

GPU-Based Volume Visualization from High-Order Finite Element Fields

Blake Nelson, Robert M. Kirby, *Member, IEEE*, and Robert Haimes

Abstract—This paper describes a new volume rendering system for *spectral/hp* finite-element methods that has as its goal to be both accurate and interactive. Even though high-order finite element methods are commonly used by scientists and engineers, there are few visualization methods designed to display this data directly. Consequently, visualizations of high-order data are generally created by first sampling the high-order field onto a regular grid and then generating the visualization via traditional methods based on linear interpolation. This approach, however, introduces error into the visualization pipeline and requires the user to balance image quality, interactivity, and resource consumption. We first show that evaluation of the volume rendering integral, when applied to the composition of piecewise-smooth transfer functions with the high-order scalar field, typically exhibits second-order convergence for a wide range of high-order quadrature schemes, and has worst case first-order convergence. This result provides bounds on the ability to achieve high-order convergence to the volume rendering integral. We then develop an algorithm for optimized evaluation of the volume rendering integral, based on the categorization of each ray according to the local behavior of the field and transfer function. We demonstrate the effectiveness of our system by running performance benchmarks on several high-order fluid-flow simulations.

Index Terms—Volume visualization, high-order finite element methods, *spectral/hp* elements, GPU ray-tracing

1 INTRODUCTION

HIGH-ORDER finite element methods (a variant of which are the *spectral/hp* element methods considered in this work [1]) have reached a level of sophistication that they are now commonly applied to many real-life engineering problems, such as those found in fluid mechanics, solid mechanics, and electromagnetics [2], [3]. An attractive feature of these methods is that convergence can be obtained by reducing the mesh size (h adaptivity), increasing the polynomial order within an element (p adaptivity), or using a combination of both approaches. This leads to meshes that, while consisting of fewer and larger elements, have levels of accuracy that are comparable to more traditional finite element methods based on low-order elemental expansions.

The direct visualization of the high-order fields produced by *spectral/hp* finite element methods using linear primitives (e.g., those used by existing volume rendering packages such as Voreen [4], ImageVis3D [5], and VTK [6]) is not possible since these primitives are unable to directly represent the high-order field. To use these systems, engineers must first create compatible linear approximations of the high-order data. Since linear approximations do not, in general, faithfully represent high-order data, this

approximation step introduces error into the visualization pipeline. This error can be reduced by creating linear approximations with smaller spacing between samples, but this comes at the expense of increased computation time to sample the high-order data and increased memory usage to store the resulting data set. In practice, it is difficult to eliminate the error introduced by linear approximations through the use of increased sampling.

In this work, we propose a new direct volume rendering method that uses the properties of *spectral/hp* finite element fields with the goal of producing images that are both *accurate* and *interactive*. We obtain accuracy by using the properties of *spectral/hp* elements to categorize each ray based upon the properties of the transfer function composed with the scalar field. We prove that, in the optimal case of fields that are smooth along the ray, the evaluation of the volume rendering integral will generally exhibit second-order convergence, with a worst case of first-order convergence. We use these properties to develop an optimized high-order volume rendering algorithm, in which we first categorize each ray based on the local properties of the transfer function and scalar field, then evaluate the volume-rendering integral with a quadrature method optimized for the category. While our primary motivation is the desire to generate *pixel-exact* images, which we define as an image which does not change with additional refinement of the volume rendering integral, performance is also an important consideration for a usable system. We have, therefore, implemented our system on the GPU, using a combination of NVIDIA's OptiX [7] ray-tracing framework and Cuda.

- B. Nelson is with Space Dynamics Laboratory, Utah State University Research Foundation, Logan, UT 84341. E-mail: Blake.Nelson@sdl.usu.edu.
- R.M. Kirby is with the School of Computing, Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT 84112. E-mail: kirby@cs.utah.edu.
- R. Haimes is with the Department of Aeronautics and Astronautics, MIT, Cambridge, MA 02139. E-mail: haimes@mit.edu.

Manuscript received 15 Aug. 2012; revised 10 Jan. 2013; accepted 17 June 2013; published online 1 July 2013.

Recommended for acceptance by P. Cignoni.

For information on obtaining reprints of this article, please send e-mail to: tcvg@computer.org, and reference IEEECS Log Number TVCG-2012-08-0160. Digital Object Identifier no. 10.1109/TVCG.2013.96.

2 RELATED WORK

While direct volume rendering of low-order data is treated extensively in the literature [8], [10], there are relatively few

methods designed specifically for high-order data. In this section, we provide a brief overview of existing high-order visualization methods.

Since the solution to the volume rendering equation does not, in general, have an analytic solution, it is not sufficient to use the high-order data directly to guarantee pixel-exact images. This is because the volume rendering integral will be approximated with numerical quadrature, which will introduce error into the visualization. An exception to this is when rendering linear tetrahedra [11], for which the volume rendering integral can be solved analytically. While highly accurate results can be obtained for quadratic tetrahedra [12] and hexahedra [11], other element types and polynomial orders cannot be solved analytically and require numerical quadrature. More recently, direct volume rendering of arbitrary elements and polynomial order was introduced [13], where GPU clusters are able to produce interactive results. In the special case of volume rendering fields with no transfer function and no emissive component, high-order quadrature techniques can be used to evaluate the attenuation portion of the volume rendering integral without error, resulting in pixel-exact images [14]. Another approach uses point-based samples to approximate the volume rendering integral [15], but sacrifices accuracy for execution speed.

Higher order integration methods have been investigated in an attempt to achieve faster convergence of the volume rendering integral and to provide better error bounds on the resulting calculation. Conservative bounds on the number of samples required for a given level of accuracy have been developed for trapezoidal and Simpson’s rule, as well as for a power series approach [16]. An investigation of Simpson’s rule applied to voxel-based data [17] showed improved accuracy at the expense of increased memory usage in the preintegration table. Our system is designed for high-order (spectral/hp) finite elements, which range in polynomial order from degree one to degree 14 (approximately). Assuming a preintegration sampling of 256 samples for each degree of freedom, this would result in tables as large as 256^{15} , which, while is theoretically possible, is not practical at this time. For lower order data and integration methods, however, preintegration requires modestly sized tables, and is, therefore, a popular choice for interactive volume rendering.

In addition to volume rendering, other types of visualization techniques have also been created directly for high-order data. These include cut-surfaces and contour line generation [18], [19] and the generation of isosurfaces using ray-tracing [20], particles [21], and combined particle/ray-tracing [22].

To avoid the computational and storage penalties associated with the straightforward linear tessellations mentioned in the introduction, several adaptive subdivision schemes have been developed [23], [24] to reduce the number of linear primitives required. Subdivision is guided through the use of error estimates, which results in few subdivisions for elements well represented by linear approximation, and many subdivisions for elements that are not. Adaptive subdivision does not eliminate the error introduced into the simulation pipeline, and reducing the error to arbitrary levels can still produce unacceptably large data sets and long execution times.

3 BACKGROUND AND OVERVIEW

Our method is designed for the direct volume rendering of spectral/hp finite element fields that are produced by the continuous Galerkin (CG) or discontinuous Galerkin (DG) methods. For these methods, the domain Ω over which the partial differential equations of interest are being solved is tessellated. This tessellation frequently consists of hexahedra, tetrahedra, prisms, and pyramids.

Each element is associated with a reference element on which the high-order scalar field is defined. A mapping function $\mathbf{x} = \Phi(\eta)$ is defined for each element that maps points from a “reference” space (η) to world space (\mathbf{x}). In our work, the final “reference” space is the tensor (Cartesian) space as used in [1] and discussed in the next section. Since we are interested in evaluating the scalar field $f(\eta)$ defined over the reference space at arbitrary points in world space, we must find the element containing \mathbf{x} and invert the mapping before evaluating the high-order FEM function $f(\Phi^{-1}(\mathbf{x}))$. While this mapping does exist for each element, it is not, in general, a linear transformation. Consequently, the field along the ray may not be a polynomial (and, in general, is not). This field is, however, guaranteed to be smooth along the interior of an element.

We use the *emission-absorption* optical model [25] for direct volume rendering, in which the irradiance along a ray segment is given by

$$I(a, b) = \int_a^b \kappa(f(t)) \tau(f(t)) e^{-\int_a^t \tau(f(u)) du} dt, \quad (1)$$

where a and b are the segment endpoints, κ and τ are the user-defined emissive (color) and density transfer functions, respectively, and $f(t)$ is the scalar field at a point t along the segment. We refer to the integral $-\int_a^t \tau(f(u)) du$ as the *inner* integral and I as the *outer* integral.

While there are no theoretical constraints on a transfer function’s form, we assume that transfer functions are piecewise-linear functions. While transfer functions using different bases exist (e.g., Gaussian basis functions [26]), piecewise-linear transfer functions are consistent with the transfer function interfaces provided by many existing volume rendering systems [27], [4], [5]. We limit our attention to the case of 1D transfer functions.

Piecewise-linear transfer functions are specified in terms of a finite number of points $(c_0, T_0), (c_1, T_1), \dots, (c_n, T_n)$, where c_i is called a *breakpoint* and T_i is the corresponding value of the transfer function at that point. Breakpoints correspond to the transitions between linear pieces in the transfer function. While the transfer function is continuous at the breakpoints, the derivative is undefined at these points.

Because the transfer function has breakpoints where the derivative is undefined, so does the composition of the transfer function with the scalar field along the ray, $\kappa(f(t))$ and $\tau(f(t))$. Given a smooth function f , the result of composing the transfer function with f is a piecewise-smooth function which is only C^0 at a finite number of breakpoints. This is applicable to both CG and DG methods because, as will be discussed in Section 5, we evaluate the volume rendering integral on a per-element basis.

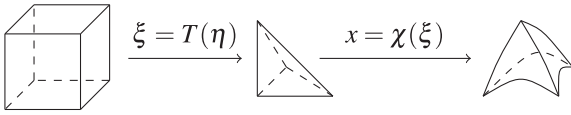


Fig. 1. Illustration of the mapping between tensor (left), reference (middle), and world space (right) for a tetrahedron. Tensor points are denoted by η , reference points by ξ , and world space points by x .

