INFORMATICA, 2026, Vol. 0, No. 0, 1–27

© 2026 *Vilnius University*
DOI: https://doi.org/10.15388/26-INFOR642

Visualization of Mandelbulbs Associated with the Zeta Functions

Lukas KUZMA*, Igoris BELOVAS, Martynas SABALIAUSKAS

Institute of Data Science and Digital Technologies, Faculty of Mathematics and Informatics, Vilnius University, Akedemijos 4, Vilnius, LT-04812, Lithuania
e-mail: lukas.kuzma@mif.vu.lt, igoris.belovas@mif.vu.lt, martynas.sabaliauskas@mif.vu.lt

Received: March 2026; accepted: July 2026

Abstract. This research builds upon our previous work on efficient algorithms for calculating and visualizing fractal structures associated with zeta functions, shifting the focus from two-dimensional to three-dimensional structures. Specifically, we introduce Zetabulbs – a class of Mandelbulbs (three-dimensional analogues of Mandelbrot sets) derived from the zeta functions. The visualization of Mandelbulbs poses substantial computational and algorithmic challenges, while also carrying practical relevance in areas such as digital visual effects, where Mandelbulb-based structures have been used in the production of high-budget films. By introducing Zetabulbs, we expand the landscape of zeta-function–driven fractal geometry and open new opportunities for its application in both artistic visualization and scientific exploration.

Key words: fractal structures, numerical algorithms, zeta functions, visualizations, Zetabulbs.

1. Introduction

Since their introduction, Mandelbrot sets have been studied extensively and have captivated both mathematicians and fractal enthusiasts. It was therefore natural to seek a three-dimensional analogue of the Mandelbrot set. A direct three-dimensional counterpart, however, cannot exist in a canonical sense, as there is no true three-dimensional analogue of the complex plane. To overcome this limitation, researchers developed alternative structures based on spherical coordinates, enabling the visualization of a three-dimensional representation of Mandelbrot-type dynamics. These structures became known as Mandelbulbs (cf. Ugwu *et al.*, 2024; Kumar and Bhaskar, 2025).

Beyond their mathematical appeal, Mandelbulbs have demonstrated practical relevance. They have been employed in the creation of digital visual effects for high-budget films, illustrating that these structures are not merely objects of mathematical aesthetics, but also tools with concrete applications in computational art and media.

Recently, we have further developed (see Belovas *et al.*, 2022) an efficient algorithm originally proposed by Borwein (2000) for the computation of the Riemann zeta function.

*Corresponding author.

This algorithm is particularly well suited for the evaluation of multiple zeta values (cf. Algorithm 1 in Belovas *et al.* (2022)). The Mandelbrot set associated with the Riemann zeta function exhibits intricate and highly nontrivial structures in the neighbourhoods of its pole and zeros (see Belovas *et al.*, 2022). These observations naturally lead to the problem of a three-dimensional analogues, namely, Mandelbulbs associated with zeta functions.

How do the resulting fractal structures behave in neighbourhoods of poles and nontrivial zeros of zeta functions? Do such structures possess mathematical and practical significance? The present study aims to advance our understanding of these questions. Specifically, we seek to construct and investigate Mandelbulbs associated with the Riemann zeta function and Dirichlet L-functions, which we refer to as *Zetabulbs*. To achieve this objective, the following tasks must be undertaken:

- implementation of GPU-based (Graphics Processing Unit) versions of efficient algorithms for the computation of zeta functions (modified Borwein's and Šleževičienė's algorithms);
- development of efficient algorithms for the real-time generation and rendering of Mandelbulb images;
- construction of Mandelbulb dynamics derived from zeta functions;
- visualization of Zetabulbs and exploration of newly received fractal structures.

This paper can be divided into three main parts: algorithms for the calculation of zeta functions (Sections 2–3), algorithm parallelization (Section 4), and algorithm application for visualization of Zetabulbs (Sections 5–6). Each part contains its own methodology and results.

2. Calculation of the Riemann Zeta Function

The Riemann zeta-function is a function of a complex variable $s = \sigma + it$ defined by the ordinary Dirichlet series or the Euler product formula,

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} = \prod_{p \text{ prime}} \left(1 - \frac{1}{p^s}\right)^{-1}, \quad \Re s > 1,$$

and by analytic continuation elsewhere except for a simple pole at $s = 1$. It satisfies the functional equation

$$\zeta(s) = 2^s \pi^{s-1} \sin(\pi s/2) \Gamma(1-s) \zeta(1-s),$$

implying that $\zeta(s)$ has simple zeros at $s = -2n$, $n \in \mathbb{N}$, known as the trivial zeros. The Riemann hypothesis states that all other zeroes, called non-trivial, lie on the critical line $\Re s = 1/2$.

To investigate the Riemann zeta function numerically, one needs an algorithm enabling him to effectively compute multiple values over the complex plane. Moreover, the algorithm must be well-suited for parallel CPU (Central Processing Unit) and GPU computations. We have employed modifications of Borwein's efficient algorithm. The idea of the

method comes from the alternating series convergence and properties of Chebyshev polynomials (see Borwein, 2000). It is important to note that the algorithm is nearly optimal in the sense that there is no sequence of n-term exponential polynomials that can converge to $\zeta(s)$ very much faster than those of the algorithm (cf. Theorem 3.1 in Borwein, 2000).

2.1. *MB-Algorithm*

The modified Borwein's algorithm (*MB*-algorithm) applies to complex numbers with $\Re s \geqslant 1/2$ and arbitrary $\Im s$. In this modification (see Belovas *et al.*, 2022), the Riemann zeta function is represented by the alternating series

$$\zeta(s) = \frac{1}{1-2^{1-s}} \sum_{k=0}^{n-1} \frac{(-1)^k \psi_{n,k}}{(k+1)^s} + R_n(s). \tag{1}$$

Here

$$\psi_{n,k} = 1 - \frac{H_k}{H_n}$$

and

$$l_{\max} = \arg \max_{0 \leqslant k \leqslant n} \frac{(n+k-1)!4^k}{(n-k)!(2k)!}, \quad n \in \mathbb{N},\ 0 \leqslant k \leqslant n,$$

while

$$\begin{aligned}
H_l &= H_{l-1} + \exp\bigl(T_l - T_{l_{\max}} + (l - l_{\max})\log 4\bigr),\\
H_0 &= \exp(T_0 - T_{l_{\max}} - l_{\max}\log 4),\\
T_l &= T_{l-1} + \log\frac{(n-l+1)(n+l-1)}{(2l-1)(2l)},\\
T_0 &= -\log n, \quad 1 \leqslant l \leqslant n.
\end{aligned}$$

The error term and the computational complexity are closely linked to the problem of selecting the minimal number of terms in the series (1). Theorem 1 in Belovas *et al.* (2022) states that the series (1) to compute the Riemann zeta-function with d decimal digits of accuracy require a number of terms

$$n = \left\lceil \frac{\pi t}{2\log(3+\sqrt{8})} + \frac{\log 10}{\log(3+\sqrt{8})} d + \frac{\log\frac{2}{1-2^{-\varepsilon}}}{\log(3+\sqrt{8})} \right\rceil, \tag{2}$$

here $|s - s_k| \geqslant \varepsilon > 0$ and $s_k = 1 + 2i\pi k/\log 2$.

2.2. *NA-Algorithm*

The limit theorem for the coefficients of *MB*-series enables us to derive the following normal approximation for the coefficients $\psi_{n,k}$ of the series (1) (cf. Belovas *et al.*, 2022)

$$\psi_{n,k} = 1 - \Phi\left(\frac{k-\mu_n}{\sigma_n}\right) + O\left(\frac{1}{\sqrt{n}}\right), \quad \mu_n = \frac{n}{\sqrt{2}}, \quad \sigma_n = \frac{\sqrt{n}}{\sqrt[4]{32}}.$$

Here $\Phi(x)$ is the cumulative distribution function of the standard normal distribution. This result allows us to select the number of terms k_0 in the series (1),

$$k_0 = \lceil \mu_n + z\sigma_n \rceil, \quad z = \Phi^{-1}\left(1 - 10^{-d}\right), \; n > 10^3. \tag{3}$$

Note that for fixed $\Re s$ and d, we have $k_0 \sim 0.63\Im s$.

2.3. *EMB-Algorithm*

Kuzma (Belovas *et al.*, 2022) proposed the following empirical (*EMB*) formula for the minimal number of terms n in the series (1) for calculations with d decimal digits of accuracy

$$\begin{aligned} &n = \lceil at + b\sqrt{t} + c \rceil, \quad a = 0.451, \\ &b = 1.407\sqrt{d} - 0.245, \quad c = 0.371d + 0.195. \end{aligned} \tag{4}$$

The empirical formula (4) has been verified for $0 \leqslant t \leqslant 10^4$.

3. Calculation of the Dirichlet L-Functions

Let $s \in \mathbb{C}$ and $\Re s > 1$, then Dirichlet L-series are defined as follows

$$L(s, \chi) = \sum_{n=1}^{\infty} \frac{\chi(n)}{n^s}, \quad \Re s > 1.$$

Here χ is a Dirichlet character modulo q. By analytic continuation, this function extends to a meromorphic function on the whole complex plane and is known as a Dirichlet L-function. The Dirichlet L-function is a generalization of the Riemann zeta function and thus has similar properties. If χ is principal, the corresponding Dirichlet L-function has a simple pole at $s = 1$; otherwise, it is holomorphic on the whole complex plane. The Generalized Riemann hypothesis asserts that all nontrivial zeros of Dirichlet L-function lie on the critical line $\Re s = 1/2$.

