Verifying Volume Rendering Using Discretization Error Analysis

Tiago Etiene, Daniel Jönsson, Timo Ropinski, Carlos Scheidegger, João Comba, L. Gustavo Nonato, Robert M. Kirby, Anders Ynnerman, and Cláudio T. Silva

UUSCI-2013-001

Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA

March 12, 2013

Abstract:

We propose an approach for verification of volume rendering correctness based on an analysis of the volume rendering integral, the basis of most DVR algorithms. With respect to the most common discretization of this continuous model (Riemann summation), we make assumptions about the impact of parameter changes on the rendered results and derive convergence curves describing the expected behavior. Specifically, we progressively refine the number of samples along the ray, the grid size, and the pixel size, and evaluate how the errors observed during refinement compare against the expected approximation errors. We derive the theoretical foundations of our verification approach, explain how to realize it in practice and discuss its limitations. We also report the errors identified by our approach when applied to two publicly-available volume rendering packages.

Verifying Volume Rendering Using Discretization Error Analysis

Tiago Etiene, Daniel Jönsson, Timo Ropinski, Carlos Scheidegger, João Comba, L. Gustavo Nonato, Robert M. Kirby, *Member, IEEE*, Anders Ynnerman, and Cláudio T. Silva, *Fellow, IEEE*

Abstract—We propose an approach for verification of volume rendering correctness based on an analysis of the volume rendering integral, the basis of most DVR algorithms. With respect to the most common discretization of this continuous model (Riemann summation), we make assumptions about the impact of parameter changes on the rendered results and derive convergence curves describing the expected behavior. Specifically, we progressively refine the number of samples along the ray, the grid size, and the pixel size, and evaluate how the errors observed during refinement compare against the expected approximation errors. We derive the theoretical foundations of our verification approach, explain how to realize it in practice and discuss its limitations. We also report the errors identified by our approach when applied to two publicly-available volume rendering packages.

Index Terms—discretization errors, volume rendering, verifiable visualization, verification, testing

1 Introduction

In the last several decades, the visualization and graphics communities have developed a wide range of volume rendering techniques. As they are used in several different disciplines of science, and thus form a basis for new scientific insights, it is essential to assess their reliability and identify errors. Furthermore, the increasing complexity of volume rendering algorithms makes the correctness of the algorithm itself as well as its potentially error-prone implementations complementary and equally important issues. Being that volume rendering is essential in areas such as medical imaging, where accuracy and precision play a crucial role, a formal methodology for assessing correctness is highly desirable [18], [32]. While verification has been widely adopted in many different branches of computer science – see model checking [3], fuzzing [12], and convergence analysis [38] – there has not been significant work accomplished on a formalized praxis for asserting the correctness of visualization techniques. In this article we present a new verification approach for direct volume rendering techniques as well as its theoretical background. We use the word verification in the same sense as Babuska and Oden [1]: "verification is the process of determining if a computational model, and its corresponding numerical solution, obtained by discretizing the mathematical model (with corresponding exact solution) of a physical event, and the code implementing the computational model can be used to represent the mathematical model of the event with sufficient accuracy". The presented methodology is based on order-of-accuracy and convergence analysis [38] which we can apply after deriving the expected behavior of the algorithms under observation.

To allow the verification of volume rendering algorithms, we start with an analysis of the volume rendering integral and the most common discretization of this continuous model – Riemman summation. This analysis gives us insight into the expected behavior of the observed algorithms, which is essential to perform verification [14]. In this sense, our main assumption, serving as a foundation for the proposed verification approach, is that discretization errors of the implementations under verification should behave as the errors introduced by the discretization of the volume rendering integral. Based on this, we can mathematically derive the expected behavior from the discretization of the volume rendering integral and verify existing implementations through convergence analysis by comparing their actual behavior to the expected behavior. Based on the results of this comparison, we can assess the correctness of the implementation under verification. To get further insights about deviations from the expected behavior, we present an investigation of the sensitivity of this method. We can demonstrate that our methodology is capable of increasing the confidence in volume rendering algorithms. To our knowledge, the proposed approach is the first step towards the verification of DVR algorithms. Thus, it can be seen as an important contribution towards a formal verification methodology of volume rendering techniques [35]. Our main contributions are:

- we derive the theoretical foundations necessary for verifying volume rendering with order-of-accuracy and convergence analysis. We analyze the volume rendering integral and its (common) discretization using Riemann summation to derive an algorithm's expected behavior when being subject to parameter changes;
- we explain how to exploit these theoretical foundations to perform a practical verification of implemented volume rendering algorithms, such that it can be easily used for the verification of existing volume rendering frameworks;
- we discuss the limitations of the proposed concepts by analyzing frequently occurring errors and by documenting those errors we could identify when applying the pre-

- *Etiene and Kirby are with University of Utah, USA*
  E-mail: {tetiene,kirby}@cs.utah.edu
- *Jönsson, Ropinski and Ynnerman, are with Linköping University, Sweden*
  E-mail: {daniel.jonsson,timo.ropinski,anders.ynnerman}@liu.se
- *Nonato is with Universidade de São Paulo, Brazil*
  E-mail: gnonato@icmc.usp.br
- *Comba is with Universidade Federal do Rio Grande do Sul, Brazil*
  E-mail: comba@inf.ufrgs.br
- *Scheidegger is with AT&T Labs – Research*
  E-mail: cscheid@research.att.com
- *Silva is with NYU Poly, USA*
  E-mail: csilva@nyu.edu

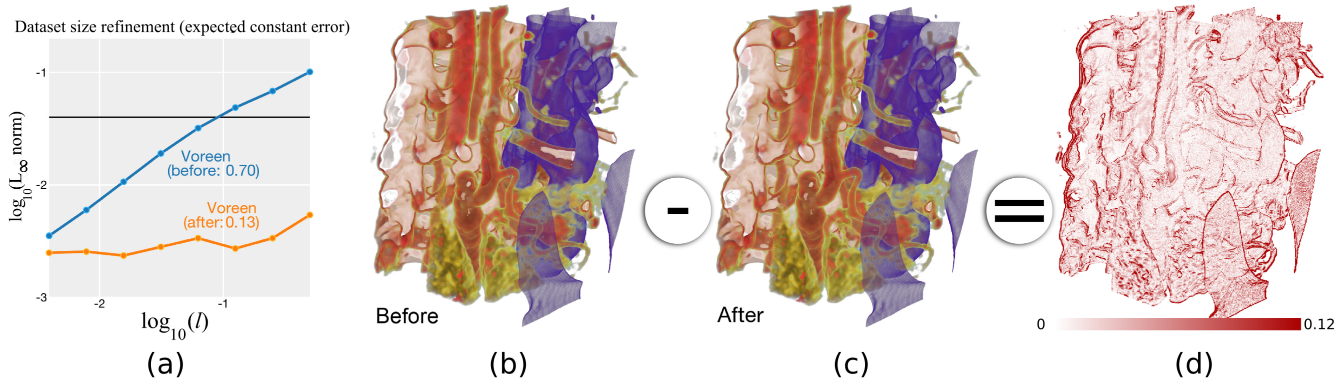


Fig. 1. (a) shows the result of our verification procedure for dataset refinement. The blue line corresponds to the initial behavior, which deviates from the expected slope (solid dark line). After fixing the issues, we obtain the orange curve, with a slope closer to the expected one. (b) and (c) show a human torso, displaying the blood vessels and the spine, before and after our changes. (d) shows the difference between (b) and (c).

sented methodology to two widely used volume rendering frameworks, VTK [39] and Voreen [27] (see Figure 1).

2 Related Work

Critical decisions in fields such as medical imaging often rely on images produced by volume rendering algorithms, where it is of utmost importance that the results are correct [4]. The multitude of algorithm components and their interactions make this guarantee a challenge. As a consequence, many authors focus on specific aspects of the problem such as numerical aspects of the evaluation of the volume rendering integral, shading, transfer functions, and interpolation schemes. The quality of volume rendering has always been of central interest to the community, and relying on visual inspection is a common practice. Meissner *et al.* [26] evaluate volume rendering techniques using the human visual system as a reference while, more recently, Smelyanskiy *et al.* [40] present a domain expert guided comparison scheme. While those approaches are valuable, the need for a more systematic evaluation is discussed in several papers [11], [15], [16], [18]. See Pommert and Höhne [32], [33] for a survey.

Among several aspects to consider in the correctness of volume rendering algorithms, one of the most important is the approximation of the volume rendering integral. The solution with linearly interpolated attributes is presented by Williams and Max [44], with further discussions on its numerical stability by Williams *et al.* [45]. Interpolant approximations and errors [5], [28], [29], [31], gradient computation [42] and opacity correction [23] are also the subject of analysis with regard to numerical accuracy. The idea of *pre-integration* enables high-quality, accurate and efficient algorithms using graphics hardware [7], [22], [37]. Similarly, VTK currently uses partial pre-integration, in particular for unstructured grids [30]. Note that although there has been work on high-order and high-accuracy volume rendering – to the best of our knowledge – none of these approaches attempted to evaluate the convergence rate of the standard discretization process of the volume rendering integral.

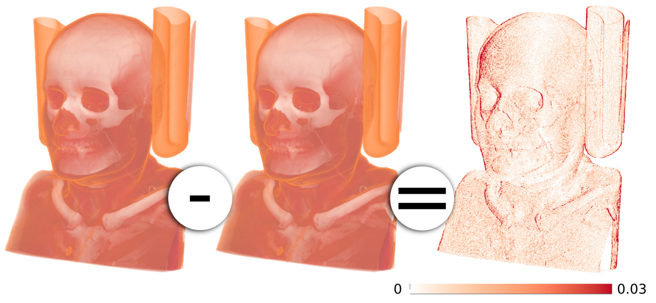


Fig. 2. Left: the volume rendering of the torso dataset using an incorrect trilinear interpolation. Middle: Same dataset with the correct interpolation. Right: difference between the two images.

The use of a verification framework has only recently been discussed in scientific visualization, despite the vast literature on verification in computer science. Globus and Uselton [11] first pointed out the need to verify not only visualization algorithms but also their implementations, and Kirby and Silva suggested a research program around verification [18]. The verification of isosurface algorithms was discussed by Etiene *et al.* [8], [9], where a systematic evaluation identified and corrected problems in several implementations of isosurface extraction techniques. Zheng *et al.* [47] address CT reconstruction and interpolation errors in direct volume rendering algorithms using a verifiable framework based on projection errors. In contrast, our work focuses on the verification of the final image produced through direct volume rendering.

