Investigation of Smoothness-Increasing Accuracy-Conserving Filters for Improving Streamline Integration through Discontinuous Fields

Michael Steffen, Sean Curtis, Robert M. Kirby, *Member*, *IEEE*, and Jennifer K. Ryan

Abstract—Streamline integration of fields produced by computational fluid mechanics simulations is a commonly used tool for the investigation and analysis of fluid flow phenomena. Integration is often accomplished through the application of ordinary differential equation (ODE) integrators—integrators whose error characteristics are predicated on the smoothness of the field through which the streamline is being integrated, which is not available at the interelement level of finite volume and finite element data. Adaptive error control techniques are often used to ameliorate the challenge posed by interelement discontinuities. As the root of the difficulties is the discontinuous nature of the data, we present a complementary approach of applying smoothness-increasing accuracy-conserving filters to the data prior to streamline integration. We investigate whether such an approach applied to uniform quadrilateral discontinuous Galerkin (high-order finite volume) data can be used to augment current adaptive error control approaches. We discuss and demonstrate through a numerical example the computational trade-offs exhibited when one applies such a strategy.

Index Terms—Streamline integration, finite element, finite volume, filtering techniques, adaptive error control.

1 Introduction

Given a vector field, the streamlines of that field are lines that are everywhere tangent to the underlying field. A quick search of both the visualization and the application domain literature demonstrates that streamlines are a popular visualization tool. The bias toward using streamlines is in part explained by studies that show streamlines to be effective visual representations for elucidating the salient features of the vector fields [1]. Furthermore, streamlines as a visual representation are appealing because they are applicable for both two-dimensional (2D) and three-dimensional (3D) fields [2].

Streamline integration is often accomplished through the application of ordinary differential equation (ODE) integrators such as predictor-corrector or Runge-Kutta schemes. The foundation for the development of these schemes is the use of the Taylor series for building numerical approximations of the solution of the ODE of interest. The Taylor series can be further used to elucidate the error characteristics of the derived scheme. All schemes employed for streamline integration that are built using such an approach exhibit error characteristics that are predicated on the smoothness of the field through which the streamline is being integrated.

Low-order and high-order finite volume and finite element fields are among the most common types of fluid flow simulation data sets available. Streamlining is commonly applied to these data sets. The property of these fields that challenges classic streamline integration using Taylor-series-based approximations is that finite volume fields are piecewise discontinuous and finite element fields are only C^0 continuous. Hence, one of the limiting factors of streamline accuracy and integration efficiency is the lack of smoothness at the interelement level of finite volume and finite element data.

Adaptive error control techniques are often used to ameliorate the challenge posed by interelement discontinuities. To paraphrase a classic work on the subject of solving ODEs with discontinuities [3], one must 1) detect, 2) determine the order, size, and location of, and 3) judiciously "pass over" discontinuities for effective error control. Such an approach has been effectively employed within the visualization community for overcoming the challenges posed by discontinuous data at the cost of an increased number of evaluations of the field data. The number of evaluations of the field increases drastically with every discontinuity that is encountered [3]. Thus, if one requires a particular error tolerance and employs such methods for error control when integrating a streamline through a finite volume or finite element data set, a large amount of the computational work involved is due to handling interelement discontinuities and not the intraelement integration.

As the root of the difficulties is the discontinuous nature of the data, one could speculate that if one were to filter the data in such a way that it was no longer discontinuous,

- *M. Steffen and R.M. Kirby are with the School of Computing and the Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT 84112. E-mail: {msteffen, kirby}@cs.utah.edu.*
- *S. Curtis is with the Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599. E-mail: seanc@cs.unc.edu.*
- *J.K. Ryan is with the Department of Mathematics, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061. E-mail: jkryan@vt.edu.*

