Real-Time Rendering of Algebraic Surfaces

Csongor Csanád Karikó[1] and Gábor Valasek[1]

Eötvös Loránd University, Hungary
{valasek, jpoiwr}@inf.elte.hu

Abstract

We investigate the problem of robust and real-time rendering of algebraic surfaces. We show that expressing the intersection of the ray and the algebraic surface as a single univariate polynomial is not robust in practice, comparing results between monomial, Bernstein, Lagrange, and Chebyshev basis fits. We show that fitting multiple polynomials over subintervals, such as a unit length subdivision of the ray extent within the region of interest, improves robustness at a negligible performance cost.

CCS Concepts

• ***Computing methodologies*** → ***Ray tracing;***

1. Introduction

Rendering surfaces defined as isocontours of general $f : \mathbb{R}^3 \to \mathbb{R}$ mappings is a challenging task. Achieving high performance and robustness simultaneously is only viable for constrained subsets of such mappings, each requiring its own specialized algorithms.

For example, if $f(x,y,z)$ is the signed distance to the closest isocountor point, or a lower bound thereof, it is possible to use sphere tracing [Har96] or one of its variants to realize both of these objectives. This property may be relaxed to f being Lipschitz-continous with a known or easily computable Lipschitz-constant, in which case sphere tracing methods remain applicable [KB89].

In general, the rendering task is formulated as finding the smallest positive $t \in \mathbb{R}$ such that $f(\boldsymbol{o} + t \cdot \boldsymbol{d}) = 0$, where $\boldsymbol{o}$ is the origin of the ray and $\boldsymbol{d}$ is its direction. A family of methods involve fitting one or more polynomials to $f(t)$, reducing the problem to one dimensional root finding [WMK24]. With this approach, there are multiple choices to be made surrounding the used polynomial basis, the fitting method, and the root finder. Rendering artifacts, such as shown on Figure 1, may originate from inaccurate fitting, numerically unstable evaluation, and imprecise root finding. We used the Barth sextic for illustration, which is defined as the zeros of

$$f(x,y,z) = 4(\phi^2x^2 - y^2)(\phi^2y^2 - z^2)(\phi^2z^2 - x^2) - (1+2\phi)(x^2+y^2+z^2-1)^2, \tag{1}$$

where ϕ is the golden ratio.

The purpose of this paper is to investigate the numeric stability of different polynomial bases when rendering algebraic surfaces in a finite subset of space. That is, we assume that $f(x,y,z)$ is a polynomial of known degree and we visualize it within e.g. an axis-aligned bounding box. Additionally, we narrow down our discussion to methods that do not rely on any kind of preprocessing to compute auxiliary data such as acceleration structures.

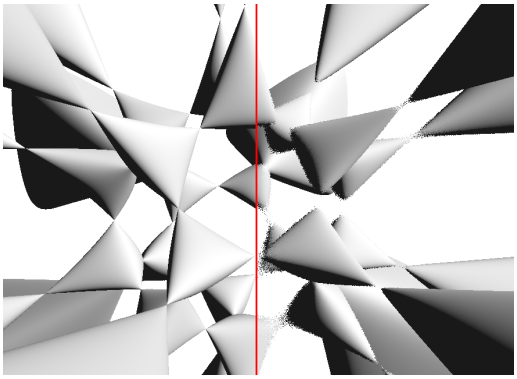


Figure 1: *Rendering the Barth sextic with a single degree 6 polynomial fit along each ray. A reference render with direct raymarching is shown on the left. We used the monomial basis for fitting a univariate polynomial on the right. Both images were rendered with ray marching. That is, we took equidistant steps along the ray until the sign of f or its 1D polynomial approximation changed. Note the noise in the center of the right image.*

2. Background

Let us denote the basis functions of a degree N polynomial with $M_i(x), L_i(x), B_i(x), T_i(x)$ and its corresponding coefficients with $c_i, i = 0,1,\ldots,N$. We investigate the following bases:

Monomial	$M_i(x) = x^i$
Lagrange	$L_i^N(x) = \prod_{\substack{k=0 \\ k\neq i}}^{N} \frac{x-x_k}{x_i-x_k}$
Bernstein	$B_i^N(x) = \binom{N}{i} x^i (1-x)^{N-i}$
Chebyshev	$T_i(x) = \cos(i\cos^{-1}(x))$