3 Verification

Before presenting our verification procedure, let us consider four of the techniques used for code verification in computational science [38]: *expert judgment*, a procedure in which a field expert determines if the output of an implementation is correct by evaluating the results; *error quantification*, which is the quantification of the discretization errors when compared to

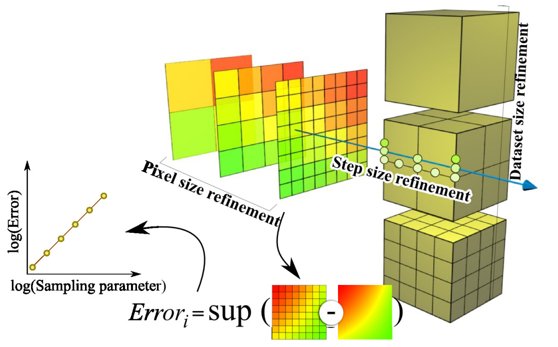


Fig. 3. Our verification procedure works by evaluating discretization error during refinement of one of three sampling parameters.

an analytical solution, a benchmark solution or some ground-truth; *convergence analysis*, a procedure in which one evaluates if the discretization errors converge to zero as a function of some parameter; and *order-of-accuracy*, a procedure where one evaluates if the discretization errors decrease according to the expected rate. In this list, the expert judgment is the least rigorous test, followed by error quantification and convergence analysis. Order-of-accuracy is widely recognized as the most rigorous code verification tool [1], [19], [35], [38]. In this paper, we focus on the latter two methods, namely, convergence analysis and order-of-accuracy. Before we dive into these methods, let us first consider some of the limitations of the expert analysis and error quantification.

In visualization, expert analysis and error quantification are, to the best of our knowledge, the only two verification tools previously employed for verification of volume rendering techniques [26], [28], [40]. Whereas it is easy to envision situations where an expert may fail to predict a code mistake, it is more difficult to see when error quantification fails. We devise the following experiment to understand potential limitations of both approaches. We artificially introduced a code mistake in a volume rendering implementation: the trilinear interpolation was changed from $p(x,y,z) = Axyz + \underline{Bxy(1-z)} + \ldots$ to $p(x,y,z) = Axyz + \underline{Axy(1-z)} + \ldots$. We then used this implementation to render an image whose analytical solution is known. Finally, we compute the maximum error between the rendered and the analytical solution, which in this case is 3.6×10^{-3}. How can one decide if this value is good enough? Does the sampling distance d or the input scalar field $s(x,y,z)$ give us enough data to make an informed decision? In this particular case, the correct interpolant generates an image with maximum error of 3.4×10^{-3}: the two images are very similar by this metric. Also, it may be challenging, even for an expert, to notice such a small deviation, as shown in Figure 2. On top of this, the maximum errors for another code mistake could be even smaller. (We point out that this particular case can be uncovered by "playing around" with the data or other *ad hoc* methods. The goal is to show that error quantification can also fail to predict code mistakes, even for a severe bug.) On the other hand, we will have enough information to make such a decision if one observes how errors behave when input parameters change instead of quantifying them from one image. The convergence and order-of-accuracy tests work in this way, and they are the focus of this paper.

We advocate the use of convergence and order-of-accuracy verification not as a replacement but as an extension of the current testing pipeline. Note that these are not the only approaches for assessing correctness of computer code. As mentioned before, verification is well-developed in computer science [3], [10], [12], [46].

We apply verification in the spirit of Babuska and Oden's procedure, which we summarize in Figure 3 [1]. It starts with a mathematical evaluation of the expected convergence of the volume rendering integral (Section 5). The result of this step is an articulation of the asymptotic error according to some discretization parameter (step size, dataset size, or pixel size). Then, we use the volume rendering implementation under verification to generate a sequence of images by successive refinement of one of the discretization parameters. Next, we compute the observed discretization errors by comparing these images against a reference – an analytical solution, if one is available, or one of the rendered images. Finally, we compare the sequence of observed outputs against expected errors to evaluate if expected and observed convergence match (Sections 6 and 7).

4 Discretization Errors

In this section, we present the mathematical model used in volume rendering algorithms and its *expected behavior*, which we write in terms of the errors involved in each discretization step. Let us assume the well-known *low albedo emission* plus *absorption* model [25]. The volume rendering integral (VRI) I, as described by Engel *et al.* [7], is:

$$I(x,y) = \int_0^D C(s(\mathbf{x}(\lambda)))\tau(s(\mathbf{x}(\lambda))) \times \exp\left(-\int_0^\lambda \tau(s(\mathbf{x}(\lambda')))\mathrm{d}\lambda'\right)\mathrm{d}\lambda, \quad (1)$$

where D is the ray length, $C(s(\mathbf{x}(\lambda)))$ is the reflected/emitted light, $\tau(s(\mathbf{x}(\lambda)))$ is the light extinction coefficient, $s(\mathbf{x}(\lambda))$ is the scalar value at position $\mathbf{x}$ in the ray parameterized by λ. There are three natural ways to discretize the equation. We will generate progressively denser ray sampling (by refining the integration *step size*), progressively larger datasets (by refining the *size of the voxel in the dataset*), and progressively higher-resolution images (by refining the *pixel size in the final image*). Each of these three variables will introduce errors that may appear in a volume rendering system, where the first two of these variables specifically impact the volume rendering integration (per pixel/ray). In the following section, we discretize the VRI using the most common approximation in literature – Riemann summation.

4.1 Errors Due to Step Size Refinement

In this section, we are interested in the errors generated by successive ray step refinements (see Figure 4). Equation (1) is commonly discretized using traditional Riemann summation

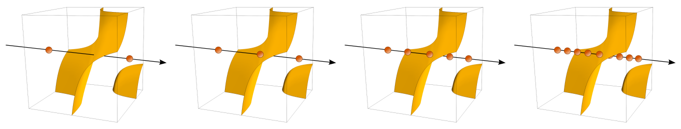


Fig. 4. Step size refinement. The figure shows an isosurface of a trilinear function defined on the volume.

for numerical integration:

$$\int_0^D f(s(\mathbf{x}(\lambda)))\mathrm{d}\lambda = \sum_{i=0}^{n-1} f(s(\mathbf{x}(id)))d + O(d), \tag{2}$$

where n is the number of sub-intervals and $d = D/n$. The proof of linear convergence follows from Taylor expansion of the integrand over small intervals d. Other methods are available and they provide different convergence rates. For instance, the Trapezoidal Rule is a 2nd order method on the integral of f.

In the case of the VRI, we approximate not only the outer integral but also the integrand $T(s(\mathbf{x}(\lambda))) = \exp\left(-\int_0^\lambda \tau(s(\mathbf{x}(\lambda')))\mathrm{d}\lambda'\right)$. Moreover, T requires two approximations: $e^{t(\lambda)}$ and the inner integral. Before we derive the convergence rate for the VRI, let us first evaluate the convergence of T. Throughout the text, we assume that all transfer functions are smooth, *i.e.,* $C(s), \tau(s) \in C^\infty$. Although this is not the case in practice, this restriction is useful for convergence evaluation and verification purposes.

4.1.1 Approximation of $T(\lambda)$

Let $T(\lambda) = T_\lambda = e^{-t(\lambda)}$, where $t(\lambda) = \int_0^\lambda \tau(\lambda')\mathrm{d}\lambda'$, and λ parameterizes a ray position. We will first approximate $t(\lambda)$ and then $T(\lambda)$. Typically, the integral is solved using Riemann summation. In the following, $d = D/n$ is the ray sampling distance, D is the ray length and n is the number of sub-intervals along the ray:

$$\int_0^\lambda \tau(\lambda')\mathrm{d}\lambda' = \sum_{j=0}^{i-1} \tau(jd)d + O(d), \tag{3}$$

where $\lambda = id$. Using Equation (3):

$$\begin{aligned} T(\lambda) &= \exp\left(-\int_0^\lambda \tau(\lambda')\mathrm{d}\lambda'\right) & (4)\\ &= \exp\left(-\sum_{j=0}^{i-1} \tau(jd)d + O(d)\right) & (5)\\ &= \left(\prod_{j=0}^{i-1} \exp\left(-\tau(jd)d\right)\right) \exp\left(O(d)\right). & (6)\end{aligned}$$

Let us define $\tau_j = \tau(jd)$. We start with a Taylor expansion of $\exp(O(d))$:

$$\begin{aligned} T_\lambda &= \left(\prod_{j=0}^{i-1} \exp\left(-\tau_j d\right)\right)\left(1 + O(d)\right) & (7)\\ &= \prod_{j=0}^{i-1} \exp\left(-\tau_j d\right) + \prod_{j=0}^{i-1} \exp\left(-\tau_j d\right) O(d). & (8)\end{aligned}$$

Part I: Let us focus on the second term in the right hand side of Equation (8). The first observation is that it contains only approximation errors, which means that we are interested only in its asymptotic behavior. Let us expand it using first-order Taylor approximation and use the fact that $\tau_j d = O(d)$:

$$\prod_{j=0}^{i-1} \left(1 - O(d)\right) O(d) = \left(1 + O(d)\right)^i O(d), \tag{9}$$

where the change in the sign is warranted because the goal is to determine the asymptotic behavior. For $i = 1$, only one step is necessary for computing the volume rendering integral along the ray, and the previous equation will exhibit linear convergence. Nevertheless, in the general case, the numerical integration requires multiple steps, hence errors accumulate, and the convergence may change. Thus we set $i = n$. Knowing that $(1 + O(d))^n = O(1)$ (see Appendix) and inserting Equation (9) into Equation (8) we obtain:

$$\begin{aligned} T_\lambda &= \prod_{j=0}^{i-1} \exp(-\tau_j d) + O(d)O(1) & (10)\\ &= \prod_{j=0}^{i-1} \left(1 - \tau_j d + O(d^2)\right) + O(d). & (11)\end{aligned}$$

Part II: We now show that the first term on the right side of Equation (11) also converges linearly with respect to d. In the course of this section, we omit the presence of the term $O(d)$ in Equation (11) for the sake of clarity. Let us define the set K as the set of indices j for which $1 - \tau_j d = 0$. The size of K is denoted as $|K| = k$. We also define $\overline{K}$ as the set of indices j for which $1 - \tau_j d \neq 0$, and $|\overline{K}| = i - k$. Equation (11) can be written as:

$$\begin{aligned} T_\lambda &= \left(\prod_{j\in\overline{K}} \left(1 - \tau_j d + O(d^2)\right)\right)\left(\prod_{j\in K} O(d^2)\right) & (12)\\ &= \left(\prod_{j\in\overline{K}} \left(1 - \tau_j d + O(d^2)\right)\right) O(d^{2k}). & (13)\end{aligned}$$

Because $1 - \tau_j d \neq 0$ for $j \in \overline{K}$:

$$T_\lambda = \left(\prod_{j\in\overline{K}} (1 - \tau_j d)\left(1 + \frac{O(d^2)}{1 - \tau_j d}\right)\right) O(d^{2k}). \tag{14}$$

From the definition of big O notation, $1/(1 - \tau_j d) = O(1)$, hence:

$$\begin{aligned} T_\lambda &= \left(\prod_{j\in\overline{K}} (1 - \tau_j d)\left(1 + O(1)O(d^2)\right)\right) O(d^{2k}) & (15)\\ &= \left(\prod_{j\in\overline{K}} (1 - \tau_j d)(1 + O(d^2))\right) O(d^{2k}) & (16)\\ &= \left(\prod_{j\in\overline{K}} 1 - \tau_j d\right)(1 + O(d^2))^{i-k} O(d^{2k}). & (17)\end{aligned}$$

In real world implementation, $k \neq 0$ implies that at least one of the terms $1 - \tau_j d = 0$. Hence the code accumulating the value of T, `T = T * (1 - t`$_j$` * d)`, will invariably return `T = 0`. This can also be seen in our theoretical analysis.