Manuscript received 22 Feb. 2007; revised 30 Aug. 2007; accepted 17 Dec. 2007; published online 2 Jan. 2008.
Recommended for acceptance by A. Pang.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-0014-0207.
Digital Object Identifier no. 10.1109/TVCG.2008.9.

streamline integration could then be made more efficient. The caveat that arises when one is interested in simulation and visualization error control is how does one select a filter that does not destroy the formal accuracy of the simulation data through which the streamlines are to be integrated. Recent mathematical advances [4], [5] have shown that such filters can be constructed for high-order finite element and discontinuous Galerkin (DG) (high-order finite volume) data on uniform quadrilateral and hexahedral meshes. These filters are such that they have the provable quality that they increase the level of smoothness of the field without destroying the accuracy in the case that the "true solution" that the simulation is approximating is smooth. In fact, in many cases, these filters can increase the accuracy of the solution. It is the thesis of this work that application of such filters to discontinuous data prior to streamline integration can drastically improve the computational efficiency of the integration process.

1.1 Objectives

In this paper, we present a demonstration of a complementary approach to classic error control—application of smoothness-increasing accuracy-conserving (SIAC) filters to discontinuous data prior to streamline integration. The objective of this work is to understand the computational trade-offs between the application of error control on discontinuous data and the filtering of the data prior to integration.

Although the filtering approach investigated here has been extended to nonuniform quadrilateral and hexahedral meshes [6] and has one-sided variations [7], we will limit our focus to an investigation of whether such an approach applied to uniform quadrilateral DG (high-order finite volume) data can be used to augment current adaptive error control approaches. We will employ the uniform symmetric filtering technique that is based upon convolution with a kernel constructed as a linear combination of B-splines [5]. We will present the mathematical foundations of the construction of these filters, a discussion of how the filtering process can be accomplished on finite volume and finite element data, and results that demonstrate the benefits of applying these filters prior to streamline integration. We will then discuss and demonstrate through a numerical example the computational trade-offs exhibited when one applies such a strategy.

1.2 Outline

The paper is organized as follows: In Section 2, an overview of some of the previous work accomplished in streamline visualization and in filtering for visualization, as well as the local SIAC filter that we will be using, is presented. In Section 3, we review the two numerical sources of error that arise in streamline integration—projection (or solution) error and time-stepping error. In addition, we will present how adaptive error control attempts to handle time-stepping errors introduced when integrating discontinuous fields. In Section 4, we present the mathematical details of the SIAC filters, and in Section 5, we provide a discussion of implementation details needed to understand the application of the filters. In Section 6, we provide a demonstration of the efficacy of the filters as a preprocessing to streamline integration, and in Section 7, we provide a summary of our findings and a discussion of future work.

2 Previous Work

In this section, we review two orthogonal but relevant areas of background research: vector field visualization algorithms and filtering. Both areas are rich subjects in the visualization and image processing literature; as such, we only provide a cursory review so that the context for the current work can be established.

2.1 Previous Work in Vector Field Visualization

Vector fields are ubiquitous in scientific computing (and other) applications. They are used to represent a wide range of diverse phenomena, from electromagnetic fields to fluid flow. One of the greatest challenges in working with such data is presenting it in a manner that allows a human to quickly understand the fundamental characteristics of the field efficiently.

