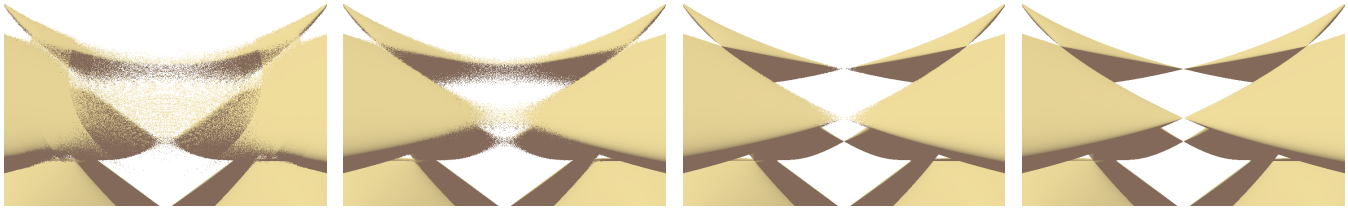
Robust Ray–Surface Intersections for Algebraic Surfaces

Péter Szente[1], Csongor Csanád Karikó[1], and Gábor Valasek[1]

[1] Eötvös Loránd University, Hungary
{szentep, kcscs, valasek}@inf.elte.hu



(a) *Naïve polynomial fitting* **(b)** *Using EFT for polynomial fitting* **(c)** *Direct surface evaluation* **(d)** *EFT and surface evaluation*

Figure 1: *Visual artifacts on the Endraß octic, an algebraic surface of degree 8, showcasing the effect of applying various levels of numerical precision improvements, using single-precision floating point numbers. EFT abbreviates the error-free transform.*

Abstract

We present a robust method for rendering algebraic surfaces on the GPU using only single-precision arithmetic. While fitting the surface function to a polynomial along the view ray is efficient, it typically suffers from numerical instability even at moderate degrees. We address this by employing error-free transforms to emulate higher precision without the performance cost of standard double-precision types. We show that the resulting polynomial fit can supply data for inferring directional Lipschitz bounds and we propose a new lower and upper bound on Bézier functions. Additionally, we propose a modification to Yuksel's bracketed Newton method that uses the fitted polynomial solely to isolate monotonous segments, while the final root refinement relies on bisection of the original implicit function. This strategy ensures numerical stability and register efficiency on consumer graphics hardware. We demonstrate our results on rendering various degree algebraic surfaces.

CCS Concepts

• ***Computing methodologies*** → ***Computer graphics; Rasterization;***

1. Introduction and Related Work

We consider the problem of rendering algebraic surfaces in dynamic scenes via univariate polynomial fitting and root finding. This approach expresses the composition of the parametric ray $\boldsymbol{p}(t) = \boldsymbol{p}_0 + t\boldsymbol{v}$ with the algebraic function $f(x,y,z)$ as a univariate polynomial in the ray parameter, that is, $f \circ \boldsymbol{p}$, and solves for $f \circ \boldsymbol{p} = 0$. Although the method has been applied previously [Han83], its numerical robustness poses challenges even at moderate degrees. Karikó and Valasek considered ray marching univariate polynomial fits [KV25], however, they addressed numerical precision issues by heuristic subdivision of the ray interval. We propose the use of error-free transforms (EFT) [Tha06] and splitting fitting matrices into two single-precision arrays to address numerical concerns without resorting to heuristics or double-precision types.

We chose the confines of single-precision floating-point numbers because double-precision computations on certain GPU architectures are either prohibitively expensive or even non-existent. One such example for the latter is the Apple M GPUs to date.

Fitting notwithstanding, the efficient and robust root finding of the univariate polynomial remains a critical component. Yuksel's bracketed Newton [Yuk22] is an algorithm that only requires a linear working set in the degree of the polynomial. Its efficient implementation on the GPU is possible via reformulations, as demonstrated by Peters [PPUJ23]. However, one has to carefully adapt to the idiosyncrasies of the chosen GPU architecture so as to avoid, e.g., register spilling. We propose a modification to their method that uses an error-corrected univariate fit to infer higher-order derivatives, that determine the monotonous segments of the original function, and a bisection-based final refinement on the original three-variate function.