For $k \neq 0$, the entire right-hand-side of Equation (17) *is* the approximation error. The larger k is – *i.e.* the more zeroes in the product of Equation (17) – the faster the sequence converges to zero due to the $O(d^{2k})$ factor. So, when $k \neq 0$, one obtains a high-order approximation of $T_\lambda = 0$. Nevertheless, because we want to recover the approximation errors for the general case ($T_\lambda \neq 0$), we set $k = 0$ in Equation (17), and $i = n$ (for the same reasons as previously stated):

$$T_\lambda = \left(\prod_{j=0}^{n-1} 1 - \tau_j d\right)(1 + O(d^2))^n. \quad (18)$$

Using the fact that $(1+O(d^2))^n = 1 + O(d)$ and $(1+O(d))^n = O(1)$ (see Appendix):

$$\begin{aligned} T_\lambda &= \left(\prod_{j=0}^{n-1} 1 - \tau_j d\right)(1 + O(d)) && (19)\\ &= \left(\prod_{j=0}^{n-1} 1 - \tau_j d\right) + O(d)\left(\prod_{j=0}^{n-1}(1 + O(d))\right) && (20)\\ &= \left(\prod_{j=0}^{n-1} 1 - \tau_j d\right) + O(d)(1 + O(d))^n && (21)\\ &= \left(\prod_{j=0}^{n-1} 1 - \tau_j d\right) + O(d)O(1) && (22)\\ &= \left(\prod_{j=0}^{n-1} 1 - \tau_j d\right) + O(d). && (23) \end{aligned}$$

We finish our derivation by adding the previously omitted $O(d)$:

$$\begin{aligned} T_\lambda &= \left(\prod_{j=0}^{n-1} 1 - \tau_j d\right) + O(d) + O(d) && (24)\\ &= \left(\prod_{j=0}^{n-1} 1 - \tau_j d\right) + O(d). && (25) \end{aligned}$$

4.1.2 Approximation of the Outer Integral

Let $\tilde{T}_i$ be the approximation of $T(\lambda_i)$. We write $T(\lambda_i) = T_i = \tilde{T}_i + O(d)$, and $C_i = C(id)$. In typical volume rendering implementations, the outer integral is also approximated using a Riemann summation. Thus:

$$\begin{aligned} I(x,y) &= \sum_{i=0}^{n-1} C(id)\tau(id)T_i d + O(d) && (26)\\ &= \sum_{i=0}^{n-1} C_i \tau_i d\left(\tilde{T}_i + O(d)\right) + O(d) && (27)\\ &= \sum_{i=0}^{n-1} C_i \tau_i \tilde{T}_i d + \sum_{i=0}^{n-1} C_i \tau_i d O(d) + O(d). && (28) \end{aligned}$$

Because both τ_i and C_i are bounded, one can write $C_i \tau_i d O(d) = O(d^2)$ and $\sum_i O(d^2) = nO(d^2) = \frac{D}{d}O(d^2) = O(d)$. The above equation can be re-written as:

$$I(x,y) = \sum_{i=0}^{n-1} C(id)\tau(id)d\tilde{T}_i + O(d). \quad (29)$$

We have now showed that the dominant error when considering step size in the VRI is of order $O(d)$. In other words, when decreasing the step size by half, the error should be reduced by a factor of a half.

TABLE 1
Effects of the different integration methods.

		Outer integral	
		Riemann	Trapezoidal
Inner integral	Monte Carlo	$O(d^{0.43})$	$O(d^{0.58})$
	Riemann	$O(d^{0.99})$	$O(d^{0.98})$
	Trapezoidal	$O(d^{1.01})$	$O(d^{2.00})$

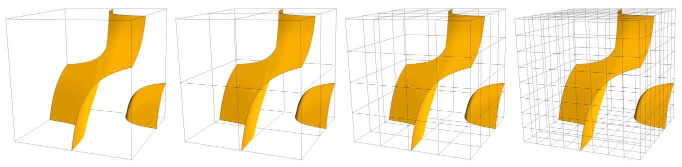


Fig. 5. Isosurface of a randomly-generated scalar field defined at different resolutions. The (piecewise) trilinear surface is the same regardless of the grid size.

4.1.3 Numerical Integration Techniques

The interplay between the approximation errors of the inner and outer integrals is non-trivial; here we demonstrate this fact with a simple numerical example. Table 1 shows the result of using different integration methods for the inner and outer integrals along a single ray. We simulate the integration along a single ray to compute these quantities numerically. For this experiment, we assume: $x \in [0,D]$, $\tau(s(x)) = \cos(s(x))$, $C(s(x)) = \sin(s(x))$, $s(x) = x$, and thus the solution for the VRI is $I = 1 - \exp(-\sin(D))(\sin(D)+1)$. To evaluate the effects of the discretization errors of the integrals, we further assume that $\exp(x)$ does not introduce errors. The computation of the convergence rate is detailed in Section 5. The results shown in Table 1 suggest that one needs to improve the accuracy of *both* the inner and outer integrals to obtain a high-order method.

4.2 Errors Due to Dataset Refinement

For the purposes of this paper, we assume that no additional errors will be created during the refinement of the scalar field. Hence, we need to find an interpolation function that fulfills the so called two-scaling property. Fortunately, B-splines fulfill the two-scale property [41]; we choose the linear B-spline, which results in the well-known trilinear interpolator. Care must be taken on the refinement step. In this paper, we will choose a refinement factor of two, which simplifies to a simple averaging of the nearest neighbors. The errors introduced so far remain unchanged:

$$I(x,y) = \sum_{i=0}^{n-1} C(\tilde{s}(\mathbf{x}_i))\tau(\tilde{s}(\mathbf{x}_i))d\tilde{T}(\tilde{s}(\mathbf{x}_i)) + O(d). \quad (30)$$

The previous equation shows that the grid size refinement errors are constant and due to sources other than the grid size N, such as the $O(d)$ term.

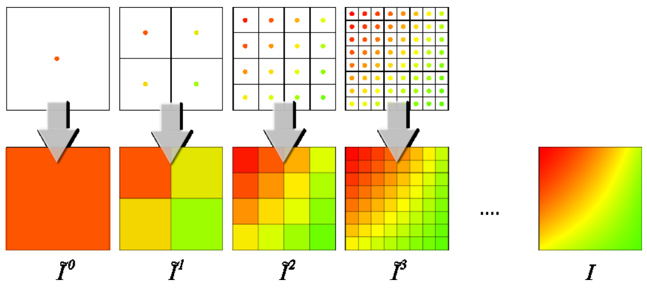


Fig. 6. Pixel refinement. Typically, the VRI is evaluated only at pixel centers (top). The value at the center is then interpolated in the domain defined by the pixel size (bottom) using nearest-neighbor interpolation to obtain $\tilde{I}^i$. In a correct implementation, as i increases, $\tilde{I}^i$ approaches the true solution I.

4.3 Errors Due to Pixel Size Refinement

The final source of errors we investigate comes from the finite number of rays sent into the image. This error is not one that arises due to the VRI discretization *per se*, but rather due to how the VRI is used to render the final image seen by the viewer. We quantify these errors by creating a sequence of images of progressively higher resolution, and then examine the supremum of the difference between values of the finite approximations of the volume-rendered image and the true solution. In this section, we assume that the derivatives along image axes of the volume rendering integral exist.

Denote the true volume-rendered image as $I(x,y)$. The approximation is constructed by sampling $I(x,y)$ in a finite subset of the domain (in our case, a square lattice of increasing resolution). At a level of detail j, $\tilde{I}^j(x,y)$ denotes the nearest-neighbor interpolation of the sampled values, and the error is measured as $E_j = \sup_{(x,y)\in[0,1]^2} |I(x,y) - \tilde{I}^j(x,y)|$. Effectively, this procedure assigns to the entire square pixel the value sampled in its center, but evaluates the error *over the entirety of the pixel values*. Figure 6 illustrates the process.

We use the notation $I = I(x,y)$, $I_i^j = I^j(x_i,y_i)$, and $\delta_i = (x,y) - (x_i,y_i)$. In what follows, the Taylor expansion assumes a fixed level of detail j. We omit the superscript j for the sake of clarity. Let us write the values of I as a Taylor series expansion ($\mathbf{H}_i = \mathbf{H}(x_i,y_i)$ is the Hessian and I_i^x and I_i^y are the partial derivatives of I_i at (x_i,y_i)):

$$\begin{aligned} I &= \tilde{I}_i + \nabla I_i^T \delta_i + \frac{1}{2}\delta_i^T \mathbf{H}_i \delta_i + \cdots && (31)\\ &= \tilde{I}_i + O(\nabla I_i^T \delta_i) && (32)\\ &= \tilde{I}_i + O((I_i^x, I_i^y)^T (x - x_i, y - y_i)), && (33) \end{aligned}$$

$\tilde{I}_i$ is a nearest-neighbor reconstruction from a square lattice for the pixel (x_i,y_i) at a given level j. In the regime where the Hessian terms are negligible, the dominant errors (and hence the supremum of the difference) occur when $(x - x_i, y - y_i) = (h,h)$, where h is half the pixel size. Thus:

$$\begin{aligned} I &= I_i + O(h) && (34)\\ &= \sum_{i=0}^{n-1} C(\tilde{s}(\mathbf{x}_i))\tau(\tilde{s}(\mathbf{x}_i))\tilde{T}d(\tilde{s}(\mathbf{x}_i))) + \\ &\qquad + O(h) + O(d). && (35) \end{aligned}$$

As can be seen, the error in the pixel approximation decays linearly with the pixel size. Equation (35) contains all the errors we examine for verification purposes, and it will be the basis for our analysis in Section 5.

