Analytic Ray Splitting for Controlled Precision DVR – Extended version –

Sebastian Weiss and Rüdiger Westermann

Technical University of Munich, Germany

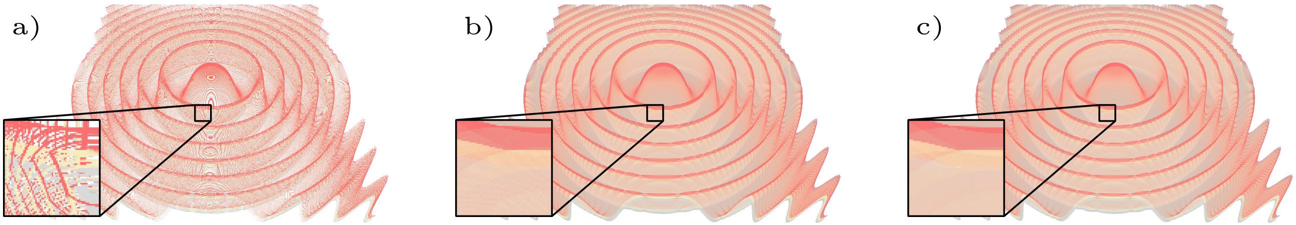


Figure 1: *DVR via post-classification of the* 128^3 *Marschner Lobb dataset to a 1920x1080 pixel raster. a) Constant stepping with a step size of* 0.25 *cells along the view rays at 16ms. b) Same as a) with a step size of* 0.02 *cells renders at 158ms. c) Our analytic ray splitting algorithm avoids sampling artefacts and converges towards the ground truth in 156ms.*

Abstract

When a transfer function is used to map interpolated scalar values in a volumetric field to optical properties, an accurate approximation of the direct volume rendering integral becomes difficult. With a constant integration step size, important properties can be missed, and a very small step size can introduce errors due to numerical precision issues. We propose an algorithm that analytically splits a ray through a cubical cell at the control points of a piecewise-polynomial transfer function. This splitting generates segments of varying lengths over which the variation of the assigned optical properties is described by piecewise cubic functions. This allows using numerical quadrature rules with controlled precision to obtain an approximation with prescribed error bound. The proposed splitting scheme can be used to find all piecewise linear or monotone segments along a ray, and it can thus be used to improve the accuracy of direct volume rendering, scale-invariant volume rendering, and multi-isosurface rendering. In several tests, we compare the results of analytic ray splitting to those using constant stepping and generating segments via polygonal isosurface rendering.

CCS Concepts

• ***Computing methodologies*** → ***Volumetric models;*** *Parallel algorithms;* • ***Mathematics of computing*** → *Quadrature;*

1. Introduction

Over the last decades, direct volume rendering of a three-dimensional (3D) scalar field under the assumption of an optical emission-absorption model [Max95, DCH88, Lev88] has been studied extensively. At the core of direct volume rendering algorithms is the numerical approximation of the low-albedo volume rendering integral [WM92]. The rendering process computes the integral of emitted light along a ray starting at the eye point and passing through the field, scaled by the optical distance from the light to the eye. Most commonly, constant stepping approaches are used for rendering, which approximate the integral by summing physical properties over a set of equidistant integration points along the rays.

For pre-classified data, Novins and Arvo [NA92] derive error bounds from the trilinear interpolant of the physical quantities in a cubical cell. When trilinear interpolation is used, the profile of the interpolated quantity along a ray passing though a cell becomes a cubic polynomial. This has been exploited for isosurface ray-casting [PSL+98, NMHW02, MKW+04], to analytically compute exact ray-isosurface intersection points. To the best of our knowledge, for post-classified data on cubic grids, analytical methods have not been used so far.

Under the assumption of piecewise linearity between the values at consecutive integration points, Kniss *et al.* [KIL$^+$03] showed that multi-dimensional Gaussian transfer functions (TFs) can be algebraically integrated. Even though this approach also works when the step size along a ray varies adaptively, due to the absence of an algorithm that can automatically adapt the step size according to the range of TF values, constant stepping was used. The scale-invariant volume rendering approach by Kraus [Kra05] also relies on constant stepping, yet it splits the integral in data space instead of physical space.

In the case where piecewise linear 1D triangular TFs are applied to the trilinearly interpolated field values in a voxel cell, a ray is split into multiple piecewise cubic segments (see Figure 2). A triangular TF is peaked at a selected isovalue and linearly falls off to 0 at a given width r to account for the fuzziness of the selected surface. The resulting segments can become very small, and false classifications occur and segments might even be missed entirely if the step size along the ray does not adapt to the segment boundaries.

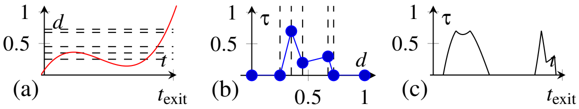


Figure 2: *(a) In a voxel cell using trilinear interpolation, the density along a ray $s(t)$ is described by a cubic polynomial. (b) A piecewise linear TF. (c) Applying the TF from (b) to the densitiy function in (a) results in a piecewise cubic function. Dashed lines indicate the density values where the TF control points (blue) are hit.*

1.1. Our Contribution

We propose an algorithm to accurately consider triangular TFs in direct volume rendering via ray-tracing, by providing an adaptive solution for analytic ray splitting in a voxel cell. A ray is split into segments that are controlled by the composite functions that map trilinearly interpolated field values to emission and absorption values. In this way, we achieve adaptive step size control over the entire spectrum of post-classifications, ranging from arbitrarily sharp TFs to reveal material boundaries (Figure 1a-c) to smooth or constant mappings in a more homogeneous material. Within each ray segment, piecewise polynomial expressions of the optical properties are derived, which enables to solve the direct volume integral with a controlled error.

For ray splitting, we use an extension of the algorithm by Marmitt *et al.* [MKW$^+$04] to solve for all intersections of a ray with the control points of the TF, i.e., the data values where the assignment to optical properties changes. For each segment, polynomial expressions are derived, so that numerical integration schemes can be applied up to arbitrary precision. Here, we employ and evaluate in particular quadrature schemes with a fixed number of control points and the adaptive Simpson quadrature code proposed by Campagnolo *et al.* [CCdF15] in the context of direct volume rendering.

It is our goal to position the proposed analytic ray-splitting algorithm as a means to generate high-quality baseline images for direct volume rendering. The results of alternative approaches can be compared qualitatively and quantitatively, and their strengths and weaknesses regarding specific TF settings and data modalities can be revealed. In particular sharp TFs pose a problem for constant stepping approaches. As we demonstrate in section 6, such approaches do not converge with decreasing step size due to the accumulation of approximation errors. Our approach can effectively hint to specific locations where such errors are high, for example by analyzing the difference between an image rendered using constant stepping and an image rendered using ray-splitting.

Furthermore, with only minor modifications the proposed algorithm can adaptively split a ray into piecewise monotone segments required by scale-invariant volume rendering (subsection 4.2). The direct volume rendering approximation via the rendering and blending of multiple polygonal isosurfaces [LJKY13] can be seens as an approximation of our approach. Our algorithm takes the idea of splitting the ray in data space, but considers precise intersections that are computed analytically during ray-tracing instead of rasterizing polygonal isosurfaces. Then, our algorithm uses the surface in the tri-linear interpolant instead of a piecewise linear approximation, and the data variation in each segment between two consecutive isosurfaces can be considered at high numerical accuracy. To summarize, we

- build upon the algorithm by Marmitt *et al.* to split a ray through a voxel grid at all control points of a piecewiese polynomial TF (section 4),
- embed controlled-precision quadratures as proposed by Novins and Arvo [NAS92] to numerically solve the volume rendering integral on a per-segment basis (section 5),
- demonstrate the use of the proposed algorithm to split a ray automatically into piecewise monotone segments that are required by scale-invariant direct volume rendering [Kra05] (subsection 4.2).

We have pursued a number of experiments to quantitatively evaluate the errors that are introduced by constant stepping approaches, scale-invariant volume rendering, and multi-isosurface rasterization. In these experiments, our method serves as a baseline regarding the numerical error in the volume rendering integral approximation. The evaluation reveals interesting properties of the considered approaches regarding quality and scalability.

All operations involved in analytic ray splitting have been implemented on the GPU, to provide interactive frame rates even for large data sets. Nevertheless, there are scenarios were constant stepping approaches give similar quality at higher rendering performance. On the other hand, we also demonstrate scenarios where the performance differences become small and convergence—irregardless of how small the step size is—cannot be achieved via constant stepping. We have made our implementations available on GitHub, so that the findings can be replicated and qualitative as well as quantitative comparisons with other algorithms can be performed.

2. Related Work

Direct volume rendering (DVR) refers to the generation of an image of a volumetric material with certain optical properties directly, i.e., without using an intermediate surface representation as in indirect methods [LC87]. In DVR via ray-casting, the perceived lightness is computed by considering an optical model, e.g., an emission-absorption model [DCH88, Lev88, Max95], and computing a numerical approximation of the resulting low-albedo volume rendering integral [WM92]. Studies have been performed both with respect to achieving computational efficiency using hierarchical acceleration structures [DH92] and GPUs [KW03], and investigating error bounds for the used numerical integration rules [NA92, WM92, dBGHM97]. Regarding the latter, Etiene *et al.* [EJR+13] study the relationships between integration step size along the rays and integration results. They propose a framework to assess the convergence of volume rendering algorithms with respect to step size, pixel size, and voxel resolution. Campagnolo *et al.* [CCdF15] propose an iterative, adaptive Simpson quadrature scheme to evaluate the volume rendering integral up to a certain accuracy. However, to avoid arbitrary many subdivisions when sharp peaks in the TF occur, an upper bound on the number of subdivisions needs to be considered.