Over the years, multiple strategies have been developed in the hope of answering the question of how best to visualize vector fields. There are iconic methods [8], image-based methods such as spot-noise diffusion [9], line-integral convolution [10], reaction-diffusion [11], [12], and streamlines [13]. Each method has its own set of advantages and disadvantages. Iconic (or glyph) methods are some of the most common, but glyphs' consumption of image real estate can make them ineffective for representing fields that exhibit large and diverse scales. Image-based methods provide a means of capturing detailed behavior with fine granularity but are computationally expensive and do not easily facilitate communicating vector magnitude. Also, extending image-based methods into 3D can make the visualization less tractable for the human eye. Reaction-diffusion and image-based methods are also computationally expensive. Streamlines, which are space curves that are everywhere tangent to the vector field, offer a good all-around solution. Well-placed streamlines can capture the intricacies of fields that exhibit a wide scale of details (such as turbulent flows). They can easily be decorated with visual indications of vector direction and magnitude. However, placing streamlines is not a trivial task, and calculating streamlines is a constant battle between computational efficiency and streamline accuracy. Placing (or seeding) the streamlines by hand yields the best results, but it requires a priori knowledge that is seldom available or easily attainable. The simplest automatic placement algorithms such as grid-aligned seeds yield unsatisfying results. There has been a great deal of research in placement strategies [14], [15], [16], [17], [18]. These strategies have continuously improved computational efficiency, as well as yielding appealing results—long continuous streamlines, nicely distributed through the field. Another concern with streamline integration is maximizing accuracy while minimizing error and computational effort. Both Runge-Kutta and extrapolation methods are commonly mentioned in the literature—with the choice of which integration technique to use being based on a multitude of mathematical and computational factors such as the error per unit of computational cost, availability (and strength) of error

estimators, etc. The lack of smoothness at element or grid boundaries can cause large errors during integration, leading some to utilize adaptive error control techniques such as Runge-Kutta with error estimation and adaptive step-size control [19], [20], [21]. The work presented in this paper benefits from all the previous streamline placement work, as our focus is on understanding and respecting the assumptions upon which the time integrators commonly used in streamline routines are built.

2.2 Previous Work in Filtering for Visualization

Filtering in most visualization applications has as its goal the reconstruction of a continuous function from a given (discrete) data set. For example, assume that f_k is the given set of evenly sampled points of some function $f(x)$. A filter might take this set of points and introduce some type of continuity assumption to create the reconstructed solution $f^*(x)$. Filtering for visualization based upon discrete data is often constructed by considering the convolution of the "true" solution, of which f_k is a sampling, against some type of spline, often a cubic B-spline [22], [23], [24], [25], [26], [27]. Much of the literature concentrates on the specific use in image processing, though there has also been work in graphic visualization [10], [28], as well as computer animation [29]. The challenge in spline filtering is choosing the convolution kernel such that it meets smoothness requirements and aids in data reconstruction without damage to the initial discrete data set.

There are many filtering techniques that rely on the use of splines in filtering. A good overview of the evaluation of filtering techniques is presented in [30]. In [31], Möller et al. further discuss imposing a smoothness requirement for interpolation and derivative filtering. In [22], the methods of nearest neighbor interpolation and cubic splines are compared. Hou and Andrews [23] specifically discuss the use of cubic B-splines for interpolation with applications to image processing. The interpolation consists of using a linear combination of five B-splines with the coefficients determined by the input data. This is a similar approach to the one discussed throughout this paper. Another method of filtering for visualization via convolution is presented in [31]. This method chooses an even number of filter weights for the convolution kernel to design a filter based on smoothness requirements. The authors also discuss classifying filters [31] and extend the analysis to the spatial domain. We can relate our filter to those evaluated by Mitchell and Netravali [25], where they design a reconstruction filter for images based on piecewise cubic filters, with the B-spline filter falling into this class. In [25], it is noted that a 2D separable filter is desirable, as is the case with the B-spline filter that we implement. Further discussion on spline filters can be found in [29], [32], and [33]. In [29], a cubic interpolation filter with a controllable tension parameter is discussed. This tension parameter can be adjusted in order to construct a filter that is monotonic and hence avoids overshoots. We neglect a discussion of monotonicity presently and leave it for future work.

