# Analytic Ray Splitting for Controlled Precision DVR

Sebastian Weiss and Rüdiger Westermann

Technical University of Munich, Germany

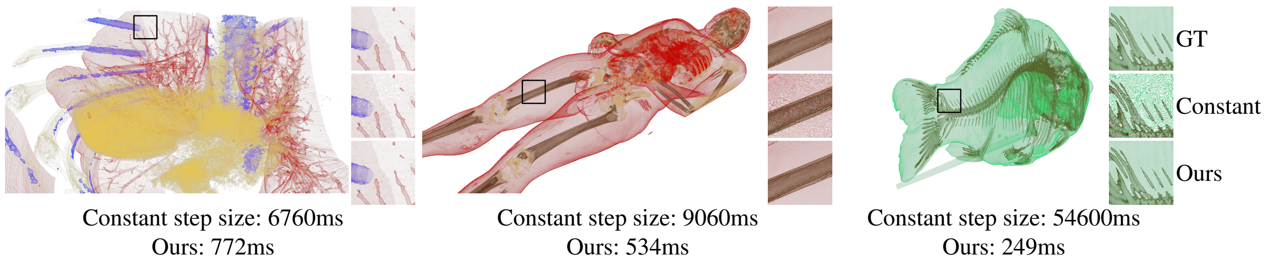
GT
Constant
Ours
Constant step size: 6760ms
Ours: 772ms
Constant step size: 9060ms
Ours: 534ms
Constant step size: 54600ms
Ours: 249ms

**Figure 1:** *DVR via post-classification. Using a constant step size (i.e.,* 0.2 *voxels) leads to sampling artifacts for high-frequency transfer functions. Analytic ray splitting (Ours) produces an image indistinguishable from the ground truth (GT) rendering, i.e., on an 8 bit color display the results are the same, and renders at less time than constant stepping.*

**Abstract**

*For direct volume rendering of post-classified data, 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 over which the variation of the optical properties is described by piecewise cubic functions. This allows using numerical quadrature rules with controlled precision to obtain an approximation with prescribed error bounds. The proposed splitting scheme can be used to find all piecewise linear or monotonic 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.*

**CCS Concepts**

• ***Computing methodologies*** $\rightarrow$ ***Volumetric models;*** *Parallel algorithms;* • ***Mathematics of computing*** $\rightarrow$ *Quadrature;*

## 1. Introduction

In direct volume rendering (DVR), the perceived lightness is determined 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 along the rays of sight [WM92]. In previous works, error bounds for the used numerical integration rules have been investigated [NA92, WM92, dBGHM97, CCdF15]. Etiene *et al.* [EJR+13] study the relationships between integration step size and accuracy of the results, and they propose a framework to assess the convergence of DVR algorithms with respect to step size, pixel size, and grid resolution.

Novins and Arvo [NA92] assume that both an emission and absorption field is given, i.e., the initial data values are pre-classified via transfer functions (TFs). When using trilinear interpolation in the cubical cells of a voxel grid, in each cell the profile of the interpolated quantities along a ray become cubic polynomials [PSL+98]. For these polynomials, error bounds for the numerical integration have been derived. 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 algebraic solutions for ray-isosurface intersections in the trilinear interpolant.

When post-classification is used, i.e., the initial data 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 a linear variation of the optical properties within each cell, Williams and Max show that the integral can be computed algebraically [WM92]. Pre-integration [RKE00, EKE01] builds upon the assumption that the data values between adjacent sample points along a ray vary linearly. Then, integrals—including post-classification—can be pre-computed and used depending solely on the values at the sample points. 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 monotonic change of the data values is assumed.

In the common case where piecewise linear 1D TFs are applied to the trilinearly interpolated data values, a ray is split into multiple piecewise cubic segments (see Figure 2). Then, piecewise linearity or monotony between adjacent sample points can only be justified for very small step sizes. In general, false classifications can occur and segments might even be missed entirely if the step size does not adapt to the segment boundaries.

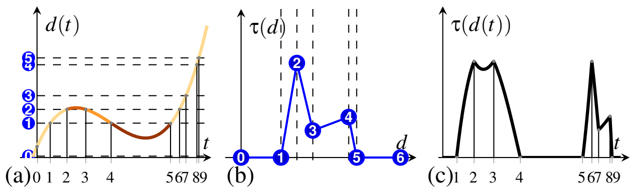

**Figure 2:** *(a) In a cell with trilinear interpolation, the data values along a ray $s(t)$ are described by a cubic polynomial. (b) A piecewise linear TF. (c) Applying the TF from (b) to the data values in (a) results in a piecewise cubic function. Dashed lines indicate the data values where the TF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

**Contribution** We propose an algorithm for adaptive step size control in DVR of post-classified data on a voxel grid. For post-classification, we consider piecewise linear TFs (Figure 2b). For cubical cells with trilinear interpolation, this splits a ray into multiple piecewise cubic segments. By determining these segments analytically, adaptive step size control is achieved. In particular, we

- build upon the algorithm by Marmitt *et al.* to split a ray through a voxel grid at all control points of a piecewise defined TF (subsection 2.1),
- apply controlled-precision numerical integration of post-classified data to solve the volume rendering integral on a per-segment basis (subsection 2.2),
- demonstrate the use of the proposed algorithm to split a ray into piecewise monotonic segments that are required by scale-invariant DVR (subsection 2.3).