For ground truth, we use ray marching on the three-variate function, i.e., taking equidistant steps along the ray until the first de-

DOI: 10.2312/egs.20261005

tected sign change. While this approach does not suffer from basis conversion errors, it may skip over surface details that are smaller than a step size. More robust approaches exist [KB89] that require the higher-order interrogation of the surface. We consider one such method in our paper and show that the polynomial fit is a viable proxy to query directional derivative data. In the remainder, we assume we can only evaluate the values of $f(x,y,z)$ and query its total degree. We do not impose any restriction on the temporal complexity of the coefficients, that is, our method is applicable to animated algebraic polynomials as is.

We propose a robust Lipschitz constant bound computation to guarantee convergence without missing surface features [Har96]. This is facilitated by a new method that achieves parameter-dependent bounds on polynomials in the Bernstein basis without requiring subdivision, shown in Section 3.

In Section 2, we describe the methods used for robust polynomial fitting and root finding. In Section 4, we present the results of our numerical and performance evaluation. We conclude the paper with the discussion of the results.

2. Method

In algebraic surface rendering, the surface is given in the form $f(x,y,z)=0$, where $f:\mathbb{R}^3\to\mathbb{R}$ is a polynomial in x,y,z. Our goal is to find the intersection of the ray $\mathbf{p}(t):\mathbb{R}\to\mathbb{R}^3$ and the surface for each pixel. We do so by converting the problem to root finding on the $f\circ\mathbf{p}$ univariate function via fitting. The main stages of our algorithm are as follows:

1. Precompute a pair of single-precision matrices for polynomial fitting that approximates the double-precision fitting matrix.
2. Use EFT for matrix-vector multiplication during fitting.
3. Find the root using the surface function f for value evaluation and the univariate $f\circ\boldsymbol{p}$ proxy for higher order derivatives.

We assume that a bounding sphere determines the region of interest within space where we render the algebraic surface. We employ two approaches for the root finding stage: a modification of Yuksel's method and an improved Lipschitz constant estimation method, using bounds on the fitted polynomial. The former is discussed next, and the second is presented in Section 3.

2.1. Polynomial Fitting

In general, fitting a degree n polynomial to $n+1$ samples in an arbitrary $h_i(x):\mathbb{R}\to\mathbb{R}$ basis is carried out by solving the following system of $n+1$ equations:

$$\forall j\in\{0,\dots,n\}:\sum_{i=0}^{n}c_i h_i(x_j)=s_j\ , \tag{1}$$

where x_j are distinct parameters and s_j are samples. In this paper, we consider the cases of the monomial ($h_i(x)=x^i$) and the Bernstein ($h_i(x)=\binom{n}{i}(1-x)^{n-i}x^i$) basis. The parameters are the Chebyshev nodes $x_i=\cos\frac{i\pi}{n}$, and $s_j=f(\boldsymbol{p}(x_j))$. Let H denote the matrix with entries $H_{ij}=h_j(x_i)$ and the sample vector as $\boldsymbol{s}=[s_0,\dots,s_n]^T$.

We precomputed the H^{-1} matrices using double-precision floating point libraries in Python. Let $fl(x)$ be the float32 representation of x. The matrix elements are stored in a pair of float32 numbers $(fl(x),fl(x-fl(x)))$ as shader code. Notably, Chebyshev nodes on $[-1,1]$ yield lower condition numbers for H in the monomial basis than those reported in [KV25] (Table 1) where $[0,1]$ was used. As the x_j nodes are fixed, this precomputation is only needed to be done once for each degree.

Basis	Degree 3	5	10
monomial	6.70	36.94	2882.20
Bernstein	4.32	16.56	510.28

Table 1: *L^2 condition numbers of fitting matrices generated from different bases using Chebyshev nodes.*

The product $H^{-1}\mathbf{s}$ is computed via error-free transformations [Tha06, ROO08] using the error-free transform of simple arithmetic operations and the compensated summation algorithm. As illustrated in Figure 1, this approach is critical for numerical stability, as the other strategies proposed herein cannot mitigate errors introduced at this step on their own.

2.2. Root Finding

Yuksel's bracketed Newton method iteratively constructs intervals on higher derivatives bottom-up to bracket the monotone segments of the original function. We propose to use the error-corrected univariate fit for the segment bracketing and replace the final Newton iteration with a bisection-based refinement that directly interrogates the original trivariate function to achieve float32 precision. This allows us to compensate for errors accumulated during the fitting and root-finding steps, and consequently, incorrect brackets.

