



Lipschitz-agnostic, efficient and accurate rendering of implicit surfaces

Rene Winchenbach¹ · Michael Möller² · Andreas Kolb²

Accepted: 28 November 2023 / Published online: 8 January 2024
© The Author(s) 2024

Abstract

In this paper, we propose an accurate and controllable rendering process for implicit surfaces with no or unknown analytic Lipschitz constants. Our process is built upon a ray-casting approach where we construct an adaptive Chebyshev proxy along each ray to perform an accurate intersection test via a robust and multi-stage searching method. By taking into account approximation errors and numerical conditions, our methods comprise several pre-conditioning and post-processing stages to improve the numerical accuracy, which potentially applied recursively. The intersection search is performed by evaluating a QR decomposition on the Chebyshev proxy function, which can be done in a numerically accurate way. Our process achieves comparable accuracy to other techniques that impose more constraints on the surface, e.g., knowledge of Lipschitz constants, and higher accuracy compared to approaches that impose similar constraints as our approach.

Keywords Rendering · Implicit surfaces · Ray casting · Root finding · Chebyshev approximation

1 Introduction

Implicit surfaces have a long tradition in computer graphics as they are a powerful abstraction of many geometric models [6, 23] and as they provide construction operations on the underlying shapes either directly, e.g., using metaballs [5], or indirectly, e.g., using skeleton-based representations [2]. Implicit surfaces are commonly defined as an iso-contour of a 3D scalar field $s(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^3$, where $\mathbf{x}$ is inside and outside the object, if $s(\mathbf{x}) < 0$ and $s(\mathbf{x}) > 0$, respectively.

Direct rendering of implicit surfaces, e.g., without conversion into a mesh, is, however, a non-trivial task, and is commonly handled with ray-casting [26, 33, 44]. This involves finding the first root of $s(\mathbf{r}(\lambda))$ along a ray $\mathbf{r}(\lambda) = \mathbf{o} + \lambda \mathbf{d}$ cast from a viewpoint $\mathbf{o}$ in direction $\mathbf{d}$. For polynomial scalar fields s of degree ≤ 4 , several accurate or even analytic approaches for root finding exist, e.g., interval arithmetic [45], which can also be utilized by approximating s by polynomials of degree ≤ 4 [28, 43] or Taylor expansions [45]. This often induces an approximation error that is hardly controllable, as finding a sufficiently accurate approximation of $s(\mathbf{r}(\lambda))$ is a very challenging task, especially when

being restricted to low degree approximations. Alternatively, if the global or local Lipschitz constant for the scalar field is known, more accurate root finding algorithms can be used, e.g., LG-surfaces [29] and Sphere Tracing [26]. However, while Lipschitz constants are known when rendering, e.g., SDFs, they are generally not known a priori when rendering surfaces based on arbitrary implicit functions.

In this paper, we propose a rendering approach for implicit surfaces that approximates the scalar field along rays with controllable accuracy and does not require any knowledge of Lipschitz constants. The only technical requirement is, that the scalar field is globally C^0 -continuous. The specific contributions of this paper are:

- (a) An efficient and accurate approach to find the first root along a ray, which involves an efficient Chebyshev approximation method, comprising several pre-conditioning and post-processing stages, as well as an optional, recursive root finding, to achieve as-optimal-as-possible scalar field approximations along rays,
- (b) a performance analysis of our method using scalar fields of varying types, e.g., polynomial and non-algebraic, regarding both, computational aspects and achievable numerical accuracy, which are compared to existing rendering approaches, and
- (c) an open source CPU-based C++ implementation of the Chebyshev proxy construction process, the rendering

✉ Rene Winchenbach
rene.winchenbach@gmail.com

¹ Technical University Munich, Munich, Germany

² University of Siegen, Siegen, Germany

approach and an implicit surface rendering framework that also includes implementations of other common rendering approaches, namely LG-Surfaces [29], Sphere Tracing [26], Adaptive Marching Points [44], a local quartic polynomial approximation process [43], Segment Tracing [21] and a simple ray marching approach.

Limitations. Our method is not well-suited for functions that are defined piecewise, e.g., discrete Signed Distance Fields, due to their large number of discontinuities in some derivative terms, which involve many recursions and high polynomial degrees, yielding a significant drop in runtime performance.

2 Related work

Implicit surfaces are described based on a position in space $\mathbf{x} \in \mathbb{R}^3$ and an arbitrary scalar field $S(\mathbf{x})$ [6], where the surface is some iso-contour of S , commonly $S(\mathbf{x}) = 0$. We distinguish three distinct ways to visualize implicit surfaces: (i) polynomial approximations that locally approximate the surface using piecewise polynomials of various degrees, (ii) rendering approaches that require prior knowledge of the Lipschitz constant of the scalar field, either locally or globally, and (iii) interval refinement methods using root isolation techniques.

Polynomial approximations. For polynomial scalar functions along rays up to degree 4 closed-form solutions can be applied [24], while for higher-degree polynomials no general analytic solutions can exist. Instead, for higher degree, and/or piecewise polynomial surfaces, numerical root finding approaches can be utilized, e.g., Bézier clipping [30, 38]. For arbitrary scalar fields, approximating polynomials can be constructed along a ray, to which various root finding methods are applied [28, 43], involving, e.g., Taylor expansion [45] or Bernstein polynomials [30]. However, increasing the degree of a monomial polynomial arbitrarily does not necessarily yield a better approximation due to Runge's phenomenon [41], which is a fundamental limitation when constructing polynomial approximations using interpolation methods in existing rendering approaches. Our method addresses these limitations by utilizing Chebyshev polynomials that do not suffer from Runge's phenomenon and which can scale to much greater numerical accuracy for arbitrary implicit surfaces.

Lipschitz constant-based approaches. Knowing the Lipschitz continuity of the scalar function and its Lipschitz constant k significantly simplifies the root finding procedure. Kalra and Barr [29] utilized the Lipschitz continuity of the surface function and of the gradient of the surface function to construct an approach in which surfaces are guaranteed to be intersected. If only a global Lipschitz constant is known, sphere tracing [26] can be applied, which provides a robust render-

ing approach that iterates along a ray based on the current function magnitude. Several extensions have been proposed to sphere tracing, e.g., using overrelaxation [32], locally defined Lipschitz constants [22], or nonlinearly deformed signed distance fields [42]. Sphere tracing-like approaches can, however, be arbitrarily slow for grazing rays, which can be diminished using local Lipschitz constant, e.g., using Segment tracing [21]. Our method circumvents these limitations by not relying on a known Lipschitz constant.

Interval refinement. For arbitrarily defined polynomial functions a robust ray tracing approach is to utilize interval-based approaches [14, 37], where the main goal is to perform a root isolation, i.e., finding an interval along the ray that contains a single root, notably the first root for surface rendering. This can be done in a variety of ways, e.g., using Budan's theorem [16], Descartes' rule [25] or Sturm's theorem [48], but these approaches generally rely on the surface function being polynomial or monomial polynomial approximations of the surface function that may not be accurate for challenging surface functions. Several improvements over these approaches have been proposed in the past, e.g., using affine arithmetics to obtain tighter interval bounds [19], Taylor tests for root bracketing [40] or GPU-based algorithms [34]. Recently, Keeter [31] proposed an interval refinement approach that realizes a hierarchical approach to render implicit surfaces involving thousands of arithmetic operations efficiently on GPUs. Our method is able to work with traditional arithmetic operations, while preserving numerical robustness, by directly incorporating numerical error approximations into the rendering procedure.

3 Chebyshev proxy rendering

Rendering an implicit surface function $s(\mathbf{x})$ can be formulated as finding intersections along a ray $\mathbf{r}(\lambda) = \mathbf{o} + \lambda \vec{\mathbf{d}}$, that are cast from a viewpoint $\mathbf{o}$ in direction $\vec{\mathbf{d}}$, where the intersections are the roots along r . Consequently, we can reformulate the problem of intersection testing along rays as a root search problem for an arbitrary unary function $f : \mathbb{R} \rightarrow \mathbb{R}$. More specifically, for rendering problems we are only interested in intersections along a compact interval, i.e., intersections in front of the camera up to some maximum depth $b < \infty$ with a minimum distance $a \geq 0$, which means that we need to find roots of an arbitrary function $f : [a, b] \rightarrow \mathbb{R}$. However, finding roots of arbitrary functions is a challenging problem and, accordingly, we aim to transform the problem into a form that can be readily handled.

Our approach comprises five separate steps as

- (a) Constructing a Chebyshev proxy f_p , which allows for robust root finding, see Sect. 3.1.

- (b) Finding the roots of the Chebyshev proxy f_p , see Sect. 3.2.
- (c) Improving the pre-conditions for the approximation accuracy of f_p to enhance the numerical accuracy of the root finding for f , see Sect. 3.3.
- (d) Post-processing steps to refine the estimated root, see Sect. 3.4.
- (e) A recursive process to improve numerical accuracy in cases where extreme numerical conditions lead to visual artifacts, see Sect. 3.5.

3.1 Chebyshev proxy construction

In $\mathbb{R}$ the Weierstrass Approximation Theorem [49] guarantees the existence of a proxy function f_p for every C^0 continuous function f over a compact interval $[a, b] \subset \mathbb{R}$ with arbitrary precision ϵ , i.e., $\|f - f_p\|_\infty \leq \epsilon$, using a polynomial function of finite degree n , depending on ϵ . To construct f_p , we utilize Chebyshev polynomials as these do not suffer from Runge's phenomenon [41], i.e., increasing the degree results in better approximations, and as they can be efficiently constructed using DFTs. In general, Chebyshev basis polynomials of the first kind T_n , of degree n , are recursively defined via

$$T_0(x) = 1; \quad T_1(x) = 2x; \quad T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x),$$

Based on this orthogonal basis over $[-1, 1]$, a Chebyshev polynomial $p(x) = \sum_{i=0}^n c_i T_i(x)$, with Chebyshev coefficients $C = \{c_0, \dots, c_n\}$, can be constructed, which can be used for approximating a C^0 -function f . The corresponding coefficients C are calculated by sampling f on a Chebyshev-Lobato and performing a discrete Fourier transform (DFT) with $\mathcal{O}(n \log n)$ time complexity [8]. Sampling at the Chebyshev-Lobato grid, defined over $[-1, 1]$ as $x_k = -\cos(\frac{k\pi}{n})$, $k = 0, 1, \dots, n$, guarantees uniform convergence for analytic functions with an approximation error $\mathcal{O}(e^{-n\mu})$ [4] in the mathematical sense, with n being the polynomial degree and μ being a positive constant. Note, that arbitrary compact finite intervals can be transformed to the interval $[-1, 1]$ via an affine transformation.

While the Weierstrass Approximation Theorem prescribes the existence of a polynomial degree $n_{\epsilon_u}^{\text{opt}}$ with an L_∞ approximation error of ϵ for $f : [a, b] \rightarrow \mathbb{R}$, there is no general approach to find this lower bound degree. Therefore, heuristic approaches, e.g., [1, 3, 7, 9, 15, 18, 27, 35, 36, 46], exist to determine $n_\epsilon \approx n_{\epsilon_u}^{\text{opt}}$, e.g., using the interstitial error [12] as an approximation of the L_∞ error.

In our proposed method, we utilize the approach of Aurentz and Trefethen [1] that employs the round-off plateau behavior of the Chebyshev coefficients, and has been well tested for a large variety of functions and is integrated into the widely utilized Chebfun library [17]. Using n_ϵ , a uniquely

determined proxy function f_p exists as

$$f_p = \sum_{i=0}^{n_\epsilon} c_i T_i, \quad \text{with } \|f - f_p\|_\infty \lesssim \epsilon_u. \quad (1)$$

Machine Precision & Discretization. At this point, it is important to note that in computer science we always operate on the finite set $\mathbb{M}$ of machine numbers and that we deal with the machine approximation $f_m : \mathbb{M} \rightarrow \mathbb{M}$ of the “real” function f given by the numerical implementation. As we do not address this level of approximation, we assume $f_m = f$, i.e., f_m represents the “real” function whose nearest root we are looking for. We, therefore, use f and f_m for analytic aspects with infinite precision and numerical aspects with finite precision. Moreover, there are limits in transferring “infinitely precise” analytical concepts to finite precision machine operators and representations, which we will address in the following. Specifically, there is always an underlying limitation relating to the achievable computational precision, e.g., based on the number and magnitude of coefficients.