All operations involved in analytic ray splitting have been implemented on the GPU to provide interactive frame rates even for large data sets. We have made our implementations available on GitHub[†], so that all findings can be reproduced. A video showing the method and results in animation is available on YouTube[‡].

## 2. Solving the Volume Rendering Integral

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 $v$ along a ray $s(t)$ – $d(t) = v(s(t))$ –, and $C : [0,1] \to \mathbb{R}_0^+$ the assigned color, both specified via a TF. With $g(v) = \tau(v)C(v)$ being the self-emission, the light intensity reaching the eye along the ray segment from $t = a$ to $b$ is computed as

$$L(a,b) = \int_a^b g(v(s(t))) \exp\left(-\int_a^t \tau(v(s(u)))\mathrm{d}u\right)\mathrm{d}t. \quad (1)$$

We assume that the data values are given at the vertices of a voxel grid $v$. Within a cell, the data values are trilinearly interpolated, so that the data profile $v(s(t))$ along a ray through a cell is a cubic polynomial [PSL$^+$98]. As a consequence, there are between zero and three locations along a ray in a single cell where a selected data value $\theta$ can be hit. These locations are given by the roots of $v(s(t)) - \theta = 0 \in [t_{\text{in}}, t_{\text{out}}]$. Marmitt *et al.* [MKW$^+$04] and Neubauer *et al.* [NMHW02] have proposed numerical schemes to extract these roots. To avoid catastrophic cancellation when isolating the extrema of the cubic polynomial via the roots of its derivative, we use a numerically stable formulation [PTVF88, p. 184].

For cell-by-cell ray marching, we use the voxel traversal algorithm by Józsa *et al.* [JTC14], a stable reformulation of the algorithm by Amanatides *et al.* [AW$^+$87, Hoe16]. Both the accumulation of numerical errors and errors due to rounding are reduced, so that even for large data sets the ray traversal routine does not introduce any perceivable errors. The algorithm provides the sequence of visited cells in front-to-back order, as well as the per-cell entry and exit points $[t_{\text{in}}, t_{\text{out}}]$. Note that this algorithm reports multiple roots either as no intersection or two separate intersections with two single roots, eliminating special handling of this case.

### 2.1. Transfer Function-Based Ray Splitting

In contrast to previous works, our proposed ray splitting scheme needs to consider the mapping of data values via a TF. 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 so that no peaks are missed. Second, the absorption between two control points $\tau(v(s(u)))$ has to be analytically integrable. This is required to evaluate the inner integral in Equation 1 analytically and, thus, reduce the nested volume rendering integral to a single integral. The latter constraint holds, e.g., for all cell-wise polynomial interpolations. Specifically, we assume piecewise linear TFs as shown in Figure 2b. Absorption and emission values are given at a discrete set of $N$ data values $0 = d_1 < d_2 < \ldots < d_N = 1$. We call these the TF control points. Each control point $i$ stores the absorption $\tau_i$ 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.

Given the TF, all points where the ray takes on the values of the TF control points need to be found.We subsequently call these points, solutions of the cubic polynomial $d(t) - d_i$, the split points along a ray. For example, Figure 2 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, multiple segments can occur within a single cell, and these segments need to be sorted efficiently. Note

[†] https://github.com/shamanDevel/Ray-Splitting-for-DVR
[‡] https://youtu.be/bOLqIJd6dsw

that depending on the data values at the cell vertices and the number and width of the selected TF, up to $3N$ intersections can occur.

Furthermore, ambiguous cases regarding the occurrence of intersections need to be resolved. Such cases occur if no other intersection is in between two intersections for the same control point. Figure 3 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 2a, 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).