Beyond root isolation strategies, univariate ray tracing utilising local Lipschitz bounds remains a prevalent approach. However, the efficiency of these solvers is intrinsically linked to the tightness of the bounds. In the following section, we introduce a novel bounding technique for Bernstein polynomials that allows larger step sizes and the early culling of non-intersecting rays.

3. Lower and Upper Bernstein Bounds

In the Bernstein basis, the function values stay within the convex hull of the control data. In particular, for the extrema we have

$$m=\min\{b_i\}_{i=0}^n\le b(t)=\sum_{i=0}^{n}b_iB_i^n(t)\le\max\{b_i\}_{i=0}^n=M, \tag{2}$$

where m and M are the minimal and maximal control data. Let us consider the bounds on the function extrema given by Equation (2). These are sharp if and only if the extrema are taken at one of the endpoints. Otherwise, the effect of the interior control points is overestimated by the convex hull property. Indeed, if $b_0\le M-\epsilon$, $b_n\le M-\epsilon$ for some $\epsilon>0$ and $b_i=M$, $i=1,\dots,n-1$, then

$$b(t)\le(M-\epsilon)B_0^n(t)+\sum_{i=1}^{n-1}M\cdot B_i^n(t)+(M-\epsilon)B_n^n(t) \tag{3}$$

$$=M-\epsilon B_0^n(t)-\epsilon B_n^n(t)<M \tag{4}$$

Degree	2	3	4	5	6	7	8	9	10	15	20
Control point	47.89%	60.10%	68.07%	72.25%	72.59%	77.35%	76.38%	78.36%	79.56%	84.36%	88.45%
LU bound	28.54%	42.31%	49.73%	54.79%	58.27%	59.96%	51.31%	59.02%	50.84%	60.22%	60.71%

Table 2: *Comparison of relative errors for function extrema bounds. Errors are calculated as the difference between estimated and actual extrema, normalised by the magnitude of the extremum. "Control Point" utilises the minimum and maximum control points of univariate ($\mathbb{R} \to \mathbb{R}$) Bézier curves. "LU Bound" refers to our proposed method defined in Equation (7). Lower values indicate sharper bounds.*

Surface	Equation $f(x,y,z) =$	Note
Sphere	$x^2+y^2+z^2-R^2$	
Torus	$(x^2+y^2+z^2+R^2-r^2)^2-4R^2(x^2+y^2)$	
Flirt	$x^2-x^3+y^2+y^4+z^3-10z^4$	
Calyx	$x^2+y^2z^3-z^4$	Has a narrow cusp (Figure 2a)
Barth sextic	$(-2\phi-1)(x^2+y^2+z^2-1)^2+4(x^2\phi^2-y^2)(z^2\phi^2-x^2)(y^2\phi^2-z^2)$	50 real ordinary double points
Endraß octic	$64(x^2-1)(y^2-1)((x-y)^2-2)((x+y)^2-2)-((8(2+\sqrt{2})z^2+4+14\sqrt{2})\cdot(x^2+y^2)-4(1+\sqrt{2})(x^2+y^2)^2-16z^4+8(1-2\sqrt{2})z^2-12\sqrt{2}-1)^2$	144 real ordinary double points (Figure 2c)
Barth decic	$(5\phi+3)(r^2-1)^2(r^2+\phi-2)^2+8(x^4-2x^2y^2-2x^2z^2+y^4-2y^2z^2+z^4)(x^2-y^2\phi^4)(z^2-x^2\phi^4)(y^2-z^2\phi^4)$	300 real ordinary double points
16-Ball	$x^{16}+y^{16}+z^{16}-1$	Unit ball in ℓ_{16}-norm (Figure 2b)

Table 3: *Algebraic test surfaces. $\phi = \frac{1+\sqrt{5}}{2}$ The surface is defined by the zero-contour of the equations.*

as $\epsilon \cdot \left(B_0^n(t)+B_n^n(t)\right) = 0$ if and only if $\epsilon = 0$, since $\forall t \in [0,1]$: $B_0^n(t)+B_n^n(t) > 0$. Consequently,

$$l(t) = m+(b_0-m)(1-t)^n+(b_n-m)t^n \tag{5}$$

$$u(t) = M-(M-b_0)(1-t)^n-(M-b_n)t^n \tag{6}$$

are lower and upper bounding curves of $b(t)$, respectively. We refer to these as the LU functions. The extrema of $l(t)$ and $u(t)$ are strictly sharper than the control point extrema if the latter is not taken at b_0 or b_n. By differentiating $l(t)$, one notes that the resulting form