Several heuristics exist to estimate this approximation accuracy limit ϵ_a for Chebyshev proxys [12, 39], where we utilize the estimated proposed by Boyd [12] given as

$$\epsilon_a = n^2 \epsilon_{\text{machine}} \underbrace{\max_{i \in \{0, \dots, n\}} |c_i|}_{\text{mmc}(f_p)} \leq \|f_p - f_m\|_\infty, \quad (2)$$

where $\text{mmc}(f_p)$ is the maximum magnitude coefficient.

For our proxy construction process, we utilize the adaptive approach of Boyd [12] and the heuristic chopping process of Aurentz and Trefethen [1], which tries to achieve $\|f_m - f_p\|_\infty \leq \epsilon_u$ by appropriately setting the threshold for the round-off plateau detection. This round-off plateau is the result of the difference in magnitude of coefficients being limited by the machine precision ϵ_m , as any coefficient smaller than $\text{mmc}(f_p)/\epsilon_m$ will (most likely) not affect the result of machine precision operations. Therefore, ϵ_u can not be chosen below a value that depends on $\text{mmc}(f_p)$ and the approximation error is bound on both sides

$$\epsilon_a \leq \|f_m - f_p\|_\infty \leq \epsilon_u, \quad (3)$$

where both limits depend on $\text{mmc}(f_p)$ [1, 12]. $\text{mmc}(f_p)$ is considered as being proportional to the magnitude of the function [12]

$$\text{mmc}(f_p) \propto \text{mag}_{f_m}(a, b), \quad (4)$$

where mag denotes the absolute magnitude of the function over the given interval range. Under the assumption of Eq. 4,

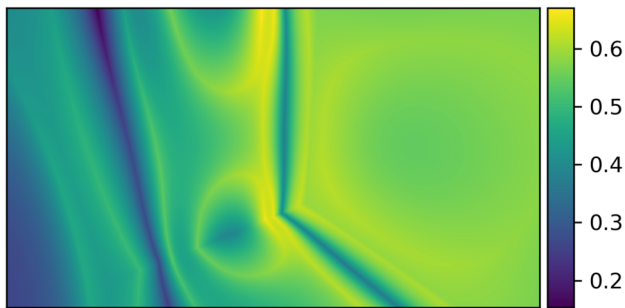


Fig. 1 The logarithmic per-pixel ratio $\log_{10}(\text{mmc}(f_p)/\text{mag}_{f_p}(a, b))$ for the Bohemian star surface (Fig. 2b), estimated using the proxy

ϵ_a and ϵ_u have to strictly relate to $\epsilon_m \text{mag}_{f_m}(a, b)$, i.e., $\epsilon_u < \epsilon_m \cdot \text{mag}_{f_m}(a, b)$ can, in general, not be achieved [12].

Since this proportionality is semi-well understood for many cases, we evaluated the proportionality hypothesis for all test functions to ensure that this also applies to rendering problems. The determination of the maximum magnitude $\text{mag}_{f_m}(a, b) = \|f_m|_{(s,t)}\|_\infty$ is challenging for an arbitrary function, as it requires root finding for derivatives. We, therefore, evaluate the assumption *indirectly* using the maximum magnitude $\text{mag}_{f_p}(a, b)$ of the proxy f_p by applying QR-based root search on the Chebyshev proxy's derivative (see Sect. 3.2). We evaluated $\log_{10}(\text{mmc}(f_p)/\text{mag}_{f_p}(a, b))$ for all 16 functions from Fig. 2 and found that the maximum difference of 0.2, i.e., a difference factor of 5, which occurs for the Bohemian Star test function (Fig. 2b), as shown in Fig. 1. Consequently, we assume that the proportionality is valid for ray casting polynomial surfaces, rational surfaces, analytic algebraic surfaces, algebraic surfaces, and non-algebraic surfaces, which we use for testing (see Sect. 4).

3.2 Root finding for chebyshev proxies

After constructing the proxy function f_p , we now need to find the roots of f_p for the intersection tests. This can be achieved using the companion matrix M , which is the uniquely defined matrix that has f_p as its characteristic polynomial. Consequently, the real Eigenvalues of M are the real roots of p . Commonly, for Chebyshev polynomials the Eigenvalues of the companion matrix are computed via a QR algorithm [47], which estimates roots of the polynomial simultaneously in time complexity $\mathcal{O}(n^3)$ [11]. Due to this computational complexity and the limited numerical accuracy for large matrices, reducing the polynomial degree has clear benefits. However, simply excluding higher polynomial terms is not feasible, as it would negatively affect the approximation accuracy ϵ_p .

We utilize the subdivision approach proposed by Boyd [10, 11] that reduces the polynomial degree during the calculation. This approach relies on the concept that, in general, a function f_m on $[a, b]$ approximated by f_p with degree

n can be alternatively approximated on the intervals $[a, d]$ and $[d, b]$, with $d \in (a, b)$, with lower degree and similar approximation accuracy. This subdivision process is especially useful for functions f that are C^∞ everywhere on $[a, b]$ besides a few C^k points. In this case, the subdivision ultimately excludes the C^k points and yields intervals over which f is C^∞ and for which the polynomial degree for approximation is significantly reduced [12].

More precisely, $f_m : [a, b] \rightarrow \mathbb{M}$ is transformed to the canonical interval $[-1, 1]$ using an affine mapping [12, 17]. $[-1, 1]$ is split at the heuristically chosen value $d = -0.004849834917525$ [17], and two new approximation functions f_p^l and f_p^r are determined over $f_p|_{[-1,d]}$ and $f_p|_{[d,1]}$, respectively. The new proxies f_p^l and f_p^r are constructed by evaluating f_p on a Chebyshev-Lobatto grid of degree n , equal to the degree of f_p , on each sub-interval, with time complexity $\mathcal{O}(n^2)$, and applying a DFT, with time complexity $\mathcal{O}(n \log n)$, to find the respective coefficients [17]. See, Boyd and Gally [13] for further numerical properties of Chebyshev root searching.

3.3 Improving pre-conditions

As discussed in Sect. 3.1, the error bounds ϵ_a and ϵ_u both depend on the maximum magnitude coefficient of the polynomial proxy $\text{mmc}(f_p)$ and, as $\text{mmc}(f_p) \propto \text{mag}_{f_m}(a, b)$, they both depend on the maximum magnitude of f_m . It is important to keep in mind that uniformly scaling the function, e.g., searching for roots of $g : x \rightarrow \frac{x}{1000} f(x)$, does not improve the root searching process as the errors are still bound in the same relative ratio to the maximum magnitudes. Consequently, to achieve smaller ϵ_a and ϵ_u we have to either exclude regions of large relative magnitude from the root search interval to reduce $\text{mmc}(f_p)$, or regions that are difficult to approximate using Chebyshev proxies, i.e., regions with C^0 locations, that require high Chebyshev proxy degrees.

Using the fact that $f_m : [a, b] \rightarrow \mathbb{M}$ has an (unknown) machine Lipschitz constant, yields that $\text{mag}_{f_m}(a, b)$ can be reduced by lowering $f_m(a)$, $f_m(b)$ and $b - a$. Reducing $f_m(a)$ and $f_m(b)$ can be efficiently achieved by excluding segments on either end of the interval that do not contain the first root of the function. Moreover, reducing the root search to a sub-interval $[a_s, b_s] \subsetneq [a, b]$ always reduces the interval length $b - a$.

As we aim to find the first root ξ_0 , we shift f_m by the approximation error ϵ_a of f_p to guarantee that ξ_0 will not be missed when using f_p for root search. This also holds, if ξ_0 is a double root of f_m . The shifted function $f_p^\epsilon : [a, b] \rightarrow \mathbb{M}$ is constructed by directly modifying the Chebyshev coefficients

c_i^p of the proxy function f_p of f_m :

$$f_p^\epsilon(x) = \sum_{i=0}^n c_i^\epsilon T_i(x), c_i^\epsilon = \begin{cases} c_i^p, & i \neq 0, \\ c_i^p - s \operatorname{sgn}(f_m(a))\epsilon, & \text{else} \end{cases}. \quad (5)$$

Note that if $|f_p(a)| \leq \epsilon$ then a was already the approximated first root and the process can be stopped.

Performing this shift requires estimating $\epsilon_a = \|f_m - f_p\|_\infty$, which we achieve using Eq. 2. Subsequently, we shift f_p by $s \cdot \epsilon_a$, where empirical experiments reveal that $s = 100$ is sufficient in all our test cases, i.e., no roots of f_m are missed.

However, this shift is not without drawbacks. The first root $\tilde{\xi}_0$ of f_p^ϵ is an estimate for the actual first root ξ_0 of f_m that is located left of ξ_0 . Depending on the shift value $s \cdot \epsilon_a$, this offset may be large. For example, $f(x) = e^{-x} - 0.5$, $x \in [-100, 100]$, yields an estimated approximation error of $\epsilon_a \approx 5.9 \cdot 10^{27}$. Moreover, the shift can generate a pseudo-root, i.e., $\tilde{\xi}_0$ does not correspond to ξ_0 . Consequently, we may have $\tilde{\xi}_0 \ll \xi_0$ in specific cases. These problems are addressed in our post-processing (Sect. 3.4) and in our recursive approach (Sect. 3.5).

3.4 Post-processing

Given a root $\tilde{\xi}$ of the proxy f_p , or f_p^ϵ , two main categories of errors exist that we try to diminish via post-processing: (i) errors on the proxy, i.e., $\tilde{\xi}$ may not be the exact first root of f_p , and (ii) errors of the approximation, i.e., $\tilde{\xi}$ may not be the exact first root of f_m . We utilize two separate approaches, i.e., iterative refinement using Newton's method and an incremental shifting.

Iterative Refinement. Newton's method is commonly used to polish estimated roots of polynomial functions [12]. This approach improves the estimate of the first root $\tilde{\xi}^0 = \tilde{\xi}$ by iteratively estimating a new approximate root $\tilde{\xi}_{n+1}$ for f_m , as

$$\tilde{\xi}_{n+1} = \tilde{\xi}_n - \frac{f_m(\tilde{\xi}_n)}{f'_m(\tilde{\xi}_n)}, \quad (6)$$

until a convergence criterion has been reached, e.g., $|f_m(\tilde{\xi}_{n+1})| \leq \epsilon$, which can be applied either to the proxy f_p or f_m itself.

Applying Newton's method to f_p is straightforward [12] as the derivative of f_p can be calculated directly from the coefficients of f_p , but it does not necessarily improve the root's accuracy relative to f_m as the approximation accuracy for f_m by f_p remains unchanged (see also Sect. 4). Applying Newton's method to f_m directly requires a derivative estimate, e.g., using finite differences, which may not be sufficiently accurate to achieve convergence. Moreover, there

is no guarantee that starting the Newton process at $\tilde{\xi}$ will converge toward the corresponding root of f_m , i.e., it may diverge or converge to a different root, potentially outside the search interval.

Considering the shifted function f_p^ϵ , the approximate root $\tilde{\xi}^\epsilon$ is generally left of the first root ξ of f_m , if $\tilde{\xi}^\epsilon$ is no pseudo-root. Accordingly, Newton's method can yield significant improvements, but it may, depending on the specific constellation, also reduce the accuracy considerably, see Sect. 4. **Incremental Shifting.** This approach successively evaluates $\tilde{\xi}_+$ or $\tilde{\xi}_-$, i.e., the previous or next number in machine precision to $\tilde{\xi}$, to improve the root estimate of f_m , or f , i.e.,

$$\tilde{\xi}^{i+1} = \begin{cases} \tilde{\xi}_+^i, & |f(\tilde{\xi}_+^i)| < |f(\tilde{\xi}^i)|, \\ \tilde{\xi}_-^i, & |f(\tilde{\xi}_-^i)| < |f(\tilde{\xi}^i)|, \\ \tilde{\xi}^i, & \text{else,} \end{cases} \quad (7)$$

which is repeated until $\tilde{\xi}^{i+1} = \tilde{\xi}^i$. This process is guaranteed to improve the error of the root estimate, but it may get stuck in a local minimum. Applying this process to f may incur significant computational costs, as the offset $\xi - \tilde{\xi}$ can be large and, thus, may require $\gg 10^6$ iterations, while applying it to f_p^ϵ requires only a few iterations. Therefore, our root finding process applies incremental shifting to f_p only, see Sect. 3.5.

3.5 Recursive proxy root finding

The main objective in rendering implicit surfaces is the visual quality and correctness; see Sect. 4.2.2. It is, however, hardly possible to robustly determine qualitative failure cases by algorithmic or numerical means. Specifically, locally nearly flat or multiple roots of the per ray scalar function can cause strong visual artefacts; see Sect. 4.3.