Effective evaluation of the volume rendering integral requires proper accounting of the breakpoints. Ideally, if the breakpoints can be found [11], we could then evaluate the volume rendering integral as follows (with t_i being the i_{th} breakpoint):

$$I = \sum_{i=0}^n \int_{t_i}^{t_{i+1}} \kappa(f(t)) \tau(f(t)) e^{-\sum_{j=0}^{i-1} \int_{t_j}^{t_{j+1}} \tau(f(u)) du - \int_{t_i}^t \tau(f(u)) du} dt, \quad (2)$$

where the integral could then be evaluated using high-order methods such as Gaussian quadrature [28]. While this approach is conceptually appealing, the high-order nature of the field makes it inefficient in practice. Finding n breakpoints along the ray is equivalent to finding the isosurfaces associated with each breakpoint. While this can be done, it is too computationally expensive for our system's interactivity requirements [20].

Since we do not know the location of each breakpoint, we cannot assume that the use of high-order quadrature methods will automatically lead to high-order convergence. The convergence properties of high-order quadrature methods assume smooth functions, which is an assumption that is violated by the breakpoints.

4 HIGH-ORDER FINITE ELEMENTS

A finite element volume is represented by the decomposition of a domain Ω into a mesh of n smaller, nonoverlapping elements Ω_i such that $\Omega = \bigcup_{i \leq n} \Omega_i$. The four basic element types that are used in this work are the hexahedron, prism, tetrahedron, and pyramid. Each element is associated with three different spaces: world, reference, and tensor. The world space represents the element in its physical position and orientation. In reference space, each element is transformed to a common, element-specific representation. The reference space element is related to the world space element through the bijective mapping $\chi: \mathbb{R}^3 \rightarrow \mathbb{R}^3$. The tensor space element is the cube $[-1, 1]^3$ for all elements, and has a mapping to the reference element $T: \mathbb{R}^3 \rightarrow \mathbb{R}^3$. The mapping between tensor space and world space, which is a bijection *a.e.*, is given by $\Phi = \chi(T(\eta))$. Given that the composition of the mappings is a bijection *a.e.* allows us to convert points between spaces as needed during visualization. An diagrammatic example of these mappings for a tetrahedron is shown in Fig. 1.

For the remainder of this paper, we will use the following notation to help distinguish between points in tensor space and points in world space: points in the world Cartesian space will be indicated by x , with individual directions will be specified by x_1, x_2, x_3 . Similarly, in the local Cartesian space associated with the element's tensor representation, points will be specified with η , and individual components as η_1, η_2, η_3 .

The solution to a high-order finite element simulation is a polynomial function $F(\xi) \in \mathcal{P}^{N_1, N_2, N_3}$ with respect to the tensor element, where N_1, N_2, N_3 denote (possibly) different polynomial orders in the three principle directions. Since Φ is a bijection *a.e.* (with special care being needed only at collapsed vertices), we can invert it to calculate the tensor point for a given world point, and then use the tensor point to calculate the field value. While Φ^{-1} exists for each element, it is, in general, not known analytically and must be calculated numerically when needed. It is also usually a nonlinear transformation that, when applied to the polynomial field in tensor space, produces a function in world space that, while smooth, is not necessarily a polynomial.

This work deals with continuous Galerkin (CG) formulations of the finite element method, which require that the fields are continuous across element boundaries, but impose no restrictions on the continuity of any derivatives. Therefore, while the field is C^p on the interior of each element (with $p \geq 1$ indicating higher levels of smoothness depending on the element's approximating polynomial order and its mapping to world space), the field over the domain Ω is C^0 .

5 HIGH-ORDER VOLUME RENDERING

We begin our discussion of our high-order volume rendering system by providing an overview of the resulting algorithm (see Algorithm 1), followed by our analysis of the volume rendering integral for piecewise-smooth functions that has motivated it.

ALGORITHM 1: High-Order Volume Rendering Algorithm

Input: A ray $R(t)$, color transfer function κ , density transfer function τ , a list of all transfer function breakpoints s , and a list of all elements E traversed by the ray, ordered by intersection distance.

```

1 foreach Element  $E_i \in E$  do
2   Determine ray entrance  $t_a$  and exit  $t_b$  for element  $E_i$ .
3   Evaluate the field on the ray segment  $[t_a, t_b]$  using interval
   arithmetic to obtain field bounds  $[f_{min}, f_{max}]$  and transfer
   function bounds  $[\kappa_{min}, \kappa_{max}]$ ,  $[\tau_{min}, \tau_{max}]$ .
4   Classify each transfer function into one of three categories
   based on these bounds: empty (E), piecewise-smooth (PS), or
   smooth (S).
5   Integrate each segment based on the classification (details are
   given in Section 5.2).
6   if  $\tau \in E$  (empty space) then
7     Skip this ray segment.
8   else if  $\kappa \in E \wedge \tau \in S$  (non-emitting occlusion) then
9     Integrate  $\tau$  using Gauss-Kronrod quadrature (Section
     5.1.3), with the number of sample points chosen based on
     the order of  $E_i$ .
10  else if  $\kappa \in E \wedge \tau \in PS$  then
11    Integrate  $\tau$  using trapezoidal rule (see Section 5.1.1).
12  else if  $\kappa \in PS \wedge (\tau \in PS \vee \tau \in S)$  then
13    Evaluate inner and outer integrals using trapezoidal rule
    (see Section 5.2.3).
14  else
15    Evaluate inner integral using trapezoidal rule and outer
    integral using Gaussian quadrature (see Section 5.2.3).
16  end
17 end

```

Steps 1 and 2 represent traversal of the finite element grid and is standard for ray-tracing methods for finite element meshes [29], [12]. Since the element geometry

(represented by Φ) can be nonlinear in high-order fields, rays can enter and exit the same element multiple times.

The contribution of this work begins on Step 4. In this step, we classify the transfer functions over $[t_a, t_b]$ into one of three categories based on the structure of the field over the segment: empty space segments (**E**), where the transfer function is zero along the entire segment; piecewise-smooth segments (**PS**), where the segment contains one or more breakpoints; and smooth segments (**S**), where the segments contain no breakpoints. We discuss how we use these categories to choose an appropriate quadrature method for the volume rendering integral in Section 5.2.

Categorization of a ray segment requires the range of the scalar field along the ray, from which we can determine if the transfer function is zero along the entire segment or if it contains any breakpoints. One approach we can use is to calculate the global min and max using standard optimization techniques. While this approach will work, and will produce accurate segment categorization, it is a computationally expensive operation that does not facilitate interactivity. Instead, we generate fast and conservative estimates of the range through the use of interval arithmetic [30]. The implication of this choice is that we may perform more work than needed during integration, but save the time computing the exact range of the function. As we discuss in further detail below, this is a reasonable tradeoff for current GPU architectures.

Using interval arithmetic, we replace operations on real numbers with operations on intervals. An interval X is defined as

$$X = [\underline{X}, \overline{X}] = \{x \in \mathbb{R} : \underline{X} \leq x \leq \overline{X}\}. \quad (3)$$

The *set image* of a function g is defined as

$$g(X) = \{g(x) : x \in X\}, \quad (4)$$

and represents the true range of g . Since g is often not easily computable, we use the *interval extension* G of g to represent the range. G is formed by evaluating g with interval numbers as arguments rather than floating point numbers [30], and has the following useful property:

$$G(X) \supseteq g(X) = \{g(x) : (x) \in X\}. \quad (5)$$

In other words, if g is evaluated using interval arithmetic on an input interval X , the result is an interval extension that is guaranteed to contain the true range of g on that interval.

We now discuss how we use interval extensions to categorize rays. Let $f(t)$ be the field along a ray segment on the interval $X = [t_a, t_b]$. We first construct the interval extension $F(X)$ (giving us an estimate of f 's true range) using interval arithmetic as described above. Let $c_i, i \leq n$ be the n breakpoints associated with the density transfer function τ . We categorize τ as follows:

$$C(\tau) = \begin{cases} E, & \text{if } F(X) = [0, 0], \\ PS, & \text{if } \exists_i : c_i \in F(X), \\ S, & \text{if } \forall_i : c_i \notin F(X). \end{cases} \quad (6)$$

The categorization for each of the color channels proceeds in an analogous fashion. It is important to note that this categorization is not precise, meaning that a ray can be

categorized as a piecewise-smooth segment when it is actually smooth. This is because the true range is a subset of the estimated range, so a breakpoint can be in the estimated range while falling outside the actual range. The implication of this is that we may handle some rays suboptimally by using the rules developed below for piecewise-smooth functions, rather than the more optimal approach for smooth functions. In this context, suboptimal means that, by classifying a smooth function as piecewise-smooth, we will use a lower order quadrature scheme and, therefore, will require more samples to achieve a given level of accuracy. If tighter bounds on the range estimate are desired, the ray segment can be broken into subsegments to produce a tighter bound [18].

5.1 Integration Techniques

In this section, we provide the motivation for the integration methods we use to evaluate the volume rendering integral. As we mentioned in Section 3, the field along the ray is a smooth function, and seems to be a perfect candidate for the use of high-order quadrature rules. However, as we show below, the existence of breakpoints along the ray, which are introduced through the composition of the field with the transfer function, limits us to linear or quadratic convergence, even when using high-order quadrature rules such as Gaussian quadrature.

As indicated in Section 3, one way to address the presence of breakpoints along the ray is to use a root-finding procedure to find the location of each breakpoint, then to use high-order quadrature on the smooth segments between breakpoints. While this approach is appealing in theory, it is not useful in practice due to both the number of breakpoints found in typical transfer functions (increasing the number of root-finding calls needed) and the high-order field along the ray (increasing the time and complexity of the root-finding routine). Therefore, in the sections that follow, we focus our attention on the case where the location of each breakpoint is unknown, and only coincides with the location of a sample point through chance.