Two practical aspects that are worth noting. In practice, the $\sup(x,y)$ of the error over a pixel (x,y) cannot be computed. Thus we use a finite high-resolution image as a proxy for the true solution. This allows us to evaluate the maximum error over a pixel. Note also that colors are evaluated at pixel center, as shown in Figure 6. Also, often the final image is not a smooth function. However, our goal is not to provide a characterization of discretization errors that can be used in any arbitrary setting, but instead one that can be used for verification purposes. Therefore, to use the analysis outlined above, one must manufacture scalar fields and transfer functions which yield a smooth function (cf. Section 6).

5 Convergence Computation

The heart of our method is the evaluation of the discretization errors. Once we have the discretization errors, we can evaluate the order-of-accuracy and convergence rate. The error computation and analysis will proceed differently depending on whether an analytical solution for the VRI is available or not. We highlight that previous frameworks for verification of visualization algorithms could benefit from the fact that analytical solutions can be easily constructed [9]. For the case of the VRI, this is no longer true, and therefore we should not rely on known solutions. We describe two ways in which to proceed with the convergence analysis. First, we will show how to calculate errors using a known solution, and then how to do so when the analytical solution is not known *a priori*.

5.1 Numerical Errors Using a Known Solution

When a solution $F(x,y)$ for the VRI is known, the procedure is equivalent to the Method of Manufactured Solutions [1]. In the previous section, we have shown that the solution F can be written as:

$$F(x,y) = I(x,y) + O(r^k) = I(x,y) + \beta r^k + \text{HOT}, \qquad (36)$$

where I is the approximated image, r is the discretization parameter and $\beta \in \mathbb{R}$ is a constant, multiplicative factor that is not a function of the dataset. An important assumption is that the HOT, or "higher order terms", are small enough that they do not affect the convergence of order $k \in \mathbb{R}$, *i.e.*, high-order derivatives of F must have negligible impact in the asymptotic convergence of I [38]. This formulation implies that not all solutions F are suitable for verification purposes, only those for which the HOT are negligible. In addition, integration methods whose approximation errors cannot be written as shown cannot be compared by only evaluating k, as we propose next. The expected value of k for the cases of step size and pixel size refinement is $k = 1$, whereas we do not expect to see error reduction when examining grid size refinement. This implies that the pixel intensity converges to the true solution at a rate determined by k, and thus the error can be written as:

$$e(x,y) = I(x,y) - F(x,y) \approx \beta r^k. \qquad (37)$$

One can evaluate the convergence for all pixels in the image using L_2, L_∞, or other norms. Henceforth, we adopt the L_∞ norm because it provides a rigorous and yet intuitive way of evaluating errors: it tells us that the maximum image error should decay at the same rate k. Mathematically, the error is then:

$$E = \sup_{x,y}(e(x,y)) = \sup_{x,y}(|I(x,y) - F(x,y)|) = \beta r^k. \tag{38}$$

We denote individual images (and the respective errors) by a subscript i. For each image I_i, we first calculate the supremum of the absolute difference $\sup_{x,y}(|F(x,y) - I_i(x,y)|)$. We then compute the observed convergence rate k by taking logarithms of both definitions of E and solving the resulting equations for $\log(\beta)$ and k in a least-squares sense:

$$\begin{aligned} \log E_i &= \log \sup_{x,y}(|F(x,y) - I_i(x,y)|) \\ &= \log(\beta) + k\log(r_i). \end{aligned} \tag{39}$$

The system of equations has as many equations as the number of images and calculated errors. We note that the solution $F(x,y)$ cannot always be computed analytically [25]. In the general case, we need an alternative method for determining the error.

5.2 Numerical Errors When The True Solution Is Unknown

In the case where the true solution is unknown *a priori*, using a numerical approximation in a high-precision context (*i.e.* a gold standard solution) to compute a reference image is a valid approach for verification [20]. The main disadvantage of this approach is that it might mask errors which appear in the reference image itself. Our slightly different approach requires neither an analytical solution nor a numerical approximation, but still retains a high sensitivity to errors. Suppose we want to verify the convergence of a sequence of images I_i with $r_{i+1} = cr_i$, where $c \in (0,1)$ is a constant factor. As we have seen in the previous section, the approximation for the solution F at resolution i and $i+1$ can be written respectively as:

$$\begin{aligned} F(x,y) &= I_i(x,y) + O(r_i^k) \\ &= I_i(x,y) + \beta r_i^k + \text{HOT}, \end{aligned} \tag{40}$$

$$\begin{aligned} F(x,y) &= I_{i+1}(x,y) + O(r_{i+1}^k) \\ &= I_{i+1}(x,y) + \beta r_{i+1}^k + \text{HOT}. \end{aligned} \tag{41}$$

Again, we assume that the HOT are negligible. Now, we subtract Equation (41) from Equation (40) to eliminate the unknown F:

$$0 = (I_{i+1}(x,y) + \beta r_{i+1}^k) - (I_i(x,y) + \beta r_i^k) \tag{42}$$

$$0 = I_{i+1}(x,y) - I_i(x,y) + \beta r_{i+1}^k - \beta r_i^k. \tag{43}$$

Thus, the convergence order k can be computed by evaluating the errors involved in the subtraction of consecutive images:

$$e_i(x,y) = I_{i+1}(x,y) - I_i(x,y) = -\beta r_{i+1}^k + \beta r_i^k \tag{44}$$

$$= \beta(1 - c^k) r_i^k. \tag{45}$$

As before, we use the L_∞ norm to compute the maximum error amongst all pixels:

$$\begin{aligned} E_i &= \sup_{x,y}(e_i(x,y)) \\ &= \sup_{x,y}(|I_{i+1}(x,y) - I_i(x,y)|) = \beta(1 - c^k) r_i^k. \end{aligned} \tag{46}$$

Thus the observed convergence is again computed by taking logarithms of both sides. We then write $y = \log \beta(1 - c^k)$ to hide the dependency of the term in k and determine y and k via least-squares:

$$\log E_i = \log \beta(1 - c^k) r_i^k \tag{47}$$

$$= \log \beta(1 - c^k) + k \log r_i \tag{48}$$

$$= y + k \log r_i. \tag{49}$$

In the case of the grid size test, the linear regression measures the constant error due to sources other than the grid size, since no approximation errors with respect to the grid size N are introduced.

Equation (49) shows us how to compute the convergence rate using only the images obtained from the VRI approximation and consequently avoiding any bias and/or limitations introduced by simple manufactured solutions or numerical approximations using reference images. We have generated sequences of images based on the refinements in the following section. The steps are shown in Algorithm 1.

Algorithm 1 A simple algorithm for verification via step size, dataset size or pixel size.

```
VERIFICATION PROCEDURE($G, \tau(s), d_0, N_0, h_0, \rho$)
1  ▷ Let $G$ be the scalar field
2  ▷ Let $\tau(s)$ be a transfer function
3  ▷ Let $d_0$, $N_0 \times N_0 \times N_0$ and $h_0$ be the initial step size,
   dataset size and pixel size respectively
4  ▷ Let $\rho \in \{\text{step}, \text{dataset}, \text{pixel}\}$
5  $F_0 \leftarrow$ VOLUMERENDERING($G, \tau(s), d_0, N_0, h_0$)
6  for $i \leftarrow 1$ to #tests
7      do REFINE($d_i, N_i$, or $h_i$ depending on $\rho$)
8          $F_i \leftarrow$ VOLUMERENDERING($G, \tau(s), d_i, N_i, h_i$)
9          if there is an analytical solution $I$:
10             then $E_i = \max_{x,y} |I(x,y) - F_i(x,y)|$
11             else  $E_i = \max_{x,y} |F_{i-1}(x,y) - F_i(x,y)|$
12 Linear regression of $E_i$ using Equations (39) or (49)
```

6 Application Examples

We present the results of applying our verification framework to two mature and widely-used libraries, namely VTK and Voreen. We stress that the goal here is first to show that our verification technique is very sensitive to changes that cause the output image to deviate from the correct solution; secondly, it is very easy to apply and thus can help developers and practitioners to gain confidence in their implementations.

6.1 Implementations Under Verification

VTK: The VTK library provides several implementations of well-known volume rendering techniques. In our tests we included two modules from version 5.6.1: `vtkVolumeRayCastMapper` (RCM) and `vtkFixedPointVolumeRayCastMapper` (FP). The RCM module accepts as input scalar fields with 8- or 16-bit precision and internal computations are performed with single or double precision. FP accepts input datasets with up to 32 bits of precision but it uses 15-bit fixed-point arithmetic internally. Both techniques use back-to-front compositing. We have also modified the VTK source to capture 15 bit and 32 bit precision images for FP and RCM respectively.

Voreen: As opposed to the tested modules in VTK, Voreen uses the graphics processing unit (GPU) and front-to-back compositing for its implementations. From the ray casting processors available within Voreen, we have chosen the `SingleVolumeRaycaster`, which is the standard processor in most Voreen workspaces. At the time of writing, version 2.6.1 is the latest, and the one we verified. We made minor modifications to the code so that floating point data of the format Nearly Raw Raster Data NRRD [17] could be imported and smaller step sizes could be used.

6.2 System Setup

The grid lies in the domain $[0,2]^3$ for VTK and $[0,1]^3$ for Voreen. The scalar values at grid nodes are chosen from a uniform random distribution. The camera is centered at the xy plane and is aimed along the z axis. We did not include shading since that gives a more complex VRI. To verify shaded results, a different theoretical analysis is necessary. The images can be generated using both perspective and parallel projections. We only use post-classification, which simplifies the analysis. In addition, we assume an identity opacity transfer function (that is, the opacity is exactly equal to the sampled scalar). We do this because for every pair of scalar field and opacity transfer function, there is another scalar field (which admittedly need to be of finer resolution) that, when combined with the identity transfer function, represents the composition arbitrarily well. The function composition arising from volume classification can increase the high-frequency content of a volume [2], and a full treatment of the impact of arbitrary transfer functions on the convergence of the integral remains a topic for future explorations. In addition, this assumption enabled much of the theoretical analysis that would not be possible otherwise, while still being stringent enough to uncover issues in the implementations.