We, therefore, propose a manually activatable root searching algorithm that recursively combines the pre-conditioning via shifting (Sect. 3.3), the proxy root finding using Chebyshev approximation (Sect. 3.1), and post-processing refinement procedures (Sect. 3.4), as depicted in Algorithm 1. This recursive procedure works by splitting the initial search interval into more granular intervals that can be processed at higher approximation accuracies. As we are only interested in finding the first root along a ray, we perform this recursion in a depth-first manner, i.e., we evaluate the left-most intervals first, as we can stop the entire process once the first root of f_m has been found.

We search for sub-intervals where f_p provides a more accurate approximation to f_m and that potentially contains a root. For example, given a function f_m over $[a, b]$, with $f_m(a) = 100$ and an approximation accuracy of $\epsilon = 1$, then, instead of searching for roots of f_m directly, searching for intervals where $f_m(x) \leq \epsilon$ and subsequently

re-approximating f_m over these intervals, yields a more accurate and efficient root search in most cases.

To perform this interval division, we shift f_m by the approximation error of f_p over the entire interval by

$$\sigma = s \max \left\{ \epsilon_u, n^2 \epsilon_m \text{mmc}(f_p) \right\}, \quad (8)$$

where $\epsilon_a = n^2 \epsilon_m \text{mmc}(f_p)$ (see Eq. 2) is a lower bound for $\|f_p - f_m\|_\infty$ and s is set to 100; see also Sect. 3.3. No root for f_p^σ is a root of f , but indicates a location at which $|f_p|$ drops below the threshold $|\sigma|$. The roots of f_p^σ define sub-intervals with lower magnitudes, and, thus, can increase the approximation accuracy. For now, we assume that $f_m(a) > \sigma$.

Algorithm 1: Recursive root finding algorithm for $f : [a, b] \rightarrow \mathbb{M}$ using a user-defined precision threshold parameter ϵ_u and a global scaling parameter s .

Procedure FirstRootRec(f_m, ϵ_u, l, r)

// Proxy construction (Sec. 3.1)

$\{c_i^p\} \leftarrow \text{ChebApprox}(f_m, l, r, \epsilon_u)$

$\epsilon_a \leftarrow n^2 \epsilon_m \max\{f_m(a), f_m(b), \text{mmc}(f_p)\}$

If $\epsilon_a \leq \epsilon_u$

// Root finding incl. splitting (Sec. 3.2)

$\mathcal{R}^\sigma \leftarrow \text{RootsQRCompanionMatrix}(c_i^p, l, r)$

If $|\mathcal{R}| > 0$ // Eq. 3

$\tilde{\xi}_0 \leftarrow \text{first root in } \mathcal{R}$

return PostProcess($\tilde{\xi}_0$) // Sec. 3.4

else if return no root

// Shifting & reduction of intervals

$\sigma \leftarrow c_e^0 - \text{sgn}(f(a))s \max\{\epsilon_a, \epsilon_u\}$ // Eq. 5

$\tilde{c}_0^p \leftarrow c_0^p - \sigma$ // f_p becomes f_p^σ due to σ -shift

$\tilde{\mathcal{R}}^\sigma \leftarrow \text{RootsQRCompanionMatrix}(\tilde{c}_i^p, l, r)$

If $|\tilde{\mathcal{R}}^\sigma| = 0$ **return no root**

Else If $|\tilde{\mathcal{R}}^\sigma| = 1$

$\tilde{\xi}_0^\sigma \leftarrow \text{first root in } \tilde{\mathcal{R}}^\sigma$

return FirstRootRec($f_m, \epsilon_u, \tilde{\xi}_0^\sigma, r$)

Else // $|\tilde{\mathcal{R}}^\sigma| \geq 2$

$\{\tilde{\xi}_0^\sigma, \tilde{\xi}_1^\sigma\} \leftarrow \text{first two roots in } \tilde{\mathcal{R}}^\sigma$

$\tilde{\xi} \leftarrow \text{FirstRootRec}(f_m, \epsilon_u, \tilde{\xi}_0^\sigma, \tilde{\xi}_1^\sigma)$

If root found: return $\tilde{\xi}$

Else If $|\tilde{\mathcal{R}}^\sigma| = 2$: **return no root**

Else

return FirstRootRec($f_m, \epsilon_u, \tilde{\xi}_1^\sigma, r$)

If no root was found for f_p^σ then no root can exist for f_m , as the function remains above the threshold σ everywhere in absolute magnitude and has no sign changes. If a single root $\tilde{\xi}_0$ is found for f_p^σ , this marks the transition from $|f_p| > \sigma$ to $|f_p| < \sigma$. In such cases, a new root search can be performed on the interval $[\tilde{\xi}_0, b]$ as $[a, \tilde{\xi}_0]$ can not contain a root of f_m . If two roots $\tilde{\xi}_0$ and $\tilde{\xi}_1$ are found for f_p^σ , $|f_p| < \sigma$ is only valid between $\tilde{\xi}_0$ and $\tilde{\xi}_1$. Consequently, a root of f_m can only occur

in this interval, and a new root search is performed on $[\tilde{\xi}_0, \tilde{\xi}_1]$. Finally, if there exist three or more roots $\{\tilde{\xi}_0, \tilde{\xi}_1, \tilde{\xi}_2, \dots\}$ for f_p^σ then, analogously, roots can occur on the intervals $[\tilde{\xi}_0, \tilde{\xi}_1]$ and $[\tilde{\xi}_2, b]$, where a root search is performed on each sub-interval.

Overall, this process recursively narrows down onto the first root, as given in Algorithm 1. As the approximation gets more accurate with each subdivision, it would be possible to reduce the scaling parameter s in each iteration. However, there was no practical benefit in our experiments and, thus, we use $s = 100$ in all cases.

In the case of $|f_m(a)| \leq \sigma$, the argumentation above is inverted, e.g., if a single root $\tilde{\xi}_0$ was found then a root exists in the interval $[a, \tilde{\xi}_0]$ instead of $[\tilde{\xi}_0, b]$. However, if no roots were found and $f(a) \leq \sigma$, then we assume that a already was a root of f_m .

4 Evaluation

Implementation and system. We implemented our proposed approach, including the adaptive Chebyshev approximation, in an open source C++ program included in the supplementary material. We perform our evaluations on a system with an AMD Ryzen 3950X CPU, an Nvidia 3090 RTX GPU and 64 GiB of host memory. The actual rendering performed on the CPU using double precision arithmetic, while the GPU is utilized only to visualize the results. Note that there are no technical limitations that would prevent implementing our procedure on a GPU, however, due to limitations regarding double precision arithmetic performance on consumer GPUs, such an implementation would not be widely useful. All images are ray traced at a 1920×1080 pixel resolution with no antialiasing or subsampling applied.

Test functions. The 17 implicit surfaces used for the evaluation are shown in Figs. 2 and 7. Their definitions are given in Appendix B, and meta information for the functions is provided in Table 1. The test surface functions are of five different classes. The classes are (a) *polynomial* surfaces (C^∞ everywhere) that can be represented exactly using Chebyshev proxies, (b) *rational* surfaces (C^∞ outside sparse singularities), e.g., using division operations, (c) *analytic algebraic* surfaces (C^∞), (d) *algebraic* surfaces (C^∞ outside sparse singularities), utilizing, e.g., the square root function, and (e) *non-algebraic (transcendental)* surfaces that may be only C^0 in large regions of the image, e.g., using max and abs.

Evaluation procedure. The evaluation consists of three distinct parts: (1) Evaluation of the proposed rendering approach on its own. This comprises the evaluation of (1.a) the required polynomial degrees and the resulting coefficient distributions (Sect. 4.1.1), and (1.b) the computational performance and effects of post-processing (Sect. 4.1.4). (2) Comparisons of

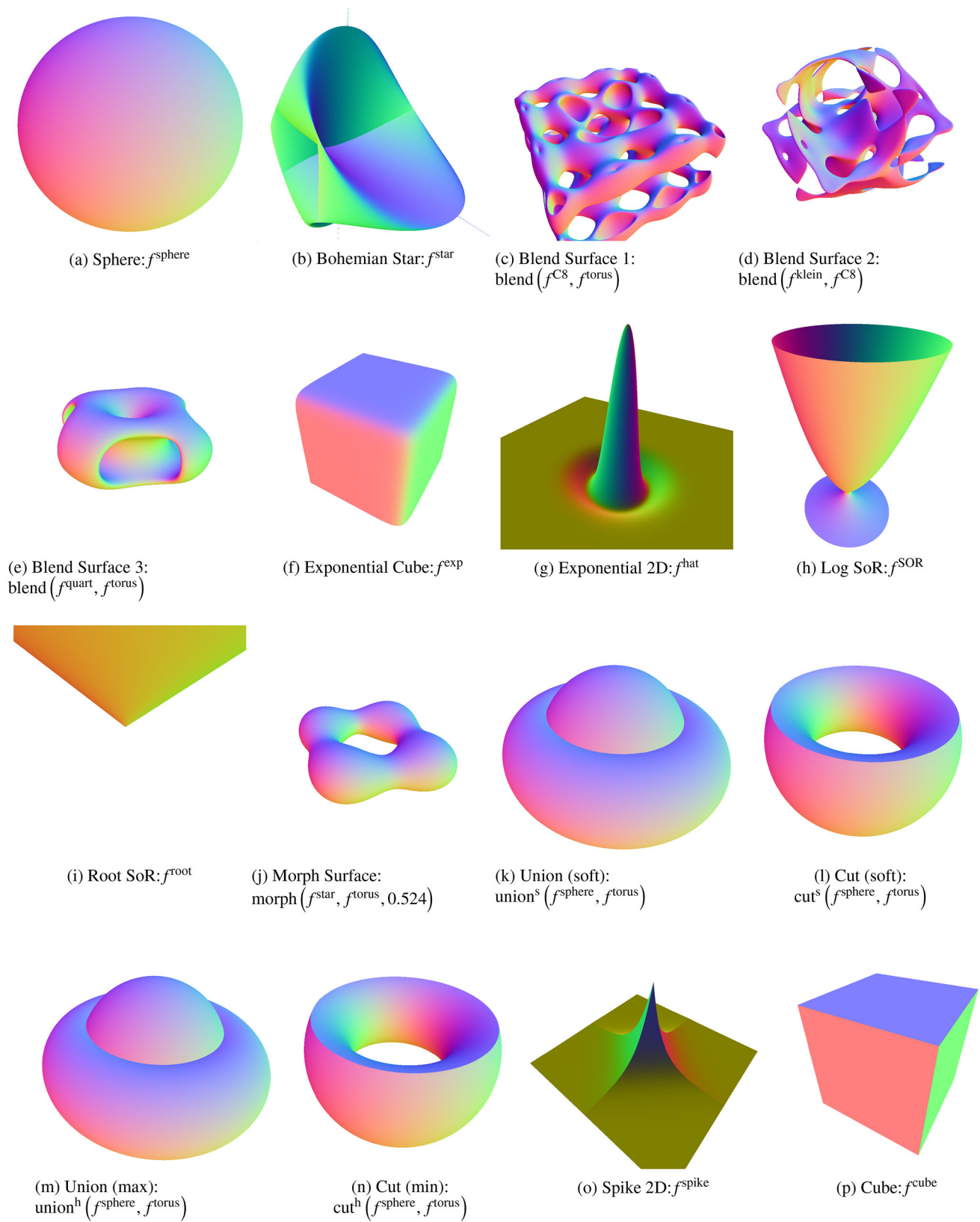


Fig. 2 Evaluated surfaces, normals color coded

the proposed rendering approach to existing rendering techniques, including (2.a) a discussion of the various visible failure modes occurring in rendering of implicit surfaces in Sect. 4.2.1, and, (2.b) a quantitative comparisons against baseline methods separated by the five categories of implicit surfaces (Sect. 4.2.2). (3) An evaluation of our recursive root searching approach applied to the Bohemian Star function in Sect. 4.3.

Precision control. Our proposed rendering technique depends on the user desired precision parameter ϵ_u^* , which transforms to $\epsilon_u = 10^{\epsilon_u^*} \epsilon_m$ (see Sect. 3.1). While $\epsilon_u^* = 0$ would be ideal in all cases, the process is bound by the approximation accuracy ϵ_a (see Eq. 3). Thus, too small ϵ_u^* enforces infeasible high polynomial degrees. We, therefore, apply a heuristic by choosing ϵ_u^* based on the portion of the image that is not C^∞ , as these regions require “allocate” the majority of computational resources. For surface functions that are C^∞ (besides a few points; Fig. 2j), we set $\epsilon_u^* = 1$, for functions with discontinuities in small regions, we set $\epsilon_u^* = 4$, and for functions with discontinuities in most of the image, i.e., the cube function (see Fig. 2p), we set $\epsilon_u^* = 6$. Note that for polynomial functions of low degrees, e.g., the sphere function (Fig. 2a), the underlying function can be exactly represented using a Chebyshev proxy f_p and thus the choice of ϵ_u^* does not matter.

