ISSN 2464-4617 (print)
ISSN 2464-4625 (online)
Computer Science Research Notes - CSRN
http://www.wscg.eu
WSCG 2025 Proceedings

Real-time Rendering of Algebraic Surfaces via Polynomial Fitting

Csongor Csanád Karikó
Eötvös Loránd University
Pázmány P. stny 1/C.
Hungary 1117, Budapest
jpoiwr@inf.elte.hu

Gábor Valasek
Eötvös Loránd University
Pázmány P. stny 1/C.
Hungary 1117, Budapest
valasek@inf.elte.hu

ABSTRACT

We investigate the problem of robust real-time algebraic surface rendering. We show that expressing the composition 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, is a viable approach to overcome the robustness issues. In addition, we discuss a heuristic method for determining split locations.

Keywords

Computer Graphics, Algebraic Surfaces, Ray Marching, Polynomial Interpolation

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 special subsets of such mappings. For example, if $f(\boldsymbol{x}) = f(x,y,z)$ is the signed distance to the closest isocontour point, or a lower bound thereof, it is possible to use sphere tracing [Hart96] or one of its variants [Kein14, Bal18, Ban23] to realize both of these objectives.

However, stipulating such a distance property on $f(\boldsymbol{x})$ is a rather constraining requirement to adhere to, especially for closed-form formulations. Instead, it can be relaxed to prescribing the existence of computable Lipschitz bounds along ray segments [Kal89, Gal20], that is, to be able to compute $L > 0$ scalars such that $|f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq L\|\boldsymbol{y} - \boldsymbol{x}\|_2$ holds for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^3$ on the ray segment. As $\frac{f(\boldsymbol{x})}{L}$ is a signed distance lower bound, these surfaces can be rendered with any of the sphere tracing techniques. Still, the actual process of deriving the Lipschitz bounds has to be worked out in a case by case or a function class by function class manner.

Our research focuses on rendering algebraic surfaces in a finite subset of space, that is, we assume that $f(\boldsymbol{x})$ is a polynomial of known degree and we visualize it within e.g. an axis-aligned bounding box. We do not assume any distance-like behavior on $f(\boldsymbol{x})$. 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.

We investigate the numeric precision and runtime performance of fitting one or more polynomials to f along each ray. In theory, choosing the same degree for the fitted polynomial as that of the original surface function provides a polynomial that exactly matches f along the ray. In practice, our experiments showed that the conversion from the trivariate formulation to the univariate polynomial is hindered by numerical errors enough to cause visual distortions on the GPU, such as in the case of the Barth sextic surface shown in Figure 1. Therefore, we mainly focus on the numeric stability of fitting and the polynomial evaluation, excluding the root-finding stage of the rendering.

We examined multiple different polynomial bases, all of which exhibited numeric issues of varying degrees. We show how splitting the polynomials improves numeric stability and simple heuristics to determine when such splits are necessary.

In the next section, we examine the polynomial fitting and evaluation methods relevant to our work. Thereafter, in Section 3, we discuss our rendering framework and the numerical issues of using a single segment and splitting into multiple segments. Section 4 presents the results of our performance evaluation and we conclude our findings in Section 5.

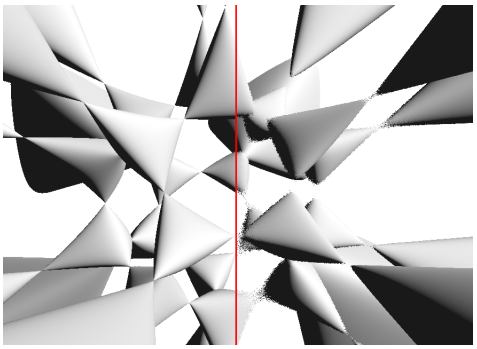


Figure 1: Rendering the Barth sextic with a single degree 6 polynomial fit along each ray within an axis-aligned bounding box of dimensions $10 \times 10 \times 10$ centered at the origin. A reference render with direct ray-marching using $f(\boldsymbol{x})$ surface function evaluations is shown on the left. We used an univariate polynomial in the monomial bases on the right and sampled it during ray marching instead of evaluating $f(\boldsymbol{x})$ directly. Note the noise in the center of the right image.

2 BACKGROUND AND RELATED WORK

2.1 Polynomial fitting

The goal of polynomial fitting is to obtain a function of the form $p(x) = \sum_{i=0}^{N} c_i b_i(x)$ matching a set of $(x_0, y_0), (x_1, y_1), \ldots, (x_N, y_N)$ data samples. The polynomial is uniquely defined by its c_i coefficients ($i = 0, 1, \ldots, N$) and $b_i(x) : \mathbb{R} \to \mathbb{R}$ basis functions.