Šleževičiene (2004), building upon Borwein's technique, proposed several algorithms (MS-algorithms) for efficient computing of Dirichlet L-functions.The following two results (Propositions 1 and 2), established in Šleževičiene (2004) (see Theorems 4 and 5,

respectively), yield algorithms for computing Dirichlet L-functions attached to a character with odd modulus.

Proposition 1. *Let q be odd, χ mod q be a non-principal character and let*

$$p_n(x) = \sum_{k=0}^{n} a_{n,k} x^{kq}$$

be an arbitrary polynomial of degree nq which does not vanish at -1. Further, let

$$c_{n,j} = (-1)^j \left(\sum_{k=0}^{j} (-1)^k a_{n,k} - p_n(-1) \right).$$

Then for $\sigma > 0$,

$$L(s, \chi) = \frac{-1}{p_n(-1)(1 - \chi(2)2^{1-s})} \sum_{l=1}^{q} \chi(l)(-1)^{l-1} \sum_{j=0}^{n-1} \frac{c_{n,j}}{(l+qj)^s} + \xi_n(s), \tag{5}$$

where

$$\xi_n(s) = \frac{-1}{p_n(-1)\Gamma(s)(1 - \chi(2)2^{1-s})} \int_0^1 \frac{p_n(s)|\log x|^{s-1}}{1 + x^q} \sum_{l=1}^{q} \chi(l) x^{l-1} dx.$$

Proposition 2. *Let q be odd. For $\sigma > 0$,*

$$L(s, \chi) = \frac{1}{2^n(1 - \chi(2)2^{1-s})} \sum_{a=1}^{q} (-1)^{a-1} \chi(a) \sum_{j=0}^{2n-1} \frac{e_j}{(a+qj)^s} + \gamma_n(s), \tag{6}$$

where

$$e_j = (-1)^j \left(\sum_{k=0}^{j-n} \binom{n}{k} - 2^n \right).$$

If $\varphi(m)$ stands for the Euler totient function, then

$$\left|\gamma_n(s)\right| \leqslant \frac{\varphi(q)}{8^n} \frac{(1 + |t|/\sigma)\exp((|t|\pi)/2)}{|1 - \chi(2)2^{1-s}|}.$$

In the next section, we introduce a modification of Šleževičienė's efficient algorithm for Dirichlet L-functions that applies a normal approximation to the coefficients of the corresponding series.

3.1. *NA-Modification of MS-Algorithm for Dirichlet L-Functions*

Associating polynomials $p_n(x)$ in Proposition 1 with Chebyshev polynomials

$$\begin{aligned} T_n(x) &= (-1)^n \cos(2n \arcsin(\sqrt{x}) \\ &= (-1)^n n \sum_{k=0}^{n} (-1)^k \frac{(n+k-1)!}{(n-k)!(2k)!} 4^k x^k, \\ p_n(x) &= \sum_{k=0}^{n} \underbrace{(-1)^{n+k} n \frac{(n+k-1)!}{(n-k)!(2k)!} 4^k}_{=a_{n,k}} x^{qk}, \end{aligned}$$

we get we following series for Dirichlet L-functions,

$$L(s,\chi) = \frac{-1}{1-\chi(2)2^{1-s}} \sum_{l=1}^{q} \chi(l)(-1)^l \sum_{j=0}^{n-1} \frac{(-1)^j c_{n,j}}{(l+qj)^s} + \xi_n(s), \tag{7}$$

where

$$c_{n,j} = 1 - \frac{\sum_{k=0}^{j} (-1)^k a_{n,k}}{p_n(-1)} = 1 - \frac{\sum_{k=0}^{j} \alpha_{n,k}}{\sum_{k=0}^{n} \alpha_{n,k}},$$

and

$$\alpha_{n,k} = \frac{(n+k-1)!}{(n-k)!(2k)!} 4^k.$$

Note that Borwein (2000) recommended similar approach for the calculation of the Riemann zeta function to make the error as small as possible. Indeed, Chebyshev polynomials $T_n(x)$, shifted to $[0, 1]$ and appropriately normalized maximization of the value $p_n(-1)$ over all polynomials of comparable supremum norm on $[0, 1]$.

Let Ω_n be a integral random variable associated with Chebyshev polynomials,

$$P(\Omega_n = k) = \frac{\alpha_{n,k}}{\sum_{j=0}^{n} \alpha_{n,j}}, \quad 0 \leqslant k \leqslant n.$$

Suppose that $F_n(x)$ is the cumulative distribution function of the random variable Ω_n. It has been proved (cf. Theorem 3.1 in Belovas and Sakalauskas, 2018), that the cumulative distribution function is

$$F_n(x) = \Phi\left(\frac{x-\mu_n}{\sigma_n}\right) + O\left(\frac{1}{\sqrt{n}}\right).$$

By Lemma 2.2 of Belovas and Sakalauskas (2018), the mean and variance are given by

$$\mu_n = E(\Omega_n) = \frac{n}{\sqrt{2}} + \frac{33}{144\sqrt{2}}\frac{1}{n} + O\left(\frac{1}{n^2}\right),$$
$$(\sigma_n)^2 = Var(\Omega_n) = \frac{\sqrt{2}}{8}n + O\left(\frac{1}{n^2}\right).$$

We now can formulate a result establishing a normal approximation for the coefficients of the series associated with Dirichlet L-functions.

Proposition 3. *The following the Gaussian approximation for $c_{n,j}$ coefficients holds*:

$$c_{n,j} = 1 - \Phi\left(\frac{j + 1/2 - \mu_n}{\sigma_n}\right) + O\left(\frac{1}{\sqrt{n}}\right). \tag{8}$$

Note that $1/2$ *is the continuity correction, and*

$$\sigma_n = \frac{\sqrt{n}}{2\sqrt[4]{2}} + O\left(\frac{1}{n^{5/2}}\right).$$

Proof. The proposition is an immediate consequence of the proof of Theorem 2 in Belovas *et al.* (2022). Besides,

$$\sigma_n = \sqrt{\frac{\sqrt{2}}{8}n + O\left(\frac{1}{n^2}\right)} = \sqrt{\frac{\sqrt{2}}{8}n} \cdot \sqrt{1 + O\left(\frac{1}{n^3}\right)} = \frac{\sqrt{n}}{2^{5/4}} + O\left(\frac{1}{n^{5/2}}\right). \quad \square$$

4. Parallel Algorithms for Calculation of the Riemann Zeta Function

Two approaches to parallelize the *MB*-algorithm can be used: optimize for specific or multiple value computation speed. Each pixel of an image of a Zetabulb will require multiple computations of the Riemann zeta function. We aim to generate at least 24 frames of HD: 720p images (a total of $720 \times 1280 = 921600$ pixels) per second. For this purpose, optimizing for multiple-value computation speed is more beneficial.

Personal computers usually have two processing units: a central processing unit (CPU) and a graphics processing unit (GPU). Both processing units can have multiple cores; however, they are specialized for different tasks. CPUs usually have a low amount (up to 24) of high-speed (up to 6GHz) cores and are used for tasks that are hard to parallelize. Whereas, GPUs usually have a high amount (up to 16000) of slower (up to 2.5GHz) cores and are used for tasks that are easy to parallelize. CUDA NVIDIA (2026) is a parallel computing platform that allows one to perform computations on both the CPU and the GPU and manages the data transfer operations between RAM (Random Access Memory) and VRAM (Video Random Access Memory). This platform is the optimal approach for real-time image generation.

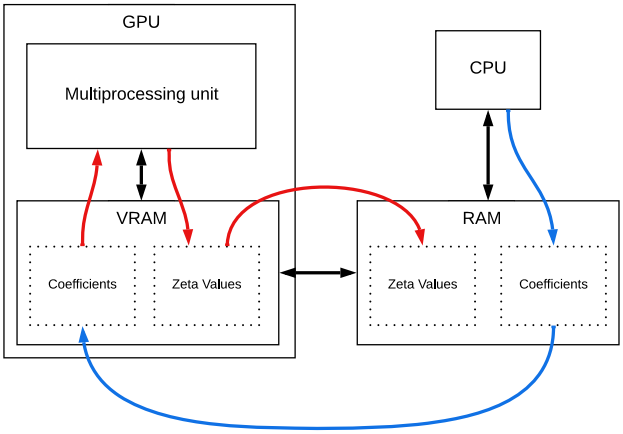


Fig. 1. Data transfer diagram for the *MB*-method. The blue arrows indicate the initialization part's data transfer flow. The red arrows indicate the computation part's data transfer flow.

The *MB*-method can be separated into two logical parts: initialization and computation. During the first part, we compute n coefficients ψ required to compute the Riemann zeta function with the maximum imaginary part $t_{\max}$ and d decimal digits of accuracy. During the second part, we use these coefficients to compute the actual value of the Riemann zeta function. Note that although both parts are linear, the coefficients computed during the initialization part can be reused when computing Riemann zeta values for arguments $t < t_{\max}$. This property of the *MB*-method makes it suitable for computing multiple values at once.

A data transfer diagram for a CUDA implementation of the *MB*-method can be seen in Fig. 1. Here, the data flow of the initialization part is indicated by blue arrows and the computation part by red arrows. During the initialization part, the coefficients ψ are computed using a CPU. After that, they are transferred from RAM to VRAM for the remainder of the program's runtime. Note that it might be more efficient to perform the initialization part on the GPU if the CPU clock speed is low. Then, whenever the CPU requests a Riemann zeta value to be computed, the GPU uses ψ to get the value and transfers it from VRAM back to RAM. An implementation of this algorithm can be found in Kuzma (2025).

4.1. *Numerical Experiments*

Modern GPUs perform arithmetic instructions much faster when using single-precision (*float*) instead of double-precision (*double*) type variables NVIDIA (2023). If the precision using *float* type variables is good enough (i.e. there is no visible difference between images generated using *float* and *double* type variables), this precision for performance trade-off is preferred. However, the *EMB*-algorithm requires to compute a large sum of values very close to zero. This summation will compound errors that occur when using single-precision variables and produce a result that is not good enough for visualization.