5.1.1 Quadrature of Piecewise-Smooth Segments

When the transfer function along a ray segment is piecewise-smooth (i.e., the segment contains at least one breakpoint), we cannot use high-order quadrature routines and expect high-order convergence. This is because the convergence analysis for these methods assumes continuity in the function's derivatives. As we noted in Section 3, we could evaluate these types of ray segments by locating the world space position of each breakpoint, then applying high-order quadrature to the smooth segments between breakpoints. This approach is not feasible due to the computational requirements of finding the location of the breakpoints. We now discuss the convergence behavior of an arbitrary n -point quadrature rule when used to estimate the integral of a piecewise-smooth function. We note that we use as our guide the classic quadrature book by Davis and Rabinowitz [28] and employ Big-Oh Notation from the perspective of "asymptotics of the infinitesimal" as used in mathematical real analysis. Formal descriptions and discussion of usage can be found in [31].

Let $f(t)$ be piecewise-smooth in $[a, b]$ with a single breakpoint at $c \in (a, b)$:

$$f(t) = \begin{cases} e(t), & t \leq c, \\ g(t), & t > c, \end{cases} \quad (7)$$

where $e(c) = g(c)$, $e'(c) \neq g'(c)$, $e \in C^\infty[a, b]$, and $g \in C^\infty[a, b]$. We consider n -point quadrature methods of the form

$$\int_a^b f(t) dt \approx \sum_{i=1}^n w_i f(t_i), \quad (8)$$

where $a \leq t_i \leq b$, and $\sum_{i=1}^n w_i = b - a$. The abscissas can be evenly spaced (corresponding to Newton-Cotes quadrature) or nonequally spaced (such as the case of Gaussian quadrature). The error between the integral's true value and this approximation is given by

$$E = \int_a^b f(t) dt - \sum_{i=1}^n w_i f(t_i). \quad (9)$$

To determine the utility of high-order quadrature for piecewise-smooth functions, we wish to determine how this error behaves as the interval $h = b - a$ becomes smaller. We proceed in a manner similar to that described in Davis and Rabinowitz ([28, Eqs. (4.3.8)-(4.3.14)]) by rewriting each term in (9) using Taylor's theorem with exact remainder, expanding e and g around the breakpoint c , which gives

$$f(t) = \begin{cases} e(t) = T_e(t) = e(c) + e'(c)(t-c) \\ \quad + \int_c^t e''(x)(t-x) dx & t \leq c, \\ g(t) = T_g(t) = g(c) + g'(c)(t-c) \\ \quad + \int_c^t g''(x)(t-x) dx & t > c. \end{cases} \quad (10)$$

Using these series to rewrite the integral gives

$$\begin{aligned} \int_a^b f(t) dt &= \int_a^c T_e(t) dt + \int_c^b T_g(t) dt \\ &= (c-a)e(c) - \frac{(a-c)^2}{2} e'(c) \\ &\quad + \int_a^c \int_c^t e''(x)(t-x) dx dt \\ &\quad + (b-c)g(c) + \frac{(b-c)^2}{2} g'(c) \\ &\quad + \int_c^b \int_c^t g''(x)(t-x) dx dt. \end{aligned} \quad (11)$$

The remainders take the form of an iterated integral

$$\int_a^c \int_c^t e''(x)(t-x) dx dt, \quad (12)$$

where the limits of integration are bounded. Since the field along the ray is a smooth function (see Section 3) with finite curvature, the integrand is bounded. We can, therefore, use Fubini's theorem [32] to swap the order of integration yielding

$$\int_a^c \int_c^t e''(x)(t-x) dx dt = \int_a^c \int_x^a e''(x)(t-x) dt dx. \quad (13)$$

After integrating the inner integral, we obtain

$$\int_a^c e''(x) \frac{(a-x)^2}{2} dx. \quad (14)$$

By the integral mean value theorem [33]

$$\begin{aligned} \int_a^c e''(x) \frac{(a-x)^2}{2} dx &= e''(\xi_e) \int_a^c \frac{(a-x)^2}{2} dx \\ &= e''(\xi_e) \frac{(c-a)^3}{6}, \end{aligned} \quad (15)$$

for some $\xi_e \in (a, c)$ and under the assumption that the second derivative of the function is bounded on (a, c) . A similar derivation shows the remainder term for g is given by

$$g''(\xi_g) \frac{(b-c)^3}{6}. \quad (16)$$

Substituting the error terms from (15) and (16) into (11), and noting that $e(c) = g(c)$, we obtain the following:

$$\begin{aligned} \int_a^b f(t) dt &= he(c) - \frac{(a-c)^2}{2} e'(c) + \frac{(b-c)^2}{2} g'(c) \\ &\quad + e''(\xi_e) \frac{(c-a)^3}{6} + g''(\xi_g) \frac{(b-c)^3}{6}. \end{aligned} \quad (17)$$

Since both $(c-a)^3$ and $(b-c)^3$ are $O(h^3)$, this simplifies to

$$\int_a^b f(t) dt = he(c) - \frac{(a-c)^2}{2} e'(c) + \frac{(b-c)^2}{2} g'(c) + O(h^3). \quad (18)$$

To approximate the integral, assume we use an n -point quadrature formula spanning the entire domain of integration where $n \geq 2$ and where we are guaranteed that the breakpoint c lies between two sample points, $c \in (t_k, t_{k+1})$. The quadrature approximation of the integral can then be written as

$$\begin{aligned} Q_n &= \sum_{i=1}^k w_i T_e(t_i) + \sum_{i=k+1}^n w_i T_g(t_i) \\ &= \sum_{i=1}^k w_i \left(e(c) + e'(c)(t_i - c) + \int_c^{t_i} e''(x)(t_i - x) dx \right) \\ &\quad + \sum_{i=k+1}^n w_i \left(g(c) + g'(c)(t_i - c) + \int_c^{t_i} g''(x)(t_i - x) dx \right). \end{aligned} \quad (19)$$

This equation is not useful in its current form, as we would like to understand how this approximation behaves in terms of h . Using the assumption that $h = b - a = \sum_{j=1}^n w_j$ and the definition that $e(c) = g(c)$, the terms not involving derivatives (denoted with the superscript zero) can be rewritten as

$$Q_n^0 = \sum_{i=1}^k w_i e(c) + \sum_{i=k+1}^n w_i g(c) = \sum_{i=1}^n w_i e(c) = he(c). \quad (20)$$

The terms involving first derivatives (denoted by the superscript one) are given by

$$Q_n^1 = e'(c) \sum_{i=1}^k w_i(t_i - c) + g'(c) \sum_{i=k+1}^n w_i(t_i - c). \quad (21)$$

To represent this equation in terms of h (i.e., the spacing between a and b), we note that

$$\begin{aligned} t_i &= a + \beta_i h, \\ c &= a + \beta_c h, \\ t_i - c &= a + \beta_i h - (a + \beta_c h) = h(\beta_i - \beta_c), \end{aligned} \quad (22)$$

where $0 \leq \beta_i \leq 1$. Substituting these relations into (21), we get

$$Q_n^1 = e'(c) \sum_{i=1}^k w_i h(\beta_i - \beta_c) + g'(c) \sum_{i=k+1}^n w_i h(\beta_i - \beta_c). \quad (23)$$

Expanding and rearranging the summation yields:

$$\begin{aligned} Q_n^1 &= h e'(c) \sum_{i=1}^k w_i \beta_i + h g'(c) \sum_{i=k+1}^n w_i \beta_i \\ &\quad - h \beta_c e'(c) \sum_{i=1}^k w_i - h \beta_c g'(c) \sum_{i=k+1}^n w_i. \end{aligned} \quad (24)$$

It is not immediately apparent how the individual weights are related to the interval spacing h . To elucidate the relationship, we first rescale the weights from $[a, b]$ to $[0, 1]$

$$w_i = \hat{w}_i(b - a) = h \hat{w}_i, \quad (25)$$

where \hat{w}_i is the rescaled weight. Substituting this relation into (24) and simplifying the expression in terms of h yields

$$\begin{aligned} Q_n^1 &= h^2 \left(e'(c) \sum_{i=1}^k \hat{w}_i \beta_i + g'(c) \sum_{i=k+1}^n \hat{w}_i \beta_i \right. \\ &\quad \left. - \beta_c e'(c) \sum_{i=1}^k \hat{w}_i - \beta_c g'(c) \sum_{i=k+1}^n \hat{w}_i \right). \end{aligned} \quad (26)$$

The remainder terms are represented as

$$\begin{aligned} Q_n^2 &= \sum_{i=1}^k w_i \int_c^{t_i} e''(x)(t_i - x) dx \\ &\quad + \sum_{i=k+1}^n w_i \int_c^{t_i} g''(x)(t_i - x) dx. \end{aligned} \quad (27)$$

Using the integral mean value theorem

$$\begin{aligned} Q_n^2 &= \sum_{i=1}^k w_i e''(\xi_e^i) \int_c^{t_i} (t_i - x) dx \\ &\quad + \sum_{i=k+1}^n w_i g''(\xi_g^{i-k}) \int_c^{t_i} (t_i - x) dx, \end{aligned} \quad (28)$$

for values $\xi_e^i \in (a, t_i)$, $1 \leq i \leq k$ and $\xi_g^{i-k} \in (c, t_i)$, $k+1 \leq i \leq n$. Integrating leads to

$$Q_n^2 = \sum_{i=1}^k w_i e''(\xi_e^i) \frac{(t_i - c)^2}{2} + \sum_{i=k+1}^n w_i g''(\xi_g^{i-k}) \frac{(t_i - c)^2}{2}. \quad (29)$$

Substituting the relation from (25) gives

$$Q_n^2 = h \sum_{i=1}^k \hat{w}_i e''(\xi_e^i) \frac{(t_i - c)^2}{2} + h \sum_{i=k+1}^n \hat{w}_i g''(\xi_g^{i-k}) \frac{(t_i - c)^2}{2}. \quad (30)$$

We note that $(t_i - c)^2$ is $O(h^2)$, which leads to $Q_n^2 = O(h^3)$. Equation (19) can now be rewritten as

$$\begin{aligned} Q_n &= h e(c) + h^2 \left(e'(c) \sum_{i=1}^k \hat{w}_i \beta_i + g'(c) \sum_{i=k+1}^n \hat{w}_i \beta_i \right. \\ &\quad \left. - \beta_c e'(c) \sum_{i=1}^k \hat{w}_i - \beta_c g'(c) \sum_{i=k+1}^n \hat{w}_i \right) + O(h^3). \end{aligned} \quad (31)$$

The error is then found by subtracting (31) from (18), which gives the following expression for the remainder term:

$$E = \frac{(b-c)^2}{2} g'(c) - \frac{(a-c)^2}{2} e'(c) - Q_n^1 + O(h^3). \quad (32)$$

Since both $(b-c)^2$ and $(a-c)^2$ are $O(h^2)$, the error of an n -point quadrature rule, given our two aforementioned assumptions (i.e., the quadrature rule consists of $n \geq 2$ samples and the breakpoint occurs between sample points), exhibits worst case second-order convergence.