We investigated the monomial, Bernstein, Lagrange, and Chebyshev polynomial bases. Despite its numeric drawbacks, monomial bases are ubiquitous and are often a default for many practitioners. The Bernstein bases offer significantly better numeric properties [Far96] and deeper geometric insights, however, traditional evaluation algorithms pose increased computational costs. From an interpolation point of view, fitting is the most trivial in the Lagrange basis, whereas the Chebyshev bases offers more efficient means of evaluation.

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

Basis	Degree 3	5	10
Monomial	85.21	2771.06	17686488.95
Bernstein	4.32	16.56	510.28
Chebyshev	66.87	2255.22	13933909.74
DCT	1.41	1.41	1.41
Lagrange	1.00	1.00	1.00

Table 1: L^2 condition numbers of interpolation matrices generated from different bases and using the Chebyshev-Lobatto grid over [0,1]. DCT represents the Chebyshev basis, but with the Discrete Cosine Transform used for the interpolation, instead of the matrix in equation 1.

In general, the fitting of a polynomial of degree N to a set of $N+1$ sample points can be accomplished by solving the following linear system of equations:

$$\underbrace{\begin{bmatrix} b_0^N(x_0) & b_1^N(x_0) & \ldots & b_N^N(x_0) \\ b_0^N(x_1) & b_1^N(x_1) & \ldots & b_N^N(x_1) \\ \ldots & \ldots & \ldots & \ldots \\ b_0^N(x_N) & b_1^N(x_N) & \ldots & b_N^N(x_N) \end{bmatrix}}_{H} \cdot \underbrace{\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_N \end{bmatrix}}_{\boldsymbol{c}} = \underbrace{\begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_N) \end{bmatrix}}_{\boldsymbol{s}} \tag{1}$$

where H is a matrix constructed from the $\{b_i^N(x)\}_{i=0}^N$ polynomial bases and x_i are the sample positions. We use the Chebyshev-Lobatto grid $x_i = \cos\left(\frac{2i+1}{2(N+1)}\pi\right), i = 0, 1, \ldots, N$ (Chebyshev nodes), as it offers optimal conditioning for the fitting matrices.

H^{-1} is precomputed in our implementation and is incorporated into the GPU shader as a constant. Thus, at runtime, we only need to multiply the sample vector $\boldsymbol{s}$ with the precalculated H^{-1} from the left. Since the embedded constant matrices depend on the x_i sample positions, their distribution cannot be changed at runtime. It is also necessary to normalize the domain of the interpolated polynomials, for example, to $[0; 1]$.

Unfortunately, for most polynomial bases H^{-1} is ill-conditioned, which means that rounding errors in $\boldsymbol{s}$ cause significant errors in the interpolation. For example, when using the monomial basis, matrix H takes the form of a Vandermonde matrix, which is notorious for being ill-conditioned. Table 1 shows a comparison between the L^2 condition numbers of different bases.

When using the Chebyshev basis, it is possible to do the interpolation with the discrete cosine transform (DCT). This results in a significantly better conditioned matrix, as shown in Table 1. The DCT matrix D corresponding to degree N interpolation can be constructed with the following formul.

$$D = \frac{1}{N+1} \cdot \begin{bmatrix} 1 & 1 & \dots & 1 \\ 2d_{2,1} & 2d_{2,2} & \dots & 2d_{2,N+1} \\ \dots & \dots & \dots & \dots \\ 2d_{N+1,1} & 2d_{N+1,2} & \dots & 2d_{N+1,N+1} \end{bmatrix}$$

where

$$d_{i,j} = \cos\left((i-1)\frac{(2(N+1-j)+1)\pi}{2(N+1)}\right) .$$

Once the D matrix is precalculated, Chebyshev coefficients are calculated by multiplying it with the samples taken at the Chebyshev-Lobatto grid: $\cos\left(\frac{2k+1}{2(N+1)}\pi\right), k = 0,1,\dots,N$. Thus, we obtain the final Chebyshev basis coefficients with $\boldsymbol{c} = D \cdot \boldsymbol{s}$.

2.2 Evaluation

In this section, we discuss the algorithms employed for evaluating the polynomials.

We used Horner's method to evaluate polynomials in the **monomial** basis, as shown in Algorithm 1.

Algorithm 1 Horner's scheme

Input: $c_0, c_1, \dots, c_N$ coefficients,
$N \in \mathbb{N}^+, x \in \mathbb{R}$
Output: $y \in \mathbb{R}$

$y \leftarrow c_N$
for i $= N-1, \dots, 0$ **do**
  $y \leftarrow y \cdot x + c_{\text{i}}$
end for
return y

We used the De Casteljau algorithm to evaluate polynomials in the **Bernstein** basis, as shown in Algorithm 2.