Table 1
First experiment results. AE stands for average root mean square error between *float* and *double* frames, ME stands for maximum square error between the *float* and *double* frames, τ_{double} – average time to compute a frame using the *double* algorithm in milliseconds, τ_{float} – average time to compute a frame using the *float* algorithm.

p	AE	ME	τ_{double}	τ_{float}
2	$2.43 \cdot 10^{-5}$	$9.25 \cdot 10^{-3}$	201.4	9.7
3	$9.38 \cdot 10^{-4}$	$8.35 \cdot 10^{-2}$	32729.3	1459.4

In this Subsection, we aim to determine the domain within which the *EMB*-algorithm struggles to produce acceptable results. The experiments have been performed on GeForce GTX 1060 (Mobile) graphics processor with 6 GB VRAM and Intel® Core™ i7-8750H 2.2 GHz (boosted to 4.0 GHz) processor with 16 GB DDR4 RAM.

4.1.1. *First Experiment*
The first experiment aims to compare the speed and accuracy of the *float* and *double* algorithms. For this experiment, we divide the complex plane into $n_f = 1000$ matrices (frames) consisting of 1280×720 uniformly located points

$$F_{p,j} = \underbrace{(0.5, 1.5)}_{\sigma} \times \underbrace{\left(k_p + jh_f, k_p + (j+1)h_f\right)}_{t}, \tag{9}$$

where $k_p = 10^{2(p-1)}$ is the minimum imaginary value of a frame, $h_f = \frac{k_{p+1} - k_p}{n_f}$ is the frame height and $0 < j \leqslant n_f$. We compute the Riemann zeta function values once with an algorithm that uses *double* precision variables and once with an algorithm that uses *float* precision variables. We find the average mean square error and the maximum mean square error of the computed values. Using both methods, we also compute the average times to compute a frame. The results of this experiment are shown in Table 1.

4.1.2. *Second Experiment*
In the second experiment, we aim to evaluate the *float* algorithm's precision on the critical line. We aim to find the maximum value $t_{\min}$ with which the *float* algorithm no longer produces d decimal digits of accuracy. In Belovas *et al.* (2022), we have shown that the most computationally expensive region of the complex plane (excluding the special points $s_k = 1 + 2\pi ki/\log 2$) for the *MB*-algorithm is the critical line.

We compute the Riemann zeta function values using both the *float* and the *double* algorithms and compute the error for each of the obtained values $E = |\zeta(F_{k,j})_{float} - \zeta(F_{k,j})_{double}|$. Then, we find the first value t with which the *float* algorithm no longer provides d decimal digits of accuracy. Here

$$\begin{aligned} F_{k,j} &= \frac{1}{2} + \underbrace{\frac{2\pi(k + j/10)}{\log 2}}_{\in[0;10^5]} i, \\ 0 \leqslant k \leqslant 1120, &\quad 0 \leqslant j \leqslant 9,\ k, j \in \mathbb{N}. \end{aligned} \tag{10}$$

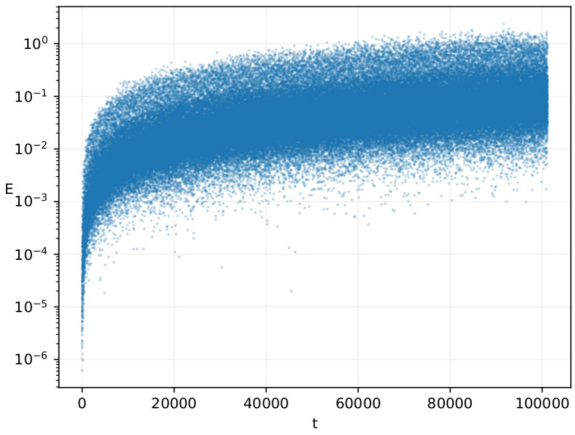


Fig. 2. Second experiment results. Each dot represents the error E at point $F_{k,j}$ (10).

Table 2
Third experiment results. Here t is the largest imaginary part of the complex number $0.5 + it$ that produces d decimal digits of accuracy.

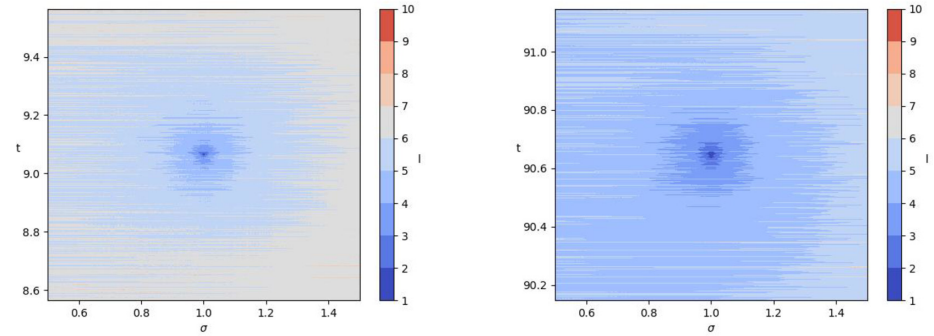
d	1	2	3	4	5
t	7433.07	652.66	72.52	18.29	Not available

The experiment results are presented in Fig. 2. As we can see, the error E increases with t. In Table 2, we have listed the largest imaginary part of the complex number $1/2+it$ that produces d decimal digits of accuracy. For example, we can expect that every t value lower than 72.52 will be computed with at least 3 decimal digits of accuracy using the *float* algorithm.

4.1.3. *Third Experiment*

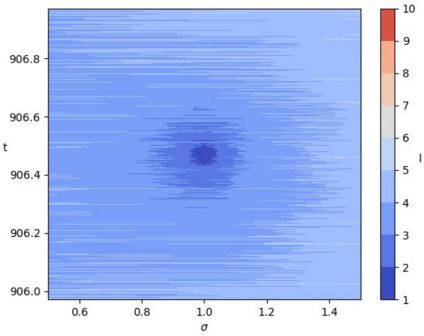
As already noted, the special points $s_k = 1 + 2\pi ki/\log 2$ are the most challenging to compute using the *MB*-algorithm. In the third experiment, we sought to estimate the radius r of a circle within which we could expect d decimal digits of accuracy could be expected for the first 200 special points s_k. To this end, we constructed a $M_{2000\times 2000}$ uniform square grid of complex points centred at each special point with $\sigma \in [1/2; 3/2]$ and $t \in [2\pi k/\log 2 - 1/2; 2\pi k/\log 2 + 1/2]$. We repeated the experiment using 2^{22} points of the Sobol sequence. The comparison showed that the resulting error distributions were qualitatively similar to those obtained with the uniform grid. The corresponding results are presented in Fig. 4. Next, we calculated the Riemann zeta function for each of these points using the *double* (with $d = 10$ decimal digits of accuracy) and *float* algorithms and created a new 2000×2000 grid of log-errors

$$\hat{d}_{n,j} = \left\lfloor -\lg \left| \zeta(M_{n,j})_{float} - \zeta(M_{n,j})_{double} \right| \right\rfloor, \quad 1 \leqslant n, j \leqslant 2000.$$

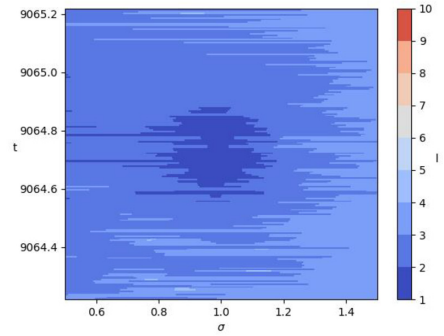


(a) $k = 1$

(b) $k = 10$



(c) $k = 100$



(d) $k = 1000$

Fig. 3. Log-error grids for several special points s_k.

Note that $\hat{d}_{n,j} \in \mathbb{N}$ and $\hat{d}_{n,j} \in [1; 10]$; however, we have obtained $\max \hat{d}_{n,j} = 4$. A visual example of this grid for the 1st, 10th, 100th and 1000th special points can be seen in Fig. 3.

Next, we find the maximum distance for each fixed $\hat{d}_{n,j} = l$,

$$r_{k,l} = \max_{\hat{d}_{n,j}=l} |M_{n,j} - s_k|, \quad 1 \leqslant l \leqslant 4,\ 1 \leqslant k \leqslant 200.$$

The experiment results are presented in Fig. 4. The maximal $r_{k,l}$ value, $r_{k,l} = 1/\sqrt{2}$, is reached at the corner of the square grid (see Fig. 4). We can see that as a special point's t value increases, so does the area around which Riemann zeta values are computed with l decimal digits of accuracy. It should also be noted that this precision area is not perfectly circular or uniform. This means we can still expect to compute higher precision Riemann zeta values within these areas.

4.2. *Parallelization Methods Comparison*

The first experiment showed that an algorithm using *float* type variables is about 22 times faster than an equivalent algorithm using *double* type variables. However, the accuracy of the computations has dropped dramatically.

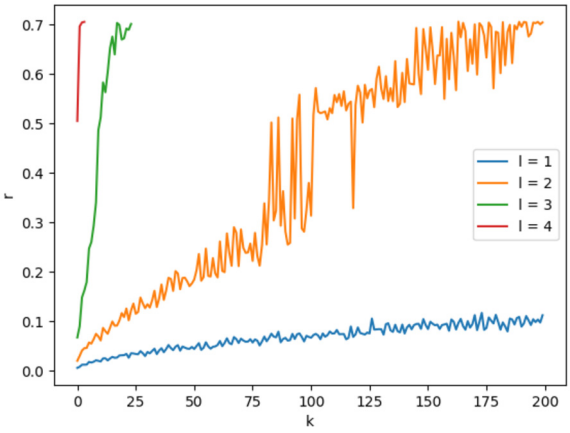


Fig. 4. The third experiment results. Each coloured line represents the calculated r for the first 200 special points for different $\hat{d}$ values.