The method that we implement uses B-splines but chooses the degree of B-spline to use based on smoothness requirements expected from the given sampled solution and uses this to aid in improved streamline integration. The mathematics behind choosing the appropriate convolution kernel is also presented. That is, if we expect for the given information to have k derivatives, then we use a *linear combination* of kth order B-splines. To be precise, we use $2k+1$ B-splines to construct the convolution filter. The coefficients of the linear combination of B-splines are well known and have shown to be unique [34], [35]. This linear combination of B-splines makes up our convolution kernel, which we convolve against our discrete data projected onto our chosen basis (such as the Legendre polynomial basis). This filtering technique has already demonstrated its effectiveness in postprocessing numerical simulations. In previous investigations, it not only filtered out oscillations contained in the error of the resulting simulation but also increased the accuracy of the approximation [5], [7], [36]. This technique has also been proven to be effective for derivative calculations [5].

The mathematical history behind the accuracy-increasing filter that we have implemented for improved streamline integration was originally proven to increase the order of accuracy for finite element approximation through postprocessing [4], with specific extensions to the DG method [5], [7], [36]. The solution spaces described by the DG method contain both finite volume and finite element schemes (as DG fields are piecewise polynomial (discontinuous) fields and thus contain as a subset piecewise polynomial C^0 fields), and hence, the aforementioned works provide us the theoretical foundations for a filtering method with which the order of accuracy could be improved up to $2k+1$ if piecewise polynomials of degree k are used in the DG approximation.

The mathematical structure of the postprocessor presented was initially designed by Bramble and Schatz [4] and Mock and Lax [37]. Further discussion of this technique will take place in Sections 4 and 5. Discussion of the application of this technique to the DG method can be found in [5], [7], and [36]. Most of this work will focus on symmetric filtering; however, in [37], the authors mention that the ideas can be imposed using a one-sided technique for postprocessing near boundaries or in the neighborhood of a discontinuity. The one-sided postprocessor implemented by Ryan and Shu [7] uses an idea similar to that of Cai et al., where a one-sided technique for the spectral Fourier approximation was explored [38].

There exists a close connection between this work and the work sometimes referenced as "macroelements." Although our work specifically deals with quadrilaterals and hexahedra, our future work is to move to fully unstructured (triangle/tetrahedral) meshes. In light of this, we will comment on why we think that there is sufficient previous work in macroelements to justify our future work. Macroelements, otherwise known as composite finite element methods, specifically the Hsieh-Clough-Tocher triangle split, are such that considering a set of triangles K, we subdivide each triangle into three subtriangles. We then have two requirements (from [39] and [40]):

$$P_K = \{p \in \mathcal{C}^1(K) : p|_{K_i} \in \mathbb{P}^3(K_i), 1 \leq i \leq 3\},$$
$$\Sigma_K = \{p(a_i), \partial_1 p(a_i), \partial_2 p(a_i), \partial_\nu p(b_i), 1 \leq i \leq 3\}.$$

Bramble and Schatz [4] proposed an extension to triangles using the following taken from the work of Bramble and Zlámal [41]; in particular, they devised a way to build interpolating polynomials over a triangle that respect the mathematical properties needed by the post-processor. These ideas are similar to the works in [42], [43], and [44]. For the proposed extension of this postprocessor to triangular meshes, convolving a *linear combination* of these basis functions for macroelements with the (low-order) approximation is used as a smoothing mechanism. This allows us to extract higher order information from a lower order solution.

3 Sources of Error and Error Control

In this section, we seek to remind the reader of the two main sources of approximation error in streamline computations: 1) solution error, which is the error that arises due to the numerical approximations that occur within a computational fluid mechanics simulation accomplished on a finite volume or finite element space, and 2) time-stepping (that is, ODE integration) error. We then discuss how adaptive error control attempts to handle the second of these errors.