Novins and Arvo [NA92] assume that both an emission and absorption field is given, i.e., the initial scalar field values are pre-classified via transfer functions (TFs), and they derive error bounds from the interpolant of both quantities in each element of a voxel grid. In this way, the variations of the quantities in each cell are restricted by the used polynomial per-cell interpolation scheme. For trilinear interpolation in a voxel cell, the profile of the interpolated quantity along a ray in that cell becomes a cubic polynomial. For isosurface raycasting, Parker *et al.* [PSL+98], Neubauer *et al.* [NMHW02], and Marmit *et al.* [MKW+04] propose exact cell-wise ray splitting schemes based on the zero crossings of the trilinear interpolant. These methods provide exact algebraic solutions for ray-isosurface intersections in the trilinear interpolant, yet they lack the capability of displaying dense volumetric materials. Attempts as by Kanodia *et al.* [KLH05] have been made to enrich the rendering of isosurfaces by some indication of the volumetric structure.

Novins and Arvo [NAS92] further propose a recursive refinement procedure using interval arithmetic, where segments of the ray with the highest estimated error are selected and refined. In this way, large uniform segments can be skipped, however, it requires an estimate of the minimum and maximum intensity and transparency per segment. This renders the approach unsuitable for the use of high-frequent TFs, as the intervals stored per cell become a very crude approximation once optical properties are mapped via high-frequency TFs.

When post-classification is used, i.e., the initial scalar field values are first interpolated and then mapped to emission and absorption via a TF, the error bounds derived by Novins and Arvo break down. For post-classification and one-dimensional (1D) transfer functions that vary linearly along the ray segments within each grid cell, Williams and Max show that the integral can be computed algebraically [WM92]. Kniss *et al.* [KIL+03] extend on this finding by demonstrating that under the same assumption of piecewise linearity, multi-dimensional Gaussian TFs can be algebraically integrated. Scale-invariant volume rendering by Kraus [Kra05] addresses ray splitting in data space. The volume rendering integral is split into segments of equal length over which a piecewise monotone change of the data values is assumed. All these approaches use an equidistant step size along a ray. We will subsequently refer to such approaches as constant stepping approaches. Thus, piecewise linearity or monotony between adjacent sample points can only be justified for very small step sizes. Lindholm *et al.* [LJKY13] generates segments of varying length along the view rays, by selecting values in data space and rendering for each value a polygonal isosurface. The values correspond to the control points in the applied TFs, i.e., the data values where the assignment to optical properties changes. While this approach adapts the segments to the selected data intervals, it assumes a piecewise constant variation of the data in these intervals.

3. Background

In the following, we first review the foundations underlying DVR using an optical emission-absorption model. Our focus is in particular on the assumptions and consequences that lead to a controlled-precision DVR algorithm.

3.1. Field Representation and Interpolation

Let $V \in \mathbb{R}^3 \to [0,1]$ be a 3D field of scalar densities. We assume that the densities are given at the vertices $\mathbf{v}_i$ of a rectangular grid, and per-vertex densities are obtained via $V(\mathbf{v}_i)$.

Let $s(t) = \mathbf{x}_0 + t\bar{\omega}$ be a ray starting at $\mathbf{x}_0$ into the unit direction $\bar{\omega}$ with ray parameter $t \geq 0$. We employ a voxel traversal algorithm [AW+87, Hoe16] to step along the ray cell-by-cell. The traversal algorithm provides the sequence of visited cells in front-to-back order, as well as the per-cell entry and exit points represented by $[t_{\text{in}}, t_{\text{out}}]$.

Within a cell, the densities are trilinearly interpolated with basis functions N_i, i.e., at a point $\mathbf{x} \in [0,1]^3$ in the local cell coordinate space the density is computed as

$$v(\mathbf{x}) = \sum_{i=1}^{8} V(\mathbf{v}_i) N_i(\mathbf{x}). \tag{1}$$

Each basis function N_i is linear in the three coordinates $\mathbf{x} = (x,y,z)^T$. The density profile along a ray through a cell is obtained by replacing $\mathbf{x}$ in Equ. 1 with the ray equation, i.e., by enforcing $\mathbf{x}$ to be on the ray. This yields a cubic polynomial

$$v(s(t)) = At^3 + Bt^2 + Ct + D. \tag{2}$$

We refer to Parker *et al.* [PSL+98] on how to efficiently compute the coefficients of this polynomial expression.

3.2. Analytical Isosurface Intersection

An isosurface for the isovalue θ is defined as

$$\Theta_\theta = \{\mathbf{x} \in \mathbb{R}^3 : v(\mathbf{x}) = \theta\}. \tag{3}$$

Since the density profile along a ray within a cell is a cubic polynomial, between zero and three intersections can occur between the ray and an isosurface. These intersections are given by the roots of $v(s(t)) - \theta = 0 \in [t_{\text{in}}, t_{\text{out}}]$. Efficient isosurface intersection algorithms are needed to split the ray at the control points of the transfer function, which are given at certain density values, see section 4.

Several closed-form solutions for the roots of the cubic equation exist, such as Cardano's formula using cubic roots [Sch13, LC96], and Viète's formula using trigonometric functions [Nic06]. However, these closed-form solutions turn out to be inefficient due to many complex arithmetic function evaluations (see our experiments in Appendix A), and they are prone to numerical instabilities. Therefore, we employ numerical schemes to extract the roots, such as proposed by Marmitt *et al.* [MKW+04].

Regular root-finding algorithms like Newton's method or the Secant method find a root of the function, but they do not necessarily find the first root in an interval $[t_{\text{in}}, t_{\text{out}}]$. The algorithm by Marmitt *et al.* guarantees to find the first intersection using the following observation: If the interval is split at the two extrema of the function, the resulting one to three segments contain exactly zero or one root. Per segment, repeated linear interpolation, a variation of the Secant method as presented by Neubauer *et al.* [NMHW02], is then used to find this root, if it exists. To isolate the extrema, the roots of the derivative of the cubic density profile, i.e. a quadratic polynomial $ax^2 + bx + c = 0$, needs to be found. This step is crucial and care has to be taken as to how to solve for the roots. In practice, the polynomials can be degenerate, so that the standard formula

$$x_{0,1} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}, \tag{4}$$

leads to catastrophic cancellation when b^2 is large, $4ac$ approaches zero and the square-root has the same sign as b. Therefore, in our algorithm we use a numerically more stable reformulation [PTVF88, p. 184]

$$\begin{aligned} x_0 &= \frac{-b - \operatorname{sign}(b) * \sqrt{b^2 - 4ac}}{2a} \\ x_1 &= c/(ax_0), \end{aligned} \tag{5}$$

which essentially catches the critical cases via the sign bit. We use this method to find all roots of the cubic polynomial within one cell that are generated by applying the composite post-shading function to the initial density values. Note that this method reports double intersection either as two intersections an epsilon apart or no intersection, depending on rounding errors in the floating-point comparisons.

3.3. The Volume Rendering Integral

In our work, we assume a low-albedo emission-absorption model for volume rendering [Max95]. Let $\tau : [0,1] \to \mathbb{R}_0^+$ be the absorption due to a given density, and $C : [0,1] \to \mathbb{R}_0^+$ the assigned self-emission, both specified via a TF (see subsection 3.4). Then, the transparency of the line segment from $t = a$ to b is written as

$$T(a,b) = \exp\left(-\int_a^b \tau(v(s(t)))\mathrm{d}t\right). \tag{6}$$

The transparency is 1 if the medium between a and b does not absorb any light and approaches zero for complete absorption. Then, the light intensity reaching the eye is

$$L(a,b) = \int_a^b g(v(s(t)))T(a,t)\mathrm{d}t, \tag{7}$$

were $g(v) = \tau(v)C(v)$. Usually, the emission is not given as a single scalar intensity, but as an RGB tuple. In this case, Equation 7 becomes a vector equation.

3.4. Piecewise Defined TFs

We assume that the TF is given as a function satisfying two constraints: First, it is defined piecewise, i.e., specified by a finite number of control points with closed-form interpolation in between. This allows splitting the ray in data space at the control points to guarantee that no peaks are missed. Second, the interpolation function between two control points—after applying the TF to the cubic density polynomial—has to be analytically integrable. This is required to evaluate the transparency integral (Equation 6) analytically and, thus, reduce the nested intensity integral (Equation 7) to a single integral. This constraint holds, e.g., for all polynomial interpolations.

In the following, we assume piecewise linear TFs as one of the most commonly used forms of TFs (see Figure 2b). Absorption and emission are given at a discrete set of N density values $0 = d_1 < d_2 < \ldots < d_N = 1$. We call these the TF control points. Each control point i stores the absorption τ_i and color C_i, with linear interpolation in between. If absorption and color are given as separate piecewise linear functions, they are combined into a single piecewise function in a pre-processing step.

4. Ray Splitting Algorithm

As illustrated in Figure 2, applying a piecewise linear TF to the cubic density profile along a ray segment results in a piecewise cubic polynomial along that segment. With constant stepping along the ray, peaks in the TF can be missed, and the resulting emission and absorption profiles can be vastly over- or under-estimated. By splitting the ray in the data domain at the control points of the TF, i.e., at the data values where the TF changes, arbitrarily wide and narrow peaks in the TF can be represented accurately. The resulting segments can be integrated independently to high precision.

We use the root finding method of Marmitt *et al.* to split a ray through a cell at the points where the ray takes on any of the data values given by the TF control points. We will subsequently call these points the split points along a ray. When applied to the initial density values, the method yields between zero and three intersections between a ray and a selected data value. The resulting emission and absorption profiles between each pair of split points are cubic polynomials.