The second and third experiments aimed to evaluate the accuracy of the *float* algorithm. In the second experiment, we showed that the accuracy decreases as t increases. This makes sense because, as t increases, so does n. This means that the sum that gives us the Riemann zeta value contains more operations and, subsequently, more rounding errors. We have also noticed that the highest errors were obtained near the special points. In the third experiment, we further investigated the areas around special points. From the results, we can conclude that within a radius $r = 1/\sqrt{2}$ starting from the 25th special point, we can only expect 3 decimal digits of accuracy.

The low accuracy of the *float* algorithm makes it less suitable for any high-precision Riemann zeta value computations. However, it might still be suitable for visualizations where low precision might be enough. This phenomenon will be investigated further in the next section.

5. Visualization of 3D Fractals

Before rendering a Mandelbulb, recall how a two-dimensional Mandelbrot set is generated. The set consists of complex numbers c for which the function $f_c(z) = z^2 + c$ does not diverge when iterated from $z = 0$. The pixels representing these points are marked with a plus sign in Fig. 5. The other points are usually coloured according to the iteration number, at which the value exceeds some preset threshold. For a 2D fractal, this approach works effectively: pixels can be coloured on a plane, and the observer perceives the overall fractal structure from above (cf. Belovas *et al.*, 2022). In contrast, while considering a three-dimensional analogue, the observer is placed within the diverging or non-diverging regions. In this context, the observer primarily perceives the boundary between the two regions rather than the regions themselves. Consequently, the traditional method of colouring pixels based on iteration count is insufficient for 3D visualization.

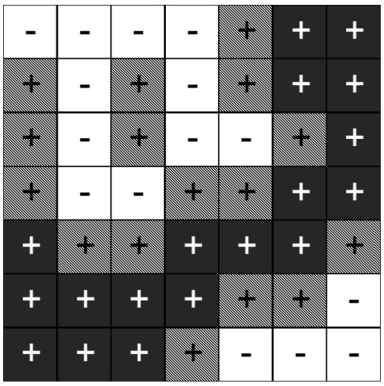


Fig. 5. Closeup segment of the Mandelbrot set. The cells marked with the plus sign belong to the Mandelbrot set, and those with the minus sign do not. The striped cells represent the edge of the Mandelbrot fractal.

In this section, we outline two approaches to the visualization of Mandelbulbs, addressing these challenges and enabling the rendering of three-dimensional fractal structures.

5.1. *Ray Casting Approach*

A classic method of visualizing 3D objects is using a ray casting graphics engine. This requires creating a polygon mesh representing an approximation of the object's shape. A polygon mesh is a set of vertices connected by edges where three edges create a face. In the case of a Mandelbulb, the polygon mesh would have to represent the surface area between the regions that do and do not belong to the Mandelbulb. For this task, we can take inspiration from the 2D Mandelbrot fractal. In Fig. 5, we can see a closeup of the Mandelbrot set where the points that belong to the set are marked with a plus sign and the points that do not are marked with a minus sign. If we were a 2D observer, when placed in the minus region of the image, we could only see the edge of the fractal (the red cells). These red cells can be easily identified if they belong to the Mandelbrot set and have at least one adjacent cell that does not belong to the Mandelbrot set. For a 3D equivalent, we can divide the volume into cubes. Then, we can calculate whether the centre point of the cube belongs to the Mandelbulb and, thus, whether the cube should be visible. This approach would create a blocky polygon mesh. Surface smoothing algorithms could be used to smooth the surface of the polygon. This polygon might be suitable with a high enough resolution and looking from far enough away. However, we would be limited by the mesh's size if we were to explore the fractal details of a Mandelbulb. The created model is also very rigid, meaning that if we would like to view a Mandelbulb of a different power, we would have to create an entirely new mesh. For these reasons, the ray casting approach is unsuitable for our task. However, it could still be useful for printing a Mandelbulb.

5.2. *Ray Marching Approach*

A technique called ray marching is a way to visualize three-dimensional objects without using a polygon mesh of the object. This method instead uses signed distance functions

(SDFs) to estimate the distance to an object. An SDF of an object takes a point's coordinates as an argument and returns the distance to the closest point of the surface of that object. Then we modify a simple ray casting engine to use these SDFs to render an image of an object. SDFs can also be used to calculate surface normals of an object. This means that objects rendered using the ray marching method can be shaded. We can use ray marching to render Mandelbulbs. For that purpose, we need an SDF for a Mandelbulb, which luckily exists. Algorithm 1 (Syntopia, 2011) gives us the SDF for a Mandelbulb.

Algorithm 1 Signed distance function of a Mandelbulb. Here s – the three-dimensional coordinates of the origin point, p – the power of the Mandelbulb, k – the maximum number of iterations.

function SDFMANDELBULB(s, p, k)
  $z \leftarrow s$
  $d \leftarrow 1$
  for $j \leftarrow 0...k$ **do**
    $r \leftarrow |z|$
    if $r > 2$ **then**
      break
    end if
    $P \leftarrow r^{p-1}$
    $d \leftarrow Ppd + 1$
    $\vartheta \leftarrow p \arccos(z_z/r)$
    $\varphi \leftarrow p \arctan(z_y/z_x)$
    $z_r \leftarrow rP$
    $z \leftarrow (z_r \sin(\vartheta)\cos(\varphi), z_r \sin(\vartheta)\sin(\varphi), z_r \cos(\vartheta))$
    $z \leftarrow z + s$
  end for
  return $r \log r/(2d)$
end function

6. Zetabulb Visualization

As it has been concluded, we use the ray marching technique to render Zetabulbs. Several free ray marching engines that can visualize Mandelbulbs are available. However, none allow efficient CUDA utilization for the Riemann zeta function calculation. Therefore, we will create a basic ray marching engine from scratch. The engine's source code and the *MB*-algorithm's implementation can be found in Kuzma (2025). The new instrument will exhibit the following features:

- object rendering (for a given SDF);
- object colouring with a multi-colour gradient;

- object shading (for the object including ambient lighting);
- FXAA anti-aliasing (using fast approximate anti-aliasing algorithm);
- enabling camera movement with keyboard and mouse controls.

6.1. *Zetabulb Signed Distance Function*

Zetabulb visualization requires associating the Mandelbulb signed distance function with the Riemann zeta function. When combining a fractal with another object (e.g. shape, fractal, surface), we transform the fractal's space by the object's properties and then visualize the fractal in this new space. Thereby, we will combine the Mandelbulb with the surface of the Riemann zeta function.

The procedure is nontrivial. The Mandelbulb has a fractal surface; however, its volume is finite, whereas the Riemann zeta function's domain is an infinite complex plane. Thus, the question arises: how do we combine a finite volume and an infinite plane? We will solve this problem by adding two more degrees of freedom: movements in the complex plane of the Riemann zeta function. This would allow the Mandelbulb to inherit the properties of a small section of the Riemann zeta functions surface rather than the entire plane. We can implement the new degrees of freedom by adding a complex point c. This point will associate the Mandelbulb with the Riemann zeta function.

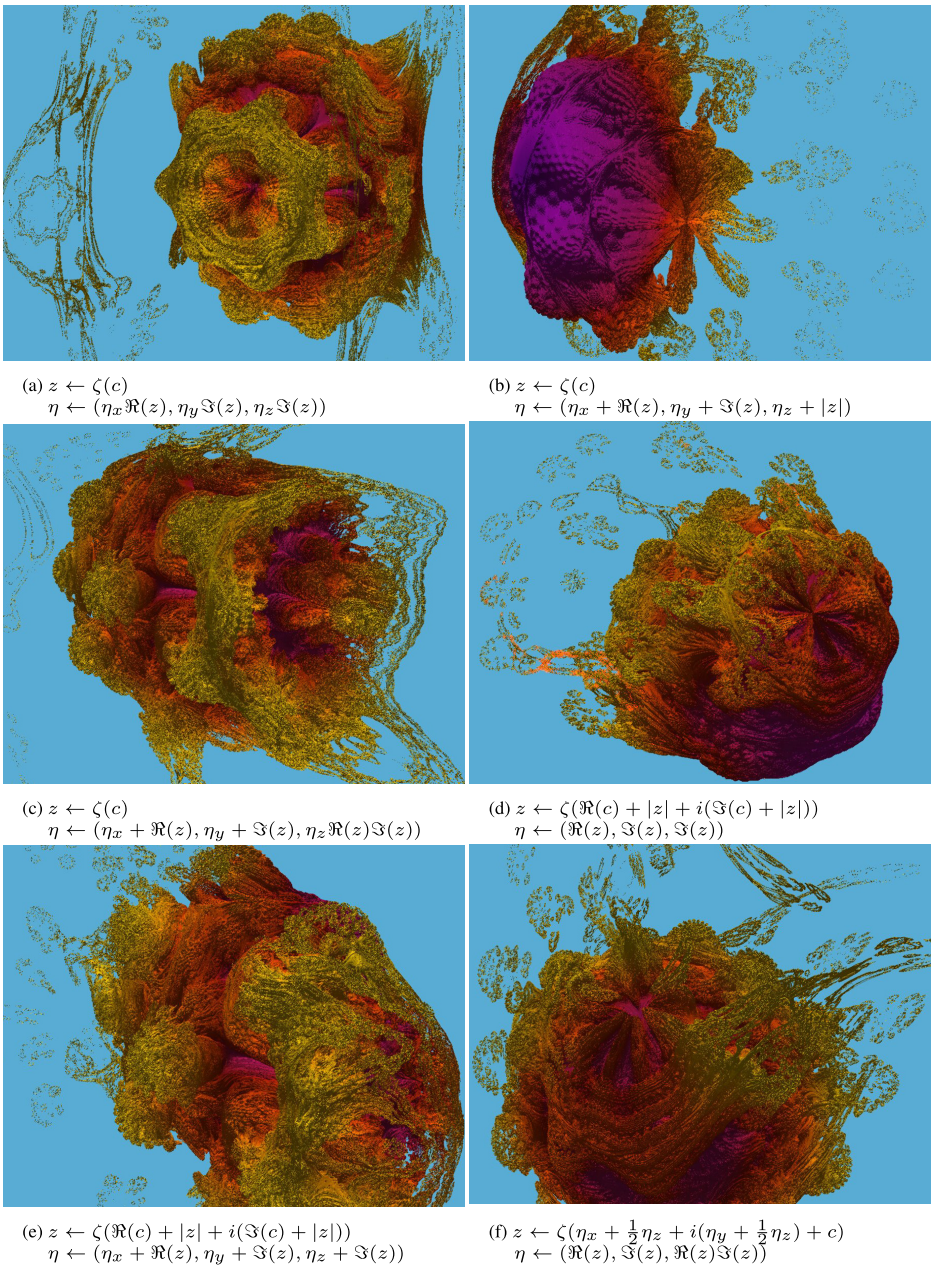
The association of the complex plane and the Cartesian coordinate system can be accomplished in two steps. First, we compute the Riemann zeta function value z for an argument related to the point c. Next, we construct a new point η and compute the SDF of a Mandelbulb for this point. An example of this association is presented in Algorithm 2. Figure 6 displays Zetabulbs obtained by several association methods. We will use the same principle when associating Mandelbulbs with Dirichlet L-functions.