3.1 Solution Error

The solution error is the difference (in an appropriately chosen norm) between the true solution and the approximate solution given by the finite volume or finite element solver. Consider what is involved, for instance, in finite element analysis. Given a domain Ω and a partial differential equation (PDE) that operates on a solution u that lives over Ω, the standard finite element method attempts to construct a geometric approximation $\tilde{\Omega} = \mathcal{T}(\Omega)$ consisting of a tessellation of polygonal shapes (for example, triangles and quadrilaterals for 2D surfaces) of the domain Ω and to build an approximating function space $\tilde{\mathcal{V}}$ consisting of piecewise linear functions based upon the tessellation [45]. Building on these two things, the goal of a finite element analysis is to find an approximation $\tilde{u} \in \tilde{\mathcal{V}}$ that satisfies the PDE operator in the Galerkin sense. The solution approximation error is thus a function of the richness of the approximation space (normally expressed in terms of element shape, element size, polynomial order per element, etc.) and the means of satisfying the differential or integral operator of interest (for example, Galerkin method). In the case of standard Galerkin (linear) finite elements, the approximation error goes as $\mathcal{O}(h^2)$, where h is a measure of the element size. Hence, if one were to decrease the element size by a factor of two, one would expect the error to decrease by a factor of four [45]. The use of high-order basis functions can admit $\mathcal{O}(h^k)$ convergence [46] (where h is a measure of element size, and k denotes the polynomial order used per element).

Even if analytic time-stepping algorithms were to exist, these approximation errors would still cause deviation between the "true" streamline of a field and a streamline produced on the numerical approximation of the answer. The filters we seek to employ are filters that do not increase this error (that is, are accuracy conserving)—in fact, the filters we will examine can under certain circumstances improve the approximation error.

3.2 Time-Stepping Error

In addition to the solution approximation error as previously mentioned, we also must contend with the time-stepping error introduced by the ODE integration scheme that we choose to employ. To remind the reader of the types of conditions upon which most time integration techniques are based and provide one of the primary motivations of this work, we present an example given in the classic time-stepping reference [47]. Consider the following second-order Runge-Kutta scheme as applied to a first-order homogeneous differential equation [47, p. 133]:

$$k_1 = f(t_0, y_0), \tag{1}$$

$$k_2 = f\left(t_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_1\right), \tag{2}$$

$$y_1 = y_0 + hk_2, \tag{3}$$

where h denotes the time step, y_0 denotes the initial condition, and y_1 denotes the approximation to the solution given at time level $t_0 + h$. To determine the order of accuracy of this scheme, we substitute (1) and (2) into (3) and accomplish the Taylor series to yield the following expression:

$$y(t_0 + h) - y_1 = \frac{h^3}{24}\Big(f_{tt} + 2f_{ty}f + f_{yy}f^2 + 4(f_t f_y + f_y^2 f)\Big)(t_0, y_0) + \ldots,$$

where it is assumed that all the partial derivatives in the above expression exist. This leads us to the application of Theorem 3.1 [47, p. 157] applied to this particular scheme, which states that this scheme is of order $k = 2$ if all the partial derivatives of $f(t, y)$ up to order k exist (and are continuous) and that the local truncation error is bounded by the following expression:

$$\|y(t_0 + h) - y_1\| \leq Ch^{k+1},$$

where C is a constant independent of the time step. The key assumption for this convergence estimate and all convergence estimates for both multistep (for example, Adams-Bashforth and Adams-Moulton) and multistage (for example, Runge-Kutta) schemes is the smoothness of the right-hand-side function in terms of the derivatives of the function. Hence, the regularity of the function (in the derivative sense) is the key feature necessary for high-order convergence to be realized.

Streamline advection through a finite element or finite volume data set can be written as the solution of the ODE system:

$$\frac{d}{dt}\vec{x}(t) = \vec{F}(\vec{x}(t)),$$
$$\vec{x}(t = 0) = \vec{x}_0,$$

where $\vec{F}(\vec{x})$ denotes the (finite volume or finite element) vector-valued function of interest, and $\vec{x}_0$ denotes the point at which the streamline is initiated. Note that although the streamline path $\vec{x}(t)$ may be smooth (in the mathematical sense), this does not imply that the function $\vec{F}(x(t))$ is smooth. Possible limitations in the regularity of $\vec{F}$ directly

impact the set of appropriate choices of ODE time-stepping algorithms that can be successfully applied [47].