4.1. Transfer Function-Based Splitting

In our scenario, the proposed ray splitting algorithm needs to consider the mapping of density values via a TF. First, this means to find all points where the ray takes on the values of the TF control points. For example, Figure 3 illustrates a situation where control point ❶ has three intersections at t_1, t_4, t_5, and control point ❸ has one intersection at t_7. Thus, the number of occurring segments within a single cell can be vastly increasing, and these segments need to be sorted efficiently. Note here that depending on the density values at the cell vertices and the number and width of the selected TF, up to $3N$ many intersections can occur in principle. This can happen in particular in cells spanning a large range of data values, i.e., where the gradient is high, and when multiple narrow peaks are selected in the TF.

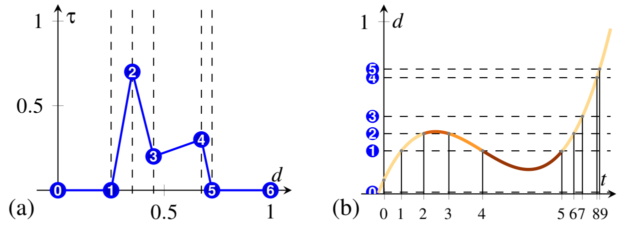


Figure 3: *Detailed analysis of the isosurface intersection (b) at the density values specified by the transfer function (a). The segments in (b) are colored based on the cases later described in Figure 4.*

Secondly, ambiguous cases regarding the occurrence of intersections need to be resolved. These cases occur if two intersections with the same data value (i.e., given by the same TF control point) occur right after each other without another intersection falling in between. Figure 4 illustrates all possible cases regarding the order in which these intersections can occur. Cases IIIa and IIIb are ambiguous, since just from the data values at two consecutive intersections along a ray the order cannot be established. Such a case occurs in Figure 3a, where control point ❷ has intersections at t_2 and immediately again at t_3. To modulate the density polynomial with the transfer function, however, it is important to know if the segment from t_2 to t_3 visits the TF segment ❷–❸ (case IIIa) or ❶–❷ (case IIIb).

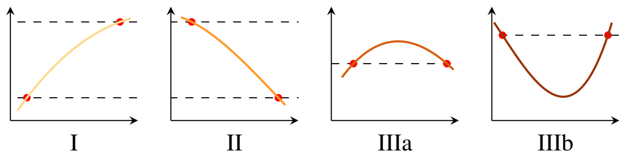


Figure 4: *Four cases of consecutive isosurface intersections.*

Task one – sorting all computed split points along a ray – can be realized in $O(N \log N)$ using standard sorting algorithms, where N is the number of control points falling into the data range covered by the current cell. Task two – ambiguous cases – can be decided, in principle, by evaluating the density polynomial in the middle of the interval between two consecutive split points, and testing whether the value is greater or smaller than the data value at the first point. This approach, however, requires an additional evaluation of the density polynomial, which can be avoided as shown below.

In the following, we present a ray splitting algorithm that can iterate over the split points in order $O(N)$, and handles the ambiguous cases directly without extra evaluations. The pseudocode of the algorithm can be found in Algorithm 1.

Algorithm 1 Analytic ray splitting algorithm

```
 1: input: density polynomial $d(t)$, entry and exit $t_0, t_1$, control point values $d_0, \dots d_{N-1}$
 2: roots = List[$N$]    ▷ an array of length $N$, each entry is a list with maximal three entries for the roots of that $d_i$
 3: Compute all isosurface intersections with the control points $d_0, \dots d_{N-1}$, store in roots[0] to roots[$N-1$]
 4: Find $i$ such that $d(t_0) \in [d_i, d_{i+1}]$, the initial control point interval
 5: $i_{\text{last}} = -1$
 6: loop    ▷ Loop until no more intersections are found
 7:     $t_{\text{lower}}$ = top(roots[i]) or $\infty$    ▷ query next intersection, if it exists
 8:     $t_{\text{upper}}$ = top(roots[i+1]) or $\infty$
 9:     if $t_1 < t_{\text{lower}}$ and $t_1 < t_{\text{upper}}$ then    ▷ exit the cell
10:         EMITINTERVAL($i$, $i+1$, $t_0$, $t_1$)
11:         return
12:     else if $t_{\text{lower}} < t_{\text{upper}}$ then    ▷ lower isovalue first (with index $i$)
13:         pop(roots[i])
14:         if $i = i_{\text{last}}$ then
15:             EMITINTERVAL($i$, $i+1$, $t_0$, $t_{\text{lower}}$)    ▷ case IIIa
16:         else
17:             EMITINTERVAL($i_{\text{last}}$, $i$, $t_0$, $t_{\text{lower}}$)    ▷ case II
18:         end if
19:         $i_{\text{last}} = i,\ t_0 = t_{\text{lower}},\ i = i-1$
20:     else    ▷ upper isovalue first (with index $i+1$)
21:         pop(roots[i+1])
22:         if $i+1 = i_{\text{last}}$ then
23:             EMITINTERVAL($i+1$, $i$, $t_0$, $t_{\text{upper}}$)    ▷ case IIIb
24:         else
25:             EMITINTERVAL($i_{\text{last}}$, $i+1$, $t_0$, $t_{\text{upper}}$)    ▷ case I
26:         end if
27:         $i_{\text{last}} = i+1,\ t_0 = t_{\text{upper}},\ i = i+1$
28:     end if
29: end loop
```

First, the density at the cell entry point along the ray (t_0) is evaluated. Then, the TF segment (d_i, d_{i+1}) that contains this density value is found via (binary) search. In the example in Figure 3, this is the segment ⓿–❶. The next split points can now only be with the data values corresponding to these two control points and, thus, only those have to be tested. Let the next split point at t_1 be with respect to the data value at control point ❶, the case for the lower control point d_i follows immediately by symmetry considerations. Now, since this data value is approached from a lower density value, the next intersection at t_2 can only be with the same data value at control point ❶ again or with the next larger data values at control point ❷. The latter situation is shown in the example. Similarly, for t_3 the data values at control points ❷ and ❸ are tested, but now the data value at ❷ is visited again. This indicates case IIIa, line 15 in Algorithm 1, and the search direction is changed. The next split can now only occur according to the data value at ❶ (case II) or ❷ (case IIIb), as we approach them from a larger density value. This process is repeated until all split points are processed and the ray exits the cell at t_{exit}.

In the pseudocode (Algorithm 1), for each ray segment that is computed by the algorithm the procedure EMITINTERVAL is called.

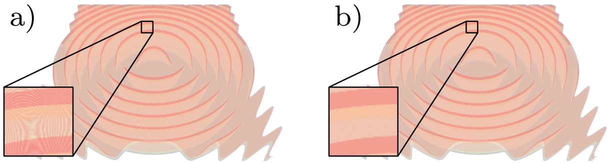


Figure 5: *a) Scale-invariant DVR with a constant step size of* 0.1 *cells suffers from sampling artifacts and erroneously assumed monotony of the density in each segment. b) Analytic ray splitting finds the exact locations of piecewise monotone segments along the rays.*

This procedure performs the per-segment numerical quadrature of the volume rendering integral (see section 5). The resulting color and opacity values are blended in front-to-back order with accumulated per-ray color and opacity, implemented in EMITINTERVAL.

4.2. Scale-Invariant and Multi-Isosurface DVR

Kraus [Kra05] introduced a model for scale-invariant volume rendering, in which the volume rendering integral in physical space is replaced by the scale-invariant integral in data space. This can be seen as the limit case where infinitely many semi-transparent isosurfaces are blended. With this model, the scale-invariant volume rendering integral over an interval $[a,b]$ in physical space becomes

$$L(a,b) = \left| \int_{v(s(a))}^{v(s(b))} g(x) \exp\left(- \left| \int_{v(s(a))}^{x} \tau(x') \mathrm{d}x' \right| \right) \mathrm{d}x \right|. \quad (8)$$

This model builds upon the assumption that the density field is piecewise monotone within the interval $[a,b]$ in physical space. Kraus uses constant stepping and argues that this assumption is approximately fulfilled. However, if the step size is too large, the assumption is violated and rendering artifacts occur (see Figure 5a.)

In our framework, scale-invariant DVR including an adaptive step size that adheres to the underlying assumption can be easily realized (Figure 5b). To enforce monotony over each integration interval, additional split points need to be included at the extrema of the density profiles across the segments. These split points, however, are computed already by the algorithm proposed by Marmitt *et al.* (see subsection 3.2), which splits each per-cell interval at the extrema of the density function to obtain zero or one root per sub-interval. As the additional split points do not interfere with the mapping via the selected TFs, they can be considered in the procedure EMITINTERVAL in addition to the current interval bounds. Per segment, the ray splitting algorithm provides the cubic polynomial of the absorption and color that is obtained by splitting with respect to the piecewise TF. The additional splits with respect to the extrema of the density allow dropping the absolute values from the inner integral in Equation 8, while the polynomials for absorption and color are not affected.

The proposed splitting scheme can also be used directly to render multiple semi-transparent isosurfaces in front-to-back order. Therefore, the determination of ambiguous cases in Algorithm 1 can be avoided. Instead, whenever a split point is computed, a procedure is called that shades the point using gradient information from the density field, and blends its contribution with the already accumulated color and opacity. Additionally, sharp isosurfaces with transparency and soft tissue in between can be rendered jointly (see the accompanying video for a demonstration).

5. Controlled Precision DVR

We now describe the use of the proposed ray splitting algorithm for controlled precision direct volume rendering. While the quadrature schemes for a single cell and a single linear transfer function were already proposed by Novins and Arvo [NA92], to our best knowledge, a consistent formulation regarding the integration along an entire ray and, in particular, including numerical quadrature on the post-classified quantities using piecewise linear transfer functions for both the emission and absorption is missing.

5.1. Mapping Density

When the cubic polynomial of the density per cell $v(s(t))$ is inserted into the piecewise linear absorption $\tau(d)$, the result is a piecewise cubic polynomial $\tau(v(s(t)))$ (Figure 2c). Similarly, $g(v(s(t)))$ results in a piecewise polynomial of degree six.