Algorithm 2 De Casteljau's algorithm

Input: $c_0, c_1, \dots, c_N$ coefficients,
$N \in \mathbb{N}^+, x \in \mathbb{R}$
Output: $y \in \mathbb{R}$

$q \leftarrow [c_0, c_1, \dots, c_N]$ ▷ Length $N+1$
for i $= 1, \dots, N$ **do**
  for j $= 0, \dots, N-$ i **do**
    $q_{\text{j}} \leftarrow (1-x) \cdot q_{\text{j}} + x \cdot q_{\text{j}+1}$
  end for
end for
return q_0

We used the Clenshaw algorithm [Cle55] for polynomials in the **Chebyshev** basis, as shown in Algorithm 3.

Algorithm 3 Clenshaw's algorithm

Input: $c_0, c_1, \dots, c_N$ coefficients,
$N \in \mathbb{N}^+, x \in \mathbb{R}$
Output: $y \in \mathbb{R}$

$b_1, b_2, b_c \leftarrow 0$
for i $= N, \dots, 1$ **do**
  $b_c \leftarrow c_{\text{i}} + 2b_1 \cdot x - b_2$
  $b_2 \leftarrow b_1$
  $b_1 \leftarrow b_c$
end for
return $c_0 + b_1 \cdot x - b_2$

Lagrange polynomials were evaluated with the barycentric interpolation formula. Barycentric weights $w_0, w_1, \dots, w_N$ were precomputed and stored in the shader as a constant array. These weights depend on the set of sample positions $x_0, x_1, \dots, x_N$ which are the Chebyshev nodes in our case. They are given by the following formula:

$$w_i = \prod_{\substack{k=0 \\ k \neq i}}^{N} \frac{1}{x_i - x_k} . \tag{2}$$

Algorithm 4 describes the part of the process implemented in the shader.

Algorithm 4 Barycentric Lagrange interpolation

Input: $c_0, c_1, \dots, c_N$ coefficients,
$n_0, n_1, \dots, n_N$ Chebyshev nodes,
$w_0, w_1, \dots, w_N$ barycentric weights,
$N \in \mathbb{N}^+, x \in \mathbb{R}$
Output: $y \in \mathbb{R}$

$y, \text{denom} \leftarrow 0$
for i $= 0, \dots, N$ **do**
  if $|x - n_i| < 10^{-6}$ **then**
    return c_i
  end if
  $d \leftarrow \frac{w_i}{x - n_i}$
  denom $\leftarrow$ denom $+ d$
  $y \leftarrow y + d \cdot c_i$
end for
return $\frac{y}{\text{denom}}$

3 RENDERING FRAMEWORK

Our goal is to render algebraic surfaces without preprocessing the input geometry.

For each ray, we compute the intersection between the ray and the bounding box of the algebraic surface. If the total degree of the algebraic surface is N, we take samples of it at $N+1$ Chebyshev nodes. The samples are

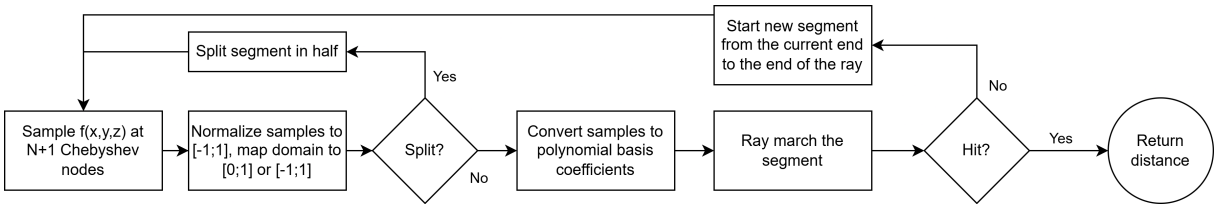


Figure 2: Our rendering pipeline. Note that there is a limit on the length a segment can be split into.

normalized by division with the largest absolute value among them. Then, a degree N polynomial is fit to the $N+1$ normalized samples. The control data are computed with precomputed inverted H matrices. The coefficients obtained in this way represent the algebraic surface along the ray for the given frame. Note that these may change in subsequent frames, as the methods tested here allow arbitrary temporal complexity in how the coefficients of the algebraic polynomials change.

Once an univariate polynomial is obtained, we apply ray marching to find its smallest positive root. Our goal is to investigate the interpolation and evaluation, therefore, we constrain ourselves to the simplest possible root-finding method: ray marching. This way, it is possible to directly compare the accuracy of the interpolated polynomial with the original surface function.

The complete pipeline is shown in Figure 2.

3.1 Single segment results

We used the Barth sextic as our main test surface. It is defined as the zero contour of the following function:

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

where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.

We ray march univariate polynomial fits to compare the exact reproduction capabilities of various polynomial bases. Figure 3 shows a comparison of the results achieved with the four bases as well as an image rendered via directly ray marching the trivariate surface function f. Tracing was done with the same world-space step length and the same maximum number of iterations, ensuring a fair comparison between the methods.