4.1 Evaluation of the proposed method

We evaluate our method regarding different properties specific to our method, focusing on both numerical aspects, e.g., the polynomial degrees required for different functions, as well as computational aspects, e.g., the computational cost of constructing Chebyshev proxies.

4.1.1 Polynomial degree distributions

Regarding the polynomial degree of the proxy f_p , we need to consider both, the maximum degree of all pixels, which requires the “highest approximation effort” for a pixel-ray, and the distribution of the degrees, which relates more to the average approximation effort for a surface. As mentioned before, using a Chebyshev-Lobato grid guarantees uniform convergence for analytic functions with an approximation error $\mathcal{O}(e^{-n\mu})$ [4], with n being the polynomial degree and μ being a positive scaling coefficient and approximating non-analytic functions, i.e., functions that are not C^∞ everywhere, is more challenging [12]. Consequently, functions that are polynomial, e.g., a sphere, require only few coefficients to be approximated, whereas C^0 -continuous non-algebraic functions require significantly more coefficients.

Table 1 describes the distribution of polynomial degrees across different surface functions, revealing the strong variation in the required polynomial degree. In general, we observe

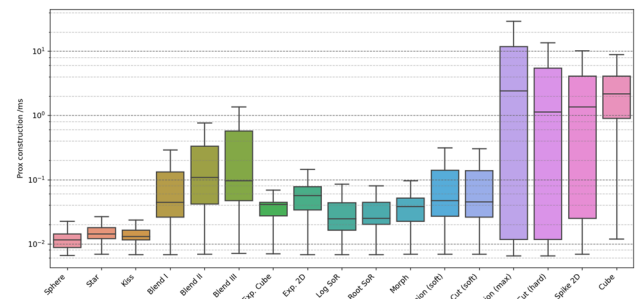


Fig. 3 Computational cost of the proxy construction in milliseconds per ray for our proposed rendering technique

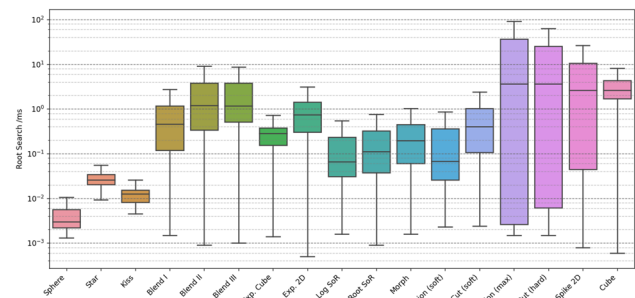


Fig. 4 Computational cost of the root search, excluding proxy construction, in milliseconds per ray for our proposed rendering technique

that the polynomial surfaces require only very low polynomial degrees, as they can be represented exactly using a Chebyshev proxy, which is in line with the expected behavior. Analytic surfaces require moderate polynomial degrees everywhere, with a relatively low ratio of the maximum polynomial degree to the median polynomial degree, which is in line with the expected behavior as the function is only difficult to approximate in some regions. Rational surfaces require higher polynomial degrees everywhere and exhibit a much higher ratio of maximum to median polynomial degree. This behavior is exacerbated for algebraic surfaces, e.g., the *Morph* surface (Fig. 2j) exhibits a relatively low median degree of 25, but requires at worst a degree 5755 polynomial, where the high degree polynomials occurs for rays close to the four points in which the function is C^0 -continuous, see Fig. 2j. Finally, for non-algebraic surfaces the overall required polynomial degree increases significantly with similar outliers in the maximum polynomial degree, e.g., the *Cut (min)* surface (Fig. 2n) requires high degree approximations along the seam between the two input surfaces, shifting the overall median degree up to 508, while yielding a significantly lower approximation accuracy than the *Morph* surface Fig. 2j.

Table 1 Meta information for the test surfaces: User desired precision ϵ_u in terms of machine precision ($\epsilon_u = 10^{\epsilon_u} \epsilon_m$), the estimates of the dynamic range dr and the Lipschitz constant k using a polynomial proxy. Coefficients: The number of polynomial coefficients of the proxy function as median (med), lower (Q_1) and upper (Q_3) quartile, and maximum (max) over all pixels

		Meta-Information				Coefficients			
		Figures	ϵ_u^*	dr	k	Q_1	med	Q_3	max
Poly	Figure 2a	1	1	6.83984	5.0727	3	3	3	3
	Figure 2b	1	1	15.3801	75637.2	9	9	9	9
	Figure 7	1	1	11.0607	3270.72	6	6	6	6
Rational	Figure 2c	1	1	6.24209	0.0930553	18	37	119	365
	Figure 2d	1	1	6.55949	0.576778	31	115	467	1619
	Figure 2e	1	1	5.4152	0.736367	39	109	484	2758
Analytic	Figure 2f	1	1	10.6895	2.09062e+06	22	28	33	37
	Figure 2g	1	1	7.49514	7.83121	37	56	76	102
	Figure 2h	1	1	6.10179	8.33593	10	14	31	84
Algebraic	Figure 2i	4	4	5.82641	1.94538	11	19	38	13506
	Figure 2j	1	1	7.05396	66.9928	14	26	53	5756
	Figure 2k	4	4	6.82806	3.62877	19	40	145	16100
Non-alg	Figure 2l	4	4	7.31895	65.8066	18	36	130	7949
	Figure 2m	4	4	7.61522	12.531	3	508	2171	7853
	Figure 2n	4	4	8.03372	63.233	5	295	1274	4780
	Figure 2o	4	4	6.38274	8.0803	11	335	936	12024
	Figure 2p	6	6	6.25081	1.65233	290	487	601	3910

Table 2 Accuracy evaluation of some example functions. $\tilde{\xi}_N$ is the result of applying Newton's method to $\tilde{\xi}$ on f_m . Q_1 and Q_3 denote the lower and upper quartile values and med the median for the *signed*

distribution of errors. The max value denotes the *absolute* maximum magnitude value and its sign

Fn.	$f(\tilde{\xi})$				$f(\tilde{\xi}_N)$			
	Q_1	Med	Q_3	max	Q_1	Med	Q_3	max
Sphere: f_p	2.22e−16	4.44e−16	6.66e−16	2.22e−15	−1.11e−16	0.00e+00	0.00e+00	8.88e−16
Sphere: f_p^ϵ	5.35e−12	5.81e−12	6.59e−12	1.28e−11	−1.11e−16	0.00e+00	0.00e+00	8.88e−16
Blend II: f_p	−5.90e−17	8.67e−18	8.33e−17	1.48e−15	−4.16e−17	−6.94e−18	1.39e−17	−1.18e−15
Blend II: f_p^ϵ	7.52e−11	1.67e−10	3.45e−10	−1.18e−16	−1.39e−17	1.69e−19	6.25e−16	5.09e−13
Union (soft): f_p	−1.37e−10	−1.98e−11	6.31e−11	−8.76e−04	−1.11e−16	0.00e+00	1.11e−16	7.51e−01
Union (soft): f_p^ϵ	7.50e−09	1.91e−08	7.73e−08	−8.76e−04	0.00e+00	1.11e−15	5.02e−14	7.46e−01

4.1.2 Proxy construction

The computational costs of the proxy construction are mostly determined by the sampling of the surface function on a Chebyshev–Lobatto grid and the subsequent DFT. In our proposed rendering approach, we utilize the adaptive Chebyshev proxy construction of Boyd [12], where the function is evaluated on Chebyshev–Lobatto grids of size $2^k + 1$, starting with $k = 4$, and performing a subsequent chopping of the coefficients using the approach of Aurentz and Trefethen [1]. Performing the DFT of the Chebyshev–Lobatto has time complexity $\mathcal{O}(n \log n)$, independent of the computational complexity of f . Accordingly, the computational requirements of constructing the proxy function are tightly coupled to the polynomial degree of f_p . Furthermore, as our test surface functions are computationally efficient to evaluate, the DFT is dominant in the overall computational require-

ments of the proxy construction in our experiments. For an overview of the computational requirements for each method see Fig. 3, and for more detailed results for each individual implicit function, please see the supplementary material.

4.1.3 Root finding

Regarding the computational performance of the root searching, we observe that the overall computational cost mainly depends on the polynomial degree of f_p and not on the underlying surface function. However, the overall computational costs are significantly higher due to the computational complexity of the QR decomposition of $\mathcal{O}(n^3)$, which is dominating the $\mathcal{O}(n \log n)$ for the DFT in the proxy construction. For an overview of the computational requirements for each method see Fig. 4, and for more detailed results for

each individual implicit function, please see the supplementary material.

4.1.4 Post-processing

To assess the impact of the post-processing, we investigate three exemplary functions, i.e., the polynomial surface *Sphere*, the rational surface *Blend II*, and the algebraic surface *Union (soft)*, by evaluating the implicit surface function f at the estimated roots $\tilde{\xi}$ and $\tilde{\xi}^\epsilon$ found for the proxy f_p and the shifted proxy f_p^ϵ function, respectively, see Table 2. Note that the quartiles Q_1 and Q_3 indicate errors away from and toward the ray origin, respectively, as the functions are positive outside the surface. For the *Sphere* surface, the error without any shifting is very low, i.e., in the order of ϵ_m , where applying the post-processing yields a median error of exactly zero. Finding the root for the shifted function f_p^ϵ yields a positive error, as the root is shifted toward the ray origin. Applying the post-processing yields errors identical to the non-shifted and post-processed function.

For the *Blend II* surface, the behavior is similar for both the non-shifted and shifted variant, i.e., the error for the non-shifted variant is close to machine precision whereas shifting the function yields a similar increase in error magnitude of about 6 orders of magnitude. We see that applying a Newton iteration based post-processing on the shifted function, yielding the root estimate $\tilde{\xi}_N$, does not yield the same accuracy as for the non-shifted variant of the *Blend II* surface, in fact, the median error magnitude increases by about an order of magnitude.

For the *Union (soft)* surface, the error of the non-shifted variant is significantly higher than for the previous two functions and can become $f(\tilde{\xi}) = -8.76 \times 10^{-4}$ in outlier cases. The shifted variant yields a smaller relative absolute error as the non-shifted variant, i.e., the error only increases by 3 orders of magnitude regarding the median instead of 4 or 8. Similar to the other examples, the estimated root is also pushed toward the ray origin and applying the post-processing yields a significant improvement in the error magnitudes for both variants regarding Q_1 , median and Q_3 , but the maximum error magnitude increases significantly. This behavior indicates that for this specific function, the estimated root $\tilde{\xi}$ of f_p is not a good starting point for a Newton-based post-processing and generates outliers with significantly higher error.

Overall, applying the shift to the proxy function yields the expected behavior, i.e., the estimated intersection is pushed toward the ray origin, and applying a Newton iteration-based post-processing does significantly increase the accuracy in most cases. However, the post-processing, also in combination with shifting, does not robustly improve the accuracy in all cases. Accordingly, the post-processing should be com-

Table 3 Comparison of the requirements of the different rendering techniques regarding Lipschitz constants, derivatives, and the number of user-supplied hyperparameters that need to be adjusted on a per function basis for optimal results

Method	Lipschitz?	$\frac{df}{dx}$?	# Params
Ours	No	None	0
Ray-March	No	None	2
AMP-Sign [44]	No	None	4
AMP-Taylor [44]	No	$\frac{df}{dx}$	4
Sherstyuk [43]	No	$\frac{df}{dx}$	2
Sphere [26]	f Global	None	1
Segment [21]	f Local	None	1
LG [29]	$f, \frac{df}{dx}$ Local	$\frac{df}{dx}, \frac{d^2f}{dx^2}$	1

bined with a sanity check, i.e., if the absolute magnitude of f actually decreased after applying the post-processing.

4.2 Comparison against baseline methods

To compare our method against existing approaches, we distinguish existing methods based on their requirements regarding the prior knowledge of Lipschitz constants and derivatives, and the number of tuning parameters to be specified per surface function; see Table 3. We compare against two methods that do not require any prior knowledge, i.e., ray marching (*Ray March*) and Adaptive Marching Points (*AMP-Sign*) [44], that uses a sign-test to isolate roots. All other methods either require Lipschitz constants (*Sphere Tracing* (*Sphere*) [26] and *Segment Tracing* (*Segment*) [21]), derivatives (*Adaptive Marching Points* (*AMP-Taylor*) [44] using a Taylor-test to isolate roots and a piecewise polynomial approach (*Sherstyuk*) [43]), or both (*LG Surfaces* approach (*LG*) [29]).