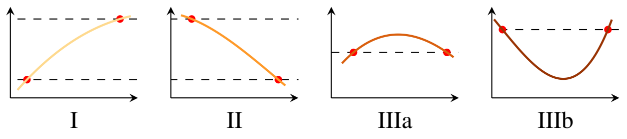

**Figure 3:** *Four cases of consecutive isosurface intersections.*

Sorting all computed split points along a ray can be realized in $O(N' \log N')$ operations using standard sorting algorithms, where $N'$ is the number of control points falling into the data range covered by the current cell. Ambiguous cases can be decided, in principle, by evaluating the 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 polynomial and should be avoided. The following adaptive ray splitting algorithm can iterate over the split points in $O(N')$ operations, and handles the ambiguous cases directly without extra evaluations.

First, the data value at the cell entry point ($t_0$) 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 2, this is the segment ⓿–❶. Now due to continuity, the next split point can only be at the data values corresponding to these two control points. W.l.o.g. let the next split point at $t_1$ be at the data value of control point ❶, the case for the lower control point $d_i$ follows similarly by symmetry considerations. Since this data value is approached from a lower data value, the next intersection at $t_2$ can only be at the same data value at control point ❶ again or at 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 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 data value. This process is repeated until all split points are processed and the ray exits the cell at $t_{\text{exit}}$. For each ray segment that is computed, the volume rendering integral is solved via quadrature, and the resulting color values are blended in front-to-back order.

## 2.2. Controlled Precision DVR

Novins and Arvo [NA92] propose various quadrature schemes for a single cell and a single linear transfer function. For a piecewise

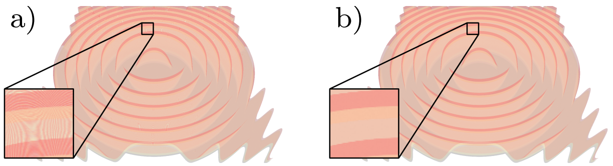

**Figure 4:** *a) Scale-invariant DVR with a step size of* 0.1 *voxels suffers from artifacts due to under-sampling and erroneously assumed per-segment monotony. b) Analytic ray splitting finds the exact locations of piecewise monotonic segments along the rays.*

TF, however, their method, cannot be directly applied to the entire cell. Instead, using our proposed ray splitting algorithm, smaller segments within a cell bounded by the control points of the TF are extracted. For each such segment, we apply the algorithm by Novins and Arvo to evaluate the integral per segment and accumulate the results over the segments and cells. In the benchmarks, we use Simpson quadrature with a fixed number of 10 quadrature points to reduce branch divergence in CUDA. Using more quadrature points does not improve the results.

## 2.3. Scale-Invariant 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 (Figure 4). This model builds upon the assumption that the density field is piecewise monotonic within the interval $[a,b]$ in data space, an assumption that is only fulfilled at small step sizes.

In our framework, scale-invariant DVR including an adaptive step size that splits the data into piecewise monotonic segments can be easily realized. To enforce monotony over each integration interval, additional split points need to be included at the extrema of the density profiles along the ray segments. In the root finding method by Marmitt *et al.*, these split points are computed in-turn to split a ray segment into sub-segments in which only zero or one root of the polynomial is located.

## 3. Evaluation on Synthetic Datasets

We compare the quality and performance of adaptive ray-splitting to DVR via ray-casting with constant step size and pre-integrated DVR [RKE00, EKE01] with a 32 bit floating point table of size $512^2$. The Marschner Lobb dataset [ML94] on a $128^3$ voxel grid is rendered to a $512^2$ viewport. Two triangular TFs with three peaks of width 0.05 and 0.002, respectively, are used to analyze the sensitivity of the DVR variants to the characteristics of the mapping function. To generate ground truth images, we use constant stepping with a step size of 0.0001 voxels using double-precision floats.

To obtain insights as to where approximation errors occur, the Marschner Lobb is rendered using different DVR algorithms (see Figure 5). At a peak width of 0.05, the transitions between the features in the image are smooth, yet constant stepping already shows

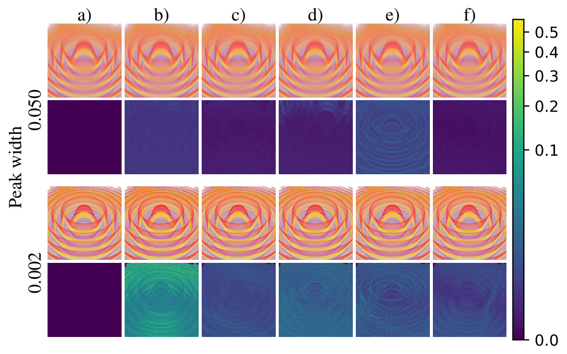

**Figure 5:** *Comparison of different DVR algorithms. (a) Ground truth, (b,c) and (d,e) constant stepping and pre-integration, respectively, with a step size of 0.1 (b,d) and 0.01 (c,e) voxel, (f) adaptive step size with Simpson quadrature. The first and third row show the rendering, the second and forth row the per-pixel $L_2$-norm of the color difference to the baseline, scaled using a power norm of $x^{0.3}$.*