Algorithm 2 Signed distance function of a Zetabulb. Here η – the three-dimensional coordinates of the origin point, p – the power of the Mandelbulb, k – the maximum number of iterations, c – the complex number used to associate the Mandelbulb with the real and imaginary surfaces of the Riemann zeta function.

1: **function** *SDFZetabulb*(η, p, k, c)
2: $\quad z \leftarrow \zeta(\Re c + \eta_x + i(\Im c + \eta_y))$
3: $\quad \eta \leftarrow (\Re(z), \Im(z), \eta_z)$
4: $\quad$ **return** *SDFMandelbulb*(η, p, k)
5: **end function**

6.2. *Visualization Quality*

We have discovered that algorithms using *float* type variables perform ~ 22 times faster than equivalent ones employing *double* type variables. However, the second experiment showed that this gain comes with a significant loss in accuracy. In this section, we will determine how this loss of accuracy impacts Zetabulb image quality.



(a) $z \leftarrow \zeta(c)$
$\eta \leftarrow (\eta_x \Re(z), \eta_y \Im(z), \eta_z \Im(z))$

(b) $z \leftarrow \zeta(c)$
$\eta \leftarrow (\eta_x + \Re(z), \eta_y + \Im(z), \eta_z + |z|)$

(c) $z \leftarrow \zeta(c)$
$\eta \leftarrow (\eta_x + \Re(z), \eta_y + \Im(z), \eta_z \Re(z) \Im(z))$

(d) $z \leftarrow \zeta(\Re(c) + |z| + i(\Im(c) + |z|))$
$\eta \leftarrow (\Re(z), \Im(z), \Im(z))$

(e) $z \leftarrow \zeta(\Re(c) + |z| + i(\Im(c) + |z|))$
$\eta \leftarrow (\eta_x + \Re(z), \eta_y + \Im(z), \eta_z + \Im(z))$

(f) $z \leftarrow \zeta(\eta_x + \frac{1}{2}\eta_z + i(\eta_y + \frac{1}{2}\eta_z) + c)$
$\eta \leftarrow (\Re(z), \Im(z), \Re(z)\Im(z))$

Fig. 6. Zetabulbs constructed using different association methods with identical c values. Values z and η correspond to the rows 2 and 3 of Algorithm 2.

Algorithm 3 Ray marching function used for the custom rendering tool. This function fires a ray from *position* in *direction* until it hits or misses the object. Returns a colour for a pixel. It is assumed that the centre of the object defined by the SDF is at coordinates (0, 0, 0).

function RAYMARCH(*position*, *direction*, SignedDistanceFunction)
    $maximumSteps \leftarrow 1000$
    $minimumHitDistance \leftarrow 2 \cdot 10^{-6}$
    $maximumTravelDistance \leftarrow 10$
    $distanceTravelled \leftarrow 0$
    $pixel \leftarrow$ RGB(0, 0, 0) ▷ Pixel colour in RGB format
    for $j \leftarrow \{0..maximumSteps\}$ **do**
        $currentPosition \leftarrow position + distanceTravelled \cdot direction$
        $distanceToObject \leftarrow SDFZetabulb(currentPosition)$
        if $distanceToObject < minimumHitDistance$ **then** ▷ Ray hits the object
            $light \leftarrow$ Light($currentPosition$) ▷ Computes light intensity at the point based on a surface normal
            $distanceToCenter \leftarrow \|currentPosition\|$
            $colour \leftarrow Colour(currentPosition)$ ▷ Computes pixel colour at the point from a multi-colour gradient based on distance from the point to the centre of the object
            $pixel \leftarrow colour \cdot light$
            break
        end if
        $distanceTravelled \leftarrow distanceTravelled + distanceToObject$
        if $totalDistanceTravelled > maximumTravelDistance$ **then** ▷ Ray misses the object
            $pixel \leftarrow$ RGB(89, 173, 211) ▷ Pixel get the background colour
            break
        end if
    end for
return *pixel*
end function

Algorithm 3 defines the process of how each pixel is coloured. In total, the Riemann zeta function is called up to *maximumSteps* times and an additional 6 times when computing a surface normal if the ray hits the Zetabulb. Errors from having non-accurate Riemann zeta values can quickly accumulate and produce incorrect pixel colour values. We will perform an experiment to test how inaccurate the computed values are. First, we will generate ten Zetabulb images with $1 \leqslant d \leqslant 10$ decimal digits of accuracy with the *double* algorithm and one with the *float* algorithm. We will call the image with 10 decimal digits of accuracy our benchmark and the others our test images. Then, we will calculate the colour accuracies of each image when compared to the benchmarks' corresponding pixels.

Table 3
E values of the test images for Scenario 1. d – the decimal digits of accuracy with which an image was generated, and f stands for the *float* algorithm. Other columns lists the percentage of pixels that fall into each $\triangle E$ category when compared with the benchmark image.

	$\Delta E \in$					
d	0	(0; 1]	(1; 2]	(2; 10]	(10; 50]	(50; 100]
1	31.93	1.68	1.52	11.47	39.96	13.44
2	35.04	3.71	3.15	14.13	33.97	10.00
3	43.39	5.58	3.34	14.29	27.59	5.80
4	59.41	8.83	4.74	13.65	11.50	1.86
5	72.65	15.18	4.49	5.42	1.96	0.29
6	86.45	10.94	1.10	1.08	0.38	0.05
7	95.02	4.21	0.30	0.33	0.12	0.02
8	98.55	1.23	0.08	0.09	0.04	0.00
9	99.81	0.17	0.01	0.01	0.00	0.00
f	90.32	7.89	0.63	0.67	0.32	0.15

To compute each pixel's colour accuracies ΔE, we will use the CIEDE2000 algorithm (see Sharma *et al.* (2005)), an industry-standard, to evaluate how similar colours are, $\Delta E \in [0; 100]$. Empirically, these values can be categorized into the following groups:

- $\Delta E = 0$. The colours are the same.
- $\Delta E \in (0; 1]$. The colour differences are not perceptible by the human eye.
- $\Delta E \in (1; 2]$. Perceptible through close observation.
- $\Delta E \in (2; 10]$. Perceptible at a glance.
- $\Delta E \in (10; 50]$. Colours are more similar than different.
- $\Delta E \in (50; 100]$. Colours are more different than similar.

In our experiment, we compute the percentages of pixels of our test images that fall in each group. By comparing the percentages of the *float* algorithm to the other test images, we should be able to evaluate the adequate visualization accuracy of the *float* algorithm.

Earlier, we have established that our implementation of the *MB*-algorithm will compute the coefficients $\{\psi_{n,k}\}$ for a maximum imaginary part $t_{\max}$ and d decimal digits of accuracy. The same coefficients $\{\psi_{n,k}\}$ will be used to compute Riemann zeta function values for $t < t_{\max}$. Naturally, these values will be computed with higher decimal accuracy than d. This property of our algorithm makes it problematic to compute the Riemann zeta function values for arguments with different imaginary parts while maintaining exactly the same accuracy. For this reason, as a benchmark, we use values computed using the *double* algorithm with $d = 10$ decimal digits of accuracy. Next, by rounding down the benchmark values, we obtain Riemann zeta function values with lower accuracies.

For this experiment, we evaluate the visualization accuracy in two scenarios, with $\Im c = 5$ and with $\Im c = 500$. We also produce a grayscale image of the *float* image with highlighted pixels that fall into the groups $\Delta E \in (10; 50]$ and $\Delta E \in (50; 100]$. The computed ΔE values are presented in Tables 3 and 4. Figure 7 presents the generated images for each scenario.

According to Table 3, in Scenario 1 ($\Im c = 5$) the effective visualization quality could be achieved between $d = 6$ and $d = 7$. However, we can notice an unusually high number

Table 4
$\triangle E$ values of the test images for Scenario 2. d – the decimal digits of accuracy with which an image was generated, and f stands for the *float* algorithm. Other columns lists the percentage of pixels that fall into each $\triangle E$ category when compared with the benchmark image.