In general, there is no direct option to estimate the Lipschitz constant for a given black-box function $f : [a, b] \rightarrow \mathbb{M}$, i.e., finding the maximum derivative of f , besides finding all roots of f'' on $[a, b]$. In other words, computing the Lipschitz constant of f is as expensive as computing all roots of f . Consequently, it is not possible to apply methods that require Lipschitz constants, such as sphere tracing, without a *second method that does not require Lipschitz constants* to estimate the same. Note that while approximations could be used for the Lipschitz constant, significantly underestimating the Lipschitz constant would lead to missing surface intersections and overestimating Lipschitz constants would increase computational costs. Accordingly, using an accurate value for the Lipschitz constants is important for a fair comparison with baseline methods that rely on this constant.

Exclusively for the purpose of *comparing with methods that require Lipschitz constants*, e.g., *Sphere Tracing*, we

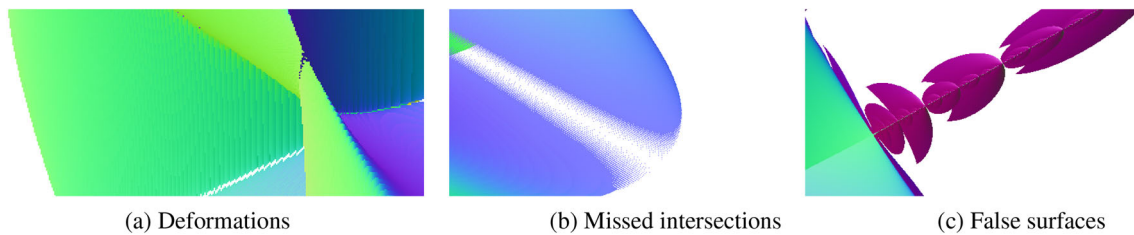


Fig. 5 Examples of the different qualitative errors observed in ray-surface intersections for the Bohemian Star function (see Fig. 2b). The left image (using *Ray Marching*) displays a jagged outline of the surface

and a discontinuous surface. The middle image (using *Ray Marching*) displays large regions of missed surface. The right image (using *AMP-Taylor*) displays significant amounts of false surfaces

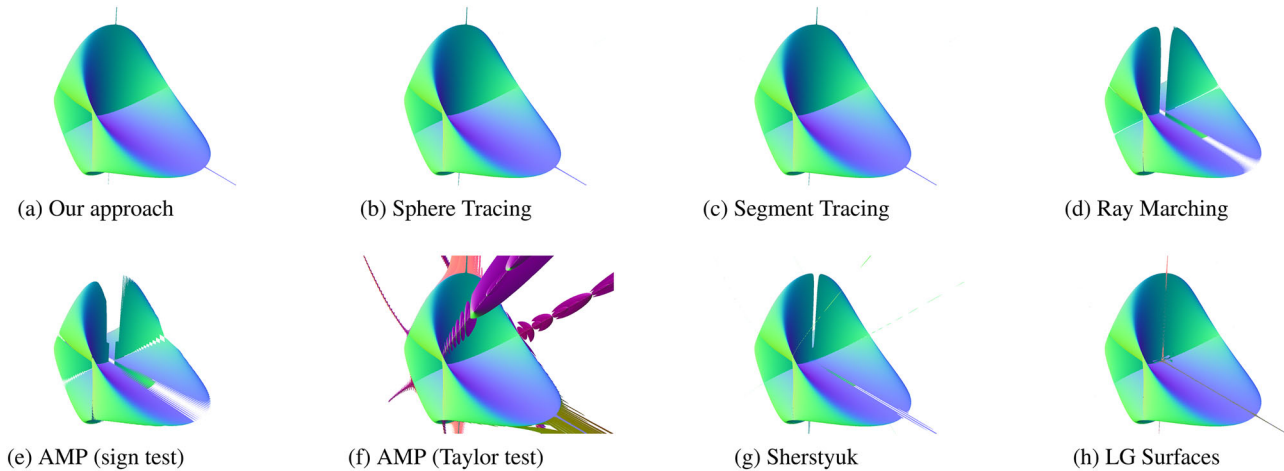


Fig. 6 The Bohemian Star surface rendered using different approaches with surfaces normal colored

construct a Chebyshev proxy f_p of degree n_ϵ for all surface functions f and estimate their Lipschitz constant using our methods for root finding. Our results indicate, that the high root finding accuracy of our approach yields Lipschitz constants with comparably marginal errors.

4.2.1 Qualitative evaluation

When evaluating the different surface functions, we identified three distinct categories of *qualitative errors* in the visualized surface (see Fig. 5): (i) *Deformations*, e.g., the surface was visible dilated or showed a jagged instead of a smooth boundary/silhouette, (ii) *Holes* in the surface, e.g., a rendering method missed the ray-surface intersection, and (iii) *False Surfaces*, e.g., the rendering method found false surface intersections. A good example of the various kinds of error modes is the Bohemian Star function, due to the large amount of multiple roots that comprise the surface of the function (see Fig. 6). When using the shifted function f_p^σ as described in Sect. 4.1.4, our method does not exhibit any noticeable qualitative errors in the surface rendering, however, if no shifting was applied then our method also misses some surface intersections (see Fig. 9a); see also the detailed discussion in Sect. 4.3.

As depicted in Fig. 6, the only other methods that do not display any noticeable errors for the Bohemian Star are the *Sphere Tracing* and *Segment Tracing* approaches, which require additional knowledge about the Lipschitz constant. Using a naïve ray marching approach with sign tests yields significant gaps in the surface, with similar behavior as *AMP-sign*. While the *AMP-Taylor* approach yields no holes in the surface, it generates a large amount of false surfaces close to the multiple roots of the surface. The approach by Sherstyuk [43], yields both, false surfaces and holes in the surface, especially close to the coordinate axis. Finally, *LG surfaces* exhibited some false surfaces, but this error is mainly due to difficulties in estimating the Lipschitz constant and the gradient of the function, and is not necessarily a conceptual problem of the method itself. The overall qualitative results are summarized in Table 4, which highlights that the only two methods not exhibiting any visually noticeable problems are our proposed method and *Sphere Tracing*, which requires a globally known Lipschitz constant and which can be arbitrarily slow for grazing rays.

Table 4 Visual evaluation of the results for the different rendering techniques for the different surface functions. Each cell indicates if there are either visual deformations of the surface, e.g., if the surface boundary is different, or holes in the surface or if there are spurious surface elements displayed

Method	Polynomial			Rational			Analytic			Algebraic				Non-algebraic				Overall			
	Sphere			Blend I			Blend II			Blend III			Log SoR				Union (soft)				
	Star	Kiss		Blend I	Blend II	Blend III	Exp. Cube	Exp. 2D	Log SoR	Root SoR	Morph	Union (soft)	Cut (soft)	Union (max)	Cut (min)	Spike 2D	Cube				
Deformations	Ours	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
	Ray-March	-	×	-	-	-	-	-	-	-	-	-	-	-	×	-	-	×			
	Sphere [26]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
	Segment [21]	-	-	-	-	-	-	-	-	-	-	-	×	-	×	×	-	×			
	AMP-sign [44]	-	×	-	-	-	-	-	-	-	-	-	-	×	-	-	-	×			
	AMP-Taylor [44]	-	×	-	-	×	-	-	-	×	-	-	-	-	-	×	-	×			
	Sherstyuk [43]	-	-	-	-	×	-	-	-	-	-	-	-	-	-	-	-	×			
Gaps in surface	LG [29]	-	×	-	-	-	-	-	-	-	-	-	-	-	-	-	-	×			
	Ours	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
	Ray-March	-	×	-	-	-	-	-	-	-	-	×	-	-	-	-	-	×			
	Sphere [26]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
	Segment [21]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
	AMP-sign [44]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	×			
	AMP-Taylor [44]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
False surfaces	Sherstyuk [43]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	×			
	LG [29]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
	Ours	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
	Ray-March	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
	Sphere [26]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
	Segment [21]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
	AMP-sign [44]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
	AMP-Taylor [44]	-	×	-	-	-	-	-	-	-	-	-	-	-	-	-	-	×			
	Sherstyuk [43]	-	×	-	×	×	-	-	-	-	-	-	-	-	×	×	-	×			
	LG [29]	-	×	-	-	-	-	-	-	-	-	-	-	-	-	-	-	×			

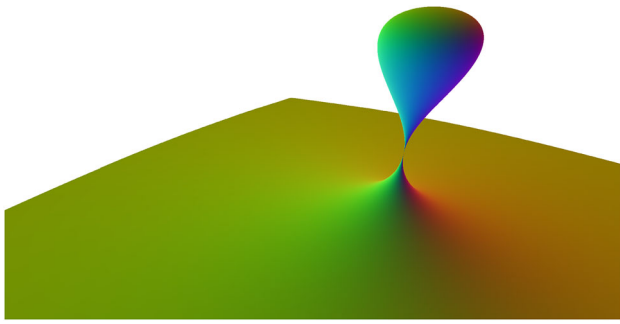


Fig. 7 The Kiss surface ($f(x, y, z) = (1 - z)z^4 - y^2 - x^2$) rendered using our method

4.2.2 Quantitative evaluation

Evaluate the quantitative accuracy of our method, and the baseline methods, is done by evaluating an intersection point $\tilde{\xi}$ using the absolute value of the actual surface function f , i.e., $|f(\tilde{\xi})|$. The resulting error distributions are visualized as box plots in Fig. 8. It is important to note that the actual magnitude of the error $|f(\tilde{\xi})|$ depends on the Lipschitz constant of the underlying surface function. The *Bohemian Star* (see Fig. 6) and *Kiss* (see Fig. 7) surfaces, for instance, have large Lipschitz constants (see Table 1), which means that a small error in the estimation of $\tilde{\xi}$ may yield a large error in the actual function magnitude. Furthermore, to evaluate the computational performance, we only measure the actual root searching process, i.e., for methods that require evaluating Lipschitz constants (either locally or globally), the execution time of these steps is excluded as there is no fair comparison possible as finding the Lipschitz constant of a black-box function is a broader problem that is beyond the scope of our paper. Moreover, as we evaluate the Lipschitz constant, and derivatives along rays, using a Chebyshev proxy (of high numerical accuracy), including this in the evaluation of baseline methods would lead to an unfair comparison regarding computational performance as our method does not require this additional root search and thus has to be necessarily faster.

General Observations. Considering the results in Fig. 8, we observe several effects *across all surface categories*, which we summarize here. Firstly, *Sphere* and *Segment* tracing exhibit a very tight spread on the achieved error in the rendering which is primarily based on the stopping criterion of these methods, i.e., they use an absolute threshold value ϵ and make no further attempts in improving the results after achieving this threshold. Conversely, these methods tend to show large variations in computational performance, especially for *Sphere Tracing*, as their computational performance is directly coupled to the Lipschitz constant of the underlying surface and especially for grazing rays *Sphere Tracing* required up to 10^{11} iterations. Secondly, while the *LG* surface

approach also shows a similar thresholding behavior for the error, the observed accuracy can be much greater as this is an upper bound for the method and not a stopping criterion, i.e., the error can be much lower than the threshold value. Finally, the ray marching approaches (*Ray-March*, *AMP-Sign*, *AMP-Taylor* and *Sherstyuk*) show relatively consistent computational performance across the different surfaces as their performance only directly depends on the computational cost of evaluating f .

Polynomial Surfaces. Our proposed method achieves a numerical accuracy comparable to the best baseline methods, e.g., *Sphere Tracing*. Regarding the computational performance, our method is better than most methods, i.e., only *Sphere Tracing* and the adaptive marching point approaches occasionally outperform our method. For this surfaces category, the Lipschitz constants can be fairly large and, consequently, the computational performance of *Sphere Tracing* can vary greatly for different parts of the rendered image due to grazing rays. Finally, *LG-Surfaces* demonstrates the worst overall computational performance in this category.