If there are two breakpoints in the interval, a similar analysis indicates that an n -point quadrature rule, where $n \geq 3$ and in which the breakpoints lie intertwined between quadrature points, will exhibit worst case linear convergence. For two breakpoints $c_0, c_1 \in (a, b)$:

$$f(t) = \begin{cases} f_0(t), & a \leq t \leq c_0, \\ f_1(t), & c_0 < t \leq c_1, \\ f_2(t), & c_1 < t \leq b. \end{cases} \quad (33)$$

Using Taylor's theorem as we did in (10), we can rewrite $f(t)$ as

$$f(t) = \begin{cases} T_0(t) = f_0(c_0) + f_0'(c_0)(t - c_0) \\ \quad + \int_{c_0}^t f_0''(x)(t - x) dx & a \leq t \leq c_0, \\ T_1(t) = f_1(c_0) + f_1'(c_0)(t - c_0) \\ \quad + \int_{c_0}^t f_1''(x)(t - x) dx & c_0 < t \leq c_1, \\ T_2(t) = f_2(c_1) + f_2'(c_1)(t - c_1) \\ \quad + \int_{c_1}^t f_2''(x)(t - x) dx & c_1 < t \leq b, \end{cases} \quad (34)$$

and can then evaluate the integral as

$$\begin{aligned} \int_a^b f(t) dt &= \int_a^{c_0} T_0(t) dt + \int_{c_0}^{c_1} T_1(t) dt + \int_{c_1}^b T_2(t) dt \\ &= (c_0 - a)f_0(c_0) + (c_1 - c_0)f_1(c_0) \\ &\quad + (b - c_1)f_2(c_1) - \frac{(a - c_0)^2}{2} f_0'(c_0) \\ &\quad + \frac{(c_1 - c_0)^2}{2} f_1'(c_0) + \frac{(b - c_1)^2}{2} f_2'(c_1) + O(h^3). \end{aligned} \quad (35)$$

In the expression above and with what is to follow, we acknowledge that we are being mathematically loose in the

use of asymptotic notation. Given the way we have set up this mathematical problem, we cannot study the integration error behavior in the limit as h decreases to zero due to the fact that one of the breakpoints will eventually leave an interval of size h as there is a fixed distance between the breakpoints. Making this analysis rigorous is beyond the scope of this work. Its utility, however, is in helping to provide an intuition as to a possible estimation of the integration error as h decreases.

If we then consider (in this loose sense) that both $(a - c_0)^2$ and $(b - c_1)^2$ are both $O(h^2)$, and $(c_1 - c_0)^2$ is also $O(h^2)$ due to the two breakpoints lying between a and b , this can be simplified to

$$\int_a^b f(t) dt = (c_0 - a)f_0(c_0) + (c_1 - c_0)f_1(c_0) + (b - c_1)f_2(c_1) + O(h^2). \quad (36)$$

We next rewrite the quadrature formula, assuming that the breakpoints are between two sample points, $c_0 \in (t_j, t_{j+1})$ and $c_1 \in (t_k, t_{k+1})$, yielding:

$$Q_n = \sum_{i=1}^j w_i T_0(t_i) + \sum_{i=j+1}^k w_i T_1(t_i) + \sum_{i=k+1}^n w_i T_2(t_i). \quad (37)$$

The terms not involving derivatives are given by

$$Q_n^0 = \sum_{i=1}^j w_i f_0(t_i) + \sum_{i=j+1}^k w_i f_1(t_i) + \sum_{i=k+1}^n w_i f_2(t_i). \quad (38)$$

We can then use the relation in (25) to express this in terms of h

$$Q_n^0 = h \sum_{i=1}^j \hat{w}_i f_0(t_i) + h \sum_{i=j+1}^k \hat{w}_i f_1(t_i) + h \sum_{i=k+1}^n \hat{w}_i f_2(t_i). \quad (39)$$

Subtracting (38) from (36), and using the fact that $f_0(c_0) = f_1(c_0)$, we get

$$E = f_0(c_0)(c_1 - a) - h f_0(c_0) \sum_{i=1}^k \hat{w}_i + f_2(c_1)(b - c_1) - h f_2(c_1) \sum_{i=k+1}^n \hat{w}_i + O(h^2). \quad (40)$$

Since $(c_1 - a)$ and $(b - c_1)$ are $O(h)$, we expect that the error of an n -point quadrature rule when there are two breakpoints in the domain will decrease linearly upon refinement as long as there are two breakpoints in the interval (a, b) . Eventually, when the interval becomes small enough that only one breakpoint exists in the interval, the error decreases quadratically upon refinement; hence, the asymptotic convergence rate becomes second order, as shown in (32). In Fig. 2, we show the convergence of the volume rendering integral over a synthetic field (described in Section 7) where each ray consists of a transfer function with two breakpoints with a spacing of $w = 0.2, 0.02, 0.002$ between them. What we see is that the image converges linearly while the sample spacing is large enough to contain at least two breakpoints. Once the sampling falls below that threshold, the anticipated second-order convergence is observed. Similar analysis indicates that this result holds for $n > 2$ breakpoints as well.

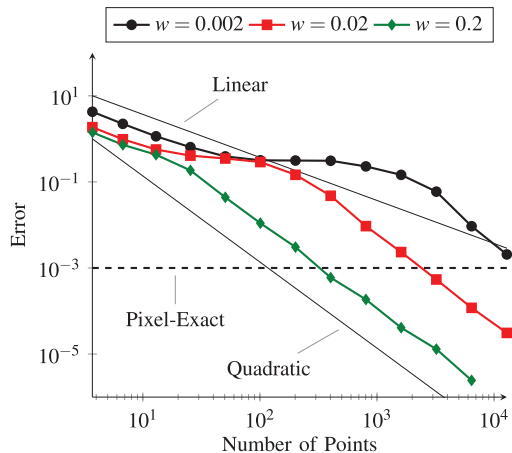


Fig. 2. Convergence rates for transfer functions with closely spaced breakpoints (spacing indicated by w). Second-order convergence is only possible when every quadrature interval contains at most one breakpoint. The closer the breakpoints, the longer it takes to converge to a pixel-exact image.

We can see from the results in (32) and (40) that the quadrature error exhibits worst case quadratic and linear convergence, respectively, under the assumption that the locations of the breakpoints do not coincide with any of the sample points, and that the quadrature rule uses two or more samples. While the assumption of two or more sample points covers many popular types of quadrature schemes (e.g., Newton-Cotes, Gaussian), it does not apply to two commonly used methods based on a single sample: Riemann quadrature and the midpoint rule. Neither of these rules will enable faster convergence; however, since, in the best case as applied to smooth functions, Riemann quadrature exhibits worst case linear convergence and the midpoint rule exhibits worst case quadratic convergence.

5.1.2 Composite Quadrature of Piecewise-Smooth Segments

In practice, the volume rendering integral along a ray is evaluated by breaking up the domain into N subintervals and then applying an n -point quadrature rule to each subinterval. Consider the quadrature of a smooth function (i.e., there are no breakpoints) on $[A, B]$ where the size of each subinterval is specified as $h = (B - A)/N$. If we use a quadrature rule with a local truncation error of $O(h^n)$ for each interval, then the overall error of this composite rule is given by

$$E = \sum_{i=1}^N O_i(h^n) = NO(h^n) = O(h^{n-1}), \quad (41)$$

where O_i represents the order of the error in the i th subinterval. In contrast, consider a function that is not smooth and consists of N_1 intervals containing no breakpoints and N_2 intervals containing a single breakpoint. From the analysis above, we can see that the composite error will be

$$E = \sum_{i=1}^{N_1} O_i(h^n) + \sum_{i=1}^{N_2} O_i(h^m), \quad (42)$$

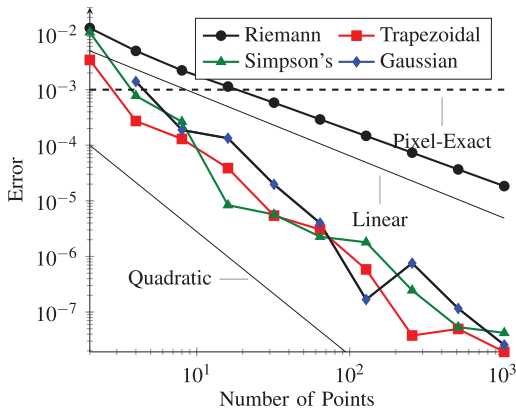


Fig. 3. Convergence of composite quadrature methods when applied to a function containing a single breakpoint, using error as defined in (9). Ideal quadratic and linear convergence rates are provided to give a baseline, as well as a cut-off point where the error has achieved pixel-exact status.

where $m = 2$ if the interval contains a single breakpoint, and $m = 1$ if it contains two or more. Therefore, the convergence of the entire integral is of order m .