The ill-conditioned fitting matrices of the monomial and non-DCT Chebyshev bases cause the rendered image to contain a significant amount of noise around thinner parts of the surface. Despite having a better conditioned fitting matrix, the Bernstein basis still exhibits similar artifacts, as shown in Figure 3d. The same errors are also present in Image 3e, which was rendered using Lagrange interpolation. Considering that the Lagrange basis has the best conditioned fitting matrix possible, that is, the identity matrix, the question arises as to whether the bottleneck of precision lies in the evaluation.

To investigate this, we extracted function samples and polynomial evaluations from the shader. The resulting plots are displayed in Figure 4. We draw two conclusions from this data. The first is that after sample normalization, the negative range of the polynomial around the root becomes very small (smaller than the machine epsilon for 32 bit floating point numbers). This makes root-finding an immensely difficult and unstable task. Secondly, rounding errors in the samples seem to affect the accuracy of interpolation significantly compared to the negative range of the function.

3.2 Splitting into multiple segments

As discussed in the previous section, the main issue with the single-segment method lies in the range of the interpolated function over the domain of the ray. On one hand, the normalization of samples before interpolation is necessary to avoid the precision issues arising from doing arithmetic with large floating point numbers. On the other hand, this normalization may decrease the absolute range of the function around the root too much, making the root almost impossible to find. Such a case is shown in Figure 4, where normalization decreases the negative range around the root as much that the rounding errors of the samples cause the interpolated polynomial to miss the root completely.

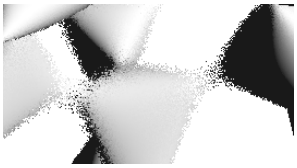
This problem can be remedied by splitting the ray into multiple segments and constructing separate interpolating polynomials covering each segment. This way the range of values covered by each polynomial is reduced, as shown in Figure 5. The questions we intend to investigate in this section are how to determine whether we need to split an interval and where exactly the ray should be split. We investigated a multitude of possible basis-dependent or independent splitting heuristics. In this section, we cover the most reliable method according to our experiments.

3.2.1 Uniform length segments

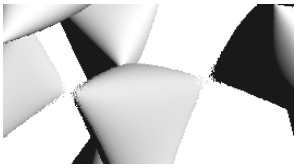
The simplest way to split the ray is to create M segments of uniform length at each pixel. In our experiments, shown in Table 2, ray marching clearly dominates the runtime cost of polynomial interpolation. Therefore, it is practical to use this method for this root-finding algorithm. The only two parameters are the maximum number of segments and the length of each segment.

(a) Reference direct ray march at 1920x1080.

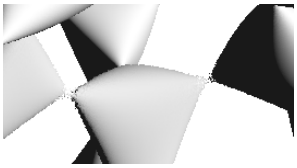
(b) Direct ray marching (reference)



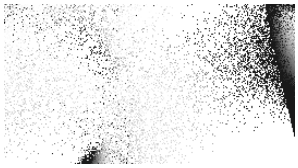
(c) Monomial



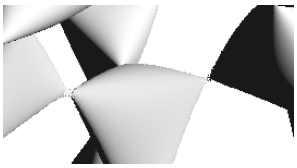
(d) Bernstein



(e) Lagrange



(f) Chebyshev



(g) Chebyshev (DCT)

(h) Chebyshev (uniform multi-segment)

Figure 3: The Barth sextic rendered with different polynomial bases. Figure 3a shows the complete rendered image. Figures 3c, 3d, 3e, 3f and 3g show the part highlighted with a red border, rendered with a single polynomial segment per ray in different bases. Note the noise around thin parts on these images. Figure 3h shows how splitting into multiple segments increases stability, even when using the same unstable interpolation method as on 3f.

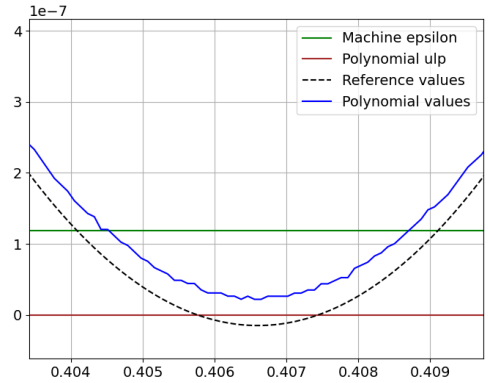


Figure 4: Comparison of surface function and its interpolating polynomial around the closest root at a noisy pixel of Figure 3e. The polynomial is of degree 6 and it was computed via barycentric Lagrange interpolation. The world-space distance covered by the ray is approximately 10.88 units. Note the range of this figure.

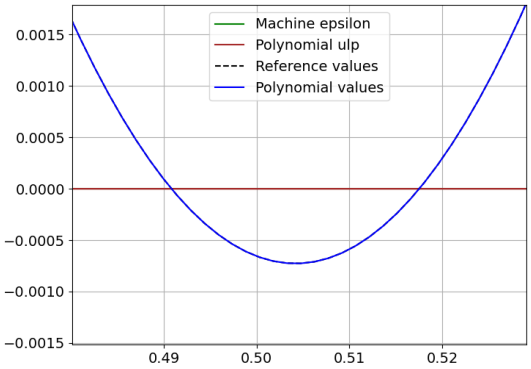


Figure 5: The same roots as shown in Figure 4, but interpolated over a smaller segment covering a distance of approximately 0.68 world-space units. Note that the location of the roots changed due to both the domain being normalized to $[0, 1]$ and the range normalized by the maximum absolute sample.