Rational Surfaces. For this category, our proposed method outperforms all baseline methods considering the achieved numerical accuracy by several orders of magnitude. However, our methods has the second highest computational cost, i.e., only *LG-Surfaces* exhibits higher computational efforts. **Analytic Surfaces.** In this category, the performance of all methods is comparable to the *rational* surfaces with a few notable differences. Our proposed method performs notably worse for the exponential cube function regarding numerical accuracy, as the function is approximated with a much lower polynomial degree, than the other *analytic* surfaces, see Table 1. Moreover, as this function also has a Lipschitz constant that is noticeably smaller than the other surfaces of this category, *Sphere Tracing* also performs notably worse, regarding computational performance. Finally, the computational performance of our proposed methods and of *LG-Surfaces* is comparable across the category with all other methods performing noticeably better.

Algebraic Surfaces. This category shows behavior analogous to the *analytic* surfaces in quantitative aspects and requires no further specialized discussion.

Non-Algebraic Surfaces. In this category, our method still yields a better numerical accuracy than the other non-Lipschitz methods, i.e., *Ray Marching* and adaptive marching points, while performing worse than all other baselines by relatively small margins, with the single exception of the *Sherstyuk* approach, which performs exceptionally well for the cube surface. Regarding computational complexity, our proposed rendering approach performs worse than most baselines methods besides *Segment Tracing* and *LG-Surfaces*. It also can be observed, that there is a wide spread of computational performance, specifically for *Segment Tracing* and *LG* for surface category.

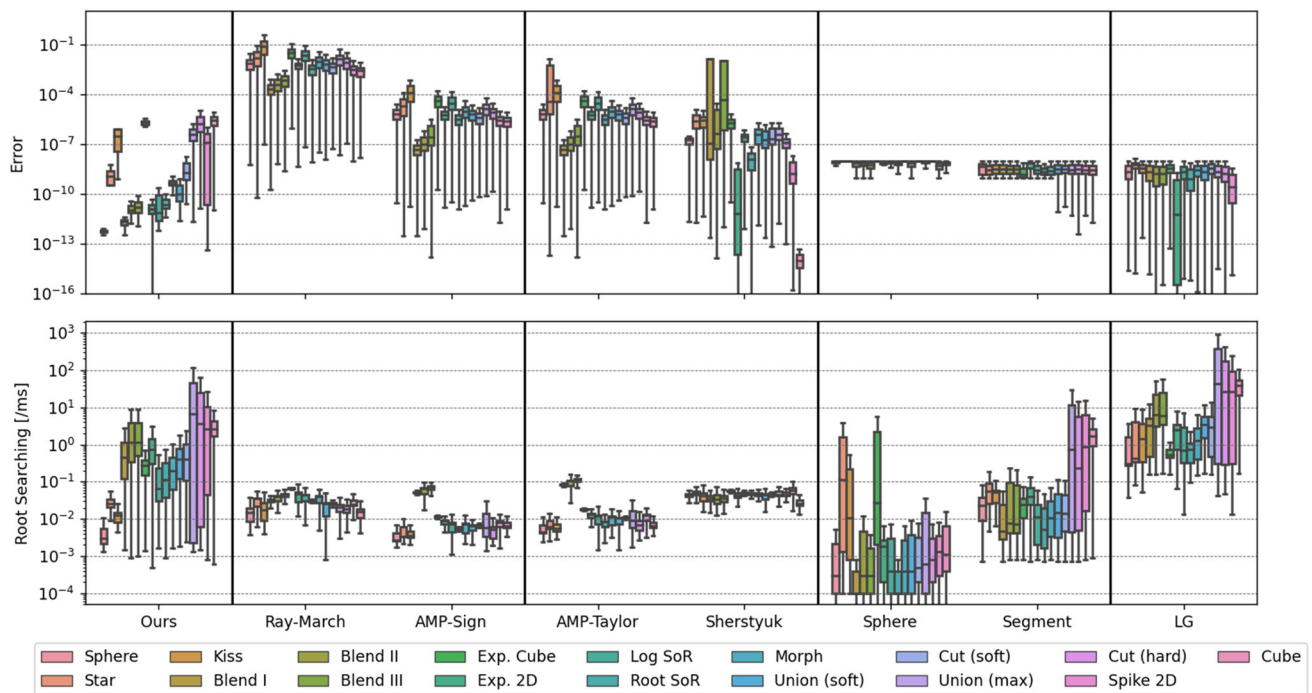


Fig. 8 This figure shows the absolute errors $|f|$ evaluated at the computed ray-surface intersection (top) and the overall timing of the ray-surface intersection test (bottom) as statistics for the entire frame for all the different surface functions grouped by the respective rendering approaches

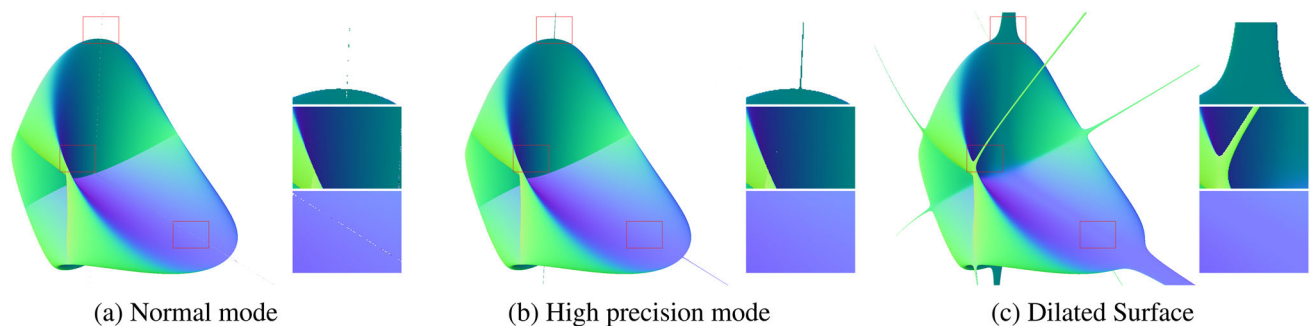


Fig. 9 The Bohemian Star surface visualized using our standard rendering process (left), using the high precision mode (middle) and with a dilated surface using f_p^ϵ and setting $s = 10^8$ (right) with detail views for the bulk surface, axis aligned roots and diagonal aligned roots

4.3 Recursive root searching

The Bohemian Star is a challenging implicit surface function as it comprises axis aligned lines and diagonally oriented curves of roots with even multiplicity, see Sect. 3.5 and Fig. 9. Visualizing the bulk surface is straight forward as these regions have single roots, i.e., there is a change of sign with large enough magnitudes on either side of the surface. On $y = 0$, $f^{\text{star}}(x, y, z)$ gets

$$f^{\text{star}}(x, 0, z) = 16x^2z^2, \quad (9)$$

which is 0 if x and/or z are 0, and all of these roots have an even multiplicity. Evaluating this function close to these axes yields discretization errors of 0, leading to root intervals for

f . For $x = \pm z$, $f^{\text{star}}(x, y, z)$ gets

$$f^{\text{star}}(x, y, x) = \left(2x^2(y^2 - 2) + y^4\right)^2, \quad (10)$$

which is zero for $y = \pm\sqrt{\sqrt{x^2(x^2 + 4)} - x^2}$, see the dilated surface in Fig. 9 for a visualization of these curves of roots with even multiplicity. These curves of roots are numerically rarely rounded to 0 in $\mathbb{M}$ if evaluated in proximity to these curves. Using our recursive root search approach, a few single pixels containing these surface segments can be identified. Consequently, very few, if any, rays hit these points making them difficult to visualize, even with knowledge of their existence. Applying a shift to the function ensures that

even multiplicity roots are found, and the surface is closed in the main region (Fig. 9, bottom detail view). In this case the lines of roots along the main axes are clearly visible, whereas the diagonal curves are not clearly visible as no rays intersect these curves. Note, that applying Newton's method to pseudo-roots as post-processing in case of roots with even multiplicity may diverge significantly, as, for instance, there are no roots along the ray to converge to.

5 Conclusions

In this paper we presented a novel rendering approach for black-box implicit surface functions based on Chebyshev proxy function. Our approach can handle a wide variety of surfaces, ranging from polynomial to non-algebraic surfaces, all without requiring any knowledge of the underlying function. Furthermore, by utilizing numerically accurate and fast to evaluate Chebyshev proxy functions, our method is able to outperform methods that require similar amounts of knowledge of the underlying implicit function regarding numerical accuracy. Moreover, our proposed rendering method achieves better visual results and numerical accuracy than methods with similar requirements on the implicit function being rendered. Overall, our proposed method can serve as a useful baseline method for the evaluation of the numerical accuracy of other methods and can be used to accurately visualize complex implicit surfaces, e.g., polynomial surfaces with large regions of very thin surfaces, without requiring any derivation of properties of the function, such as their local Lipschitz constants. In the future we intend to expand this approach, e.g., using beam tracing, to reliably find small features that no individual ray is likely to intersect.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data availability The full source code to reproduce the dataset will be available publically after publication at <https://github.com/wi-re/chebyshevSurfaces>, this also includes high resolution images of all analyzed rendered images for reference.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Source Code The source code will be available publically after publication at <https://github.com/wi-re/chebyshevSurfaces>.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence,

unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

Appendix A: Finite precision root search

In general, the set of all roots for a C^0 continuous function $f : [a, b] \subset \mathbb{R}^2 \rightarrow \mathbb{R}$, is defined as

$$\mathcal{R}(f) = \{x \mid f(x) = 0\}. \quad (\text{A.1})$$

Our goal is to identify the *first*, i.e., leftmost root. Note that we ignore the cases of $f(a) = 0$ and $f(b) = 0$ as these can be readily detected and require no specific treatment. Assuming the cardinality of $\mathcal{R}(f)$, $n > 0$, we strictly order $\mathcal{R}(f)$ the roots ξ_i as $a < \xi_0 < \dots < \xi_{n-1} < b$ and partition $[a, b]$ into sub-intervals Ξ_i

$$\mathcal{I} = \{(\underbrace{a, \xi_0}_{\Xi_0}), \dots, (\underbrace{\xi_{n-1}, b}_{\Xi_n})\}, \quad (\text{A.2})$$

Consequently, the i -th root ξ_i has two adjacent intervals Ξ_i and Ξ_{i+1} , which can be used to describe the behavior of the function around the root ξ_i . We define the *sign of the interval* $\text{sgn}_f(\Xi)$ and the *magnitude of the function* $\text{mag}_f(\Xi)$ over the interval $\Xi = (s, t]$ as

$$\begin{aligned} \text{sgn}_f(\Xi) &= \text{sgn} \left(f \left(\frac{s+t}{2} \right) \right) \text{ and } \text{mag}_f(\Xi) \\ &= \|f|_{(s,t]}\|_{\infty}. \end{aligned} \quad (\text{A.3})$$

We furthermore define the *dynamic range* of a root ξ_i as

$$\begin{aligned} \text{dr}_f(\xi_i) &= \begin{cases} \log_{10} \frac{\text{mag}_f(a,b)}{\min(\text{mag}_f(\Xi_i), \text{mag}_f(\Xi_{i+1}))}, & \text{sgn}_f(\Xi_i) \neq \text{sgn}_f(\Xi_{i+1}) \\ \infty, & \text{else,} \end{cases} \end{aligned} \quad (\text{A.4})$$

which describes the relative range of values before and after a root compared to the overall magnitude of the function. Boyd [12] shows that, a Chebyshev proxy f_p of f is bound in its precision relative to the supremum of the function over the entire interval $[a, b]$, i.e., $\|f_{p,(a,b)} - f_{(a,b)}\|_{\infty} \approx \|f_{(s,t]}\|_{\infty} \epsilon$, that is, the highest achievable dynamic range by an approximation is bound by ϵ , which for $\epsilon = \epsilon_m$ in double precision arithmetic is approximately 16.5.

The explicit determination of $\mathcal{R}(f)$ for $f : [a, b] \rightarrow \mathbb{R}$, $a, b \in \mathbb{R}$ is not generally possible within finite precision, i.e., over $\mathbb{M}$. Therefore, f is discretized using a

machine implemented variant $f_m : [a, b] \rightarrow \mathbb{M}$, assuming that $a, b \in \mathbb{M}$, utilizing finite precision arithmetic over the machine numbers $\mathbb{M}$. Using f_m for root finding for f needs specific consideration.