$$l'(t) = n\left(-(b_0-m)(1-t)^{n-1}+(b_n-m)t^{n-1}\right) \tag{7}$$

possesses a closed-form solution for the root, namely,

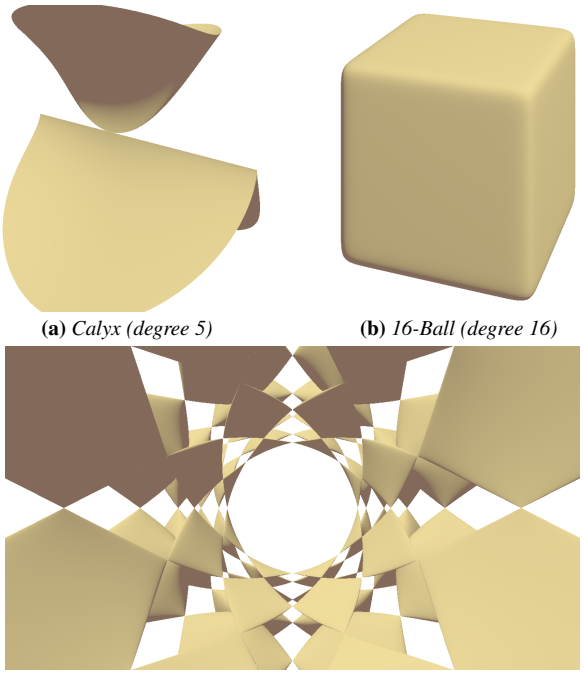
$$t = \left(1+\sqrt[n-1]{\frac{b_n-m}{b_0-m}}\right)^{-1}. \tag{8}$$

A similar result can be obtained for $u(t)$. We refer to these as LU bounds. Table 2 shows an empirical comparison of using these bounds to estimate the extrema of a Bézier curve versus the extrema of the control points. The tests were carried out on curves with random coefficients in $[0,1]$, averaging the statistics over 100 curves. The test was implemented in Python, using NumPy. On average, our method provided about 20% sharper bounds on the extrema than the estimations based on the extremal control points.

It is also worth noting that this bound can be trivially adapted to subintervals of $[0,1]$. Indeed, if the extrema of the bound is not in $[a,b]$, then it is taken at either a or b.

4. Test Results

We implemented the proposed modifications to Yuksel's root finder and the Lipschitz bound estimation in an NVIDIA Falcor 8 application, using its DirectX 12 backend. We ran our tests on an NVIDIA 1660 Ti mobile GPU. We tested our method on a selection of challenging surfaces (Table 3). Three of them are shown on Figure 2.



(a) *Calyx (degree 5)* **(b)** *16-Ball (degree 16)*

(c) *Endraß octic (degree 8)*

Figure 2: *Algebraic surfaces used for testing*

We visually ensured that the surfaces are visualised without rendering artifacts. The resulting image is notably more accurate around thin features than the ray march algorithm used as the ground truth. Our code is available at https://github.com/szentep/Tracers.

The performance measurements for rendering with the modified Yuksel method are detailed in Table 4. On average, it provides a 23.6× speedup compared to ray marching. The computational overhead of the EFT and direct surface evaluation (EVAL) was ap-

Surface	Degree	Total runtime	EFT		EVAL		Ray march	
		ms	ms	%	ms	%	ms	speedup
Sphere	2	0.23	0.09	39%	-0.02	-8%	8.7	38.4×
Torus	4	0.69	0.17	25%	0.11	16%	22.9	33.2×
Calyx	5	1.44	0.49	34%	0.17	12%	37.1	25.7×
Barth sextic	6	2.62	0.85	32%	0.45	17%	67.8	25.9×
Endraß octic	8	3.07	0.30	10%	0.12	4%	79.3	25.8×
Barth decic	10	4.56	1.14	25%	0.79	17%	57.8	12.7×
16-Ball	16	4.12	0.19	5%	0.38	9%	13.8	3.3×

Table 4: *Comparison of single-frame GPU dispatch times across test surfaces compared to ray marching. Total runtime (ms) indicates the proposed method in the monomial basis using the modified bracketed Newton root finder. Performance overheads of EFT and direct surface evaluation (EVAL) were isolated by disabling the feature and are presented as a percentage of the total. Ray marching has an iteration cap at* 10 000 *steps to ensure resolution of fine surface details.*