It must be noted that ray marching should be replaced with faster, more advanced root-finding methods in practice, as it does not provide robust first root identification properties. This may change the distribution of the necessary calculations such that the polynomial interpolation becomes the dominant part of the algorithm. This is our motivation to reduce the number of segments whenever splitting does not yield additional visual improvements to the final image.

3.2.2 Basis-agnostic split heuristics

Algorithm 5 describes a framework for splitting the polynomial with heuristics based on samples taken from the surface function. The benefit of such heuristics is that they are inexpensive to calculate as the coefficients need not be interpolated. This is especially beneficial when multiple splits occur in sequence, since this way we skip the unnecessary interpolations before the final segment is determined for root-finding.

Algorithm 5 Rendering framework employing sample based splitting heuristics

Input: $f : \mathbb{R}^3 \to \mathbb{R}$ surface function,
  $n_0, n_1, \ldots, n_N$ Chebyshev nodes,
  $N \in \mathbb{N}^+$, $\boldsymbol{o} \in \mathbb{R}^3$ ray origin,
  $\boldsymbol{d} \in \mathbb{R}^3$ ray direction,
  $t_{\min}, t_{\max}$ ray ends, ε step length
Output: $r \in \mathbb{R}$ root distance

begin $\leftarrow t_{\min}$
end $\leftarrow t_{\max}$
$i \leftarrow$ max_iters
$s \leftarrow 0$ ▷ split counter
while $i \geq 0$ & $s \leq$ max_splits **do**
  $S \leftarrow [0, 0, \ldots, 0]$ ▷ $N+1$ samples
  for $k = 0, \ldots, N$ **do**
    $S_k \leftarrow f(\boldsymbol{o} + \boldsymbol{d} \cdot (n_k \cdot (\text{end} - \text{begin}) + \text{begin}))$
  end for
  $\alpha \leftarrow \max_{k=0}^{N} |S_k|$ ▷ normalization factor
  $S \leftarrow \left[\frac{S_0}{\alpha}, \frac{S_1}{\alpha}, \ldots, \frac{S_N}{\alpha}\right]$
  do_split, split_pos $\leftarrow$ SplitHeuristics(S)
  if do_split & $s <$ max_splits **then**
    end = split_pos
    $s \leftarrow s + 1$
  else
    $C \leftarrow$ Interpolate(S) ▷ coefficients
    $l \leftarrow \frac{\varepsilon}{\text{end} - \text{begin}}$ ▷ step length
    found, root, iters $\leftarrow$ RayMarch(C, i, l)
    if found **then**
      return root $\cdot$ (end $-$ begin) $+$ begin
    else
      begin $\leftarrow$ end
      end $\leftarrow t_{\max}$
      $i \leftarrow i -$ iters
    end if
  end if
end while
return -1 ▷ no root found

The range normalization problem discussed in Section 3.1 inspires a heuristic condition directly. If there is a large enough difference between the magnitude of surface function samples, the ray should be split into two. The condition is recursively checked until we arrive at a set of samples that fail the criterion or the length of the segment falls below a minimum length. Algorithm 6 describes this condition in more detail.

Another basis agnostic method would be to calculate the interstitial error of the approximation. The main problem of this and other error measurement based methods is that the magnitude of the function spans a wide range, making any difference calculations numerically unstable. Figure 4 shows that after normalization, the difference between the polynomial and the surface function is below the machine epsilon of the used single precision floating point numbers.

Algorithm 6 Sample magnitude based split heuristics

Input: $t_{\min}, t_{\max} \in \mathbb{R}$ segment ends,
  $S_0, S_1, \ldots, S_N \in \mathbb{R}$ samples,
  $N \in \mathbb{N}^+$ degree, $\alpha \in \mathbb{R}$ threshold
Output: do_split, split_pos $\in \mathbb{R}$

$a \leftarrow \max_{k=0}^{N} |S_k|$
$b \leftarrow \min_{k=0}^{N} |S_k|$
if $\log_{10}(a) - \log_{10}(b) > \alpha$ **then**
  do_split $\leftarrow$ True
  split_pos $\leftarrow \frac{t_{\min} + t_{\max}}{2}$
  $l \leftarrow$ MinStepLength($t_{\min}$)
  if split_pos $- t_{\min} < l$ **then**
    split_pos $\leftarrow \min(\text{split\_pos} + l, t_{\max})$
  end if
else
  do_split $\leftarrow$ False
end if
  return do_split, split_pos