3.3 Adaptive Error Control

One suite of techniques often employed in the numerical solution of ODEs is time stepping with adaptive error control (for example predictor-corrector and RK-4/5). As pointed out in [32], such techniques can be effectively utilized in visualization applications that require time stepping. It is important, however, to note two observations that can be made when considering the use of error control algorithms of this form. First, almost all error control algorithms presented in the ODE literature are based upon the Taylor series analysis of the error and hence tacitly depend upon the underlying smoothness of the field being integrated [47]. Thus, error control algorithms can be impacted by the smoothness of the solution in the same way as previously discussed. The second observation is that the error indicators used, such as the one commonly employed in RK-4/5 [48] and ones used in predictor-corrector methods, will often require *many* failed steps to "find" and integrate over the discontinuity with a time step small enough to reduce the local error estimate below the prescribed error tolerance [3]. This is because no matter the order of the method, there will be a fundamental error contribution that arises due to integrating over discontinuities. If a function f has a discontinuity in the pth derivative of the function (that is, $f^{(p)}$ is discontinuous), the error when integrating past this discontinuity is on the order of $C[[f^{(p)}]](\Delta t)^{p+1}$, with $p \geq 0$, C being a constant, and $[[\cdot]]$ being the size of the jump. Thus, streamline integration over a finite volume field having discontinuities at the element interfaces (that is, $p = 0$ on element boundaries) is formally limited to first-order accuracy at the interfaces. Adaptive error control attempts to find, through an error estimation and time-step refinement strategy, a time step sufficiently small that it balances the error introduced by the jump term.

These two observations represent the main computational limitation of adaptive error control [3]. The purpose of this work is to examine whether the cost of using SIAC filters as a preprocessing stage can decrease the number of refinement steps needed for adaptive error control and thus increase the general efficiency of streamline integration through discontinuous fields.

4 Theoretical Discussion of Smoothness-Increasing Accuracy-Conserving Filters

In this section, we present a mathematical discussion of the filters we employ in this work. Unlike the mathematics literature in which they were originally presented, we do not require that the filtering be accuracy increasing. It is only a necessary condition for our work that the filters be smoothness-increasing and accuracy-conserving. In this section, in which we review the mathematics of the filter, we will explain these filters in light of the original motivation for their creation (in the mathematics literature)—increasing the accuracy of DG fields.

As noted in Section 2, the accuracy-conserving filter that we implement (provably) reduces the error in the L^2 norm and increases the smoothness of the approximation. The L^2 norm is the continuous analogy of the commonly used discrete root-mean-square (weighted euclidean) norm [49]. Unless otherwise noted, our discussion of accuracy is in reference to the minimization of errors in the L^2 norm. We have made this choice as it is the norm used in the theoretical works referenced herein and is a commonly used norm in the finite volume and finite element literature. The work presented here is applicable to general quadrilateral and hexagonal meshes [6]. However, in this paper, we focus on uniform quadrilateral meshes as the presentation of the concepts behind the postprocessor can be numerically simplified if a locally directionally uniform mesh is assumed, yielding translation invariance and subsequently localizing the postprocessor. This assumption will be used in our discussion and will be the focus of the implementation section (Section 5).

The postprocessor itself is a convolution of the numerical solution with a linear combination of B-splines and is applied after the final numerical solution has been obtained and before the streamlines are calculated. It is well known within the image-processing community that convolving data with B-splines increases the smoothness of the data and, if we are given the approximation itself, the order of accuracy the numerical approximation [22], [23], [30]. Furthermore, we can increase the effectiveness of filtering by using a linear combination of B-splines. Exactly how this is done is based on the properties of the convolution kernel. Indeed, since we are introducing continuity at element interfaces through the placement of the B-splines, smoothness is increased, which in turn aids in improving the order of accuracy depending on the number of B-splines used in the kernel and the weight of those B-splines. This work differs from previous work in visualization that utilizes B-spline filtering in that we are using a *linear combination* of B-splines and in that we present a discussion of formulating the kernel for general finite element or finite volume approximation data to increase smoothness and accuracy. As long as the grid size is sufficiently large to contain the footprint of the filter, SIAC properties will hold. Given the asymptotic results presented in [4] and [5], there will be some minimum number of elements N above which accuracy enhancement will also occur.