Now we consider a single segment $[t_0,t_1]$ with the TF control points a,b, as extracted by the ray-splitting algorithm. Let $\tau(d)$ be defined by the two control points $\tau(d_a) = \tau_a$ and $\tau(d_b) = \tau_b$ with linear interpolation in between. Then, $\tau(v(s(t)))$ is given by

$$\begin{aligned} &\tau(t) := \tau(v(s(t))) = b_0 + tb_1 + t^2 b_2 + t^3 b_3 \\ &b_0 = \tau_a + \alpha(a_0 - d_a), b_1 = \alpha a_1, b_2 = \alpha a_2, b_3 = \alpha a_3 \\ &\text{with } \alpha = \frac{\tau_a - \tau_b}{d_a - d_b}, \end{aligned} \quad (9)$$

where the coefficients a_i come from the cubic form of v, i.e., $v(s(t)) = a_0 + ta_1 + t^2 a_2 + t^3 a_3$, Similarly, $C(v(s(t)))$ can be computed, and $g(t) := \tau(v(s(t)))C(v(s(t)))$ is obtained by polynomial multiplication.

The linear interpolation from Equation 9 is only defined for $d_a \neq d_b$, i.e., the data values at the interval points come from different TF control points. However, this is not directly the case for the cases IIIa and IIIb in Figure 4, were the density polynomial visits the same data value (i.e., control point) again with no other control point "in between". We can, however, make use of the following observation: Let d_a be the current control point and the segment boundaries at $t_0 < t_1$, i.e., $d(t_0) = d(t_1) = d_a$. We also know that all values in (t_0,t_1) are greater than d_a (case IIIa) or lower than d_a (case IIIb). Then, we can use the next (previous) control point d_{a+1} (d_{a-1}) for the linear interpolation, although the ray never hits the density value of these control points, but returns to d_a. This case is handled in Algorithm 1 in lines 15 and 23, so that the linear interpolation (Equation 9 and the following quadrature (subsection 5.2)—which are both implemented in EMITINTERVAL— do not need to handle this special case.

5.2. Numerical Quadrature

Once a ray has been split into segments and the per-segment polynomials have assembled, for each segment a cubic polynomial describing the absorption $\tau(t)$ and a sextic polynomial describing the emission $g(t)$ in some interval $[t_0, t_1]$ is given. In this section we describe how to solve $T(t_0, t_1) = \exp(-\int_{t_0}^{t_1} \tau(t) \mathrm{d}t)$ (Equation 6) analytically, and $L(t_0, t_1) = \int_{t_0}^{t_1} g(t) T(t_0, t) \mathrm{d}t$ (Equation 7) numerically. The final output color is then computed by combining the individual contributions using front-to-back blending.

Since $\tau(t)$ is a cubic polynomial, we can evaluate the integral for the transparency analytically:

$$\begin{aligned} T(t_0, t_1) &= \exp\left(-\int_{t_0}^{t_1} \tau_0 + \tau_1 t + \tau_2 t^2 + \tau_3 t^3 \mathrm{d}t\right) \\ &= \exp(\mathcal{T}_{t_0}(t_1)). \end{aligned} \tag{10}$$

The integral for the light intensity is given by

$$L(t_0, t_1) = \int_{t_0}^{t_1} g(t) \exp(\mathcal{T}_{t_0}(t)) \mathrm{d}t. \tag{11}$$

Novins and Arvo [NA92] state that integrals of the form $\int_a^b g(x) e^{p(x)} dx$ cannot be solved analytically and present various numerical quadrature schemes – trapezoid rule, Simpson rule, and using the power series – with bounds on the precision.

For the trapezoid and Simpson rule, the number of required quadrature steps for a given error bound can be computed using bounds on the second or fourth derivative of f, respectively. We show in Appendix B how to compute those bounds during ray-tracing, but this increases the computation time by a factor of 10x to 100x. Furthermore, in Table 1 (Appendix B) we show that for Simpson quadrature, typically only around 10 quadrature steps are needed. Hence, it often suffices to fix the number of quadrature steps to a constant. As we shown in a number of qualitative and quantitative evaluations (section 6), already with a low number of quadrature steps the volume rendering integral can be almost perfectly solved. This is because the ray-splitting algorithm ensures that no sharp peaks in the TF are missed. Alternatively, we have also used the iterative adaptive Simpson scheme by Campagnolo *et al.* [CCdF15] to evaluate the emission integral up to a certain error bound. Our tests in subsection 6.4 have shown, however, that in the current scenario no further quality or performance increases can be achieved.

6. Evaluation and Results

In the following, we perform a number of tests and comparisons of the analytic ray splitting algorithm and alternative volume rendering approaches. In particular, we compare the quality and performance to constant stepping DVR via ray-casting with different step sizes, as well as multi-isosurface rendering via polygon surface rasterization [LJKY13]. For instance, "stepping 0.1" refers to constant stepping with a step size of 0.1 cells. Multi-isosurface rendering renders one isosurface for each data value given by the TF control points, and blends the rasterized fragments in correct visibility order. Blending assumes a piecewise constant approximation of the volume rendering integral between consecutive fragments. With respect to visual quality, this approach works well for high-frequent TFs, but it requires further subdivision of the data points when wider TFs are used. In this case, we specify the subdivision factors that are used, e.g., "multi-isosurface 4" splits each piecewise linear segment in the TF uniformly into four linear segments. We call the number of generated elements the subdivision factor.

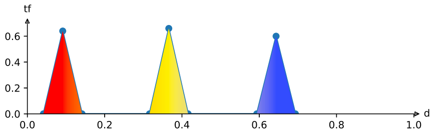


Figure 6: *The transfer function used to render the synthetic datasets. Different widths of the TF peaks are used in the evaluation.*

6.1. Synthetic Datasets

We first evaluate the quality of the rendering approaches on three synthetic datasets. All datasets were sampled to a Cartesian grid with a resolution of 128^3, and they are rendered to a 512^2 viewport. A triangular TF with three peaks is used to map certain data ranges to emissive colors red, yellow, and blue, as well as absorption (see Figure 6). To analyze how the rendering algorithms can handle varying frequencies in the TF, we use different widths for the TF peaks. I.e., initially the width of all peaks is set to 0.05, and they are then scaled to 0.002 to obtain more narrow peaks. In all quantitative evaluations of constant stepping, the TF control points are stored explicitly, and emission and absorption values are retrieved for a data value via linear interpolation between the emission and absorption values at the two closest control points. In the performance evaluations of constant stepping, however, the TF is stored as a 1D texture to enable fastest access using hardware-accelerated linear interpolation.

The first synthetic test case "Sphere" is a sphere. The density is one at the center of the grid, and falls down smoothly to zero in a radially symmetric way towards the boundaries. It is given by the function

$$v(x, y, z) = 1 - \sqrt{x^2 + y^2 + z^2}. \tag{12}$$

The second test dataset "Tube" is a tube with periodically increasing and decreasing density, given by

$$v(x, y, z) = 10(0.1 - \sqrt{y^2 + z^2}(0.9 - 0.5\cos(7x))^3). \tag{13}$$

It is used to demonstrate the behavior of the rendering algorithms if the structures in the dataset become thinner and thinner. As a final synthetic test case, we use the Marschner Lobb dataset [ML94] with the standard parameters.

6.2. Ground Truth Rendering

To verify the accuracy of the used volume rendering algorithms, we require a ground truth rendering. However, simply using constant stepping with a very small step size does not always converges. When accumulating small emission values along a ray, numerical errors are introduced due to the loss of significant bits. Figure 7 illustrates the convergence behavior of regular constant stepping with decreasing step size for Marschner Lobb.

Therefore, we employ the following approach to circumvent the aforementioned effect: The volume is traversed cell-by-cell, and the volume rendering integral is evaluated separately for each cell using a small step size of 0.0001 cells. Then, the integral values per

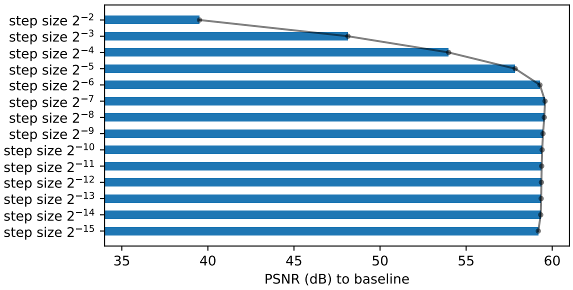


Figure 7: *Ray casting Marschner Lobb with decreasing yet constant step size. Convergence is not achieved, and below a certain step size the quality even decreases due to the accumulation of numerical errors.*

cell are blended together to obtain the final baseline color. Column (a) in Figure 10 shows the three synthetic datasets rendered with the ground truth method and different TF settings.

6.3. Qualitative Evaluation

The qualitative evaluation compares the three different rendering techniques on the datasets Sphere and Tube (see Figure 8). First, ray casting with a constant step size is used. While it can generate high quality results for smooth TFs (a.1), it introduces artifacts when the TF peaks are narrowed (b.1 and b.2). Multi-isosurface rendering, on the other hand, works well for high-frequent TFs (b.3), but for smooth TFs a further subdivision of the TF segments is necessary (a.2 and a.3). Our proposed analytic ray splitting algorithm resolves the volume rendering integral up to a user-specified precision, independent of the frequency of the TF (see the quantitative evaluation in subsection 6.4 below).