The practical meaning of this result is that, when evaluating the volume rendering integral, if we can detect that a breakpoint exists along the ray, then it is sufficient to use the trapezoidal rule to evaluate the integral. Using methods designed for higher order convergence of the form given by (8) will not produce faster convergence and may introduce additional error due to unnecessary floating point calculations.

To illustrate, we present two examples. In Fig. 3, we show the convergence of several quadrature methods when applied to the function

$$f(t) = \begin{cases} -2t^3 + 4t^2 - t + .688528, & t \leq 0.54, \\ 2t^3 - 4t^2 + t + 1.311472, & t > 0.54, \end{cases} \quad (43)$$

which contains a single breakpoint at $t = 0.54$. Notice that Simpson's rule and Gauss-Legendre quadrature both exhibit second-order convergence, even though, in all intervals except the interval with the breakpoint, they approximate the integral exactly.

5.1.3 Quadrature of Smooth Segments

When the transfer function along a ray is smooth (i.e., does not contain any breakpoints), we can take advantage of the structure of the high-order field to use high-order quadrature to evaluate the volume rendering integral. Since the field is defined by high-order polynomials in reference space, a natural choice of integration method is Gaussian quadrature, which can exactly integrate a $2n - 1$ order polynomial with n function evaluations. Since the field along the ray is only guaranteed to be smooth, Gaussian quadrature will be unable to evaluate the integral exactly. The resulting error can be reduced by subdividing the ray and using Gaussian quadrature on each subinterval. The disadvantage of this approach is that the points from the subintervals do not coincide with the original evaluation points, requiring additional samples at all points each time a subdivision is performed.

Since we would like high-order convergence combined with the reuse of sample points, we turn to Gauss-Kronrod quadrature [34]. This method consists of an n point

Gaussian quadrature estimate, followed by an $n + 1$ point Kronrod extension, for a total of $2n + 1$ function evaluations. This rule is exact for polynomials up to degree $3n + 1$. Since the field along the ray is smooth, we can obtain an error estimate by looking at the difference between the n point Gaussian rule and the $2n + 1$ Kronrod extension.

5.2 Evaluation of the Volume Rendering Integral

We now discuss how we apply the concepts from the previous section to evaluate the volume rendering integral along a ray.

5.2.1 Empty Space Skipping

Empty space skipping is an important acceleration technique that has shown considerable performance improvements on data sets where sampling is cheap, such as voxel-based volumes where sampling involves trilinear interpolation. Space skipping is even more important in the context of high-order volume rendering because sampling the field is considerably more expensive. Recall from Section 3 that sampling a point requires the numerical inversion of the mapping function Φ as well as the evaluation of a high-order polynomial. Therefore, performance improvements can be realized by accurately detecting segments along the ray that do not contribute to the volume rendering integral.

Empty space skipping is performed on a ray segment in which τ is zero along the entire segment, which indicates that the segment does not contribute to the volume rendering integral. The performance implications of this optimization are dependent on the nature of the high-order field and of the transfer function. Transfer functions that classify large portions of the range of the scalar field will not need to skip many sections, while transfer functions that classify only targeted segments of the field will see performance improvements.

5.2.2 Occlusion Only

We evaluate occlusion only when the density transfer function has a value ($\tau \in \mathbf{PS} \vee \tau \in \mathbf{S}$) and the color transfer function is zero along the ray ($\kappa \in \mathbf{E}$). In this case, we do not need to evaluate the outer integral, as there is no emissive component to the volume rendering integral. We do, however, need to evaluate the accumulated opacity along this segment. We do this by using the trapezoidal rule if the transfer function is piecewise-smooth, and Gauss-Kronrod quadrature if it is smooth, as discussed in Sections 5.1.2 and 5.1.3.

5.2.3 Evaluating the Outer Integral

For all remaining cases, we must evaluate both the inner and outer integrals, as both the density and color transfer functions contribute to the final color. The convergence rate of the outer integral depends on the type of quadrature chosen for the outer and inner integrals. To see why this is the case, let Q_n be the quadrature rule used for the outer integral with error $O(h^n)$, and I_m be the quadrature rule used for the inner integral, with error $O(h^m)$. Evaluating the outer integral using Q_n gives

$$\int_a^b f(t) dt = \sum_i (w_i f(t_i) + O(h^m)) + O(h^n). \quad (44)$$

The term $O(h^m)$ in the summation comes from evaluating the inner integral on $[a, t_i]$ using quadrature rule I_m . This constrains the convergence of the outer integral to be no greater than the convergence of the inner integral.

If either κ or τ are piecewise-smooth, then the entire outer integral will have either second- or first-order convergence, depending on the location of the breakpoints. Therefore, even if one of κ or τ is smooth, we use the trapezoidal rule to evaluate both the outer and inner integrals. This has the added benefit of using the same set of sampling points for both integrals, reducing the overall number of computations required.

If κ and τ are both smooth, then we can obtain high-order convergence by applying Gaussian quadrature to both the outer and inner integrals. In practice, however, this does not work very well, since the evaluation points of the outer integral do not, in general, correspond to the points required to evaluate the inner integral. So, for each sample t_i in the outer integral, we would need to resample all points to evaluate the inner integral, which negates the anticipated performance improvements of high-order quadrature. We, therefore, evaluate the outer integral using Gaussian quadrature, but evaluate the inner integral using the trapezoidal rule. While this limits us to first- or second-order convergence, we show in Section 7.1 that it does give us better accuracy for a given number of samples.

5.2.4 Adaptive Quadrature

From the analysis presented above, we cannot expect to achieve better than second-order convergence when evaluating the volume rendering integral when using approximation methods of the form given by (8). A natural next step is to consider using adaptive quadrature to both reduce the number of samples required to evaluate the integral, and to generate pixel-exact images through the use of error estimators. This approach is especially appealing considering that transfer functions are often designed to ignore portions of the scalar field to allow the user to focus on features of interest. Transfer functions that do this contain one or more segments, $\tau(s) = 0, s \in [s_{min}, s_{max}]$, where the transfer function does not contribute to the integral's result. By using adaptive quadrature, we hope to avoid sampling these areas, and instead concentrate our samples on the portions of the transfer function that do contribute to the result.

Using adaptive quadrature, pixel-exact images are generated by refining the integral until the error estimate falls below the tolerance required for pixel-exact images. Since there are 255 different color levels for each channel in a standard 24-bit color image, we consider each color channel to be pixel-exact once the estimated error in the channel's integration falls below $1/255 = 0.0039$. To provide a buffer against underestimation of the error, we use 0.001 as our threshold for a pixel-exact image. Therefore, when an adaptive volume rendering has been performed, the resulting image can be known to be pixel-exact and, because we were able to sample adaptively, we are also able to reduce the number of samples required to generate the image.

We attempted several implementations of adaptive trapezoidal quadrature on the GPU, and in each case we found that we were able to reduce the number of samples required to evaluate the integral and generate pixel-exact images. This came, however, at the cost of an overall

increase in image generation time. In some cases, execution time doubled when using adaptive quadrature compared to nonadaptive quadrature, even though adaptive quadrature reduced the number of samples required. This performance result can be attributed to the GPU architecture we are using. When using Cuda for sampling the field, the simple trapezoidal rule with constant spacing h between samples can be implemented efficiently by first loading the basis functions into memory, then evaluating each of the samples in parallel. With adaptive quadrature, however, we interfere with the GPU's ability to evaluate the samples in parallel. At each step in an adaptive quadrature algorithm, we must determine if the current segment of the ray meets our predetermined accuracy requirements and, if it does not, subdivide the segment in a recursive manner until it does. In this manner, even though we may be evaluating fewer samples overall, our GPU is not working as efficiently as it could, and the overall execution time is slower.

At this time we have not found a way to perform adaptive quadrature on the GPU in a way that improves performance. The methods we have tried are able to reduce the number of samples taken at the expense of increased execution time, which defeats the purpose of reducing the number of samples required. So while adaptive quadrature is an attractive approach in principle, we have not yet found a feasible implementation.

6 IMPLEMENTATION

The implementation of the algorithm shown in Algorithm 1 is done using a combination of OptiX and Cuda. We use OptiX to perform the ray-tracing in Step 2. The ray-tracer is responsible for traversing the volume on an element by element basis. During each iteration, it stores the current element and the entrance and exit points $[t_a, t_b]$ for the element along the ray. Our initial implementations evaluated the volume rendering integral in OptiX as well. This was problematic for two reasons. First, the code necessary to perform the ray-tracing is compiled by the OptiX engine at runtime. The code for evaluating the volume rendering integral resulted in runtime initialization times on the order of 10 to 20 minutes. Second, and more importantly, evaluating the volume rendering integral in OptiX prevented us from using the GPU to its fullest extent. In particular, we have been able to achieve better performance by using shared memory, which is unavailable for use within an OptiX kernel.

Once the ray-tracer has completed, a Cuda kernel is launched to evaluate the volume rendering integral on each ray segment. The kernel needs to be able to access the data stored by the OptiX program, but the current Cuda implementations do not allow for the direct sharing of memory between Cuda contexts (OptiX is built on a Cuda context). While memory can be copied from OptiX to main memory, and then from main memory to the Cuda integration context, this is far too expensive for our interactivity requirements. Therefore, to share data between the OptiX and Cuda contexts, we create OpenGL pixel buffer objects to store the intersection points and element information. This approach requires the active GPU to be connected to a display to achieve best performance. If the active GPU is not connected to a display, such as is the case for GPUs used primarily for their computational capabilities, then the pixel buffer object will be allocated on a GPU

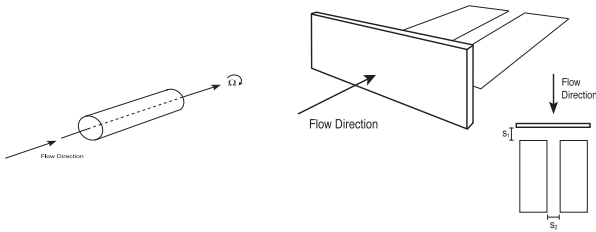


Fig. 4. Schematic overview of the fluid flow data sets considered. Left: Rotating canister traveling through an incompressible fluid. Right: Fluid flow past a block and an array of splitter plates.

connected to a display. When the pixel buffer is used in either the OptiX or Cuda context, it is first copied to the active GPU, resulting in a significant performance loss.