To form the appropriate kernel to increase smoothness, we place certain assumptions on the properties of the convolution kernel. These assumptions are based upon the fact that finite element and finite volume approximations are known to have errors that are oscillatory; we would like to filter this error in such a way that we increase smoothness and improve the order of accuracy. The first assumption that allows for this is a kernel with compact support that ensures a *local* filter operator that can be implemented efficiently. This is one of the reasons why we have selected a linear combination of B-splines. Second, the weights of the linear combination of B-splines are chosen so that they reproduce polynomials of degree $2k$ by convolution, which ensures that the order of accuracy of the approximation is not destroyed. Last, the linear combination of B-splines also ensures that the derivatives of the convolution kernel can be expressed as a difference of quotients. This property will be useful in future work when alternatives to the current derivative filtering [5] will be explored for this postprocessor. The main option for high-order filtering follows from the work of Thomée [34] and is similar to that of Möller et al. [31].

In order to further the understanding of the postprocessor, we examine the one-dimensional (1D) postprocessed solution. This solution can be written as

$$f^*(x^*) = \frac{1}{h}\int_{-\infty}^{\infty} K^{2(k+1),k+1}\left(\frac{y-x^*}{h}\right) f_h(y)\,dy, \tag{4}$$

where $K^{2(k+1),k+1}$ is a linear combination of B-splines, f_h is the numerical solution consisting of piecewise polynomials, and h is the uniform mesh size. We readily admit that the notation we have used for the kernel appears awkward. It has been chosen to be identical to the notation used in the theoretical works upon which this work is based (for example, [36]) to assist those interested in connecting this application work with its theoretical foundations. As this is the common notation used in that area of mathematics, we hope to motivate it so that people will appreciate the original notation. The rationale behind the notation $K^{a,b}$ is that a denotes the order of accuracy that we can expect from postprocessing and b denotes the order of accuracy of the numerical method (before postprocessing).

The symmetric form of the postprocessing kernel can be written as

$$K^{2(k+1),k+1}(y) = \sum_{\gamma=-k}^{k} c_\gamma^{2(k+1),k+1}\psi^{(k+1)}(y-\gamma), \tag{5}$$

where B-splines of order $(k+1)$ is obtained by convolving the characteristic function over the interval $[-\frac{1}{2},\frac{1}{2}]$ with itself k times. That is

$$\psi^{(1)} = \delta_0|_{[-1/2,1/2]}, \quad \psi^{(k+1)} = \psi^{(k)} * \delta_0|_{[-1/2,1/2]}. \tag{6}$$

The coefficients, $c_\gamma^{2(k+1),k+1}$, in (5) are scalar constants obtained from the need to at least maintain the accuracy of the approximation. That can be achieved by

$$K^{2(k+1),k+1} * p = p, \quad \text{for } p = 1, x, x^2, \cdots, x^{2k}, \tag{7}$$

as shown in [36] and [50]. We note that this makes the sum of the coefficients for a particular kernel equal to one:

$$\sum_{\gamma=-k}^{k} c_\gamma^{2(k+1),k+1} = 1.$$

Once we have obtained the coefficients and the B-splines, the form of the postprocessed solution is then

$$f^*(x^*) = \frac{1}{h}\int_{-\infty}^{\infty}\sum_{\gamma=-k}^{k} c_\gamma^{2(k+1),k+1}\cdot\psi^{(k+1)}\left(\frac{y-x^*}{h}-\gamma\right) f_h(y)\,dy. \tag{8}$$

Although the integration in (8) is shown to be over the entire real line, due to the compact nature of the convolution kernel, it will actually just include the surrounding elements. A more specific discussion of this detail will take place in the next section.