To obtain further insights as to where the errors occur, the Marschner Lobb is rendered using the different DVR algorithms (see Figure 9) and TF peak widths. At a peak width of 0.05, the transitions between the three peaks of the TF are relatively smooth, yet constant stepping and multi-isosurface rendering already show noticeable pixel errors over the whole image. Analytic ray splitting, in particular when used with Simpson quadrature, produces significantly lower numerical errors, which are in particular not visible any more. It can be clearly seen, that with a decreasing width of the TF peaks to 0.002, the errors introduced by both constant stepping and multi-isosurface rendering become unacceptably high. Constant stepping with a step size of 0.1 fails to resolve the thin structures, leading to ringing artifacts. Even for a step size as low as 0.001, the artefacts cannot be resolved. Multi-isosurface rendering shows highest errors at the silhouettes, due to the piecewise linear surface approximation in the reconstruction process, and suffers from z-fighting due to imprecision in the rasterizer. Ray splitting with Simpson quadrature and 10 quadrature points can successfully resolve the DVR integral for both TF ranges. Neither does the algorithm suffer from noise as the constant stepping approach, nor from rasterization errors at the silhouettes. It leads to the best results over all test cases shown here.

6.4. Quantitative Evaluation

The findings from the qualitative assessment are further supported by a quantitative evaluation using Marschner Lobb, Sphere and Tube. The results are shown in Figure 10. First, we evaluate the results of the different rendering algorithms for a smooth (column b1) and a sharp (column b2) TF, by plotting the percentage of pixels within a certain allowed error (i.e., so-called regression error characteristic (REC) curves). One can see that the ray splitting algorithm using Simpson quadrature can always achieve that 100% of the pixels are within an error of $1/256$ (dashed vertical line), i.e. where pixel errors occur when using 8-bit color values. For the used datasets and TF settings, constant stepping approaches actually never achieve this, i.e., even with smaller and smaller step size there are always many pixel where the error of $1/256$ is overshoot.

We also analyzed whether analytic ray splitting or the used numerical quadrature rules is the decisive factor regarding accuracy. The methods “interval - midpoint” and “interval - rectangle-3” perform analytic ray splitting as proposed, but then use the midpoint rule or the rectangular rule with three control points to evaluate the integral along the ray segments. As one can see,“interval-simple” is able to accurately approximate the rendering integral, especially with sharp TFs. This leads to the insight that analytic ray splitting at the data values selected by the TF control points is crucial for obtaining high numerical accuracy. On the other hand, higher-order quadrature is needed for convergence on TFs with both wide and narrow peaks. As shown in the REC curves, using a fixed number of ten quadrature points with Simpson's rule is enough so that 100% of pixels are within an error of $1/256$. In the difficult test case of the Tube, constant stepping approaches only achieve 100% at a much higher error rate.

6.5. Timings

Finally, we report the execution times of the different rendering algorithms in relation to the numerical accuracy they achieve. All algorithms have been implemented in CUDA, and performance measures were taken on a NVidia RTX 2070 GPU. Timings are averaged over 10 different frames of resolution 512^2, by moving the camera randomly around the datasets. No acceleration structures were used, even though any empty-space skipping strategy can be integrated in a straight forward way.

The last column (c) in Figure 10 shows, as expected, the linear performance scale of constant stepping in the step size. Analytic ray splitting lies between constant stepping with a step size of 0.1 and a step size of 0.01. It is interesting to note that the quadrature scheme and the number of quadrature steps has almost no effect on performance. Detailed profiling has shown that by far the most time is spent in the solve step, to accurately determine the points along the ray where the data values selected by the TF control points are taken, and the procedure that builds the piecewise polynomials in the volume integrals.

Multi-isosurface rendering cannot compete with the ray stepping approaches. This is due to the fact that a huge amount of fragments is generated (more than 10^8), which requires to perform tile-based rendering so that the fragment buffer fits into GPU memory. Furthermore, especially for Marschner Lobb a large number of frag-

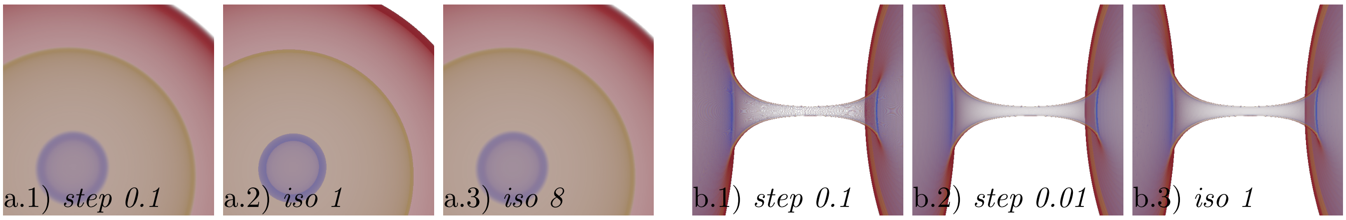


Figure 8: *Quality comparison of DVR algorithms, convergence is assumed below a maximal numerical pixel error of* $1/256$*. For smooth volumes, i.e., Sphere, constant stepping converges with a step size of* 0.1 *(a.1). Multi-isosurface rendering without further subdivision is not sufficient (a.2), and requires a subdivision factor of at least 8 (a.3) for convergence. For datasets with higher frequencies, i.e. Tube, constant stepping requires a smaller step size of around* 0.01 *(b.2), multi-isosurface rendering achieves convergence without further subdivision (b.3).*

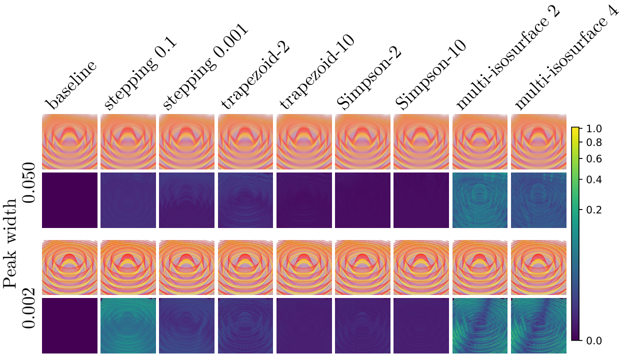


Figure 9: *Comparison of different volume rendering algorithms against the ground truth rendering result. Two TFs with wider and more narrow TF peaks are used, i.e., TF range of* 0.39 *(top) and* 0.01 *(bottom). In each group, the first row shows the rendering and the second row the per-pixel* L_2*-norm of the color difference to the baseline, scaled using a power norm of* $x^{0.3}$ *(see legend to the right).*

ments per pixel can be produced for certain views. This can make the sorting step costly and dominating the overall performance.

6.6. Real-World Datasets

We further evaluate the quality and performance of the DVR algorithms on three CT scans: A stag beetle at a size of $832 \times 832 \times 494$, a human thorax at a size of $512 \times 512 \times 286$, and a full scan of the human body at a size of $512 \times 512 \times 1884$. Additionally, we used the Ejecta dataset, a supernova simulation at a size of 512^3. The results are given in Figure 11.

We compared the ray-splitting algorithm with Simpson-10 quadrature to ray tracing with a constant step size and measure the mean absolute error to the baseline and the execution time. In a first test of constant stepping, we used a coarse step size of 0.25 cells. The coarse step size, however, leads to ringing artifacts and noise. Therefore, the step size was globally halved until the rendering has converged, i.e. the output does not change within an error of $1/256$ anymore. The results for the converged constant stepping is depicted in the third row of Figure 11 with the final step size, the average error to the baseline, and rendering time.

One can see that constant stepping with a coarse step size is much faster than the proposed ray-splitting algorithm. But if the step size should be chosen small enough for convergence, it becomes slower than the ray-splitting algorithm with Simpson quadrature. At the same time, Simpson-10 results in a lower average error as well.

7. Conclusion

We have presented an algorithm that analytically splits rays through a post-classified emission-absorption volume at the data values given by the control points of a piecewise transfer function. Per segment, the emission and absorption profiles are given as explicit cubic functions. This allows for solving analytically the absorption integral and numerically the emission integral up to a user-defined precision. It is guaranteed that no high-frequent peaks in the transfer function are missed.