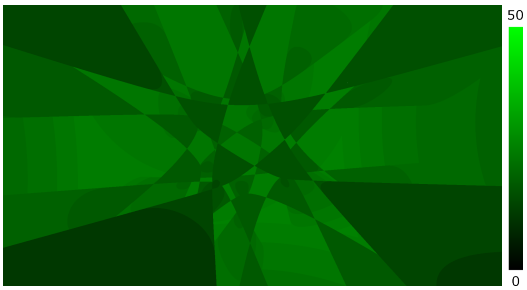
3.2.3 Split positions

We chose a simple split in half heuristic. However, the previously discussed splitting criteria are susceptible to over-splitting in practice. That is, the splitting does not stop, but goes on repeatedly after the maximum split count is reached. This creates a needlessly small segment at the beginning of the ray. The rest of the ray span within the bounding box is covered by a single segment, effectively the same as the starting segment. We need to avoid over-splitting as it wastes processing power without visual impact.

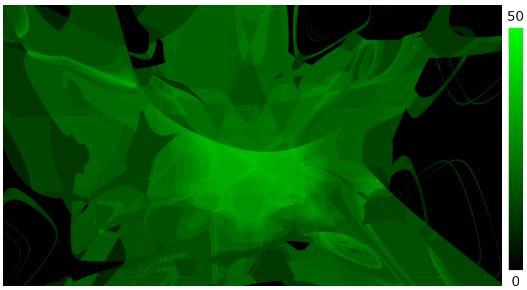
Fortunately, this situation can be conveniently avoided by introducing a minimal segment length. This could be a single parameter, or it can be used to control the level of detail at further distances. We calculate the minimal step length with

$$l(t, t_{\min}, d) = \left(\left(\frac{t - t_{\min}}{d} \right)^3 + \underbrace{0.01}_{\text{offset}} \right) \cdot d \,, \qquad (3)$$

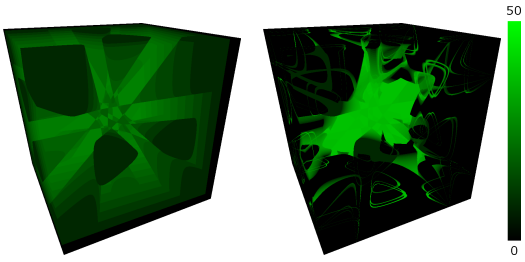
where t denotes the beginning of the current segment and $t_{\min}$ and d denote the beginning and the length



(a) Unit length segments.



(b) Split heuristics with threshold: 2.



(c) Unit length (d) Heuristics

Figure 6: The number of splits per pixel when using unit length segments and the sample based splitting heuristics. Lighter green denotes higher number of splits. Note that in the case of the split heuristics this is not equal to the final number of segments which is typically 25-30 around the center of image 6d. Unit length splitting produces maximum 13 segments on image 6c.

of the complete ray-AABB intersection, respectively. This ensures that the minimal segment length increases rapidly as the trace approaches the end of the ray. The distribution of minimal segment lengths can be changed by adjusting the exponent, while the initial minimum length is adaptable by altering the offset.

The resulting number of splits is visualized for each rendered pixel in Figure 6. The combination of the previously discussed splitting heuristics with the minimal step formula (3) generates a high number of segments in important areas where numeric stability would be low otherwise. In the case of the Barth sextic, critical areas are primarily in the center.

4 TEST RESULTS

The primary subject of our experiments was the Barth sextic. To showcase the generality and limitations of the proposed methods, we also include measurements from two additional surfaces: *tangle cube*, an algebraic surface of degree four:

$$f(x,y,z) = x^4 - 5x^2 + y^4 - 5y^2 + z^4 - 5z^2 + 11.8\ , \quad (4)$$

and *calyx* from the collection of Hauser [Hau25], which has a degree of five:

$$f(x,y,z) = x^2 + y^2z^3 - z^4\ . \quad (5)$$

These surfaces are shown in Figure 7.

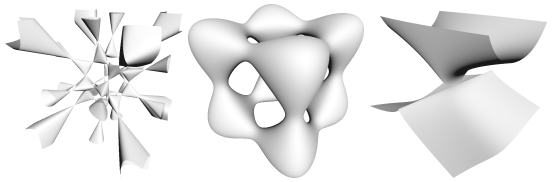


Figure 7: Evaluated surfaces. From left to right: Barth sextic, Tanglecube, Calyx.