	$\Delta E \in$					
d	0	(0; 1]	(1; 2]	(2; 10]	(10; 50]	(50; 100]
1	11.01	2.18	8.17	15.46	59.55	9.53
2	16.99	4.39	2.26	17.41	51.43	6.36
3	23.12	5.88	3.42	21.24	41.39	3.95
4	37.27	17.01	4.42	22.20	12.14	1.04
5	54.97	32.58	10.33	4.95	1.89	0.19
6	83.61	14.33	5.42	0.83	0.37	0.05
7	94.82	4.57	0.82	0.23	0.13	0.02
8	98.42	1.38	0.22	0.07	0.04	0.01
9	99.79	0.18	0.07	0.01	0.01	0.00
f	47.81	30.78	0.01	8.86	3.89	0.49

of very different pixels (groups $\Delta E \in (10; 50]$ and $\Delta E \in (50; 100]$) for these quality ranges. In Fig. 7(f), we can see these pixels highlighted by the colours red and green, respectively. We can see a small green region in the bottom right of the image. The *float* algorithm appears to have generated a segment that does not exist in the benchmark image. After further investigation, we could see that the region exists, and the *double* algorithm could not generate it. Scenario 2 ($\Im c = 500$) did not have such anomalies, and its effective precision could be evaluated as between $d = 4$ and $d = 5$ in all ΔE categories. As we can see, by increasing $\Im c$, we experienced a degradation in image visualization quality.

With the third experiment, we have discovered that the *float* algorithm is inaccurate when computing the Riemann zeta function values in the neighbourhoods of the special points s_k. Scenario 3 aims to investigate Zetabulbs generated near one of these points. Figure 8(a) was generated using the *double* algorithm with $d = 10$ decimal digits of accuracy, whereas Fig. 8(b) was generated using the *float* algorithm. We can see that the *float* algorithm generated different although similar structures.

6.3. *Visualization Performance*

In this subsection, we evaluate the real-time Zetabulb visualization performance. We measure the average FPS (frames per second) that our Zetabulb visualization instrument can achieve with *float* and *double* algorithms and different GPUs in four experiments, each containing four scenarios lasting for 30 seconds:

- Scenario A. Mandelbulb with no movement.
- Scenario B. Mandelbulb with a varying power.
- Scenario C. Zetabulb with no movement and set $t_{\max}$ value.
- Scenario D. Zetabulb with increasing $\Im c$ and set $t_{\max}$ value.

The visualization performance results are presented in Table 5. The human eye needs at least 24 FPS to perceive a video as smooth, so this is the value that we will consider satisfactory (coloured blue in Table 5). As discussed before, using *double* type variables

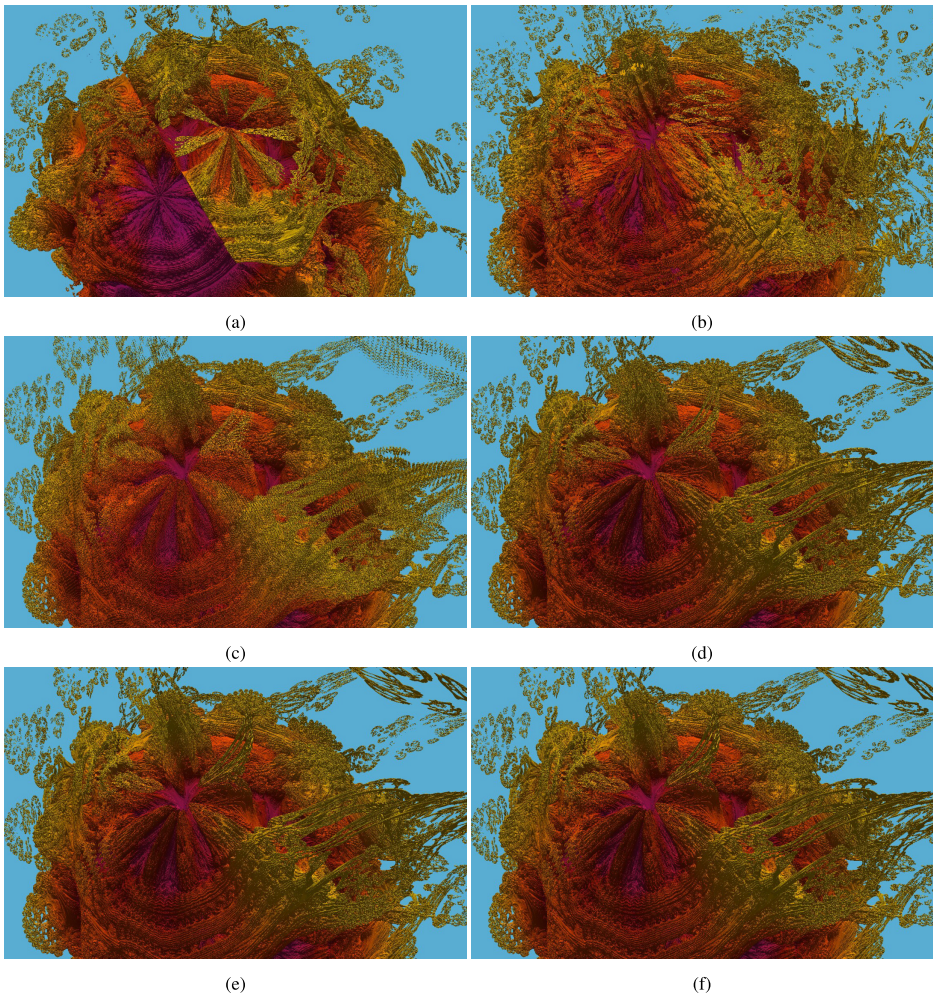
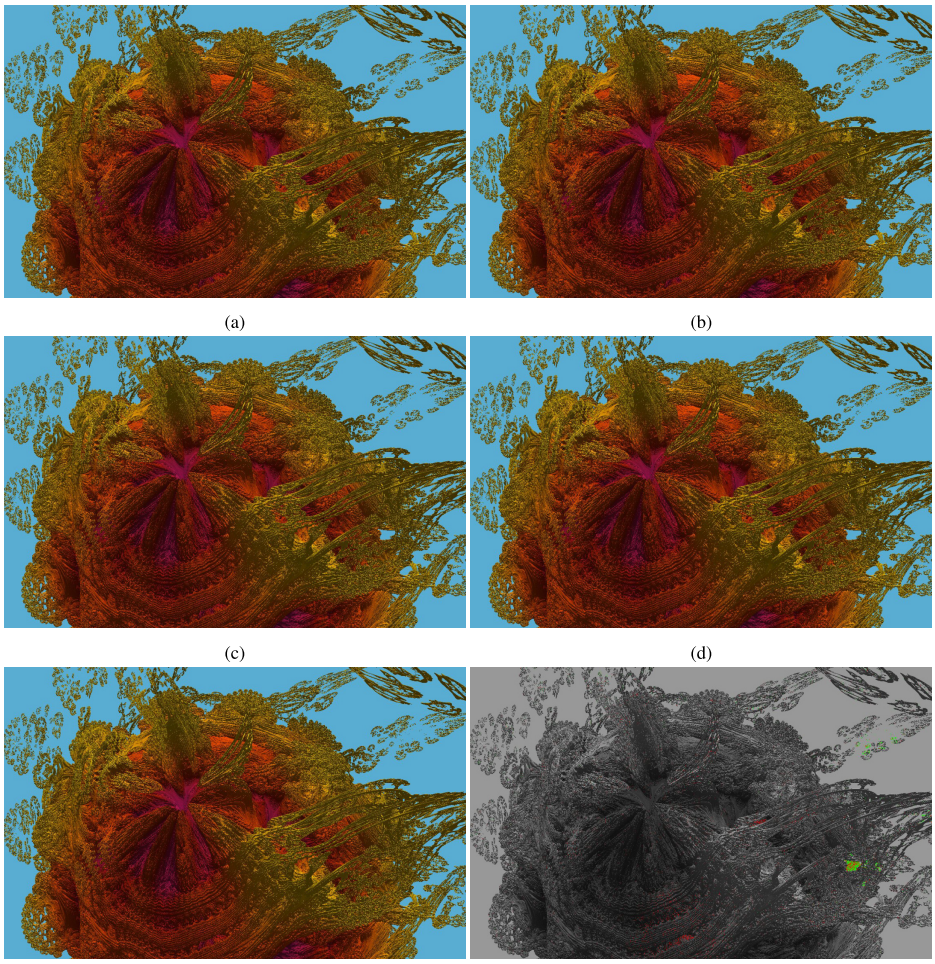


Fig. 7. Zetabulb images with different Riemann zeta value precisions for Scenario 1 ($\Im c = 5$). Subfigures (a) through (d) were generated with the *double* algorithm and lower precision. Subfigure (e) was generated with the *float* algorithm. Subfigure (f) is a grayscale version of (e) with pixels that fall into the category $\Delta E \in (10; 50]$ highlighted red and $\Delta E \in (50; 100]$ highlighted green.

in CUDA computations significantly increases performance time. We can see this effect again when comparing the results of T_1 to T_3 and T_2 to T_4 where the *double* algorithm could not achieve even a single frame per second.