To apply verification via step size refinement, we start with $d_0 = \frac{1}{2}$ and a refinement factor of half, $d_{i+1} = \frac{1}{2}d_i$. We use a dataset of size 2^3 since we have experienced that low resolution grids with random scalar fields are effective at stressing the code for debugging purposes.

Let l be the cell size. For verification via dataset refinement we start with 2^3 grid nodes, and we refine grid cells until we reach 513^3 nodes, corresponding to cell sizes $l_{i+1} = \frac{1}{2}l_i$. Step size is fixed at $d = 10^{-2}$. This is done to evaluate the effects of discretization errors due only to grid refinement.

TABLE 2
Each cell indicates where the results for each type of test can be found in the paper.

		Known sol.	**Unknown sol.**	
		Parallel	*Parallel*	*Perspective*
FP	*step size*	Fig. 8	Fig. 7(a)	Fig. 7(a)
	pixel size	Fig. 7(b)	Fig. 8	Fig. 7(b)
	dataset size	Fig. 8	Fig. 7(c)	Fig. 7(c)
RCM	*step size*	Table 3	Fig. 7(d)	Fig. 7(d)
	pixel size	Fig. 7(e)	Fig. 9(b)	Fig. 7(e)
	dataset size	Fig. 9(a)	Fig. 7(f)	Fig. 7(f)
Voreen	*step size*	N/A	N/A	Fig. 7(g)
	pixel size			Fig. 7(h)
	dataset size			Fig. 7(i)

For verification via pixel size refinement, we start by generating images with 32^2 pixels using the implementation under verification, and then continue to refine pixel size until we reach 1024^2 pixels. The pixel size h is refined according to $h_{i+1} = \frac{1}{2}h_i$. The errors are computed taking the difference between the rendered image and an analytical solution. In this case, we use an analytical solution for the volume rendering integral in the domain $[0,1]^2$. We assume the following: $s(x,y,z) = z\cos(xy)$, $\tau(s) = \sin(s)$, $\mathbf{x}(\lambda) = (x,y,\lambda)$, $C(s) = 1$ and ray length $D = 1$. The analytical solution is then:

$$I(x,y) = 1 - \exp\left(\frac{\cos(\cos(xy))}{\cos(xy)} - \frac{1}{\cos(xy)}\right). \tag{50}$$

The dataset size used is 513^3, and the step size is set at $d = 10^{-5}$ to mitigate sampling errors. Both step and dataset size are fixed to only evaluate errors due to pixel size refinement.

For VTK, we also have the following setup: no auto adjustment of the step size d; single thread; interpolation type is set to linear. For Voreen, we enabled floating point buffers in the pipeline. The Voreen version under verification does not support parallel projection.

The errors are computed using the L_∞ norm and are given by the maximum distance between two images, defined as $e_i = \max_{x,y} |I_i(x,y) - I_{i+1}(x,y)|$, where $I_i(x,y)$ is the pixel with center in (x,y) of the image I_i rendered with the implementation under verification. If a solution F is available, $e_i = \max_{x,y} |I_i(x,y) - F(x,y)|$.

In the following sections we report the results of applying the verification framework with known and unknown exact solutions to three volume rendering implementations. Table 2 indicates where one can find the verification results based on the type of solution (known or unknown), the convergence parameters, and also the projection step used.

6.3 Observed Behavior

The results of our verification procedure are summarized in Figure 7. We tested both VTK and Voreen and found unexpected behaviors. We emphasize that this *does not* immediately translate into a code mistake but only that a deeper investigation is needed. To find the reason for the unexpected behavior we analyzed the source code of the given systems. We expect linear convergence when step size or pixel size are refined ($k = 1$) and constant error when dataset refinement is used.

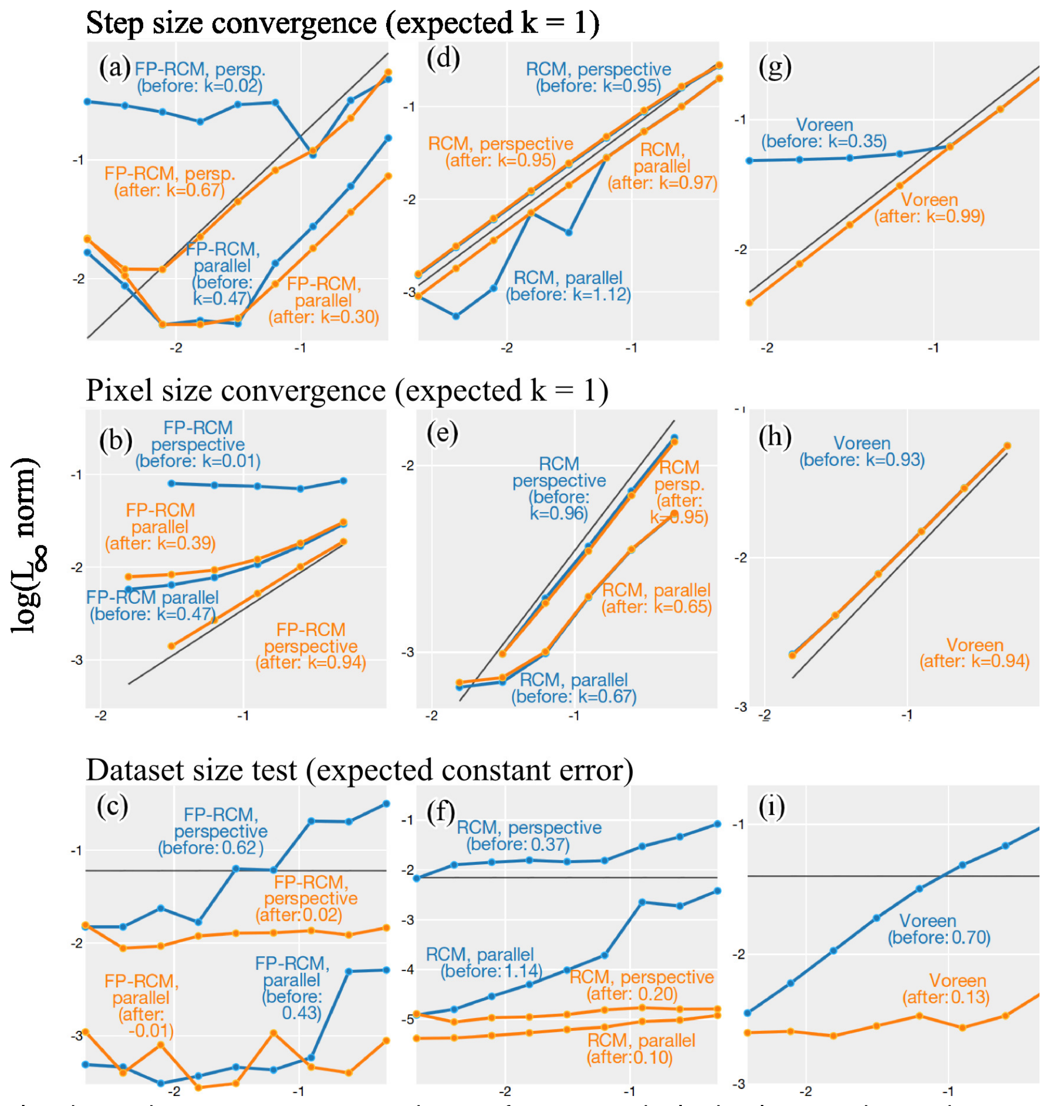


Fig. 7. Each plot shows the convergence experiments for one particular implementation and one particular type of convergence. The behavior before any changes to the source code were made are shown in blue. The results of the changes are shown by the orange lines. The black line indicates the expected *slope* from the theoretical analysis. Notice the black lines indicate only the expected *slope* of the results. Any line parallel to the black indicator line has the same slope and is equally acceptable. The convergence order is denoted by k. Notice also that the dataset refinement test does not introduce errors and thus all that is expected is a constant error.

FP: The results obtained for the FP module (blue curves in Figures 7(a), (b), and (c)) were different from expected for all tests. The 15-bit fixed-point precision could, to some extent, justify this behavior. Still, we only expected this influence to have a negative effect after a certain threshold for step size. The perspective projection curve shown in Figure 7(a), for instance, has what appears to be a constant error when using step size refinement and perspective projection. We expect the error to decrease for large values of d; we acknowledge that when d is too small, the errors due to 15-bit fixed-point precision will dominate. After investigating the reason for this deviation we found that depending on the pixel position, some rays might cross only half of the volume instead of the full volume. In other words, instead of sampling the ray in n locations, for some pixels the ray was only sampled $\frac{n}{2}$ times. This is a combination of several factors which includes domain size, step size, and ray direction. Details can be found in the supplementary material.

Using our synthetic dataset, we observed a '+' pattern shown in Figure 8 (left). The darker regions are precisely the pixels where the ray does not cover the whole domain. Artifacts may also be seen in standard datasets such as the Carp shown in Figure 10. The orange curves in Figures 7(a), (b), and (c) show the convergence results after modifying VTK's source. Notice that for step size refinement using perspective projection, the convergence curve changed from 0.02 to 0.92 for the first seven samples. For the eight and ninth samples the error slightly increases. A similar phenomenon occur in the parallel convergence curve. The curve starts to diverge in the high-resolution regime (parallel and perspective projection plot). This is likely to be due to the limit of 15-bit fixed point arithmetic. Although the pixel size refinement convergence for perspective projection substantially improved (from 0.01 to 0.94), the convergence curves for parallel projection remained similar, which can be explained by the $O(d)$ error.

RCM: The RCM module (blue curves in Figures 7(d), (e), and (f)) produces nearly linearly converging sequences when refining the step size or pixel size. However, dataset refinement with either perspective or parallel projection fails to present the expected constant error. Analyzing the source code, we found the discrepancy to be due to the number of steps taken when marching inside the volume. For instance, suppose that the step size is set in such a way that 200 steps are required to traverse the volume. Instead of 200 steps, the RCM module used values between 195 and 199 steps, depending on some conditions. The consequence of this deviation is shown in Figure 9.

The orange curves in Figures 7(d), (e), and (f) show the convergence results for the RCM module after fixing the issue that prevented code convergence. It consists of changing the epsilon values used during the computation of the number of steps. Notice that the behavior is close to the expected one and the errors are very small (10^{-5}). The convergence curve using pixel size refinement is close to linear for large pixel size but seems to be converging to some positive value. This might be due to other sources of error which become dominant after sufficient refinement.