The Cuda module then evaluates the volume rendering integral as described in Section 5. The loop continues until all rays exit the volume.

7 RESULTS

We illustrate the utility of our system, both in terms of the accuracy of the generated images and its performance characteristics, by using our system on three different data sets. The first data set consists of a single, axis-aligned hexahedron with extents $[-1, -1, -2] - [1, 1, 4]$ and spherical field $f(x, y, z) = x^2 + y^2 + z^2$. This data set provides a good baseline upon which we can evaluate the accuracy of our implementation and verify that our implementation is correct.

The second data set consists of a rotating canister traveling through an incompressible fluid. A schematic of the flow scenario under consideration is presented in Fig. 4(left). The 3D mesh consists of 5,040 hexahedra and 696 prisms, with the computational problem being solved using third-order polynomials (in each direction, with ninth total order) within each element. The solutions presented herein were computed at Reynolds number of $Re = 1,000$ and with a nondimensionalized angular velocity of $\Omega = 0.2$.

The third data set consists of flow past a block with an array of splitter plates placed downstream of the block. A schematic of the flow scenario under consideration is presented in Fig. 4(right). As the fluid impinges upon the block, it is diverted around the structure, generating vorticity along the surface. The 3D computational mesh consists of 3,360 hexahedra and 7,644 prisms, with the computational fluid mechanics problem being solved with sixth-order polynomials (in each direction) per element. All computations were accomplished at $Re = 200$.

All tests were performed on a desktop workstation equipped with an NVIDIA Tesla C2050 GPU and Intel Xeon W3520 quad-core processor running at 2.6 GHz. We used OptiX Version 2.1.1 and Cuda Version 4.0, using 32-bit floating point precision for all code executed on the GPU. Performance when using 64-bit precision varies depending on the card.

7.1 Convergence Results

The goal of our system is the generation of pixel-exact images. The convergence graphs shown in this section were obtained by first generating a pixel-exact image then

comparing this image to the images created with varying values of sample spacing. The error metric we used is the largest difference between pixel values over the image: $\text{Max}|I_r(x, y) - I_h(x, y)|$, where $I_r(x, y)$ is the pixel in the pixel-exact image at position (x, y) , and $I_h(x, y)$ is the pixel in the image generated with sample spacing h . We compare this overall image error to the total number of samples required to generate the image.

We start by verifying the theoretical convergence of our method by using the spherical synthetic data. We use this data set because we can evaluate the volume rendering integral accurately enough to guarantee a pixel-exact image, which we can then use as a “gold-standard” against which to verify our method. Even for a field as simple as this one we are unable to create an analytic expression for the volume rendering integral. However, by placing the field in an axis-aligned hexahedron, we know that the mapping function Φ from Section 3 is a linear transformation, which means that the field along the ray will be quadratic.

Generation of the pixel-exact image proceeds as follows: We start by generating a ray for each pixel. In this special case where we know the field along the ray is quadratic, we can numerically find the location of each breakpoint along the ray to obtain a list of ray segments with no discontinuities (similar to what is done by Williams et al. [11]). We then evaluate the outer integral using Gauss-Kronrod quadrature. At each of the sample points in the outer integral, we use Simpson’s rule to evaluate the inner integral exactly. After evaluating all of the points in the outer integral, we use the Gauss-Kronrod error estimator to determine if we have achieved a pixel-exact image. We have found that for this simple data set, the 15-point Gauss-Kronrod rule approximated the volume rendering integral well below the pixel-exact error threshold.

We then generated visualizations using Riemann integration (which is representative of the types of integration performed by most existing volume rendering systems) and our system to verify the expected convergence rates. To ensure that this test provides a valid comparison of convergence rates, empty-space skipping was enabled for all tests.

Results are shown in Fig. 5. As expected, Riemann quadrature converges linearly, while our method exhibits second-order convergence. Not only does our method exhibit higher order convergence than Riemann quadrature, the resulting image is also more accurate for a given number of samples.

We next test our convergence on the canister data set for which we cannot calculate the volume rendering integral exactly. Since we do not have an analytic solution, we create the pixel image by refining the sampling size until the resulting image no longer changes. We show convergence results for both Riemann quadrature and our method in Fig. 5. We can see that we achieved the expected second-order convergence, and that our method always returns an image with less error than the image produced using Riemann integration.

Finally, we illustrate the benefit of using high-order quadrature for the outer integral when possible. We applied a transfer function to the sphere data set in which neither the density function nor the color transfer function

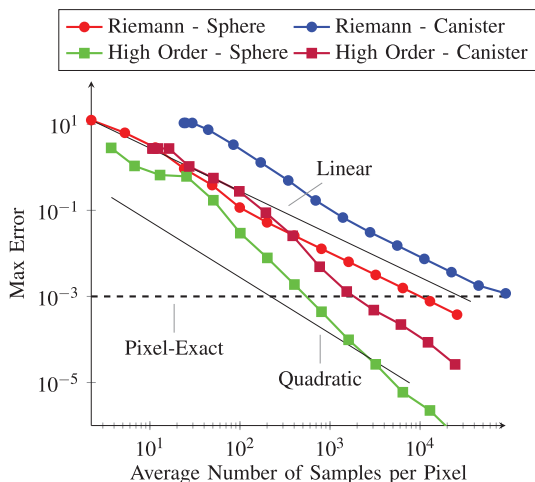


Fig. 5. Comparison of convergence rates between Riemann integration and our high-order method. Our method converges to the pixel-exact image an order of magnitude faster than simple Riemann integration. Ideal quadratic and linear convergence rates are provided to give a baseline, as well as a cut-off point where the error has achieved pixel-exact status.

contained breakpoints. As discussed in Section 5.2.3, we expect second-order convergence in this case. In Fig. 6, we compare evaluating the volume rendering integral using Gauss-Kronrod quadrature and the trapezoidal rule for the outer integral. As expected, both methods converge quadratically, but by using Gauss-Kronrod quadrature for the outer integral, we are able to obtain a more accurate image for a given number of samples. In fact, for a given level of accuracy, we see that the Gauss-Kronrod approach uses an order of magnitude fewer samples.

7.2 Accuracy

In this section, we compare the images generated by our method to those generated by the `vtkGPUVolumeRayCastMapper` class in VTK. This system is similar to many existing volume rendering algorithms in that it operates on a collection of samples on a linear grid, where the values between samples are obtained via linear interpolation. This

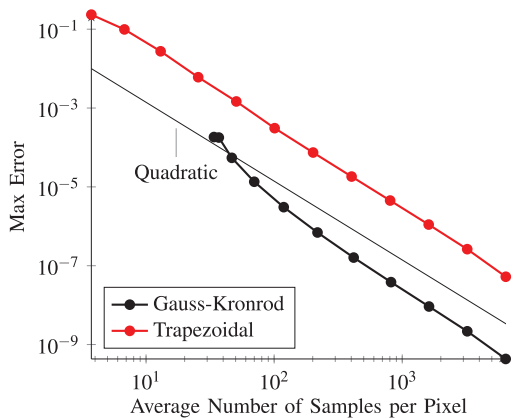


Fig. 6. Comparison of using Gauss-Kronrod quadrature to evaluate the outer integral of the sphere data set versus trapezoidal rule.

test corresponds to how most volume visualization is performed for high-order data sets. In Fig. 7a, we show a view of block and splitter data set generated by our algorithm. In Figs. 7b, 7c, and 7d, we show this same view, generated with VTK, for data sets with samples spacing of 0.074, 0.071, and 0.068, respectively. Each of these images was generated by increasing the sample spacing of the volume rendering algorithm (not the data set) until the resulting image no longer changed with additional refinement. What we can see from this example is that the image generated with a data set sampling of 0.071 is obviously incorrect, while images performed with data sets sampled both more coarsely and more finely produce images that are closer to the true result. This illustrates a significant problem with volume rendering approaches based on sampling high-order volumes; namely, that it is impossible to determine, by an evaluation of the resulting image, if the image contains significant error.

7.3 Performance

While the primary focus of our system is on accuracy, we posit that it must also be interactive to be useful. We define interactivity as achieving rendering speeds of at least one

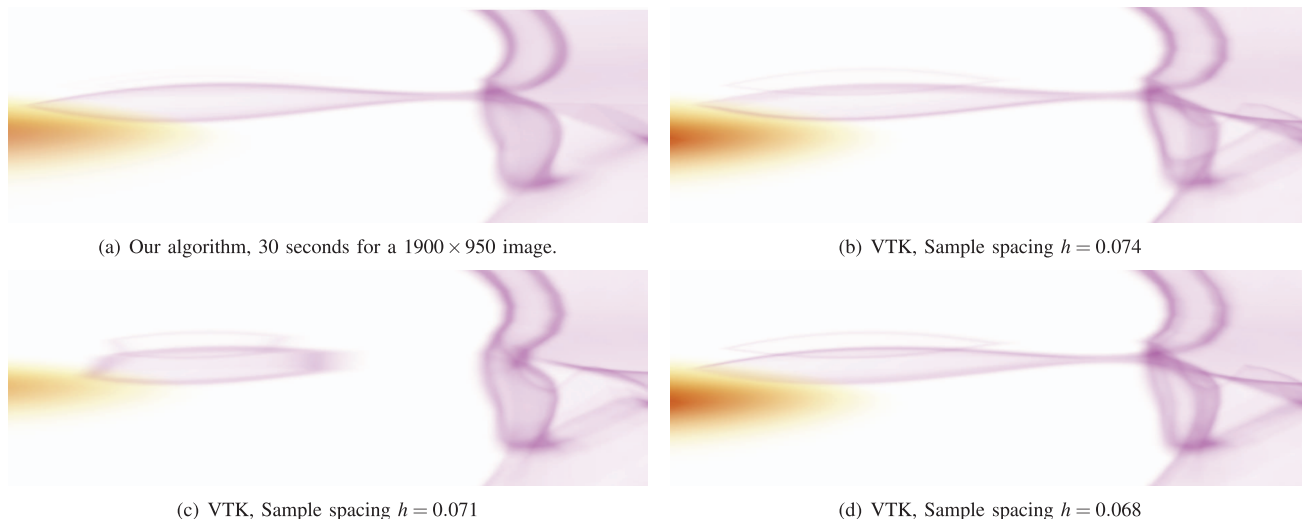


Fig. 7. In this figure, we illustrate how features can be lost in the sampling process, by showing the feature as rendered by our algorithm compared with image rendered using the GPU-based volume rendering system in VTK for a variety of sample spacings.