DOI: 10.2312/egp.20251026

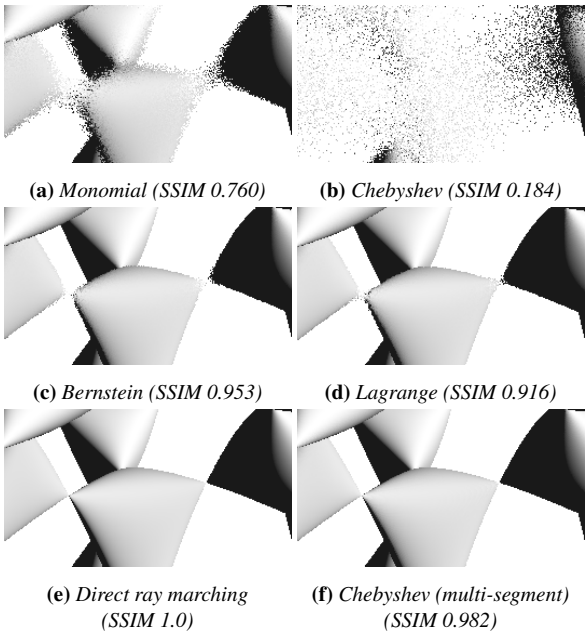
2.1. Polynomial fitting

Let $h(x) = \sum_{i=0}^{n} c_i h_i^n(x)$ denote a polynomial in some $h_i^n(t) : \mathbb{R} \to \mathbb{R}$ basis, represented by the coefficients c_i, $i = 0, \ldots, n$. If an input function is sampled at some x_i, $i = 0, \ldots, n$ points, one can compute the coefficients c_i that make $h(t)$ interpolate these by solving

$$\underbrace{\begin{bmatrix} h_0^n(x_0) & h_1^n(x_0) & \ldots & h_n^n(x_0) \\ h_0^n(x_1) & h_1^n(x_1) & \ldots & h_n^n(x_1) \\ \ldots & \ldots & \ldots & \ldots \\ h_0^n(x_n) & h_1^n(x_n) & \ldots & h_n^n(x_n) \end{bmatrix}}_{H} \cdot \underbrace{\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}}_{\boldsymbol{c}} = \underbrace{\begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}}_{\boldsymbol{f}} \tag{2}$$

by precomputing the double precision QR decomposition of H and then solving $R = Q^T \boldsymbol{f}$ with backsubstitution.

We used the Chebyshev-Lobatto grid on $[0,1]$ for x_i, as it improves the accuracy of the interpolation.



(a) *Monomial (SSIM 0.760)* **(b)** *Chebyshev (SSIM 0.184)*

(c) *Bernstein (SSIM 0.953)* **(d)** *Lagrange (SSIM 0.916)*

(e) *Direct ray marching (SSIM 1.0)* **(f)** *Chebyshev (multi-segment) (SSIM 0.982)*

Figure 2: *Part of the Barth sextic rendered with different polynomial bases. Figures 2a,2b,2c and 2d show images rendered with a single polynomial segment per ray and their structural similarity index compared to Figure 2e. Note the noise on these around thin parts. Figure 2f shows how splitting into multiple segments increases stability.*

3. Test results

To assess the stability of the different polynomial bases, we rendered challenging algebraic surfaces by fitting a single polynomial per ray and then ray marching the fitted polynomial. As showcased by Figure 2, all bases exhibit rendering artifacts. To check the origin of these errors, we rendered the same surfaces by ray marching the surface function directly (Figure 2e).

For a slight performance cost, the accuracy of rendering can be significantly improved by splitting the ray into segments, upon which separate polynomials are fitted. We used unit length ray segments (Figure 2f). We also experimented with different splitting heuristics but we are yet to find a robust, automatic method.

Basis	Single segment (ms)	Multi segment (ms)
Monomial	7.46	9.62
Bernstein	400.1	401.96
Lagrange	33.65	33.2
Chebyshev	10.33	12.2
Direct ray march	14.32 ms	

Table 1: *Comparison of the render time of a single* 1920×1080 *frame for the different bases. The step length for ray march is the same for all methods. Multi segment versions may split the ray into a maximum of 10 unit length segments The last row contains timings using ray marching on the original* $f(x,y,z)$ *function. Ray marching used a step length of 0.005 in both cases.*

The focus of this paper is on the effect of the choice of basis on stability and performance. Therefore, due to its simplicity, we used ray marching to find the root of the polynomial. This enables us to focus on the stability of interpolation and evaluation.

We used numerically stable methods for evaluating the polynomials. For monomial and Chebyshev basis polynomials, we used Horner's and Clenshaw's algorithms, respectively. Lagrange polynomials were evaluated with the barycentric evaluation formula while Bernstein polynomials with the de Casteljau algorithm.

We implemented these techniques in C++ using the Falcor system. The tests were carried out on a laptop with an AMD Ryzen 7 7840HS CPU and NVIDIA RTX 4060 laptop GPU. The reported performance figures are timing reports of the render dispatch taken in Nsight. The results are shown in Table 1.

4. Conclusions

We showed that converting algebraic surfaces to univariate polynomials along rays is a viable, albeit nontrivial way to render ray-surface intersections. In theory, sampling the surface function at $N+1$ positions along they ray provides all the necessary points to interpolate $f(t)$ exactly. In practice, however, the fitting and evaluation of such polynomials is not accurate enough, regardless of the basis. For challenging surfaces, such as the Barth sextic, it is necessary to split the ray into multiple segments. The monomial and Chebyshev bases offer better performance than the reference ray march on the three-variable input. The quadratic complexity of the de Casteljau algorithm makes it not viable for real-time rendering.

References

[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. 1

[KB89] KALRA D., BARR A. H.: Guaranteed Ray Intersections with Implicit Surfaces. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques* (July 1989), SIGGRAPH '89, ACM, pp. 297–306. 1

[WMK24] WINCHENBACH R., MÖLLER M., KOLB A.: Lipschitz-agnostic, efficient and accurate rendering of implicit surfaces. *The Visual Computer* (01 2024), 1–20. 1