2007). doi:10.1080/17513470701210485
35. Spencer, S.: Creating and modifying images using Newton's method for solving equations. In: Hart, G., Sarhangi, R. (eds.) Proceedings of Bridges 2010: Mathematics, Music, Art, Architecture, Culture. pp. 183–190. Tessellations Publishing, Phoenix, Arizona (2010)
36. Szyszkowicz, M.: Computer art generated by the method of secants in the complex plane. Comput. Graph. **14**(3–4), 509 (1990). doi:10.1016/0097-8493(90)90074-8
37. Szyszkowicz, M.: Computer art from numerical methods. Comput. Graph. Forum **10**(3), 255–259 (1991). doi:10.1111/1467-8659.1030255
38. Szyszkowicz, M.: A survey of several root-finding methods in the complex plane. Comput. Graph. Forum **10**(2), 141–144 (1991). doi:10.1111/1467-8659.1020141
39. Traub, J.: Iterative Methods for the Solution of Equations. Chelsea Publishing Company, New York (1977)
40. Yadav, M.: Common fixed points by two step iterative scheme for asymptotically nonexpansive mappings. Funct. Anal. Approx. Comput. **7**(1), 47–55 (2015)
41. Yadav, M., Tripathi, B.: Iteration scheme for common fixed points of two asymptotically nonexpansive mappings. Int. J. Pure Appl. Math. **80**(4), 501–514 (2012)
42. Ye, R.: Another choice for orbit traps to generate artistic fractal images. Comput. Graph. **26**(4), 629–633 (2002). doi:10.1016/S0097-8493(02)00096-1
43. Zhang, Y., Luo, G.: Unpredictability of the Wada property in the parameter plane. Phys. Lett. A **376**(45), 3060–3066 (2012). doi:10.1016/j.physleta.2012.08.015
44. Ziaukas, P., Ragulskis, M.: Fractal dimension and Wada measure revisited: no straightforward relationships in NDDS. Nonlinear Dyn. **88**(2), 871–882 (2017). doi:10.1007/s11071-016-3281-4