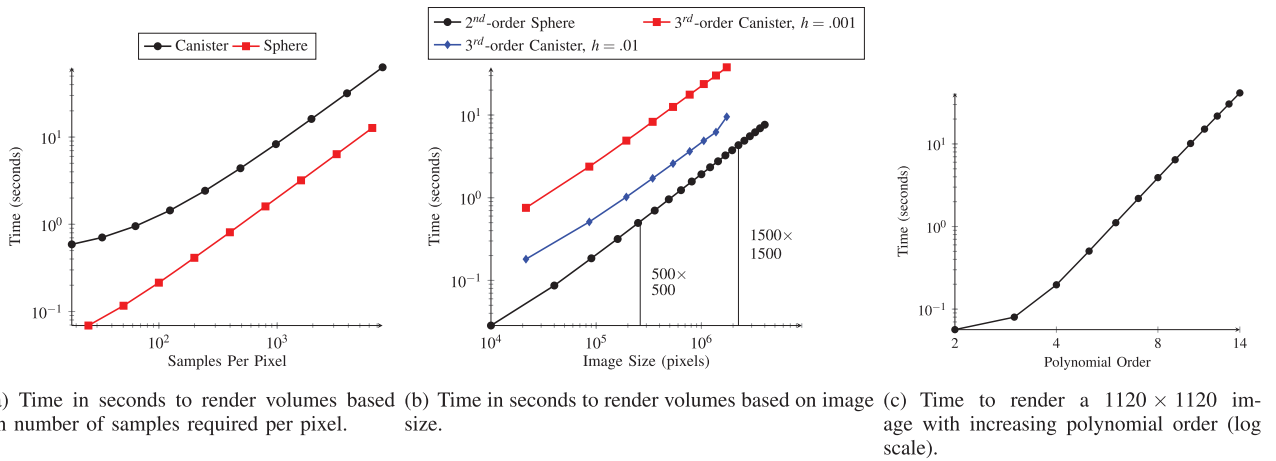


Fig. 8. Performance measures for our volume rendering algorithm. The factors with the greatest influence over execution speed are image size and number of samples.

frame per second. Overall, our system is capable of achieving these interactive frame rates for images of up to 512×512 pixels. For images larger than this, our system reduces the sampling rate during user interaction to maintain interactivity, and then generates the full, accurate image when user interaction has stopped.

The execution speed of our system is influenced by several factors: the sample spacing h along a ray, the overall image size, the polynomial order of the high-order data set, and the number of elements in the data set. To investigate the impact of each of these factors, we have performed experiments where we varied one of these parameters while holding the others constant. The results of these tests are shown in Fig. 8.

In Fig. 8a, we show the performance of our system based on the number of samples used to evaluate the volume rendering integral. Performance scales linearly with the number of samples. Combined with the convergence analysis from Section 7.1, we see that we can generally expect to double execution time to reduce the error by four. In Fig. 8b, we show performance as a function of image size. Finally, in Fig. 8c, we show performance based on the polynomial order associated

with each direction. While the time required does grow quickly with order, in practice volumes are rarely higher than sixth and eighth order per direction.

Of these parameters, we have control over h and the image size, but we do not have control over the number of elements or their order, as these are established by the engineer creating the simulation and are domain specific.

In Fig. 9, we present the times required to render a $1,900 \times 950$ image of the view shown in Fig. 7 using our method and VTK. As above, for VTK we first generated a regular grid by sampling the high-order data set, then used VTK’s `vtkGPUVolumeRayCastMapper` class to generate the images. While this class does not implement empty-space skipping, it generated images faster than those VTK classes that did. In Fig. 9, we show both the time to perform the rendering and the time required to perform the sampling compared to the time required to generate the accurate image using our method. This graph shows that, while we have met our goal of generating accurate images of high-order data, additional work is needed to obtain interactivity across all data sets and image sizes.

We can see that VTK does produce its images faster than our method; however, as shown in Fig. 10, the memory required to store the regular grid limits how fine the

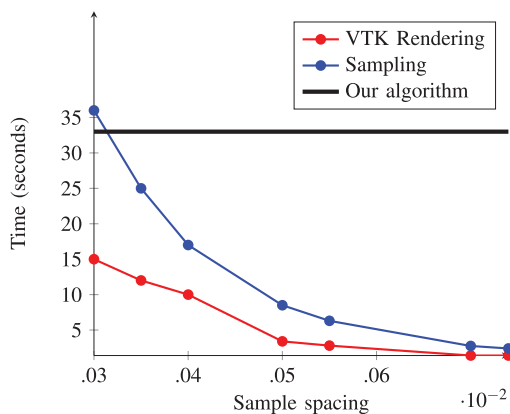


Fig. 9. Performance of our method rendering the view shown in Fig. 7 (black) versus the time to render using VTK for a variety of sample spacings (red). The blue line indicates the preprocessing time required to generate the grid used by VTK.

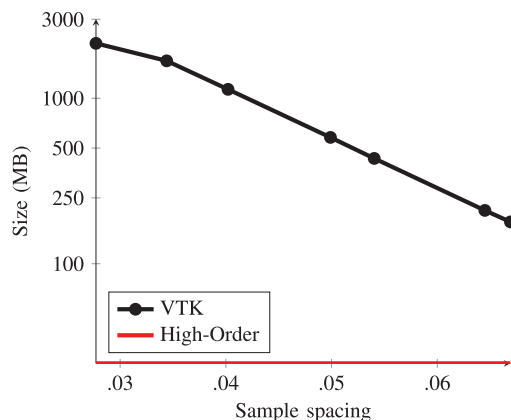


Fig. 10. Memory required for the high-order volume versus sampled regular grids. The memory required for the high-order volume is 25 megabytes.

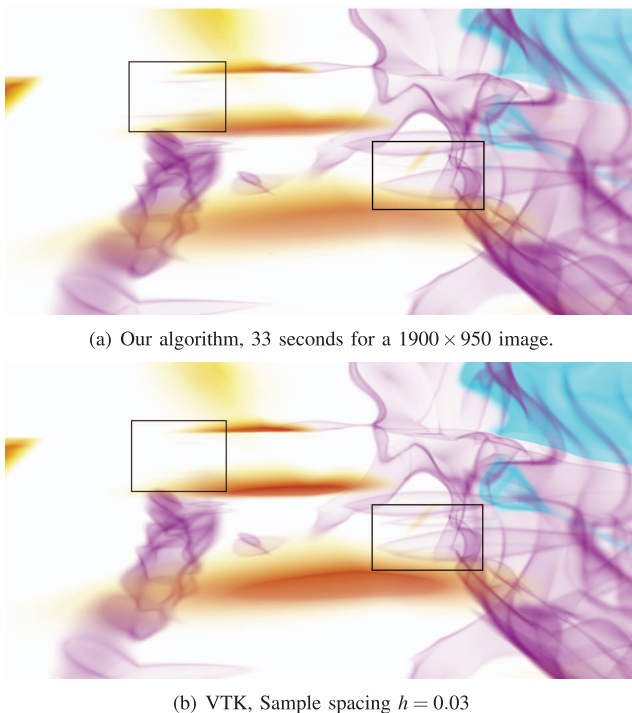


Fig. 11. Accuracy comparison between our method and the most accurate image generated by VTK, given the memory constraints of our desktop system. The boxed regions indicate areas where visible error is still present in the VTK image.

spacing can be, which then limits the level of accuracy that can be obtained. For example, in Fig. 11 we show a comparison between our method and VTK for a sample spacing of $h = 0.03$ and total volume size of 2 gigabytes, which was the largest volume we could render using VTK. With this spacing, there are still noticeable errors in the VTK image.

8 SUMMARY AND CONCLUSION

We have presented a new technique for evaluating the volume rendering integral in high-order finite element fields which attempts to address the often contradictory goals of image accuracy and interactive performance. This system uses the high-order data in its native form, thereby avoiding the approximation errors that are present when sampling onto voxel-based data structures. We have shown that, while the worst case convergence of our system is the same as that of simple Riemann integration, we generally achieve second-order convergence and are capable of producing images with less error for a given number of samples when compared to existing methods. By reducing the number of samples used to generate accurate images, we have been able to develop a system that can produce volume rendering images of high-order data efficiently on a desktop system.

We have shown that our algorithm is capable of generating accurate images of high-order data with little user interaction. This accuracy comes with a performance penalty that makes our algorithm relatively slow for all but smaller image sizes. Hence, although we have been successful in meeting our goal of accuracy, further work is needed to fully realize our goal of interactivity. This work, however, provides the first strides towards attaining

both goals in reference to volume rendering of high-order finite element fields. We are actively investigating new approaches that can take advantage of the mathematical insights gained in this work to produce interactive renderings for more representative image sizes. One promising avenue is through the use of adaptive integration. Although our system is capable of generating pixel-exact images via adaptive integration, we found that technical limitations made it far from interactive and unsuitable for general use. We are currently investigating new ways in which adaptive integration of the volume rendering integral can be framed in the context of GPU computation to restore the lost performance.