Scenarios A and B are used as benchmarks. As discussed before, this work required the creation of a custom ray marching graphics engine to test the *EMB*-algorithm efficiently. In order to show that this engine does not hinder the results, regular Mandelbulb images were generated. Scenarios A and C produce static scenes, whereas B and D produce dynamic scenes. We can see no difference in performance between static and dynamic scene results. This can be viewed as both an advantage and a disadvantage. Our tool does not have

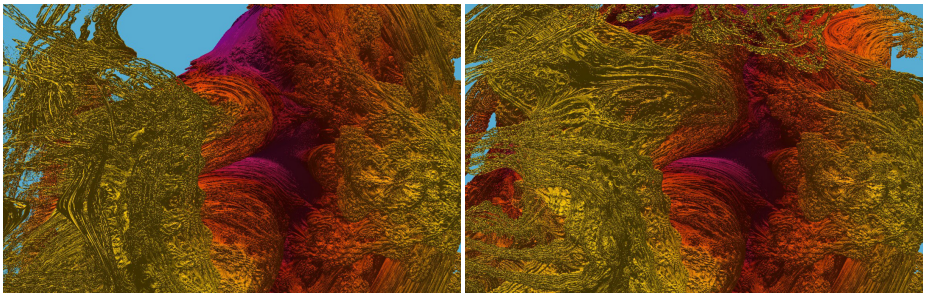


(a) (b)

(c) (d)

(e) (f)

Fig. 7. (*continued*)



(a) (b)

Fig. 8. Zetabulb images for Scenario 3 ($\Im c = 163.165$) (18th special point). Subfigure (a) was generated using the *double* algorithm with $d = 10$ decimal digits of accuracy, and (b) was generated using the *float* algorithm.

Table 5
Four tests of visualization performance (in FPS). First – *float* type variables with $t_{\max} = 50$, second – *float* type with $t_{\max} = 500$, third – *double* type with $t_{\max} = 50$, fourth – *double* type with $t_{\max} = 500$. The required memory in bytes to store the coefficients in VRAM is given for each of the tests. All the tests were performed on two GPUs: RTX 3080 and GTX 1060 Mobile. Values marked in blue denote satisfactory results.

Variable	*float* type				*double* type			
Test	T_1		T_2		T_3		T_4	
t	$t_{\max} = 50$		$t_{\max} = 500$		$t_{\max} = 50$		$t_{\max} = 500$	
Memory, bytes	192		384		1200		2400	
Scenario	*RTX*	*GTX*	*RTX*	*GTX*	*RTX*	*GTX*	*RTX*	*GTX*
A	129.71	21.31	126.71	21.10	7.12	1.97	7.17	1.98
B	137.89	22.03	142.14	21.30	6.68	1.94	7.44	1.96
C	27.35	3.99	5.78	0.97	0.11	0.03	0.02	0.005
D	27.26	3.87	5.60	0.87	0.11	0.02	0.02	0.005

Table 6
Dirichlet character $\chi_{5,4}$. Note that $\chi_{5,4}(-1) = \chi_{5,4}(4) = 1$.

n	1	2	3	4
$\chi_{5,4}(n)$	1	-1	-1	1

a feature that would cache pixel values if no scene parameters change between frames. However, this also means that changing scene parameters between frames has no impact on the performance, so we can create videos of transforming fractals at high frame rates.

The real-time Zetabulb visualization performance results were poor when using the GTX 1060; however, the RTX 3080 produced satisfactory results in T_1. It should be noted that the RTX 3080 GPU is now over three years old, and faster GPUs exist on the market. The usage of newer GPUs could produce even better performance results. We conclude that the proposed algorithm produces satisfactory results with low $t_{\max}$ values and higher-end GPUs.

6.4. *Visualization Results*

In this subsection, we will provide Zetabulb images with c values near interesting features of the Riemann zeta function surface. All of the Figs. 10(a) through 10(b) depicting Zetabulbs were generated using Algorithm 3 where the SDF is set to Algorithm 2 using both the Riemann zeta function (which we will call Riemann Zetabulbs) and the Dirichlet L-function (Dirichlet Zetabulbs) with the character $\chi_{5,4}$ (exact value shown in Table 6).

The argument value c for Algorithm 2 has to be chosen carefully. Let us consider ε-neighbourhood of the point c. The algorithm will produce normal Mandelbulbs if $|\zeta(c + \varepsilon)| \approx 0$. If $|\zeta(c + \varepsilon)$ has very large values, the algorithm will not produce any fractal structures. Ideally, we want to position the point c where $|\zeta(c+\varepsilon)| \in (0; 5]$ and where there is a lot of variance in the nearby values. The Riemann zeta function values can be divided

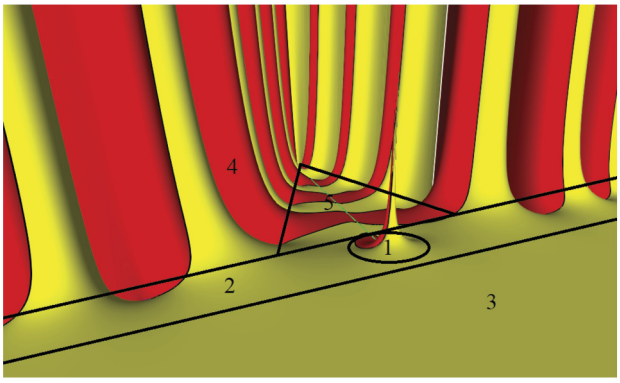


Fig. 9. The real (yellow) and the imaginary (red) parts of the Riemann zeta function and the five principal regions: the pole neighbourhood (1), the critical strip (2), the plateau to the right of the critical strip (3), the wall to the left of the critical strip (4), and the triangular plateau (5).

into five regions: the pole, the critical strip, the plateau to the right of the critical strip where $\zeta(c) \approx 1$, the wall to the left of the critical strip where $\zeta(c)$ values are very large, and the small triangular plateau to the left of the pole where $\zeta(c) \approx 0$. Figure 9 shows all five of these regions. The points that fit the criteria for generating fractal structures lie on the boundaries of all regions and the entire critical strip.

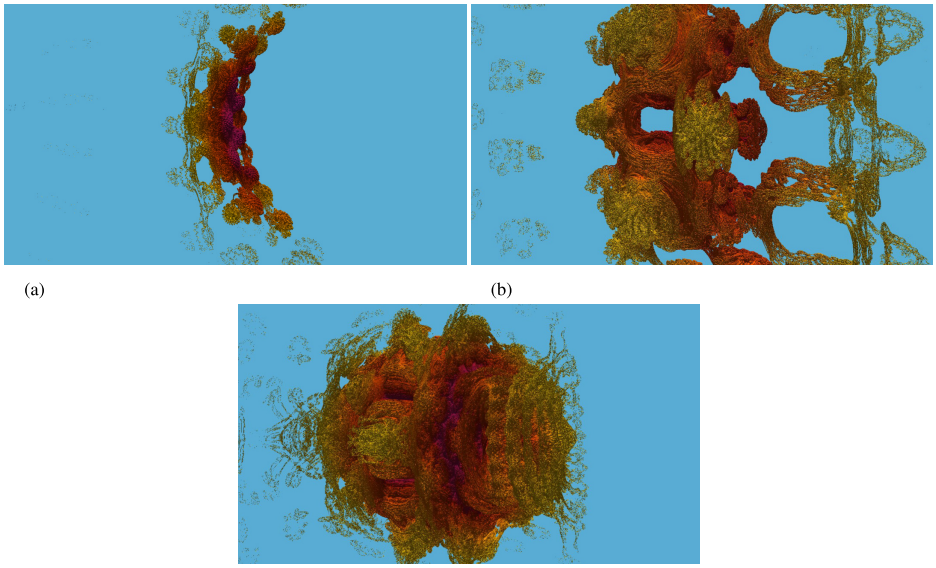
We selected three points to showcase in this work. As the first point, we chose the pole of the Riemann zeta function $c = 1$. Here, the large values of the pole region completely remove the middle of the Mandelbulb, leaving a bow-shaped structure with holes, which can be seen in Figs. 10(a) and 10(b). For comparison, the Dirichlet L-function does not have a pole, thus the Zetabulb in Fig. 10(c) appears similar to a regular Mandelbulb.

We chose the seventh trivial zero of the Riemann zeta function $c = \rho_7$ for the second point of interest. The corresponding fractal is visualized in Fig. 11(a). The Riemann zeta function has trivial zeroes at points $\rho_k = -2k, k \in \mathbb{N}$. The trivial zeroes ρ_k, with $k \in [1; 6]$ lie in the triangular plateau. An example of one such fractal is presented in Fig. 11(b). Whereas, the trivial zeroes with $k \in [8; \infty)$ belong to the wall region. This leaves only the point ρ_7 to produce distinct structures when compared to the original Mandelbulb (see Fig. 11) which is calculated using Algorithm 1.

As the final point of interest, we have selected the fourth nontrivial zero of the Riemann zeta function $c = \gamma_4 = 1/2 + 30.42487i$. At this point, most of the fractal structure forms a round blob, whereas the top of the fractal is slightly extended, creating a bottle-like shape that can be seen in Figs. 12(a) and 12(b).