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

As shown by the runtime measurements in Table 2, the monomial and both Chebyshev bases achieve real time performance on all three surfaces, while rendering with the remaining two bases runs at interactive speeds. In this case the main reason of bad performance is the costly evaluation algorithm. Therefore, splitting the polynomial into segments is viable for real-time rendering. If the surface is rendered with ray marching, the runtime is dominated by either the evaluations of the surface function or the evaluations of the interpolated polynomials. As depicted in Table 2, the performance cost of the more complicated logic of the heuristic splitting method, as well as the performance cost incurred by the extra samples taken outweigh the benefits of the significantly lower number of total segments (7,549,523 compared to 12,619,762, approx. 40% less). However, this difference could offer meaningful performance gains for other root-finding methods, such as Bézier clipping [Sed90] or the root finder devised by Yuksel [Yuk22]. When the method used for root finding is ray marching, the performance cost of splitting into more segments is relatively low compared to the polynomial evaluations during ray marching. This is shown by the relatively small difference of runtime between the single segment and the uniform segment method in Table 2.

In order to measure the precision of the different methods, we computed the Structural Similarity Index Measure [Wang04] between the rendered outputs and a reference image. We computed the reference image with

	Barth			Tanglecube			Calyx		
Basis	Single segment	Uniform segments	Heuristic segments	Single segment	Uniform segments	Heuristic segments	Single segment	Uniform segments	Heuristic segments
Monomial	9.06	**11.48**	12.16	1.97	**2.46**	4.24	4.76	**5.73**	6.07
Bernstein	457.18	461.63	499.27	54.06	57.37	84.80	191.23	182.59	196.05
Lagrange	39.05	39.59	46.39	7.72	7.56	15.32	18.40	18.99	20.53
Chebyshev	12.55	15.26	15.49	2.58	2.87	4.48	6.37	7.19	7.74
DCT	12.55	16.15	16.51	3.56	3.09	6.02	6.30	7.71	7.45
Direct ray march	17.10 ms			3.47 ms			6.96 ms		

Table 2: Comparison of the dispatch times of a single 1920×1080 frame for rendering the Barth sextic using different bases (Figure 3a). The step length for ray march is the same for all methods (0.005). The uniform segment method splits the ray into a maximum of 10 unit length segments, after which the last segment covers the rest of the ray. The heuristic method uses a split threshold of 2, with maximum 50 splits. For ray marching, all methods were limited to 5000 steps over the entire ray (enough to cover it completely). The last row contains timings using ray marching on the original trivariate $f(x,y,z)$ function. All values are in milliseconds.

ray marching the trivariate function using the same world space step length as we used with the polynomial methods. In this way, any difference between the outputs is a result of inaccuracies in the fitting and evaluation of polynomials. The results are in Table 3.

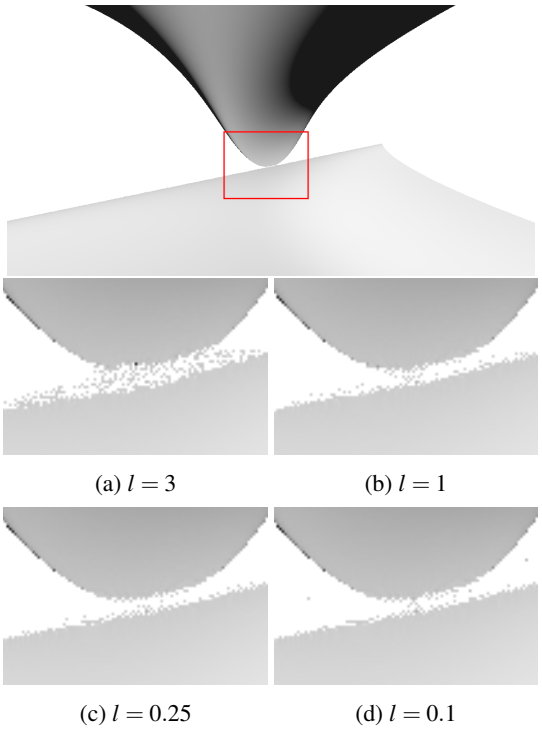
The method generally works well in easier scenarios such as the tanglecube shown in Figure 7 or other closed surfaces that lack singularities or thin parts. However in case of the calyx surface the limitations are much more apparent, as shown by Figure 8. The surface has a cusp singularity, along which significant numeric issues appear. Figure 8 shows the most problematic area rendered with a high quality ray march setting.[1]

Based on these measurements and the qualitative comparison of rendered images we conclude that the numeric stability of the polynomial evaluation and interpolation can be compensated with a larger number of smaller segments. Therefore, if ray marching is used for finding the root, it is more important to choose a basis with a high performance evaluation algorithm. Out of the tested bases, the Chebyshev basis with the discrete cosine transform interpolation method offered the best trade-off of performance and stability. It is also apparent that heuristic splitting produces less accurate results for a similar or worse runtime cost. This is mainly caused by the fact that the region of pixels in which the heuristic splits the polynomial is not tight enough, as it includes many pixels where even a single polynomial could achieve satisfactory results. This is visualized in Figure 6d. On the other hand, it fails to detect every case when a polynomial should be split further, resulting in artifacts similar to the ones shown in Figure 9.