Voreen: Our first ray refinement tests did not result in linear convergence for Voreen (blue line in Figure 7(g)) due to the early ray termination (ERT). By simply adapting the ERT threshold, we were able to obtain the expected convergence for ray refinement (orange line in the Figure 7(g)).

As can be seen in the Figure 7(i), the blue curve indicates that increasing the resolution of the dataset decreases the error. We remind the reader that using our upsampled data, as described in Section 4.2, rendering the same scalar field represented by a different number of voxels should not affect the result. For Voreen, the unexpected behavior was caused by sampling at incorrect texture locations. More specifically, internally, Voreen assumed that the texture data is node centered when, in fact, OpenGL uses grid centered data. In this case, both the volume and transfer function values were affected. In OpenGL, the texture coordinates of a texture of resolution R^m lie in the domain $[0,1]^m$, where m is the texture dimension. Since the data values are grid centered, this means that the outer most data values are located at $[\frac{1}{2R}, 1-\frac{1}{2R}]$ with the settings used in Voreen. We will refer to the domain in which the data values lie as the data domain. For volume rendering, the integration of a ray should be done over the data domain, but for Voreen, the entry and exit points of the rays went outside of that domain which caused the unexpected behavior. To obtain the expected constant convergence we apply the following transformation to the input texture coordinate p (see orange line in Figure 7(i)):

$$p' = \frac{1}{2R} + p\left(1 - \frac{1}{R}\right), p \in [0\ 1]. \tag{51}$$

Equation (51) scales and translates the texture coordinate to be in the domain $[\frac{1}{2R}, 1-\frac{1}{2R}]^m$, where the data values lie. The effect of transforming the input coordinate for a real world example can be seen in Figure 1. We provide an explanation for why this does not affect ray entry and exit point sampling, and also discuss the implications of different boundary values in the supplementary material. Although the scaling of texture coordinates has been addressed for multiresolution volumes [24], to our knowledge, it has not been applied to the sampling of transfer functions [6], [21], [36]. There are other solutions for recovering the expected convergence rate, which include changing the way we refine our sampling grid to match OpenGL grid centered data. However, we have chosen this solution for several reasons. First, it matches Voreen's initial assumption on node-centered data; it does not require special treatment at the border of the domain; and due to its simplicity, it is easy to implement. We have contacted Voreen developers and the issue found was indeed identified as a bug. The proposed solution will be adopted into Voreen's next release.

No unexpected behavior could be detected for pixel size convergence as shown in Figure 7(h), neither before nor after changing the texture coordinate sampling. Both curves lie near the expected behavior (0.93 and 0.94).

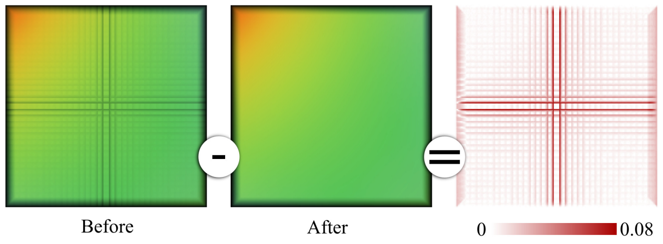
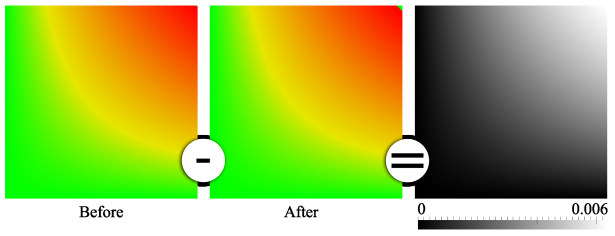


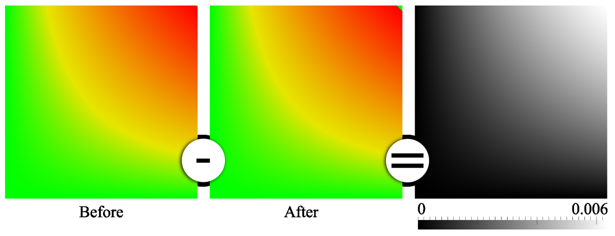
Fig. 8. The figure shows images rendered using VTK 5.6.1. In our experiments, the '+' pattern became more evident in two cases: when coarse datasets are used, and/or high number of sampling points along the ray are used. Darker pixels belong to regions where the ray traverses only half of the volume, preventing convergence. The image on the middle shows the result using our modified version of VTK. The convergence curve improved significantly after the issue was fixed. Note that this effect only occurs when perspective projection is used. For orthogonal projection, the problem is not noticeable. For the convergence analysis, we used a scalar field given by $S(x,y,z) = xyz$, $D = 1$, transfer function $\tau(s) = s$ in the domain $[0,1]^3$, and solution for the VRI given by $I(x,y) = 1 - \exp(-xy/2)$, which means the integration is along z (from zero to one).

7 Discussion

The convergence analysis presented in the previous section helped us to identify unexpected behavior in two stable and widely-used frameworks. Unexpected behavior *is not* indicative of an implementation bug but rather a warning about potential problems. For instance, some valid design decisions might affect the convergence results. Consider the widely used ERT acceleration technique. Depending on the thresholds involved, the convergence results might deviate from the ideal, and the expected curve is recovered once this feature is turned off. In this sense, the verification tool can help the developer to identify



(a) Dataset refinement. Before, a linear trend is observed. After fixing an issue, a constant error is obtained.



(b) Pixel size refinement. Exp.: $k = 1$. Before: $k = 0.37$. After: $k = 1.23$

Fig. 9. The two figures show images rendered before and after fixing an issue with the number of ray samples in the RCM module. This change was motivated by a mismatch in the dataset convergence test. Although the images are indistinguishable to the human eye, the errors (computed as the difference between images, shown on the right) are large enough to change the expected convergence rate. For both images, we applied our verification procedures on a grid with a scalar field given by $S(x,y,z) = xyz$ and transfer function $\tau(s) = s$ in the domain $[0,1]^3$. Hence, the solution for the VRI is $I(x,y) = 1 - \exp(-xy/2)$. (a) uses dataset refinement while (b) uses pixel size refinement.

portions of the code that introduce numerical errors and quantify their effect on the final image. The issue with the RCM module is another example. The dataset size convergence curve was unexpectedly linear because of a small variation in the number of steps. While this particular issue might not be harmful, we were able to learn and reason about its consequences *after* the verification process was done. Furthermore, "minor" bugs and even design decisions cannot be ignored as they can mask more complex mistakes. Therefore, one will be more confident after the design decisions that affect convergence are "turned off" and the expected convergence is recovered. The FP module, on the other hand, significantly deviates from the ideal number of steps required to march inside the volume. Although we could force VTK to march the expected number of steps, we are still investigating possible solutions to and consequences of this issue. To promote an unexpected behavior to a bug, we need interaction with the developers of the code to confirm the code mistake, which was the case with Voreen. One should be aware of the discussed issues when implementing a volume rendering algorithm as their consequences are often not discussed in the literature [6].

7.1 Test Sensitivity

A verification technique ideally should be sensitive to any deviation from the correct implementation. Unfortunately, in practice, verification has limited scope, and we gain confidence if it helps us understand the code behavior, test sensitivity, and reveal bugs. There are several ways to attain this goal: Yang *et al.* applied model checking to filesystem verification and reported *unknown* bugs [46]; Howden [13] evaluated the efficacy of dynamic and static testing for the detection of *known* real bugs of a mathematical library; Knupp and Salari [19], on the other hand, used the order-of-accuracy verification procedure to uncover *known* manufactured bugs in a proof-of-concept code. In software engineering, the process of evaluating a testing suite by injecting defects into a program is known as *mutation testing* [34].

We already presented the results of applying our verification framework to two libraries and with our experiments we confirm the previously reported sensitivity of convergence analysis [38]. We went further to explore other scenarios in volume rendering that may affect the convergence curve. Thus, in the spirit of mutation testing, we created new versions of VTK which contain known issues. Table 3 shows the results of some of the performed tests. In our experiments, we observed that some issues did not affect the observed behavior. The reason for this is that an incomplete set of tests [19] was performed, as shown with test #10 in Table 3. In that case a bug in the G and B color lookups went unnoticed because our framework only used the R channel. Once the verification framework includes all three channels, the convergence behavior does not match the expectations, hence revealing an aberrant behavior that should be investigated. For bug #9, we swapped two of the polynomial coefficients, but they were equal for the scalar field used and thus the bug was not detected. After changing the scalar field to $s(x,y,z) = 1xyz + 2xy + 3xz + \cdots$ the convergence curve no longer matches the expected one, and thus the bug is detected. Bug #11 was introduced in a matrix-vector multiplication routine which turned out to be dead code. However, for bug #12, the loop range was slightly incorrect and it was not detected, even after additional changes to the verification framework.

Aside from the defects injected into VTK, the following is a list of details known to affect the convergence curve: *ERT*, as explained before; *opacity correction*, when using the analytical solution of the volume rendering integral; *hardcoded tolerance constants*, the famous "epsilons"; *off-by-one* indexing problems (sometimes VTK does not render pixels in the first or last column of an image); *improper volume sampling* (cell centered versus grid centered scalar fields); *high-frequency transfer functions*; *high-frequency scalar fields*; *incorrect texture coordinate mapping*, as reported with Voreen; *inconsistent number of steps through the volume*, as reported with FP and RCM; etc. From all the observed situations where the step/dataset/pixel size convergence was affected, many of these are deliberate design decisions, minor mistakes during the verification procedure or minor problems with the implementation itself which can be easily fixed. Note that those issues were not all inside the ray integration routine itself, but in a variety of locations, spanning from pre-processing steps to OpenGL texture sampling of data. Our verification procedure was sensitive enough to detect all these situations.