ACKNOWLEDGMENTS

The authors thank Tiago Etienne and Mathias Schott for useful discussions about volume rendering and quadrature, and Dr. Sergey Yakovlev for reading an early draft and providing needed feedback and direction. The majority of this work was completed while the first author was a member of the Scientific Computing and Imaging Institute at the University of Utah. This work was supported under ARO W911NF-08-1-0517 and W911NF1210375 (Program Manager Dr. Mike Coyle) and the Department of Energy (DOE NET DE-EE0004449). Infrastructure support provided through NSF-IIS-0751152.

REFERENCES

- [1] G.E. Karniadakis and S.J. Sherwin, *Spectral/hp Element Methods for CFD*. Oxford Univ. Press, 1999.
- [2] A. Patera, "A Spectral Method for Fluid Dynamics: Laminar Flow in a Channel Expansion," *J. Computational Physics*, vol. 54, p. 468, 1984.
- [3] G. Karniadakis, E. Bullister, and A. Patera, "A Spectral Element Method for Solution of Two- and Three-Dimensional Time Dependent Navier-Stokes Equations," *Finite Element Methods for Nonlinear Problems*, p. 803, Springer-Verlag, 1985.
- [4] 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*, vol. 29, no. 6, pp. 6-13, <http://viscg.uni-muenster.de/publications/2009/MRMH09>, Nov./Dec. 2009.
- [5] "ImageVis3D: A Real-Time Volume Rendering Tool for Large Data," Scientific Computing and Imaging Inst. (SCI), <http://www.imagevis3d.org>, 2013.
- [6] W. Schroeder, K.M. Martin, and W.E. Lorensen, *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, second ed. Prentice-Hall, Inc., 1998.
- [7] S.G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, "Optix: A General Purpose Ray Tracing Engine," *ACM Trans. Graphics*, vol. 29, Aug. 2010.
- [8] G. Marmitt, H. Friedrich, and P. Slusallek, "Efficient CPU-Based Volume Ray Tracing Techniques," *Computer Graphics Forum*, vol. 27, no. 6, pp. 1687-1709, <http://dx.doi.org/10.1111/j.1467-8659.2008.01179.x>, 2008.
- [9] G. Marmitt, H. Friedrich, and P. Slusallek, "Interactive Volume Rendering with Ray Tracing," *Eurographics State of the Art Reports*, 2006.
- [10] K. Moreland and E. Angel, "A Fast High Accuracy Volume Renderer for Unstructured Data," *Proc. IEEE Symp. Volume Visualization and Graphics*, pp. 9-16, <http://dx.doi.org/10.1109/VV.2004.2>, 2004.
- [11] P.L. Williams, N.L. Max, and C.M. Stein, "A High Accuracy Volume Renderer for Unstructured Data," *IEEE Trans. Visualization and Computer Graphics*, vol. 4, no. 1, pp. 37-54, <http://portal.acm.org/citation.cfm?id=614269.614391>, Jan. 1998.

- [12] D.F. Wiley, H.R. Childs, B. Hamann, and K.I. Joy, "Ray Casting Curved-Quadratic Elements," *Proc. Sixth Joint Eurographics-IEEE TCVG Conf. Visualization (VISSYM '04)*, pp. 201-210, <http://dx.doi.org/10.2312/VisSym/VisSym04/201-210>, 2004.
- [13] M. Üffinger, S. Frey, and T. Ertl, "Interactive High-Quality Visualization of Higher-Order Finite Elements," *Computer Graphics Forum*, vol. 29, no. 2, pp. 337-346, <http://dx.doi.org/10.1111/j.1467-8659.2009.01603.x>, 2010.
- [14] C. Sadowsky, J. Cohen, and R. Taylor, "Rendering Tetrahedral Meshes with Higher-Order Attenuation Functions for Digital Radiograph Reconstruction," *Proc. IEEE Visualization (VIS '05)*, pp. 303-310, Oct. 2005.
- [15] Y. Zhou and M. Garland, "Interactive Point-Based Rendering of Higher-Order Tetrahedral Data," *IEEE Trans. Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1229-1236, Sept./Oct. 2006.
- [16] K. Novins and J. Arvo, "Controlled Precision Volume Integration," *Proc. Workshop Volume Visualization*, pp. 83-89, <http://doi.acm.org/10.1145/147130.147154>, 1992.
- [17] J.-F. El Hajjar, S. Marchesin, J.-M. Dischler, and C. Mongenet, "Second Order Pre-Integrated Volume Rendering," *Proc. IEEE Pacific Visualization Symp. (PacificVIS '08)*, pp. 9-16, Mar. 2008.
- [18] B. Nelson, R. Haimes, and R.M. Kirby, "GPU-Based Interactive Cut-Surface Extraction from High-Order Finite Element Fields," *IEEE Trans. Visualization and Computer Graphics*, vol. 17, no. 12, pp. 1803-1811, Dec. 2011.
- [19] M. Brasher and R. Haimes, "Rendering Planar Cuts through Quadratic and Cubic Finite Elements," *Proc. IEEE Conf. Visualization*, pp. 409-416, <http://dx.doi.org/10.1109/VISUAL.2004.91>, 2004.
- [20] B. Nelson and R.M. Kirby, "Ray-Tracing Polymorphic Multi-domain Spectral/hp Elements for Isosurface Rendering," *IEEE Trans. Visualization and Computer Graphics*, vol. 12, no. 1, pp. 114-125, <http://portal.acm.org/citation.cfm?id=1100864.1100967>, Jan. 2006.
- [21] M. Meyer, B. Nelson, R. Kirby, and R. Whitaker, "Particle Systems for Efficient and Accurate High-Order Finite Element Visualization," *IEEE Trans. Visualization and Computer Graphics*, vol. 13, no. 5, pp. 1015-1026, Sept. 2007.
- [22] C. Pagot, J. Vollrath, F. Sadlo, D. Weiskopf, T. Ertl, and J.L.D. Comba, "Interactive Isocontouring of High-Order Surfaces," *Scientific Visualization: Interactions, Features, Metaphors*, Dagstuhl Follow-Ups, H. Hagen, ed., vol. 2, pp. 276-291, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, <http://drops.dagstuhl.de/opus/volltexte/2011/3305>, 2011.
- [23] J.-F. Remacle, N. Chevaugon, É. Marchandise, and C. Geuzaine, "Efficient Visualization of High-Order Finite Elements," *Int'l J. for Numerical Methods in Eng.*, vol. 69, no. 4, pp. 750-771, <http://dx.doi.org/10.1002/nme.1787>, 2007.
- [24] W. Schroeder, F. Bertel, M. Malaterre, D. Thompson, P. Pebay, R. O'Bara, and S. Tendulkar, "Methods and Framework for Visualizing Higher-Order Finite Elements," *IEEE Trans. Visualization and Computer Graphics*, vol. 12, no. 4, pp. 446-460, July/Aug. 2006.
- [25] N. Max, "Optical Models for Direct Volume Rendering," *IEEE Trans. Visualization and Computer Graphics*, vol. 1, no. 2, pp. 99-108, <http://dx.doi.org/10.1109/2945.468400>, June 1995.
- [26] J. Kniss, S. Premoze, M. Ikits, A. Lefohn, C. Hansen, and E. Praun, "Gaussian Transfer Functions for Multi-Field Volume Visualization," *Proc. IEEE 14th Visualization (VIS '03)*, pp. 497-504, <http://dx.doi.org/10.1109/VISUAL.2003.1250412>, 2003.
- [27] M. Hadwiger, J.M. Kniss, C. Rezk-Salama, D. Weiskopf, and K. Engel, *Real-Time Volume Graphics*. A.K. Peters, Ltd., 2006.
- [28] P.J. Davis and P. Rabinowitz, *Methods of Numerical Integration*, Computer Science and Applied Mathematics, second ed. Academic Press Inc., 1984.
- [29] M.P. Garrity, "Raytracing Irregular Volume Data," *Proc. Workshop Volume Visualization*, pp. 35-40, <http://doi.acm.org/10.1145/99307.99316>, 1990.
- [30] R.E. Moore, R.B. Kearfott, and M.J. Cloud, *Introduction to Interval Analysis*. Soc. for Industrial and Applied Math., 2009.
- [31] H.H. Sohrab, *Basic Real Analysis*. Birkhäuser, 2003.
- [32] W. Rudin, *Real and Complex Analysis*, third ed. McGraw-Hill, Inc., 1987.
- [33] T.M. Apostol, *Calculus*, vol. 1, second ed. Wiley, 1967.
- [34] A. Kronrod, *Nodes and Weights of Quadrature Formulas, Sixteen-Place Tables: Authorized Translation from the Russian*. Consultants Bureau, 1965.



Blake Nelson received the BS and MS degrees in computer science from the University of Utah, and the PhD degree in scientific computing from the University of Utah, where he was a member of the Scientific Computing and Imaging Institute. He is currently a research scientist and software engineer with Space Dynamics Laboratory at the Utah State University Research Foundation. His research interests include graphics, high-performance computing, scientific visualization, algorithms, and software architecture.



Robert M. Kirby (M'04) received the MS degree in applied mathematics, the MS degree in computer science, and the PhD degree in applied mathematics from Brown University, Providence, Rhode Island, 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. He is a member of the IEEE.



Robert Haimes is a principal research engineer in the Aerospace Computational Design Laboratory of the Department of Aeronautics and Astronautics at the Massachusetts Institute of Technology. His major research focuses have been computational fluid dynamics (CFD) scientific visualization for the results from CFD simulations, distributed and high-performance computing, applied computational geometry (analytic and discrete) and the use of geometry in conceptual through final design. He has had a number of projects in these areas funded by NASA, DoD, and industry. The research is usually expressed by publication and software (made generally available) that expresses the ideas articulated in the papers.

► For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.