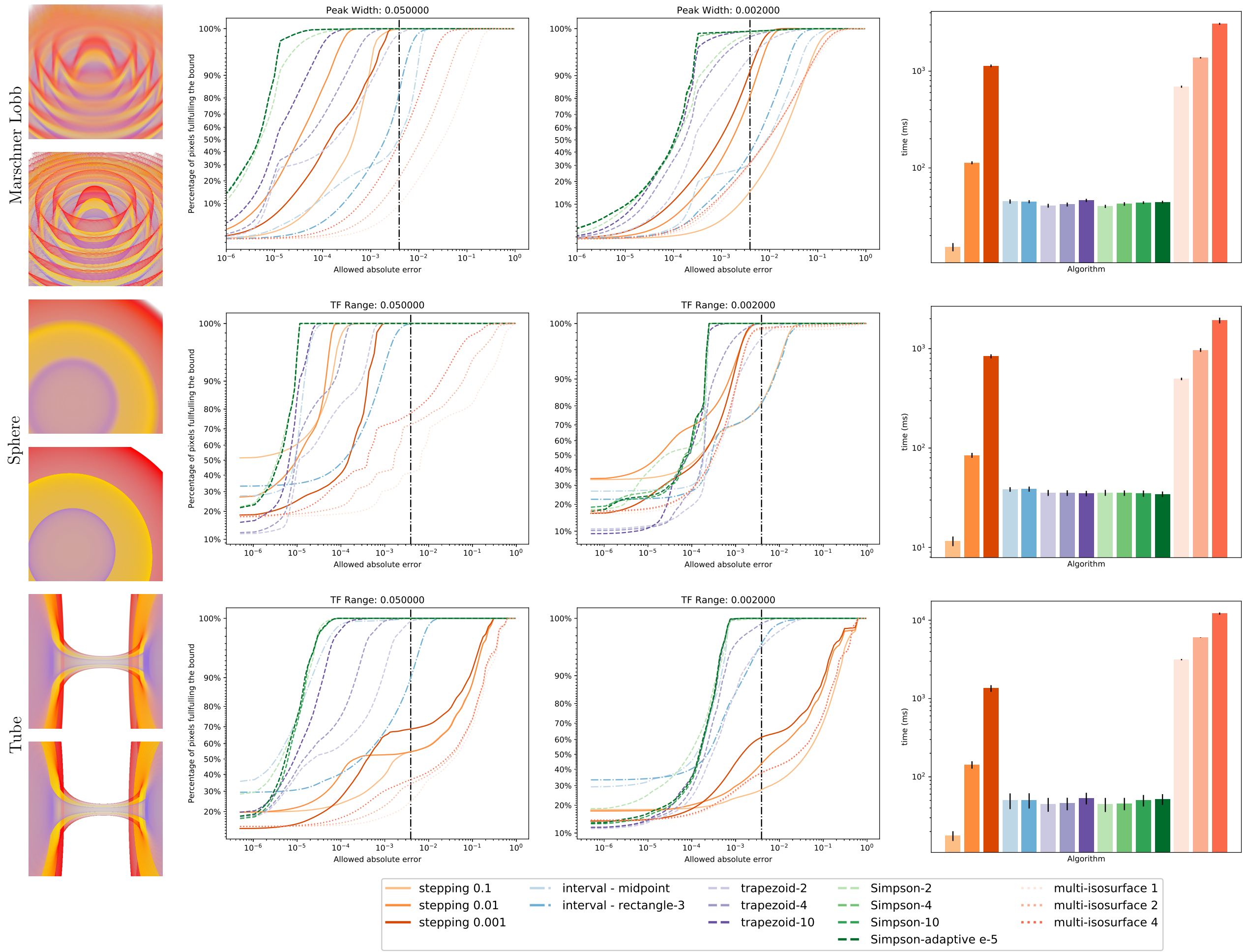
A quality and performance evaluation has shown that the performance of analytic ray splitting is even higher than that of constant stepping when triangular TFs with rather narrow peaks are used. We see the proposed renderer as an efficient and highly accurate baseline, which can be used for comparative purposes as well as a framework to integrate alternative rendering options such as scale-invariant DVR and multi-isosurface rendering.

In the future, we will in particular investigate the use of analytic ray splitting for differentiable volume rendering including post-classification. Ultimately, we plan to develop algorithms for converting physical fields in situ into a compact latent code that can be interpreted by a renderer to produce a meaningful visual representation. This require differentiable renderers that are able to compute per-pixel derivatives with respect to the post-classification process.

References

[AW⁺87] J. Amanatides, A. Woo, et al. A fast voxel traversal algorithm for ray tracing. In *Eurographics*, volume 87, pages 3–10, 1987.

[CCdF15] L. Q. Campagnolo, W. Celes, and L. H. de Figueiredo. Accurate volume rendering based on adaptive numerical integration. In *2015 28th SIBGRAPI Conference on Graphics, Patterns and Images*, pages 17–24. IEEE, 2015.

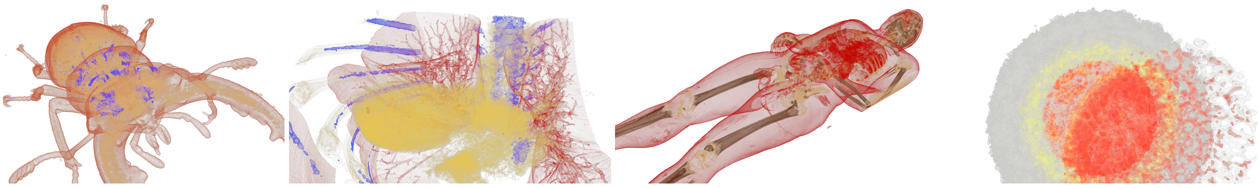


(a)
Rendering with
wide peaks (top) and
narrow peaks (bottom)

(b1) **(b2)**
Regression error characteristic curves:
The number of pixels within a specified error bound
The vertical dashed line indicates an error of 1/256

(c)
Timings of the various algorithms,
averaged over 10 frames by rotating
the camera around the object.

Figure 10: *Accuracy statistics for different synthetic datasets and rendering algorithms.*



Method	Beetle 832 × 832 × 494			Thorax 512 × 512 × 286			Human 512 × 512 × 1884			Ejecta 512^3		
	s	e	t (ms)	s	e	t (ms)	s	e	t (ms)	s	e	t (ms)
Simpson-10	–	1.73e-5	2.77e2	–	4.69e-5	7.72e2	–	1.13e-4	5.34e2	–	1.54e-5	6.14e2
Stepping 0.25	2.50e-1	3.94e-3	1.12e2	2.50e-1	1.26e-2	8.29e1	2.50e-1	2.37e-2	3.64e2	2.50e-1	4.18e-3	9.98e1
Stepping n	3.91e-3	5.37e-4	1.35e4	2.14e-3	3.24e-4	6.76e3	1.56e-2	7.47e-4	9.06e3	2.76e-3	5.93e-4	6.31e3

Figure 11: *Error and timing statistics for real-world datasets. s, e and t, respectively, indicate step size in constant stepping, mean square pixel error compared to the baseline rendering, and rendering time. Stepping n indicates that the step size is reduced until convergence. The final step size is shown in column s.*

[dBGHM97] M. W. de Boer, A. Gröpl, J. Hesser, and R. Männer. Reducing artifacts in volume rendering by higher order integration. *IEEE Visualization'97 Late Breaking Hot Topics*, pages 1–4, 1997.

[DCH88] R. A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. *ACM Siggraph Computer Graphics*, 22(4):65–74, 1988.

[DH92] J. Danskin and P. Hanrahan. Fast algorithms for volume ray tracing. In *Proceedings of the 1992 workshop on Volume visualization*, pages 91–98, 1992.

[EJR+13] T. Etiene, D. Jönsson, T. Ropinski, C. Scheidegger, J. L. Comba, L. G. Nonato, R. M. Kirby, A. Ynnerman, and C. T. Silva. Verifying volume rendering using discretization error analysis. *IEEE transactions on visualization and computer graphics*, 20(1):140–154, 2013.

[Hoe16] R. K. Hoetzlein. GVDB: Raytracing Sparse Voxel Database Structures on the GPU. In U. Assarsson and W. Hunt, editors, *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*. The Eurographics Association, 2016. doi:10.2312/hpg.20161197.

[KIL+03] J. Kniss, M. Ikits, A. Lefohn, C. Hansen, E. Praun, et al. Gaussian transfer functions for multi-field volume visualization. In *IEEE Visualization, 2003. VIS 2003.*, pages 497–504. IEEE, 2003.

[KLH05] R. L. Kanodia, L. Linsen, and B. Hamann. Multiple transparent material-enriched isosurfaces. 2005.

[Kra05] M. Kraus. Scale-invariant volume rendering. In *VIS 05. IEEE Visualization, 2005.*, pages 295–302. IEEE, 2005.

[KW03] J. Krüger and R. Westermann. Acceleration techniques for gpu-based volume rendering. In *IEEE Visualization, 2003. VIS 2003.*, pages 287–292, 2003. doi:10.1109/VISUAL.2003.1250384.

[LC87] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.

[LC96] C.-C. Lin and Y.-T. Ching. An efficient volume-rendering algorithm with an analytic approach. *The Visual Computer*, 12(10):515–526, 1996.

[Lev88] M. Levoy. Display of surfaces from volume data. *IEEE Computer graphics and Applications*, 8(3):29–37, 1988.

[LJKY13] S. Lindholm, D. Jönsson, H. Knusson, and A. Ynnerman. Towards data centric sampling for volume rendering. In *Proceedings of SIGRAD 2013; Visual Computing; June 13-14; 2013; Norrköping; Sweden*, number 094, pages 55–60. Linköping University Electronic Press, 2013.

[Max95] N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[MKW+04] G. Marmitt, A. Kleer, I. Wald, H. Friedrich, and P. Slusallek. Fast and accurate ray-voxel intersection techniques for iso-surface ray tracing. In *VMV*, volume 4, pages 429–435, 2004.

[ML94] S. R. Marschner and R. J. Lobb. An evaluation of reconstruction filters for volume rendering. In *Proceedings Visualization'94*, pages 100–107. IEEE, 1994.

[NA92] K. Novins and J. Arvo. Controlled precision volume integration. In *Proceedings of the 1992 workshop on Volume visualization*, pages 83–89, 1992.

[NAS92] K. L. Novins, J. Arvo, and D. Salesin. Adaptive error bracketing for controlled-precision volume rendering. Technical report, Cornell University, 1992.

[Nic06] R. W. D. Nickalls. Viète, descartes and the cubic equation. *The Mathematical Gazette*, 90(518):203–208, 2006. doi:10.1017/S0025557200179598.

[NMHW02] A. Neubauer, L. Mroz, H. Hauser, and R. Wegenkittl. Cell-based first-hit ray casting. In *VisSym*, pages 77–86, 2002.

[PSL+98] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. Interactive ray tracing for isosurface rendering. In *Proceedings Visualization'98 (Cat. No. 98CB36276)*, pages 233–238. IEEE, 1998.

[PTVF88] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. Numerical recipes in c, 1988.

[Sch13] J. Schwarze. Cubic and quartic roots. In A. S. Glassner, editor, *Graphics gems*, chapter VIII.1, pages 404–407. Elsevier, 2013.

[WM92] P. L. Williams and N. Max. A volume density optical model. In *Proceedings of the 1992 workshop on Volume visualization*, pages 61–68, 1992.

Supplemental Material

Appendix A: Comparison of Root-Extraction Algorithms for Multiple Isosurfaces

We compared four methods for extracting the first root of a cubic polynomial in an interval $[a,b]$. The first two methods, the formula by Viète [Nic06] using hyperbolic functions and the formula by Schwarze [Sch13] using only roots and trigonometric functions, give closed-form solutions for all roots of the cubic polynomial. From those roots, the smallest root in the given interval is selected. The other two methods, Neubauer's repeated linear interpolation [NMHW02] and its improvement by Marmitt *et al.* [MKW+04] are iterative schemes that provide – in the base form of the algorithm – only the first root in the given interval.