Basis	Single segment	Uniform segments	Heuristic segments
Monomial	0.9428	0.9995	0.9996
Bernstein	0.9979	**0.9998**	**0.9997**
Lagrange	0.9989	**0.9998**	**0.9997**
Chebyshev	0.7351	0.9992	0.9987
DCT	**0.9992**	**0.9998**	**0.9997**

Table 3: Structural similarity index measure of the different methods compared to the reference image rendered with ray marching. The parameters are the same as the ones used for performance measurements in Table 2. The rendered surface is the Barth sextic.

[1] Reference ray march renders were created by decreasing the step length and increasing the step count until further changes did not yield any visual improvement.



(a) $l = 3$ (b) $l = 1$ (c) $l = 0.25$ (d) $l = 0.1$

Figure 8: Rendering the calyx surface using uniform segment length of l. Note that decreasing l below a certain threshold does not yield any gain in quality due to the limits of floating point precision. In this case the sweet-spot is around $l = 0.25$.

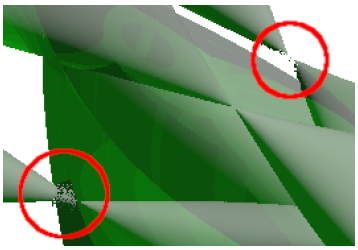


Figure 9: Split count (green) and shaded result blended together. Notice that rendering artifacts are constrained to regions where fewer splits were made by the heuristic. (Chebyshev basis without DCT, using the heuristic split method., SSIM similarity to ground truth: 0.9987.)

5 CONCLUSIONS

In this paper 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 the ray provides all the necessary points to interpolate $f \circ \boldsymbol{r}(t)$ exactly. In practice, however, the fitting and evaluation of such polynomials is not accurate enough to achieve noiseless results, regardless of the basis chosen.

For challenging algebraic 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 unsuitable for real-time rendering. Lagrange interpolation achieves the best results in terms of accuracy but falls behind the monomial and Chebyshev bases due to its slower evaluation.

Whether splitting should be done into uniform length segments or according to heuristics depends on the runtime cost of the used root-finder. In our case, we tested ray marching and the reduced total number of ray segments does not improve the runtime performance of the algorithm.

The proposed methods do not generalize well to non-algebraic surfaces as they assume that the degree of the surface is known in advance. Furthermore, they do not employ robust fitting methods to compute the polynomials. Instead, they rely on the minimal number of samples required to compute an exact fit. Since the runtime cost of ray marching is proportional to the maximum distance, these methods are only suitable for rendering small scenes.

ACKNOWLEDGMENT

Supported by the EKÖP-24 University Excellence scholarship program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation fund. Supported by ELTE Eötvös Loránd University, Budapest, Hungary.

REFERENCES

[Bal18] Bálint, C., and Valasek, G. Accelerating Sphere Tracing. Eurographics 2018 – Short Papers, The Eurographics Association, 2018.

[Ban23] Bán, R., and Valasek, G. Automatic Step Size Relaxation in Sphere Tracing. Eurographics 2023 – Short Papers, The Eurographics Association, 2023.

[Cle55] Clenshaw, C.W. A Note on the Summation of Chebyshev Series. Mathematics of Computation, Vol. 9, No. 51, pp. 118–120, 1955.

[Far96] Farouki, R.T., and Goodman, T.N.T. On the Optimal Stability of the Bernstein Basis. Math. Comput., Vol. 65, No. 216, pp. 1553–1566, 1996.

[Gal20] Galin, E., Guérin, E., Paris, A., and Peytavie, A. Segment Tracing Using Local Lipschitz Bounds. Computer Graphics Forum, Vol. 39, No. 2, pp. 545–554, 2020.

[Hart96] Hart, J.C. Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces. The Visual Computer, Vol. 12, No. 10, pp. 527–545, 1996.

[Hau25] Hauser H.: Gallery of Singular Algebraic Surfaces. Available at homepage.univie.ac.at/herwig.hauser/gallery.html, Accessed on 2025-04-29.

[Kal89] Kalra, D., and Barr, A.H. Guaranteed Ray Intersections with Implicit Surfaces. Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '89), pp. 297–306, 1989.

[Kein14] Keinert, B. et al. Enhanced Sphere Tracing. Smart Tools and Apps for Graphics – Eurographics Italian Chapter Conference, The Eurographics Association, 2014.

[Sed90] Sederberg, T.W., and Nishita, T. Curve Intersection Using Bézier Clipping. Computer-Aided Design, Vol. 22, No. 9, pp. 538–549, 1990.

[Wang04] Wang, Z., Bovik, A.C., Sheikh, H.R., and Simoncelli, E.P. Image Quality Assessment: From Error Visibility to Structural Similarity. IEEE Transactions on Image Processing, Vol. 13, No. 4, pp. 600–612, 2004.

[Yuk22] Yuksel, C. High-Performance Polynomial Root Finding for Graphics. Proc. ACM Comput. Graph. Interact. Tech., Vol. 5, No. 3, Article 27, 2022.