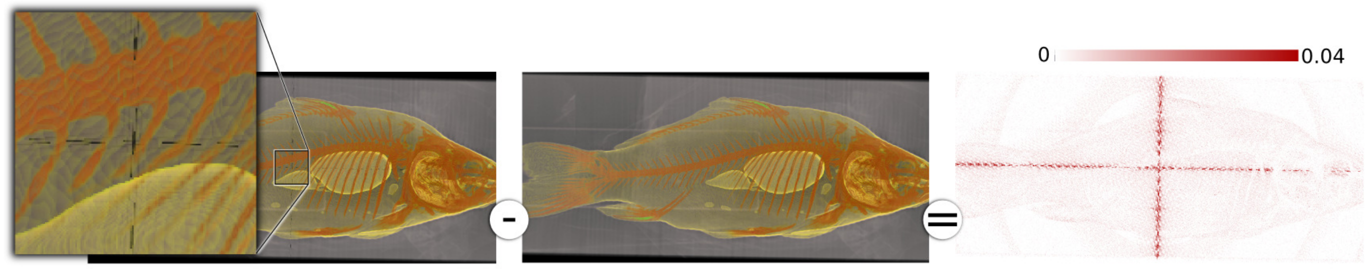
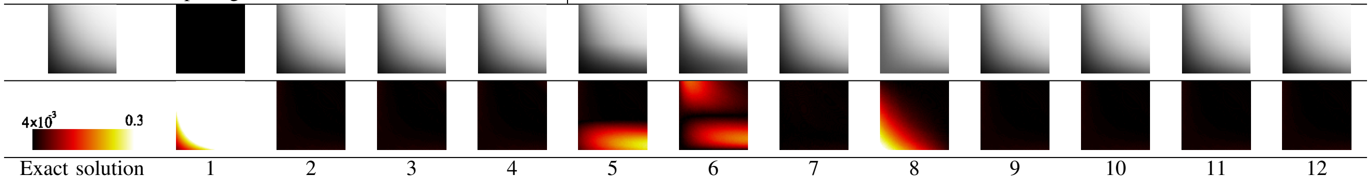


Fig. 10. A CT scan of a carp, rendered with VTK 5.6.1 and Fixed-Point Raycast Mapper (FP). On the left, we see the artifacts (dark lines) that prevented FP convergence. In the middle, we see the results after fixing the issues that prevented convergence. The artifacts are no longer visible. On the right we see the difference image.

TABLE 3
This table shows the sensitivity of convergence verification for different scenarios in a volume renderer. We applied our step size verification using a manufactured solution with a scalar field given by $S(x,y,z) = xyz + xy + xz + yz + x + y + z + 1$ and transfer function $\tau(s)$ varying linearly from zero to one for $s \in [0, \max(S(x,y,z))]$. On the right, we show what part of the volume rendering algorithm was affected by the issue. On the bottom, the first row shows the rendered images for each of the issues. The second row shows the error difference between the exact and rendered solutions. See Section 7.1 for an explanation of the undetected issues.

#	Issue	Detected	Observed behavior ($k =$)
1	Incorrect opacity accumulation	Yes	0.0
2	Incorrect ray increment	Yes	0.0
3	Small changes to early ray termination	Yes	0.1
4	Piecewise constant τ	Yes	0.0
5	Incorrect matrix-point multiplication	Yes	0.0
6	Incorrect evaluation of trilinear interpolant	Yes	0.0
7	Uninitialized pixel center offset	Yes	0.0
8	Incorrect coefficients computation 1	Yes	0.0
9	Incorrect coefficients computation 2	No	1.0
10	Incorrect color lookup	No	1.0
11	Incorrect matrix-viewpoint multiplication	No	1.0
12	Incorrect loop range	No	0.95

```
VOLUME RENDERING
  for each pixel
      do Find pixel center (#7)
         Transform rays to voxels space (#5,#11)
         for each step along the ray (#12)
             do Compute interpolant coefficients (#8,#9)
                Interpolate scalar values (#6)
                Retrieve color and opacity (#4,#10)
                Compositing (#1)
                Increment sample position (#2)
                Check for early ray termination (#3)
```



7.2 Other Volume Rendering Techniques

While we focus on ray casting, our approach can be extended to other techniques. Because the core of our method is a discretization of the VRI, the only requirement is to formulate the volume rendering algorithm as a numerical approximation to the true integral. Splatting [43], for instance, uses a reconstruction kernel before accumulating the contributions of voxels into the final image. This approach is substantially different from ray casting in the way it approximates the VRI, and so the asymptotic errors involved will have to account for errors in both accumulation and filter reconstruction [28].

Algorithmic improvements for volume rendering may require a more careful approach. For example, pre-integration computes the results of the integral with high precision over sample intervals and stores them in a look-up table. This increases efficiency and quality, since fewer steps are typically needed [7]. How the table approximates the integral will affect the convergence rate: if there is an analytical solution then no error is associated with d intervals; otherwise, a numerical approximation scheme might be used which means the error will depend on $d' = d/m$, where m is the number of sample points used in that interval and the integration method used. For example, if a linear approximation is used for the VRI during ray integration (instead of a standard sum of rectangles, as done above), the final approximation should have second order accuracy.

7.3 Manufactured Solutions

In the interest of brevity, verification via pixel size and the results presented in Table 3 were generated from an analytical solution for the volume rendering integral. Notice, still, that the use of an analytical solution for verification is known as the Method of Manufactured Solutions [1] and can be a more rigorous procedure than convergence analysis alone [38].

In this way, we can verify that the results generated by an implementation is converging at the expected rate to the *correct solution*. The disadvantage lies in the difficulty of designing solutions which are simultaneously simple (so that we can write the theoretical convergence analysis down) and yet expressive (so that the experiment analysis catches bugs).

8 Limitations

Both the discretization and verification procedures have limitations. In the discretization of the VRI equation, we assume that the solution $I(x,y)$ is smooth. Moreover, we assume that high-order terms are negligible. This assumption implies that we can safely discard all high-order terms when deriving the errors. In addition, the verification is done in a controlled fashion to avoid other error sources, as shown in Figure 7(a). Additional asymptotic analysis is necessary for each new error source. Also, I must be defined everywhere in the image plane. For instance, this condition is violated if we change the camera position and orientation. One needs to account for these transformation in $\mathbf{x}(\lambda)$, an extra complication in the generation of analytical solutions.

The verification process has the same limitations previously described but it also has practical limitations. For instance, one may be able to observe that the convergence rate may not be the expected one for low sampling rates. However, this is not due to the random scalar field generated (which is a trilinear function and thus can be represented exactly with the trilinear interpolant) but high-frequency details in τ or C. This may lead to a violation of the Nyquist rate. Because the process is iterative, the expected convergence must be recovered once the resolution is fine enough, assuming that the implementation under verification is correct. Another limitation is related to the number of rays used per pixel. Many implementations can shoot several rays per pixel, although this work assumes that only one ray is used. Also, because the verification procedure considers the code as a blackbox, it does not provide clues on the reasons for the unexpected behavior.

The scope of the mistakes that can be found by the verification procedure is not clearly defined. All we can say is that it can find bugs that actively affects the convergence of the method [19]. A common example of bugs that cannot be found by this type of procedure are bugs that affect the *performance*: the code is slower due to the mistake but the convergence rate is still the same [35]. The results shown in Table 3 are a first attempt to understand the scope of problems that can be fixed by the verification procedure.

Currently, our verification procedure is focused on the solution for the VRI without shading and other improvements on the final image quality. Hence, if one wants to use our verification procedure in an implementation that supports, for instance, shading, the feature will need to be deactivated. Lastly, for the case of dataset refinement, we assume that the underlying scalar field is defined by a piecewise-trilinear function.

9 Conclusion and Future Work

In this paper, we present verification techniques for volume rendering based on the use of convergence analysis. Using these techniques, we successfully found discrepancies in the behavior of the volume rendering algorithms of two widely-used visualization packages. We note that we do not see our techniques as a replacement for the currently used direct visual inspection or expert evaluations, but instead as a way to complement those approaches, and lead to a more comprehensive way to evaluate visualization software. By providing attractive quantitative alternatives, we hope to help make evaluation of visualization software both easier and more effective, and also contribute to a higher level of user trust in visual data analysis. We believe the use of verification techniques will be of increasing importance as the field of visualization matures and visualization methods are used in a wide range of commercial and societal areas of highest importance.

There is ample opportunity for future work. Extending our approach to deal with volume shading and level-of-detail techniques would be interesting and relevant research as these are widely used in practice. Another important problem would be to explore the verification of unstructured volume rendering techniques. Lastly, there is room for improving the approximation error for the three presented refinements. In addition, a new way for comparing the convergence curves that allows one to gain insight on the correctness of the implementation under verification is another welcomed step.

Acknowledgments

We thank Dr. Sergey Yakovlev, Dr. Blake Nelson, Stefan Lindholm, and the anonymous reviewers for their helpful comments. This work was in part supported by ARO W911NF-08-1-0517 and W911NF-12-1-0375 (Program Manager Dr. Mike Coyle), CNPq (processes 200498/2010-0, 569239/2008-7, and 491034/2008-3), NSF grants (CNS-0751152, IIS-0844572, CNS-1229185, CNS-1153503, IIS-0905385, IIS-1142013, and AGS-0835821), the Department of Energy, the Excellence Center at Linköping and Lund in Information Technology (ELLIIT), and the Swedish e-Science Research Centre (SERC).

References

[1] I. Babuska and J. Oden. Verification and validation in computational engineering and science: basic concepts. *Computer Methods in Applied Mechanics and Engineering*, 193(36-38):4057–4066, 2004.

[2] S. Bergner, T. Möller, D. Weiskopf, and D. J. Muraki. A spectral analysis of function composition and its implications for sampling in direct volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12:1353–1360, 2006.

[3] E. Clarke. The birth of model checking. In O. Grumberg and H. Veith, editors, *25 Years of Model Checking*, volume 5000 of *Lecture Notes in Computer Science*, pages 1–26. Springer Berlin / Heidelberg, 2008.

[4] J. Duncan and N. Ayache. Medical image analysis: progress over two decades and the challenges ahead. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):85–106, Jan. 2000.

[5] J.-F. El Hajjar, S. Marchesin, J.-M. Dischler, and C. Mongenet. Second order pre-integrated volume rendering. In *PacificVIS '08*, pages 9–16, 2008.

[6] K. Engel, M. Hadwiger, J. M. Kniss, C. Rezk-Salama, and D. Weiskopf. *Real-time Volume Graphics*. A K Peters, 2006.

[7] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Graphics Hardware Workshop*, pages 9–16, 2001.

[8] T. Etiene, L. G. Nonato, C. Scheidegger, J. Tierny, T. J. Peters, V. Pascucci, R. M. Kirby, and C. T. Silva. Topology verification for isosurface extraction. *IEEE Transactions on Visualization and Computer Graphics*, 18(6):952–965, 2012.

[9] T. Etiene, C. Scheidegger, L. G. Nonato, R. M. Kirby, and C. Silva. Verifiable visualization for isosurface extraction. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1227–1234, 2009.

[10] R. W. Floyd. Assigning meaning to programs. In *Proceedings of the Symposium on Applied Mathematics*, volume 19, pages 19–32. AMS, 1967.

[11] A. Globus and S. Uselton. Evaluation of visualization software. *SIGGRAPH Comp. Graph.*, 29(2):41–44, 1995.