The presented methods for ray-isosurface intersections were analyzed on the Ejecta dataset. Two views were used, a far view for performance and quality benchmarks and a close view for qualitative and quantitative precision tests, see Figure 12. A qualitative comparison of the precision of the algorithms is shown in Figure 13. One can see in the marked regions how fixed step sizes lead to ringing artifacts, how numerical instabilities occur in the pure analytic solutions, and how the numerical approximations may lead to missed intersections.

Quantitative precision statistics and timings can be found in Figure 14. As one can see, the algorithm of Neubauer or Marmitt are close in performance to constant stepping with $\Delta t = 0.2$. At this stepsize, ringing artifacts are already noticeable. The algorithm of Neubauer or Marmitt outperform constant stepping by large margins for smaller step sizes.

Based on these statistics, we selected Marmitt as the algorithm to extract isosurface intersections. This algorithm as introduced by Marmitt *et al.* [MKW+04], however, only extracts the first root in the interval. We extend the algorithm to report all roots in the given interval, not only the first root, see Algorithm 2. Marmitt requires the solution of a quadratic polynomial to isolate the extrema, see line 19 in Algorithm 2. We found it crucial to not use the standard formula for quadratic polynomials,

$$x_{0,1} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}, \tag{14}$$

denoted by "Marmitt (unstable)" in the statistics. Instead, we advise to reformulate it to be more numerically stable [PTVF88, p. 184]

$$\begin{aligned} x_0 &= \frac{-B - \operatorname{sign}(B) * \sqrt{B^2 - 4AC}}{2A} \\ x_1 &= C/(Ax_0), \end{aligned} \tag{15}$$

denoted by "Marmitt (stable)" in the statistics.

As further alternatives to the presented numerical methods, Hadwiger *et al.* [HSS+05] use a simpler bisection model, but report that this method might miss some intersections. Methods to trace arbitrary implicit surfaces were e.g. presented by Singh and Narayanan [SN09] or Knoll *et al.* [KHK+09] to name a few.

Algorithm 2 Extension of Marmitt's algorithm to find all roots in the interval $[t_{\text{in}}, t_{\text{out}}]$, Appendix A.

```
 1: function NEUBAUER($t_0, t_1, f_0, f_1$)
 2:     if sign($f_0$) = sign($f_1$) then
 3:         return no hit
 4:     end if
 5:     for $i = 1 \ldots N$ do
 6:         $t_{\text{new}} := t_0 + (t_1 - t_0)\frac{f_0}{f_0 - f_1}$
 7:         $f_{\text{new}} := f(t_{\text{new}})$
 8:         if sign($f_{\text{new}}$) = sign($f_0$) then
 9:             $t_0 = t_{\text{new}}, f_0 = f_{\text{new}}$
10:         else
11:             $t_1 = t_{\text{new}}, f_1 = f_{\text{new}}$
12:         end if
13:     end for
14:     return $:= t_0 + (t_1 - t_0)\frac{f_0}{f_0 - f_1}$
15: end function

16: function EXTENDEDMARMITT($t_{\text{in}}, t_{\text{out}}, f$)
17:     $t_0 := t_{\text{in}}, t_1 := t_{\text{out}}, f_0 = f(t_0), f_1 = f(t_1)$
18:     roots=[]
19:     Find roots of $f'(t) = 3At^2 + 2Bt + C$
20:     if $f'$ has real roots then
21:         Let $e_0 < e_1$ be the roots of $f'$
22:         if $e_0 \in [t_0, t_1]$ then
23:             if sign($f(e_0)$) $\neq$ sign($f_0$) then
24:                 add $t$ =NEUBAUER($t_0, e_0, f_0, f(e_0)$) to roots
25:             end if
26:             $t_0 = e_0, f_0 = f(e_0)$
27:         end if
28:         if $e_1 \in [t_0, t_1]$ then
29:             if sign($f(e_1)$) $\neq$ sign($f_0$) then
30:                 add $t$ =NEUBAUER($t_0, e_1, f_0, f(e_1)$) to roots
31:             end if
32:             $t_0 = e_1, f_0 = f(e_1)$
33:         end if
34:     end if
35:     if sign($f_0$) $\neq$ sign($f_1$) then
36:         add $t$ =NEUBAUER($t_0, t_1, f_0, f_1$) to roots
37:     end if
38: end function
```

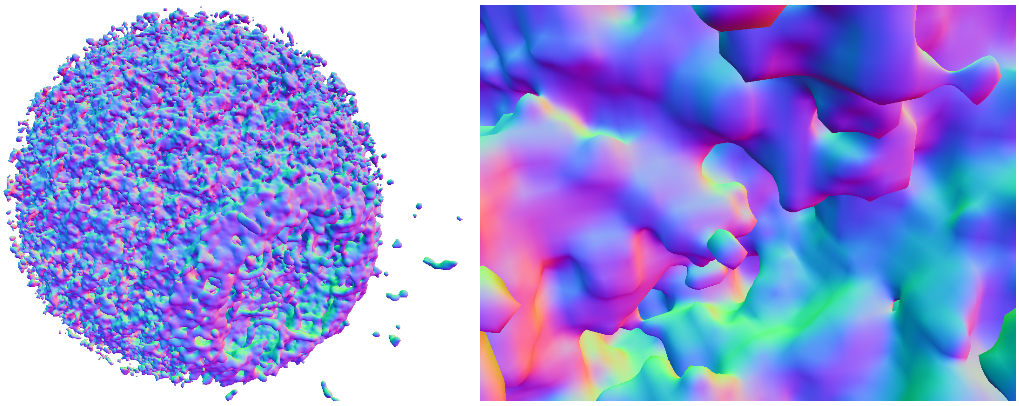


Figure 12: *The two scenes used for comparisons, statistics and benchmarks: a far and a close view of the Ejecta dataset. The resolution is* 512×512 *pixels.*

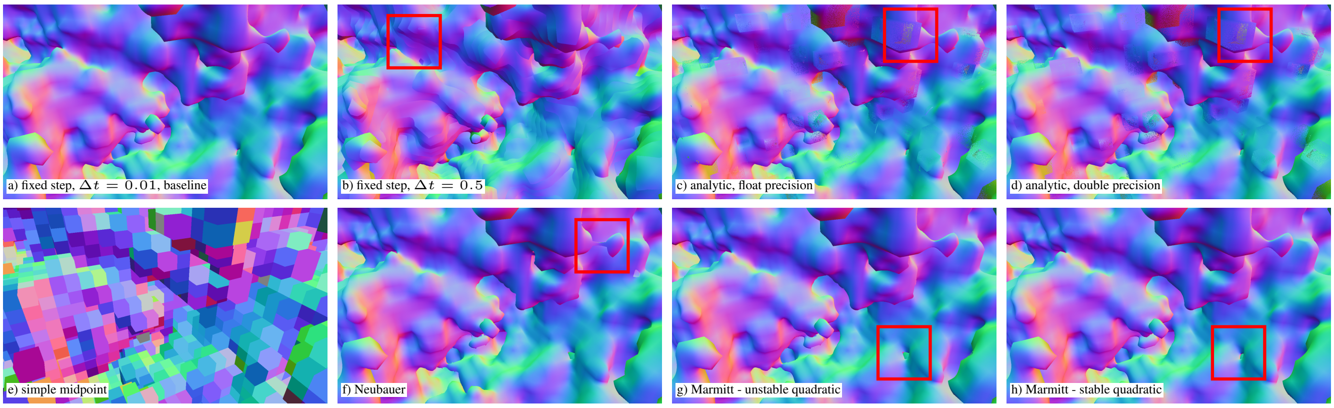

a) fixed step, $\Delta t = 0.01$, baseline
b) fixed step, $\Delta t = 0.5$
c) analytic, float precision
d) analytic, double precision
e) simple midpoint
f) Neubauer
g) Marmitt - unstable quadratic
h) Marmitt - stable quadratic

Figure 13: *Visual comparison of the different algorithms for ray-isosurface intersections and the numerical artifacts on the close view of Ejecta.*

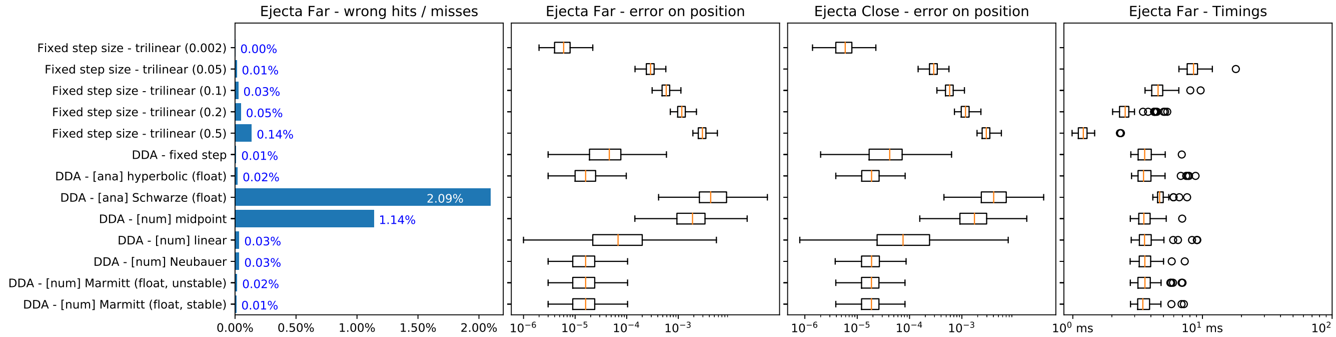


Figure 14: *Comparison of the precision and performance of the different algorithms on the close view of Ejecta. The baseline is a rendering with a fixed step size of* $\Delta t = 0.001$*. Wrong hits/misses counts how many pixels hit (or miss) the object while are a miss (or hit) in the baseline. The first two box plots show the absolute error in the isosurface position, i.e. the distance of the isosurface to the camera in world space, of pixels that are a hit in both the baseline and the method to evaluate. The third box plot shows the timings to compute the images. Note that the error measures are in log-scale.*

Appendix B: Bounds on the Quadrature Error

In the following section, we analyze the presented numerical methods for their accuracy. Novins and Arvo [NA92] present remainder terms for the trapezoid rule and Simpson rule with which the required number of quadrature steps for a given error bound can be derived,

$$r_T = \frac{b-a}{12}\hat{f}''h^2 \tag{16}$$

for the trapezoid rule and

$$r_S = \frac{b-a}{180}\hat{f}^{(t)}h^4 \tag{17}$$

for the Simpson rule, where $[a,b]$ is the integration domain and $h = (a-b)/N$ is the step size for N sample steps. The difficulty and computational expense arise from the need for an upper bound on the second derivative of f in $[a,b]$, $\hat{f}''$ for the trapezoid rule and on the fourth derivative $\hat{f}^{(4)}$ for the Simpson rule.