	Degree	LU bound	Control point	Speedup	Raymarch	
		ms	bound (ms)		ms	Speedup
Sphere	2	0.92	1.53	1.67×	8.7	9.5×
Flirt	4	2.18	3.82	1.75×	25.3	11.6×
Calyx	5	7.80	8.06	1.03×	37.1	4.8×
Barth sextic	6	31.88	32.24	1.01×	67.8	2.1×

Table 5: *Single-frame GPU dispatch times across test surfaces compared to the LU bound. Total runtime (ms) denotes the proposed method in the Bernstein basis utilising LU bounds for local Lipschitz tracing, with a maximum iteration count of* 10 000.

proximated by measuring the time difference when these features are disabled. While this alters image fidelity (Figure 1d), the theoretical impact on performance metrics is negligible. For the sphere, direct surface equation evaluation is faster than that of the degree-two univariate polynomial.

We evaluated the use of LU bounds for local Lipschitz ray tracing and efficient ray culling. We used the extrema of the derivatives as a Lipschitz constant throughout the ray. The results are shown in Table 5. While we omitted higher-degree surfaces due to running out of the iteration cap, we observed significant performance gains for lower-degree surfaces. On average, it is 4.8× faster than ray marching. LU bounds provide more robust and efficient baselines than ray marching and are consistently faster than merely using the largest magnitude coefficient of the derivative for a Lipschitz constant upper bound.

5. Conclusions

We presented a robust method for rendering algebraic surfaces in dynamic scenes on consumer GPUs using single-precision arithmetic. We employed EFTs and a hybrid root-finding strategy that uses a univariate polynomial proxy for isolation and the original function for refinement. This framework successfully mitigated the numerical instabilities inherent to polynomial fitting on our test set. Our modified Yuksel algorithm with the numerical robustness changes eliminated visual artifacts and achieved 23.6× faster rendering compared to ray marching.

Additionally, we introduced a novel bounding technique for Bernstein polynomials that provides tighter extrema estimates than standard control point bounds, improving culling efficiency for lower-degree surfaces. This provided a 7× average speedup compared to ray marching and a 1.37× speedup compared to control point-based bounds. Future work includes reducing the computational overhead of EFT through adaptive precision strategies and improving our bounding methods or extending them to higher dimensions. Moreover, we did not employ the interval-dependent nature of our LU bounding approach, which may provide additional performance improvements. Ultimately, this pipeline enables the interactive, high-fidelity visualisation of complex algebraic surfaces on standard GPU architectures.

References

[Han83] HANRAHAN P.: Ray tracing algebraic surfaces. *SIGGRAPH Computer Graphics 17*, 3 (July 1983), 83–90. 1

[Har96] HART J. C.: Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces. *The Visual Computer 12*, 10 (Dec. 1996), 527–545. 2

[KB89] KALRA D., BARR A. H.: Guaranteed Ray Intersections with Implicit Surfaces. SIGGRAPH '89, pp. 297–306. 2

[KV25] KARIKÓ C. C., VALASEK G.: Real-time rendering of algebraic surfaces via polynomial fitting. In *Computer Science Research Notes* (Jan. 2025), CSRN, University of West Bohemia, Czech Republic. doi:10.24132/csrn.2025-28. 1, 2

[PPUJ23] PETERS C., PATEL T., USHER W., JOHNSON C. R.: Ray Tracing Spherical Harmonics Glyphs. In *Vision, Modeling, and Visualization* (2023), Guthe M., Grosch T., (Eds.), The Eurographics Association. doi:10.2312/vmv.20231223. 1

[ROO08] RUMP S. M., OGITA T., OISHI S.: Accurate floating-point summation part i: Faithful rounding. *SIAM Journal on Scientific Computing 31*, 1 (Jan. 2008), 189–224. doi:10.1137/050645671. 2

[Tha06] THALL A.: Extended-precision floating-point numbers for GPU computation. In *ACM SIGGRAPH 2006 Research posters on - SIGGRAPH '06* (2006), SIGGRAPH '06, ACM Press, p. 52. 1, 2

[Yuk22] YUKSEL C.: High-performance polynomial root finding for graphics. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 5*, 3 (July 2022), 1–15. 1