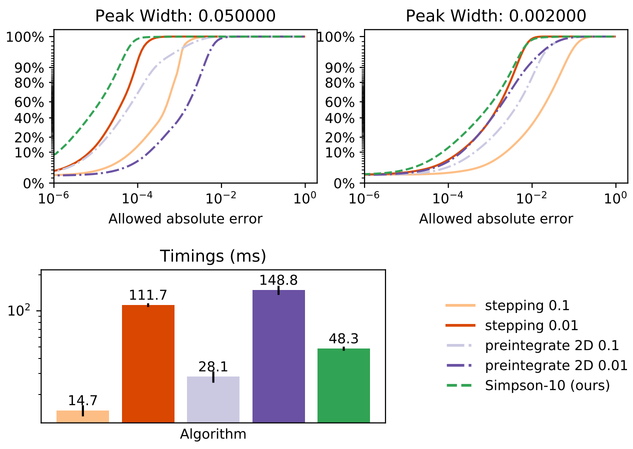

**Figure 6:** *Accuracy statistics and timings for the Marschner Lobb.*

noticeable pixel errors over the whole image. Analytic ray splitting with Simpson quadrature produces significantly lower numerical errors below the perceivable tolerance. When narrowing the peaks to 0.002, the errors introduced by constant stepping become unacceptably high. Constant stepping with a step size of 0.1 fails to resolve the thin structures, which can be improved to some degree by decreasing the step size to 0.01. Pre-integration quality is affected by sampling artifacts, discretization errors in the pre-integration table, and interpolation accuracy in the GPU texture units. Adaptive ray splitting generates results that are visually indistinguishable from the ground truth rendering, and it seems en par with constant stepping with a step size of 0.01 voxel. However, Figure 6 shows so-called regression error characteristic (REC) curves, which indicate the percentage of pixels within a certain allowed error. It can be seen that ray splitting achieves higher accuracy than all alternatives. Table 1 shows the error metrics for the images in Figure 1.

All DVR 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 structure is used. The timings in Figure 6 show the linear performance scale of constant stepping approach in the step size. Adaptive ray splitting lies between constant stepping with a step size of 0.1 and 0.01. Profiling shows 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.

| Dataset | Thorax | Human | Carp |
|---|---|---|---|
| Simpson $e$ | 4.69e-5 | 1.13e-4 | 6.60e-4 |
| Simpson $t$ | 7.72e2 | 5.34e2 | 2.49e2 |
| Stepping $s$ | 2.14e-3 | 1.56e-2 | 4.93e-4 |
| Stepping $e$ | 3.24e-4 | 7.47e-4 | 6.08e-4 |
| Stepping $t$ | 6.76e3 | 9.06e3 | 5.46e4 |

**Table 1:** *Error and timing statistics for the data sets in Figure 1. s, e and t, respectively, indicate step size in constant stepping, mean square pixel error compared to the baseline rendering, and rendering time in milliseconds.*

## 4. Real-World Datasets

We further evaluate the quality and performance of the DVR algorithms on three CT scans (Figure 1): A human thorax at a size of $512^2 \times 286$, a full scan of the human body at a size of $512^2 \times 1884$, and a carp at size $512^3$, rendered at resolution $1920 \times 1080$. We compare the ray-splitting algorithm with Simpson quadrature to ray tracing with a constant step size and measure the mean absolute error to the baseline and the execution time. For constant stepping, the step size is halved until the rendering has converged, i.e. the output does not change within an error of $1/256$ per pixel anymore. Table 1 reports the final step size, timings and error to the baseline. As one can see, adaptive ray splitting is 1-2 magnitudes faster at a lower or similar error than converged constant stepping.

## 5. 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 linear transfer function. This allows for solving analytically the absorption integral and numerically the emission integral up to a user-defined precision. Our evaluations have shown that the performance of analytic ray splitting can be even higher than that of constant stepping when triangular TFs with rather narrow peaks are used. We consider the proposed renderer as baseline 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 requires differentiable renderers that can 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.

[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.

[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.

[EKE01] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, HWWS '01, page 9–16, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/383507.383515.

[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.

[JTC14] P. Józsa, M. J. Tóth, and B. Csébfalvi. Analytic isosurface rendering and maximum intensity projection on the gpu. In *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*. Václav Skala-UNION Agency, 2014.

[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.

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

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

[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.

[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.

[RKE00] S. Rottger, M. Kraus, and T. Ertl. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *Proceedings Visualization 2000. VIS 2000 (Cat. No.00CH37145)*, pages 109–116, 2000. doi:10.1109/VISUAL.2000.885683.

[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.