Note, that in the following, all variables are assumed to be in $\mathbb{M}$. Discretizing f , yielding f_m , and subsequently approximating f_m by f_p involves four types of errors:

1. *Domain discretization*, when moving from $[a, b] \subset \mathbb{R}$ to $[a, b] \subset \mathbb{M}$
2. *Image discretization*, when outputting function image values in $\mathbb{M}$ instead of $\mathbb{R}$
3. *Evaluation errors* due to finite precision operations used during function evaluation
4. *Approximation errors* from f_m to f_p

Appendix A.1: Domain discretization

Domain discretization of $[a, b] \subset \mathbb{R}$ gets $[a, b] \subset \mathbb{M}$, and requires handling any root $\xi \in [a, b] \setminus \mathbb{M}$. To handle intervals of real numbers between adjacent machine numbers, we define the higher x_+ and lower x_- of $x \in \mathbb{M}$ as

$$x_- = \max\{y \in \mathbb{M} : y < x\}, \quad x_+ = \min\{y \in \mathbb{M} : y > x\}. \quad (\text{A.5})$$

Given the rounding operator $\lfloor x \rfloor_{\mathbb{R}} : \mathbb{R} \rightarrow \mathbb{M}$ that yields the closest value in $\mathbb{M}$ for $x \in \mathbb{R}$, we define the machine precision interval $[x_l, x_r]$ for a real number $x \in \mathbb{R}$ as

$$[x_l, x_r] = \begin{cases} [\lfloor x \rfloor, x_+], & \lfloor x \rfloor \leq x, \\ [x_-, \lfloor x \rfloor], & \text{else.} \end{cases} \quad (\text{A.6})$$

Operating on $\mathbb{M}$, a root $\xi \in]a, b[\setminus \mathbb{M}$ is not captured by the definition of $\mathcal{R}_f$ in Eq. A.1. More precisely, $\xi \in]a, b[\setminus \mathbb{M}$ can only be captured, if $\text{sgn}(f(\xi_l)) \neq \text{sgn}(f(\xi_r))$, for which the intermediate value theorem guarantees the existence of a root in (ξ_l, ξ_r) for C^0 -continuous functions. For all other cases, i.e., $f(\xi_l) = 0$, $f(\xi_r) = 0$, or $\text{sgn}(f(\xi_l)) = \text{sgn}(f(\xi_r))$, no conclusion can be made regarding roots within (ξ_l, ξ_r) . Accordingly, the definition in Eq. A.1 is expanded to

$$\mathcal{R}^d(f) = \{x \in \mathbb{M} \mid f(x) = 0 \vee f(x) \cdot f(x_+) < 0\}. \quad (\text{A.7})$$

Lipschitz argument. While we focus on rendering functions without any knowledge about Lipschitz constants, some optimizations can be applied for Lipschitz continuous functions (see Sect. 3.3). Given that f is Lipschitz continuous with Lipschitz constant $k < \infty$, i.e.,

$$|f(x_1) - f(x_2)| \leq k|x_1 - x_2|, \quad \forall x_1, x_2 \in \mathbb{R}, \quad (\text{A.8})$$

a necessary condition for the existence of a root within the interval $[s, t]$ can be deduced from $f : [s, t] \rightarrow [\max\{f(s), f(t)\} - k(t - s), \min\{f(s), f(t)\} + k(t - s)]$. Specifically, the condition

$$0 \in [\max\{f(s), f(t)\} - k(t - s), \min\{f(s), f(t)\} + k(t - s)], \quad (\text{A.9})$$

is required for f to have a root in $[s, t]$.

Appendix A.2: Image discretization

Image discretization refers to the mapping of the output values of f from $\mathbb{R}$ to $\mathbb{M}$, yielding $f_d(x) = \lfloor f(x) \rfloor$. As $0 \in \mathbb{M}$, no zeroes get lost, but as the mapping is non-bijective, non-zero values can be mapped to zero. More precisely, any value $0_-/2 < f(x) < 0_+/2$ will be mapped to zero when using f_d instead of f , yielding a pointwise error function $e_{\text{round}} : \mathbb{M} \rightarrow \mathbb{R}$:

$$f(x) = f_d(x) + e_{\text{round}}(x). \quad (\text{A.10})$$

Important note. It is not possible to distinguish if a value x with $f_d(x) = 0$, is an actual root, i.e., $f(x) = 0$, or only an apparent root due to rounding, using finite precision arithmetic. For example, consider $f(x) = x^2$, for which any $x \in (-\sqrt{0_-}/2, \sqrt{0_+}/2) (\approx 10^{22} \text{ values})$ yields $f_d(x) = 0$. Practically, any value within such an interval is treated as the *same* root for our purpose.

Appendix A.3: Evaluation errors

Evaluation errors occur when performing operations $x \text{ op}_{\mathbb{M}} y$, $x, y \in \mathbb{M}$ in finite precision, which may not yield the exact result of the infinite precision operation $x \text{ op}_{\mathbb{R}} y$ in over $\mathbb{R}$, yielding the relative error $z(x, y)$ with

$$x \text{ op}_{\mathbb{M}} y = (x \text{ op}_{\mathbb{R}} y) (1 + z(x, y)). \quad (\text{A.11})$$

$|z(x, y)| \leq \epsilon_m$ for the machine precision ϵ_m (2^{-52} in double precision) for elementary operations (e.g., $+$, $-$). Complex operations and function implementations comprise several elementary operations and may lead to catastrophic cancellation, e.g., divisions of values at extreme different scale.

Important note. As evaluation errors induced by machine implementation f_m of a mathematical function f are outside our control and the scope of this paper, we assume the machine implementation f_m to be “correct,” i.e., we assume an ideal implementation $f_m(x) = f_d(x) = \lfloor f(x) \rfloor$, $x \in \mathbb{M}$. Specifically, we assume f_m to be properly defined, i.e., it maps to $\mathbb{M} \setminus \{\pm\infty\}$. Therefore, while a (mathematical) function f may not have a Lipschitz constant $k < \infty$, f_m has

a *machine Lipschitz constant* due to the discrete representation. More precisely, any two adjacent machine representable numbers $x, x' \in \mathbb{M}$ have a finite difference in function magnitude $|f_m(x) - f_m(x')|$ and, thus, there is an upper bound for this local discrete change rate of f_m . We will use the fact that a machine Lipschitz constant exists for f_m , even though we do not require the specific constant.

Appendix A.4: Approximation errors

For the following section of this paper, we will make heavy use of approximations f_p to f_m using Chebyshev polynomials. Therefore, we discuss the impact of approximations to root finding in general. In combination with Eq. A.10, the approximation adds another pointwise error $e_{\text{approx}} : \mathbb{M} \rightarrow \mathbb{R}$, yielding

$$f(x) = f_d(x) + e_{\text{round}}(x) = f_p(x) + e_{\text{round}}(x) + e_{\text{approx}}(x). \quad (\text{A.12})$$

As we assume f_m to be an ideal implementation and as $e_{\text{round}}(x)$ is unknowable, but small, the approximation error $e_{\text{approx}}(x)$ is dominant, and we consider

$$f_m(x) = f_p(x) + e_{\text{approx}}(x), \quad (\text{A.13})$$

and we define the upper bound of the error as

$$\epsilon = \max_{x \in [a, b]} |e_{\text{approx}}(x)| = \|f_m - f_p\|_{\infty}. \quad (\text{A.14})$$

Accordingly, the (ideal) machine implementation f_m of f lays in the interval $f_p(x) - \epsilon \leq f_m(x) \leq f_p(x) + \epsilon$ and any value $|f_p(x)| \leq \epsilon$ potentially indicates a root for $f_m(x)$. Considering the set of (potential) roots of f_m when using its approximation f_p , we expand Eq. A.7 by incorporating the error bound ϵ

$$\begin{aligned} \mathcal{R}^{\text{approx}}(f_m) = \{x \in \mathbb{M} \mid & |f_p(x)| \leq \epsilon \\ & \vee (f_p(x) - \epsilon) \cdot (f_p(x_+) + \epsilon) < 0 \cdot \\ & \vee (f_p(x) + \epsilon) \cdot (f_p(x_+) - \epsilon) < 0\} \end{aligned} \quad (\text{A.15})$$

$\mathcal{R}^{\text{approx}}(f_m)$ includes significantly more values than $\mathcal{R}(f)$. *Important note.* Root searching methods commonly consider Eq. A.7 also in case they use a proxy function f_p . As the root set in Eq. A.15 is “overcomplete,” our approach presented in Sect. 3 finds the leftmost root using f_p in combination with Eq. A.15 and optionally refines the root using f_m based on Eq. A.7.

Considering the task of finding the root ξ_i of f_m using the proxy f_p requires that f_m has a magnitude of at least ϵ on

either side of the root, i.e.,

$$\begin{aligned} \text{mag}_{f_m}(\xi_i) \geq \epsilon \wedge \text{mag}_{f_m}(\xi_{i+1}) \geq \epsilon \wedge \\ \text{sgn}_{f_m}(\xi_i) \neq \text{sgn}_{f_m}(\xi_{i+1}). \end{aligned} \quad (\text{A.16})$$

In general, the approximation error ϵ from Eq. A.14 steers the set of potential roots of f_m given f_p ; see Eq. A.15. If we specify f_p to be a Chebyshev proxy function to f_m , it is possible to decrease ϵ by increasing the polynomial degree of the Chebyshev polynomial f_p for infinite precision (Weierstrass Approximation Theorem). However, higher-degree Chebyshev polynomials do not necessarily improve the accuracy of the approximation in finite precision arithmetic due to the accumulation of numerical errors inherent to evaluating larger sets of operations.

Consequently, there is a lower bound of accuracy when constructing the Chebyshev proxy f_p that is independent of the polynomial degree of f_p , and several heuristics estimate ϵ [12, 39]. We utilize the bound estimate ϵ_a proposed by Boyd [12], which is based on the coefficients c_i and degree n of the Chebyshev proxy, that, according to Boyd [12], can not provide a better approximation accuracy than ϵ_a , i.e.,

$$\epsilon_a = n^2 \epsilon_{\text{machine}} \underbrace{\max_{i \in [0, \dots, n]} |c_i|}_{\text{mmc}(f_p)} \leq \|f_p - f_m\|_{\infty}, \quad (\text{A.17})$$

where $\text{mmc}(f_p)$ is the maximum magnitude coefficient.

Furthermore, within our proxy construction process we utilized the adaptive approach of Boyd [12] and the heuristic chopping process of Aurentz and Trefethen [1], which relies on a user provided parameter ϵ_u that steers the accuracy of the proxy function that tries to achieve $\|f_m - f_p\|_{\infty} \leq \epsilon_u$. The approach of Aurentz and Trefethen [1] uses the parameter ϵ_u to set the threshold for the round-off plateau detection. This round-off plateau is the result of the difference in magnitude of coefficients being limited by ϵ_m , as any coefficient smaller than $\text{mmc}(f_p)/\epsilon_m$ will (most likely) not affect the result of machine precision operations. Therefore, ϵ_u can not be chosen below a value that depends on $\text{mmc}(f_p)$ and the approximation error is bound on both sides

$$\epsilon_a \leq \|f_m - f_p\|_{\infty} \leq \epsilon_u, \quad (\text{A.18})$$

where both limits depend on $\text{mmc}(f_p)$ [1, 12]. $\text{mmc}(f_p)$ is considered as being proportional to the magnitude of the function [12]

$$\text{mmc}(f_p) \propto \text{mag}_{f_m}(a, b), \quad (\text{A.19})$$

where the error terms are bound in terms of relative machine precision as $\epsilon_a, \epsilon_u \leq \epsilon_m \text{mag}_{f_m}(a, b)$ [12].