We now analyze how many quadrature steps would be needed to evaluate the integrals up to a specified precision using trapezoid rule or Simpson rule. For that we record all emission integrals, see subsection 3.3, occurring in the rendering of the Marschner Lobb, see subsection 6.1, into memory to perform offline statistics. We then performed experiments with five different options on how to compute those upper bounds, as shown in Table 1. Note that the presented numbers are overestimated as any attenuation due to the absorption of previous segments along the ray is ignored. The simplest way "coarse" is to use the approximation

$$\Delta(x) = \sum_{i=0}^{D} \delta_i x^i \quad \rightarrow \quad \max_{x\in[a,b]} \Delta(x) \leq \sum_{i=0}^{D} |\delta_i| c^i, \tag{18}$$

with $c = \max(|a|,|b|)$, combined with a recursive formula to compute $f^{(n)}$, as used by Novins and Arvo [NA92]. This bound can be evaluated during ray tracing, but gives very crude bounds.

The next two methods make use of the Bernstein form of the polynomials [CS66]. First, a polynomial $\Delta(x)$ is transformed from the interval $[a,b]$ to $[0,1]$, as the Bernstein form is only defined over the interval $[0,1]$, giving rise to $\tilde{\Delta}(x)$. Then the the polynomial given in the interval $[0,1]$ is transformed into Bernstein form with coefficients β_i given by

$$\beta_i = \sum_{r=0}^{i} \tilde{\delta}_r \binom{i}{r} / \binom{D}{r} \quad , i = 0,1,\ldots,D. \tag{19}$$

These Bernstein coefficients form a convex hull around the polynomial,

$$\min\{\beta_i\} \leq \tilde{\Delta}(x) \leq \max\{\beta_i\} \quad , x \in [0,1]. \tag{20}$$

Table 1: *Required quadrature steps, given as median, mean, 90% quantile and maximum, for a specified error using different quadrature schemes. The column "method" specifies the different ways on how to compute the error bounds, see Appendix B.*

Error	Method	Median	Mean	90% q.	Max
Trapezoid rule					
10^{-2}	coarse	15.0	3.6e6	2.1e3	2.4e11
	BS rec.	9.0	25.1	41.0	1.5e4
	BS expand	4.0	5.1	10.0	114.0
	roots	2.0	3.5	8.0	87.0
	numerical	3.0	3.8	8.0	45.0
10^{-3}	coarse	46.0	1.1e7	6.5e3	7.6e11
	BS rec.	28.0	78.0	129.0	4.7e4
	BS expand	11.0	14.9	31.0	359.0
	roots	6.0	10.0	25.0	276.0
	numerical	8.0	10.7	24.0	139.0
10^{-5}	coarse	453.0	1.1e8	6.5e4	7.6e12
	BS rec.	275.0	775.0	1.3e3	4.7e5
	BS expand	104.0	143.5	307.0	3.6e3
	roots	52.0	96.0	242.0	2.8e3
	numerical	74.0	99.5	225.0	813.0
Simpson rule					
10^{-2}	coarse	2.0	78.5	32.0	4.1e5
	BS rec.	2.0	3.2	6.0	78.0
	BS expand	2.0	2.2	2.0	14.0
	roots	2.0	2.1	2.0	12.0
	numerical	1.0	1.1	1.0	7.0
	adaptive	5.0	5.0	5.0	12.0
10^{-3}	coarse	4.0	138.7	56.0	7.3e5
	BS rec.	4.0	4.8	8.0	138.0
	BS expand	2.0	2.9	4.0	24.0
	roots	2.0	2.7	4.0	22.0
	numerical	1.0	1.6	3.0	13.0
	adaptive	5.0	5.2	5.0	33.0
10^{-5}	coarse	12.0	436.0	172.0	2.3e6
	BS rec.	10.0	12.5	26.0	436.0
	BS expand	6.0	6.4	12.0	76.0
	roots	4.0	5.8	10.0	64.0
	numerical	3.0	4.6	9.0	39.0
	adaptive	5.0	10.7	26.0	94.0

We can make use of this result to find better bounds on $\hat{f}''$ and $\hat{f}^{(4)}$. Novins and Arvo [NA92] present a recursive approach to compute bounds on the derivatives of f using bounds on derivatives of q and p, introduced using the coarse polynomial bounds (Equation 18). We use it together with Bernstein bounds, denoted by "BS rec." in Table 1. Even better bounds, however, can be obtained by making use of the fact that any derivative of $f(x) = q(x)e^{p(x)}$ is again of the same form, just with increasing order of q. For example, in our case with q being a sextic and p a quartic polynomial, the fourth derivative of f would lead to a polynomial q of order 18. Applying the Bernstein bounds on this expanded form leads to much tighter bounds, denoted by "BS expand" in Table 1.

The method "roots" makes again use of the fact that any derivative of $f(x) = q(x)e^{p(x)}$ is again of the same form and that

$$f(x) = 0 \Leftrightarrow q(x) = 0. \tag{21}$$

The maximum of $f^{(n)}$ in $[a,b]$ can only be reached at a,b or the roots of $f^{(n+1)}$. We compute all roots using Equation 21 and the companion matrix of q [HJ85]. For the last method, "numerical", we iteratively take more and more quadrature steps until the quadrature method has converged (up to a small epsilon). Then we record the difference to that converged solution and report the smallest number of steps after which the result lies within the specified error from the converged solution.

All those methods, however, are difficult to perform during ray tracing due to their complexity. The Bernstein bounds can be evaluated during ray tracing, but requires many additional operations that slow down the ray tracing by a factor of 10-100. As one can see in Table 1, even for a very small allowed error of 10^{-5}, at least 90% of all integrals during rendering require only 10 quadrature steps using Simpson's rule. Therefore, if one can accept that some outlier would require more quadrature steps, the number of quadrature steps can be fixed globally. This removes the need to estimate bounds on the derivatives and reduces branch divergence during ray tracing. An analysis of the errors in this case is presented in section 6.

Alternatively, we can use an iterative, adaptive Simpson scheme, Algorithm 2 by Campagnolo *et al.* [CCdF15], to evaluate the emission integral up to a certain error bound. It is denoted by "adaptive" in Table 1. As one can see, it requires slightly more quadrature steps as, e.g., expanded Bernstein ("BS expand"). It is slightly slower in performance than fixing the number of quadrature steps, see Figure 10.

References

[CCdF15] L. Q. Campagnolo, W. Celes, and L. H. de Figueiredo. Accurate volume rendering based on adaptive numerical integration. In *2015 28th SIBGRAPI Conference on Graphics, Patterns and Images*, pages 17–24. IEEE, 2015.

[CS66] G. Cargo and O. Shisha. The bernstein form of a polynomial. *Journal of Research of the National Bureau of Standards*, 70:79–81, 1966.

[HJ85] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge, MA: Cambridge University Press, 1985.

[HSS+05] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. In *Computer graphics forum*, volume 24, pages 303–312. Wiley Online Library, 2005.

[KHK+09] A. Knoll, Y. Hijazi, A. Kensler, M. Schott, C. Hansen, and H. Hagen. Fast ray tracing of arbitrary implicit surfaces with interval and affine arithmetic. In *Computer Graphics Forum*, volume 28, pages 26–40. Wiley Online Library, 2009.

[MKW+04] G. Marmitt, A. Kleer, I. Wald, H. Friedrich, and P. Slusallek. Fast and accurate ray-voxel intersection techniques for iso-surface ray tracing. In *VMV*, volume 4, pages 429–435, 2004.

[NA92] K. Novins and J. Arvo. Controlled precision volume integration. In *Proceedings of the 1992 workshop on Volume visualization*, pages 83–89, 1992.

[Nic06] R. W. D. Nickalls. Viète, descartes and the cubic equation. *The Mathematical Gazette*, 90(518):203–208, 2006. doi:10.1017/S0025557200179598.

[NMHW02] A. Neubauer, L. Mroz, H. Hauser, and R. Wegenkittl. Cell-based first-hit ray casting. In *VisSym*, pages 77–86, 2002.

[PTVF88] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. Numerical recipes in c, 1988.

[Sch13] J. Schwarze. Cubic and quartic roots. In A. S. Glassner, editor, *Graphics gems*, chapter VIII.1, pages 404–407. Elsevier, 2013.

[SN09] J. M. Singh and P. Narayanan. Real-time ray tracing of implicit surfaces on the gpu. *IEEE transactions on visualization and computer graphics*, 16(2):261–272, 2009.