[12] P. Godefroid, A. Kiezun, and M. Y. Levin. Grammar-based whitebox fuzzing. *SIGPLAN Not.*, 43(6):206–215, 2008.

[13] W. E. Howden. Applicability of software validation techniques to scientific programs. *ACM Transactions on Programming Languages and Systems*, 2:307–320, July 1980.

[14] IEEE. IEEE standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, page 1, 1990.

[15] C. Johnson. Top scientific visualization research problems. *IEEE Computer Graphics and Applications.*, 24:13–17, July 2004.

[16] C. R. Johnson and A. R. Sanderson. A next step: Visualizing errors and uncertainty. *IEEE Computer Graphics and Applications.*, 23:6–10, September 2003.

[17] G. Kindlmann. http://teem.sourceforge.net. Last accessed on April 10th, 2013.

[18] R. Kirby and C. Silva. The need for verifiable visualization. *IEEE Computer Graphics and Applications*, 28(5):78–83, 2008.

[19] P. Knupp and K. Salari. *Verification of Computer Codes in Computational Science and Engineering*. Chapman and Hall/CRC, 2002.

[20] J. Kronander, J. Unger, T. Möller, and A. Ynnerman. Estimation and modeling of actual numerical errors in volume rendering. *Computer Graphics Forum*, 29(3):893–902, 2010.

[21] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *IEEE Visualization 2003*, pages 38–43, 2003.

[22] H. Kye, B. Shin, and Y. Shin. Interactive classification for pre-integrated volume rendering of high-precision volume data. *Graphical Models*, 70(6):125–132, 2008.

[23] J. Lee and T. Newman. New method for opacity correction in oversampled volume ray casting. *Journal of WSCG*, 15:1–8, 2007.

[24] P. Ljung, C. Lundström, and A. Ynnerman. Multiresolution interblock interpolation in direct volume rendering. In *EuroVis – The Eurographics Conference on Visualization*, pages 259–266, 2006.

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

[26] M. Meissner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis. A practical evaluation of popular volume rendering algorithms. In *IEEE Volume Visualization*, pages 81–90, 2000.

[27] J. Meyer-Spradow, T. Ropinski, J. Mensmann, and K. H. Hinrichs. Voreen: A rapid-prototyping environment for ray-casting-based volume visualizations. *IEEE Computer Graphics and Applications.*, 29(6):6–13, Nov./Dec. 2009.

[28] T. Möller, R. Machiraju, K. Mueller, and R. Yagel. Classification and local error estimation of interpolation and derivative filters for volume rendering. In *IEEE Volume Visualization*, pages 71–78, 1996.

[29] T. Möller, R. Machiraju, K. Mueller, and R. Yagel. Evaluation and design of filters using a Taylor series expansion. *IEEE Transactions on Visualization and Computer Graphics*, 3:184–199, April 1997.

[30] K. Moreland and E. Angel. A fast high accuracy volume renderer for unstructured data. In *IEEE Volume Visualization and Graphics*, pages 9–16, 2004.

[31] K. Novins and J. Arvo. Controlled precision volume integration. In *ACM Volume Visualization*, pages 83–89, 1992.

[32] A. Pommert and K. H. Höhne. Evaluation of image quality in medical volume visualization: The state of the art. In *Proceedings of the 5th International Conference on Medical Image Computing and Computer-Assisted Intervention-Part II*, MICCAI '02, pages 598–605, London, UK, 2002. Springer-Verlag.

[33] A. Pommert and K. H. Höhne. Validation of medical volume visualization: a literature review. *International Congress Series*, 1256:571–576, 2003. CARS 2003. Computer Assisted Radiology and Surgery. Proceedings of the 17th International Congress and Exhibition.

[34] T. Riley, A. Goucher, R. Tim, and G. Adam. *Beautiful Testing: Leading Professionals Reveal How They Improve Software*. O'Reilly Media, Inc., 1st edition, 2009.

[35] P. J. Roache. *Verification and Validation in Computational Science and Engineering*. Hermosa Publishers, 1998.

[36] S. Röttger, S. Guthe, D. Weiskopf, T. Ertl, and W. Strasser. Smart hardware-accelerated volume rendering. In *VISSYM – Joint Eurographics - IEEE TCVG Symposium on Visualization*, pages 231–238, 2003.

[37] S. Röttger, M. Kraus, and T. Ertl. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *IEEE Visualization*, pages 109–116, 2000.

[38] C. J. Roy. Review of code and solution verification procedures for computational simulation. *Journal of Computational Physics*, 205(1):131–156, 2005.

[39] W. Schroeder, K. Martin, and W. Lorensen. *Visualization Toolkit, An Object-Oriented Approach to 3D Graphics - 2nd ed.* Prentice-Hall, 1998.

[40] M. Smelyanskiy, D. Holmes, J. Chhugani, A. Larson, D. M. Carmean, D. Hanson, P. Dubey, K. Augustine, D. Kim, A. Kyker, V. W. Lee, A. D. Nguyen, L. Seiler, and R. Robb. Mapping high-fidelity volume rendering for medical imaging to CPU, GPU and many-core architectures. *IEEE Transactions on Visualization and Computer Graphics*, 15:1563–1570, November 2009.

[41] M. Unser. Splines: a perfect fit for signal and image processing. *IEEE Signal Processing Magazine*, 16(6):22 –38, nov 1999.

[42] A. Usman, T. Möller, and L. Condat. Gradient estimation revitalized. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1495–1504, 2010.

[43] L. Westover. Interactive volume rendering. In *ACM Volume Visualization*, pages 9–16, 1989.

[44] P. L. Williams and N. Max. A volume density optical model. In *ACM Volume Visualization*, pages 61–68, 1992.

[45] P. L. Williams, N. L. Max, and C. M. Stein. A high accuracy volume renderer for unstructured data. *IEEE Transactions on Visualization and Computer Graphics*, 4:37–54, January 1998.

[46] J. Yang, P. Twohey, D. Engler, and M. Musuvathi. Using model checking to find serious file system errors. *ACM Transactions on Computer Systems*, 24:393–423, November 2006.

[47] Z. Zheng, W. Xu, and K. Mueller. VDVR: Verifiable volume visualization of projection-based data. *IEEE Transactions on Visualization and Computer Graphics*, 65(6):1515–1524, 2010.

Tiago Etiene received both the B.S. and M.S. degrees in computer science from Universidade de São Paulo, Brazil. Currently, he is a Ph.D. student at Scientific Computing and Imaging Institute, University of Utah. His research interests include scientific visualization and computer graphics.

Daniel Jönsson received the MSc degree in media technology in 2009 from Linköping University, Sweden. Since 2010, he is working toward the PhD degree in scientific visualization at the division for Media and Information Technology (MIT) at the Department of Science and Technology (ITN), Linköping University. From 2009 to 2010, he worked as a research engineer at MIT. His research interests include volume rendering, computer graphics, and scalable rendering.

Timo Ropinski is professor in interactive visualization at Linköping University, Sweden, where he is heading the scientific visualization group. His research is focused on volume rendering, biomedical visualization and interactive visual decision making. He is a member of the winning team of the IEEE Visualization contest 2010, initiator of the Voreen software project (www.voreen.org), and he has held tutorials at Eurographics, ACM SIGGRAPH and IEEE Visualization. Furthermore, he is a member of the IEEE Computer Society, ACM, and Eurographics.

Carlos Scheidegger is a researcher at AT&T Labs – Research. He has a Ph.D. in Computing from the University of Utah. Carlos has won best paper awards at IEEE Visualization in 2007, and Shape Modeling International in 2008. His research interests include data visualization and analysis, geometry processing and computer graphics.

João L. D. Comba received a PhD in Computer Science from Stanford University under the supervision of Leonidas J. Guibas, an MSc degree in Computer Science from the Federal University of Rio de Janeiro (UFRJ), Brazil, and a BSc in Computer Science was given by the Federal University of Rio Grande do Sul, Brazil. Dr Comba is currently an Associate Professor in the Graphics Group at the "Instituto de Informática" of the Federal University at Rio Grande do Sul (UFRGS), Brazil. His main research interests are in visualization, computer graphics, spatial data structures, graphics hardware and HPC.

Luis Gustavo Nonato is associate professor at the Instituto de Ciências Matemáticas e de Computação (ICMC) - Universidade de São Paulo (USP) - Brazil. He received the PhD degree in Applied Mathematics from the Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro - Brazil, in 1998. He spent a sabbatical leave in the Scientific Computing and Imaging Institute at the University of Utah from 2008 to 2010. Besides having served in several program committees he is currently in the editorial board of Computer Graphics Forum, he is the president of the Special Committee on Computer Graphics and Image Processing of Brazilian Computer Society and leads the Visual and Geometry Processing Group at ICMC-USP. His research interests include visualization and geometry processing.

Robert M. Kirby (M'04) received the M.S. degree in applied mathematics, the M.S. degree in computer science, and the Ph.D. degree in applied mathematics from Brown University, Providence, RI, in 1999, 2001, and 2002, respectively. He is currently an Associate Professor of computer science with the School of Computing, University of Utah, Salt Lake City, where he is also an Adjunct Associate Professor in the Departments of Bioengineering and Mathematics and a member of the Scientific Computing and Imaging Institute. His current research interests include scientific computing and visualization.

Anders Ynnerman received the BSc and PhD degrees in physics, respectively, from the University of Lund in 1986 and Gothenburg University in 1992. He has directed the national supercomputing organizations NSC and SNIC. Since 1999, he has been a professor of scientific visualization at Linköping University. He is the founder and director of the Norrköping Visualization Center – C. His current primary research interests include the area of visualization of large datasets and multimodal interaction techniques. He is a member of the IEEE, the IEEE Computer Society, Eurographics, and the ACM.

Claudio T. Silva is Head of Disciplines of the newly established NYU CUSP (Center for Urban Science and Progress), Professor of Computer Science and Engineering at NYU Poly, affiliate faculty at NYU Courant, and a visiting professor at Linköping University in Sweden. From 2003 to 2011, he was with the School of Computing and the Scientific Computing and Imaging Institute at the University of Utah. He has also held research positions at industrial and government labs including AT&T Labs-Research, IBM Research, Sandia National Laboratory and Lawrence Livermore National Laboratory. He received the BS degree in mathematics from the Federal University of Ceara, Brazil, in 1990, and the PhD degree in computer science from the State University of New York at Stony Brook in 1996. He coauthored more than 180 technical papers and ten U.S. patents, primarily in visualization, geometry processing, computer graphics, scientific data management, HPC, and related areas.