Appendix B: Function definitions

The blend functions used throughout the evaluation in Sect. 4 are given by:

$$\text{morph}(f, g, \alpha) = \alpha f + (1 - \alpha)g \quad (\text{B.1})$$

$$\text{blend}(f, g) = 0.01 - \frac{0.019}{1 + f^{\frac{f}{0.589}} + g^{\frac{g}{0.943}}} \quad (\text{B.2})$$

$$\text{union}^s(f, g) = \frac{1}{2} \left(f + g - \sqrt{f^2 + g^2} \right) + \beta \quad (\text{B.3})$$

$$\text{union}^h(f, g) = \min(f, g) + \beta \quad (\text{B.4})$$

$$\text{cut}^s(f, g) = \frac{1}{2} \left(f + g + \sqrt{f^2 + g^2} \right) + \beta \quad (\text{B.5})$$

$$\text{cut}^h(f, g) = \max(f, g) + \beta \quad (\text{B.6})$$

The functions used throughout the evaluation of the paper:

$$f^{\text{sphere}}(x, y, z) = x^2 + y^2 + z^2 - 1 \quad (\text{B.7})$$

$$f^{\text{star}}(x, y, z) = y^4 x^4 + 2y^6 x^2 + 2y^4 x^2 z^2 + y^8 + 2y^6 z^2 + y^4 z^4 - 4y^4 x^2 - 4y^4 z^2 - 16x^2 z^2 y^2 + 16x^2 z^2 \quad (\text{B.8})$$

$$f^{\text{root}}(x, y, z) = \sqrt{x^2 + y^2} - z \quad (\text{B.9})$$

$$f^{\text{const}}(x, y, z) = \sqrt{(x-1)^2 + y^2 + z^2} \cdot \sqrt{(x+1)^2 + y^2 + z^2} \cdot \sqrt{(x^2 + (y-1)^2 + z^2)} \cdot \sqrt{(x^2 + (y+1)^2 + z^2) - 1.1} \quad (\text{B.10})$$

$$f^{\text{torus}}(x, y, z) = (x^2 + y^2 + z^2 + 1 - 0.25)^2 - 4(x^2 + y^2) \quad (\text{B.11})$$

$$f^{\text{SOR}}(x, y, z) = x^2 + y^2 - \log(z + 3.2)^2 - 0.02 \quad (\text{B.12})$$

$$f^{\text{exp}}(x, y, z) = e^{10x-1.5} + e^{10y-1.5} + e^{10z-1.5} - 1 \quad (\text{B.13})$$

$$f^{\text{cube}}(x, y, z) = \max\{|x|, |y|, |z|\} - 1 \quad (\text{B.14})$$

$$f^{\text{spike}}(x, y, z) = e^{-5|x-0.5|-5|y-0.5|} - z \quad (\text{B.15})$$

$$f^{\text{hat}}(x, y, z) = \frac{1}{\sigma\pi} \left(1 - \frac{x^2 + y^2}{2\sigma^2} \right) e^{-\frac{x^2 + y^2}{2\sigma^2}} - z, \sigma = \frac{1}{5} \quad (\text{B.16})$$

$$f^{\text{quart}}(x, y, z) = x^2 y^2 + y^2 z^2 \frac{x^2}{100} + \frac{z^2}{100} - \frac{1}{100} \quad (\text{B.17})$$

$$f^{\text{C8}}(x, y, z) = 64x^8 - 128x^6 + 80x^4 - 16x^2 + 2 + 64y^8 - 128y^6 + 80y^4 - 16y^2 + 64z^8 - 128z^6 + 80z^4 - 16z^2 \quad (\text{B.18})$$

$$f^{\text{kiss}}(x, y, z) = 16xz \left(x^2 + y^2 + z^2 - 2y - 1 \right) + \left(x^2 + y^2 + z^2 + 2y1 \right) \left[\left(x^2 + y^2 + z^2 - 2y - 1 \right)^2 - 8z^2 \right] \quad (\text{B.19})$$

$$f^{\text{kiss}}(x, y, z) = (1 - z)z^4 - y^2 - x^2 \quad (\text{B.20})$$

Functions (B.8), (B.17), (B.18) and (B.19) were taken from [20], while (B.10) and (B.12) are from [50].

References

1. Aurentz, J.L., Trefethen, L.N.: Chopping a chebyshev series. *ACM Trans. Math. Softw.* **43**(4), 1–21 (2017)
2. Aydinlilar, M., Zanni, C.: Fast ray tracing of scale-invariant integral surfaces. In: *Computer Graphics Forum*, Wiley Online Library (2021)
3. Battles, Z., Trefethen, L.N.: An extension of matlab to continuous functions and operators. *SIAM J. Sci. Comput.* **25**(5), 1743–1770 (2004)
4. Bernstein, S.: Quelques remarques sur l'interpolation. *Math. Ann.* **79**(1), 1–12 (1918)
5. Blinn, J.F.: A generalization of algebraic surface drawing. *ACM Trans. Graph.* **1**(3), 235–256 (1982)
6. Bloomenthal, J., Bajaj, C., Blinn, J., Wyvill, B., Cani, M.-P., Rockwood, A., Wyvill, G.: *Introduction to implicit surfaces*. Morgan Kaufmann, (1997)
7. Boyd, J.P.: A chebyshev polynomial interval-searching method ("lanczos economization") for solving a nonlinear equation with application to the nonlinear eigenvalue problem. *J. Comput. Phys.* **118**(1), 1–8 (1995)
8. Boyd, J.P.: *Chebyshev and Fourier spectral methods*. Courier Corporation (2001)
9. Boyd, J.P.: Computing zeros on a real interval through chebyshev expansion and polynomial rootfinding. *SIAM J. Numer. Anal.* **40**(5), 1666–1682 (2002)
10. Boyd, J.P.: Computing real roots of a polynomial in chebyshev series form through subdivision. *Appl. Numer. Math.* **56**(8), 1077–1091 (2006)
11. Boyd, J.P.: Computing real roots of a polynomial in chebyshev series form through subdivision with linear testing and cubic solves. *Appl. Math. Comput.* **174**(2), 1642–1658 (2006)
12. Boyd, J.P.: *Solving Transcendental Equations: The Chebyshev Polynomial Proxy and Other Numerical Rootfinders, Perturbation Series, and Oracles*, vol. 139. SIAM (2014)
13. Boyd, J.P., Gally, D.H.: Numerical experiments on the accuracy of the chebyshev-frobenius companion matrix method for finding the zeros of a truncated series of chebyshev polynomials. *J. Comput. Appl. Math.* **205**(1), 281–295 (2007)
14. Caprani, O., Hvidegaard, L., Mortensen, M., Schneider, T.: Robust and efficient ray intersection of implicit surfaces. *Reliable Comput.* **6**(1), 9–21 (2000)
15. Clenshaw, C.W., Curtis, A.R.: A method for numerical integration on an automatic computer. *Numer. Math.* **2**(1), 197–205 (1960)
16. Collins, G.E., Loos, R.: Real zeros of polynomials. In: *Computer Algebra*. Springer, pp. 83–94 (1983)
17. Driscoll, T.A., Hale, N., Trefethen, L.N., editors.: *Chebfun guide*, Pafnuty Publications, Oxford (2014)
18. Fraser, W., Wilson, M.: Remarks on the clenshaw-curtis quadrature scheme. *SIAM Rev.* **8**(3), 322–327 (1966)

19. Fryazinov, O., Pasko, A., Comninos, P.: Fast reliable interrogation of procedurally defined implicit surfaces using extended revised affine arithmetic. *Comput. Graph.* **34**(6), 708–718 (2010)
20. Galaad Team, U. o. N. S. A.: Implicit algebraic surfaces. Accessed on 25 Sep 2021
21. Galin, E., Guérin, E., Paris, A., Peytavie, A.: Segment tracing using local lipschitz bounds. *Comput. Graph. Forum* **39**(2), 545–554 (2020)
22. Gènevaux, J.-D., Galin, E., Peytavie, A., Guérin, E., Briquet, C., Grosbellet, F., Benes, B.: Terrain modelling from feature primitives. *Comput. Graph. Forum* **34**(6), 198–210 (2015)
23. Gomes, A., Voiculescu, I., Jorge, J., Wyvill, B., Galbraith, C.: *Implicit Curves and Surfaces: Mathematics, Data Structures and Algorithms*. Springer Science & Business Media (2009)
24. Gourmel, O., Pajot, A., Paulin, M., Barthe, L., Poulin, P.: Fitted bvh for fast raytracing of metaballs. *Comput. Graph. Forum* **29**(2), 281–288 (2010)
25. Hanrahan, P.: Ray tracing algebraic surfaces. In: *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pp. 83–90 (1983)
26. Hart, J.C.: Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *Vis. Comput.* **12**(10), 527–545 (1996)
27. Jacobs, S.J.: A pseudospectral method for two-point boundary value problems. *J. Comput. Phys.* **88**(1), 169–182 (1990)
28. Jin, X., Tai, C.-L., Zhang, H.: Implicit modeling from polygon soup using convolution. *Vis. Comput.* **25**(3), 279–288 (2009)
29. Kalra, D., Barr, A.H.: Guaranteed ray intersections with implicit surfaces. *ACM Siggraph Computer Graphics* **23**(3), 297–306 (1989)
30. Kanamori, Y., Szego, Z., Nishita, T.: Gpu-based fast ray casting for a large number of metaballs. *Comput. Graph. Forum* **27**(2), 351–360 (2008)
31. Keeter, M.J.: Massively parallel rendering of complex closed-form implicit surfaces. *ACM Trans. Graph.* **39**(4), 141–1 (2020)
32. Keinert, B., Schäfer, H., Korndörfer, J., Ganse, U., Stamminger, M.: Enhanced sphere tracing. *STAG Smart Tools Graph.* **8**, 4 (2014)
33. Knoll, A., Hijazi, Y., Hansen, C., Wald, I., Hagen, H.: Interactive ray tracing of arbitrary implicits with simd interval arithmetic. In: *2007 IEEE Symposium on Interactive Ray Tracing*, IEEE, pp. 11–18 (2007)
34. Knoll, A., Hijazi, Y., Kensler, A., Schott, M., Hansen, C., Hagen, H.: Fast ray tracing of arbitrary implicit surfaces with interval and affine arithmetic. *Comput. Graph. Forum* **28**(1), 26–40 (2009)
35. Mason, J.C., Handscomb, D.C.: *Chebyshev Polynomials*. CRC Press (2002)
36. Mavriplis, C.: Adaptive mesh strategies for the spectral element method. *Comput. Methods Appl. Mech. Eng.* **116**(1–4), 77–86 (1994)
37. Mitchell, D.P.: Robust ray intersection with interval arithmetic. *Proc. Graph. Interface* **90**, 68–74 (1990)
38. Nishita, T., Nakamae, E.: A method for displaying metaballs by using bézier clipping. *Comput. Graph. Forum* **13**(3), 271–280 (1994)
39. O'Hara, H., Smith, F.J.: Error estimation in the clenshaw-curtis quadrature formula. *Comput. J.* **11**(2), 213–219 (1968)
40. Rawat, P. S., Sukumaran-Rajam, A., Rountev, A., Rastello, F., Pouchet, L.-N., Sadayappan, P.: Associative instruction reordering to alleviate register pressure. In: *SC18: International conference for high performance computing, networking, storage and analysis*, IEEE, pp. 590–602 (2018)
41. Runge, C.: Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten. *Zeitschrift für Mathematik und Physik* **46**(224–243), 20 (1901)
42. Seyb, D., Jacobson, A., Nowrouzezahrai, D., Jarosz, W.: Non-linear sphere tracing for rendering deformed signed distance fields. *ACM Trans. Graph.* **38**, 6 (2019)
43. Sherstyuk, A.: Fast ray tracing of implicit surfaces. *Comput. Graph. Forum* **18**(2), 139–147 (1999)
44. Singh, J.M., Narayanan, P.: Real-time ray tracing of implicit surfaces on the GPU. *IEEE Trans. Vis. Comput. Graph.* **16**(2), 261–272 (2009)
45. Singh, J.M., Wasnik, P., Ramachandra, R.: Hessian-based robust ray-tracing of implicit surfaces on gpu. In: *SIGGRAPH Asia 2018 Technical Briefs*. ACM, 1–4 (2018)
46. Sloan, I., Smith, W.: Product integration with the clenshaw-curtis quadrature scheme. *SIAM Rev.* **8**, 322–327 (1980)
47. Trefethen, L.N., Bau, D., III.: *Numerical Linear Algebra*, vol. 50. Siam (1997)
48. Van Wijk, J.J.: Ray tracing objects defined by sweeping a sphere. *Comput. Graph.* **9**(3), 283–290 (1985)
49. Weierstrass, K.: Über die analytische Darstellbarkeit sogenannter willkürlicher Functionen einer reellen Veränderlichen. *Sitzungsberichte der Königlich Preussischen Akademie der Wissenschaften zu Berlin* **2**, 633–639 (1885)
50. Wikipedia. Implicit algebraic surfaces. Accessed 25 Sep 2021

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Rene Winchenbach received his doctoral degree from the University of Siegen in 2022 and is currently working as a post-doctoral researcher at the Technical University Munich. His research interests include mainly particle-based physics, graph neural networks, their combination, as well as rendering and visualization techniques.



Michael Möller Professor for Computer Vision at the University of Siegen, Germany, received his doctoral degree from the University of Münster in 2012. His research interests include machine learning, energy minimization methods, their combination, and efficient optimization algorithms with applications in image processing and computer vision.



Andreas Kolb received the Ph.D. degree from the University of Erlangen, Germany, in 1995. He is currently the Head of the Computer Graphics and Multimedia Systems Group, University of Siegen, Germany. His research interests include computer graphics, machine learning and computer vision, specifically related to particle-based simulation and rendering, as well as real-time simulation, processing, accumulation, and visualization of sensor data.