7. Conclusions

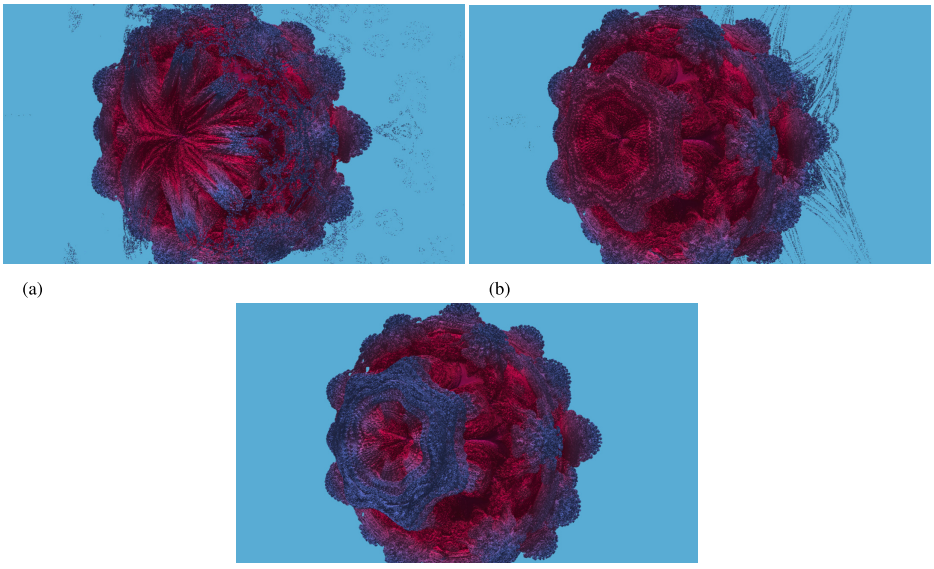
This work aimed to create efficient algorithms, which would be the base of a software instrument for visualizing Mandelbulb fractals associated with the Riemann zeta function and the Dirichlet L-function (Zetabulbs). To the author's knowledge, this is the first inves-



(a) (b)

(c)

Fig. 10. (a) Riemann Zetabulb centered on the pole. (b) Side view of the same Zetabulb. (c) Dirichlet Zetabulb centered on the pole.



(a) (b)

(c)

Fig. 11. (a) Riemann Zetabulb centred on the 7th trivial zero. (b) Riemann Zetabulb centred on the 5th trivial zero. (c) Original Mandelbulb.

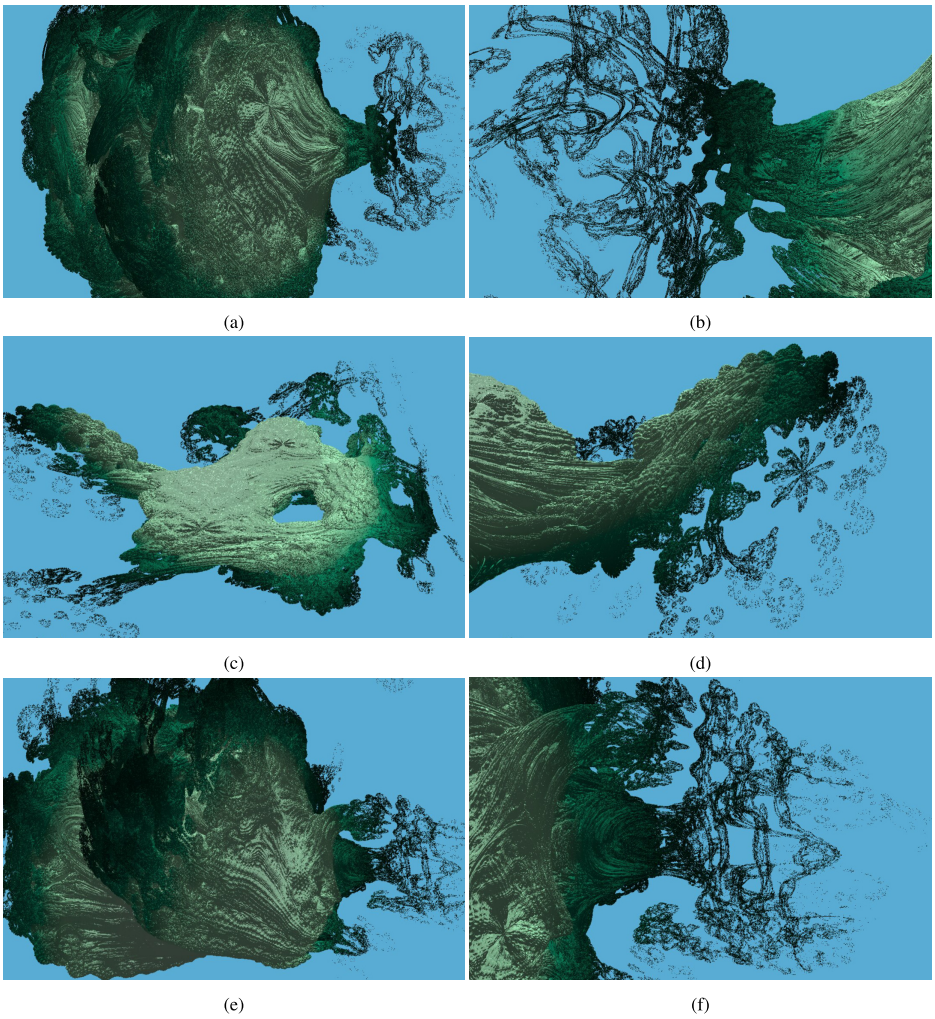


Fig. 12. (a) Riemann Zetabulb centred on the fourth non-trivial zero $c = \gamma_4$. Note a bottle-like shape. (b) Closeup image of the bottleneck of the previous Zetabulb. (c) and (d) Dirichlet Zetabulbs centred on the fourth non-trivial zero of the Riemann zeta function. (e) and (f) Dirichlet Zetabulbs centred on the fourth non-trivial zero of the Dirichlet L-function ($c = 1/2 + 16.03382i$).

tigation and application of Zetabulbs. To achieve this goal, parallel technologies have been applied to speed up *MB*-algorithm and technologies designed for fractal visualization and application have been studied and implemented.

7.1. *MB-Algorithm Parallelization Conclusions*

- Two versions of the optimal approach (one using *float* type variables and one using *double* type variables) have been compared. The conclusion is that the *float* algorithm is

much faster ($\sim$ 22 times) while being less accurate and unsuitable for precise Riemann zeta function computation.
- Next, we have noticed that the *float* algorithm accuracy correlates with the imaginary part of the argument of the Riemann zeta function. Moreover, the highest errors have been obtained in the ε-neighbourhoods of the special points. Hence, the *float* algorithm should not be used for any precise Riemann zeta function calculations on these domains.

7.2. *Zetabulb Visualizations Conclusions*

- Six different signed distance functions (SDFs) to produce Zetabulb images have been proposed. All the proposed functions have produced satisfactory results in different but still aesthetically appealing fractal structures.
- The problem of sufficient (practical) accuracy of the *float* algorithm compared to *double* algorithm for Zetabulb visualization has been considered. The conclusion is that the *float* algorithm is applicable for Zetabulb visualization (for non-scientific, e.g. cinematographic purposes) despite a certain accuracy deterioration with an increase of the imaginary part of the argument.

The presented 3D fractal (Zetabulb) images generating computer graphics software tool could be used as a base for SFX and CGI in film-making, as well as the investigation of underlying properties of the Riemann zeta function.

References

Belovas, I., Sakalauskas, L. (2018). Limit theorems for the coefficients of the modified Borwein method for the calculation of the Riemann zeta-function values. *Colloquium Mathematicum*, 151(2), 217–227. https://doi.org/10.4064/cm7086-2-2017.

Belovas, I., Sabaliauskas, M., Kuzma, L. (2022). Series with binomial-like coefficients for the investigation of fractal structures associated with the Riemann zeta function. *Fractal and Fractional*, 27(1). https://doi.org/10.3390/fractalfract6060300.

Borwein, P. (2000). An efficient algorithm for the Riemann zeta function. In: *Constructive, experimental, and nonlinear analysis; Canadian Mathematical Society Conference Proceedings; Théra, M., Ed.; American Mathematical Society, Providence, RI, USA*, Vol. 27, pp. 29–34.

Kumar, A., Bhaskar, L. (2025). Dynamic Mandelbulb fractals: a mathematical framework for time-evolving 3D fractals using distance estimation. *Chaos, Solitons and Fractals*, 199(3), 116829. https://doi.org/10.1016/j.chaos.2025.116829.

Kuzma, L. (2025). Zetabulber source code. https://github.com/Zetabulb/Zetabulber.

NVIDIA (2023). CUDA Arithmetic Instructions performance table. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#arithmetic-instructions.

NVIDIA (2026). CUDA. https://developer.nvidia.com/cuda.

Sharma, G., Wu, W., Dalal, E.N. (2005). The CIEDE2000 color-difference formula: implementation notes, supplementary test data, and mathematical observations. *Color Research & Application*, 30(1), 21–30. https://doi.org/10.1002/col.20070.

Syntopia (2011). Syntopia, Distance Estimated 3D Fractals (V): The Mandelbulb & Different DE Approximations. http://blog.hvidtfeldts.net/index.php/2011/09/distance-estimated-3d-fractals-v-the-mandelbulb-different-de-approximations/.

Ugwu, C.I., Caruso, E., Lanz, O. (2024). Fractals as pre-training datasets for anomaly detection and localization. *Fractal and Fractional*, 8(11), 661. https://doi.org/10.3390/fractalfract8110661.

Šleževičiene, R. (2004). An efficient algorithm for computing Dirichlet L-functions. *Integral Transforms and Special Functions*, 15(6), 513–522. https://doi.org/10.1080/1065246042000272072.

L. Kuzma received his master's degree in computing from Vilnius University in 2024. He is currently a PhD student at the Institute of Data Science and Digital Technologies, Faculty of Mathematics and Informatics, Vilnius University. His research interests include computational problems related to zeta functions.

I. Belovas received his doctoral degree in mathematical sciences from Vilnius Gediminas Technical University and the Institute of Mathematics and Informatics in 2004. He is currently a professor at the Institute of Data Science and Digital Technologies, Faculty of Mathematics and Informatics, Vilnius University. His research interests include problems in analytical number theory and financial engineering.

M. Sabaliauskas received his doctoral degree in technological SCiences from Vilnius University in 2017. He is currently an associate professor at the Institute of Data Science and Digital Technologies, Faculty of Mathematics and Informatics, Vilnius University. His research interests include problems in 3D visualization.