As an example, consider forming the convolution kernel for a quadratic approximation, $k=2$. In this case, the convolution kernel that we apply is given by

$$\begin{aligned} K^{(6,3)}(x) = {} & c_{-2}^{6,3}\psi^{(3)}(x+2) + c_{-1}^{6,3}\psi^{(3)}(x+1) + c_0^{6,3}\psi^{(3)}(x) \\ & + c_1^{6,3}\psi^{(3)}(x-1) + c_2^{6,3}\psi^{(3)}(x-2). \end{aligned} \tag{9}$$

The B-splines are obtained by convolving the characteristic function over the interval $[-1/2,1/2]$ with itself k times. In this case, we have

$$\psi^{(3)}(x) = \delta_0 * \delta_0 * \delta_0 = \begin{cases} \frac{1}{8}(4x^2+12x+9), & -\frac{3}{2}\le x\le -\frac{1}{2}, \\ \frac{1}{8}(6-8x^2), & -\frac{1}{2}\le x\le \frac{1}{2}, \\ \frac{1}{8}(4x^2-12x+9), & \frac{1}{2}\le x\le \frac{3}{2}. \end{cases} \tag{10}$$

The coefficients, $c_\gamma^{2(k+1),k+1}$, are found from convolving the kernel with polynomials of degree up to $2k$ $(K*p=p)$. The existence and uniqueness of the coefficients is discussed in [36]. In this case, $p=1,x,x^2,x^3,x^4$. This gives us a linear system whose solution is

$$\begin{bmatrix} c_{-2} \\ c_{-1} \\ c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} \frac{37}{1920} \\ -\frac{97}{480} \\ \frac{437}{320} \\ -\frac{97}{480} \\ \frac{37}{1920} \end{bmatrix}.$$

Notice the symmetry of the B-splines, as well as the coefficients multiplying them. Now that we have determined the exact form of the kernel, we can examine the kernel itself. The 1D function is pictured in the top of Fig. 1. Notice that the emphasis of the information is on the current element being postprocessed, and the neighboring cells contribute minimally to the filtering process.

Fig. 1 plots not only the 1D convolution kernel used in the quadratic approximation but also the projection of the postprocessing mesh (that is, the support of the kernel, where mesh lines correspond to the knot lines of the B-splines used in the kernel construction) onto the approximation mesh in two dimensions. That is, the top mesh shows the mesh used in the quadratic approximation, $f_h(x,y)$, obtained by the finite element or finite volume approximation with the mesh containing the support of the B-splines used in the convolution kernel, $K^{(6,3)}(x,y)$ (given above), superimposed (shaded area). The bottom mesh is the supermesh that we use in order to obtain the filtered solution, $f^*(x^*,y^*)$. Integration is implemented over this supermesh as an accumulation of the integral over each supermesh element. In the lower image in Fig. 1, we picture one element of the supermesh used in the postprocessing with the Gauss points used in the integration. Note that Gauss integration on each supermesh element admits exact integration (to machine precision) since both the kernel and approximation are polynomials over the subdomain of integration.

We note that our results are obtained by using the symmetric kernel discussed above. Near the boundary of a computational domain, a discontinuity of the vector field, or an interface of meshes with different cell sizes, this symmetric kernel should not be applied. As in finite difference methods, the accuracy-conserving filtering technique can easily be extended by using one-sided stencils [7]. The one-sided version of this postprocessor is performed by simply moving the support of the kernel to one side of the current element being postprocessed. The purely one-sided postprocessor has a form similar to the centered one, with different bounds on the summation and new coefficients

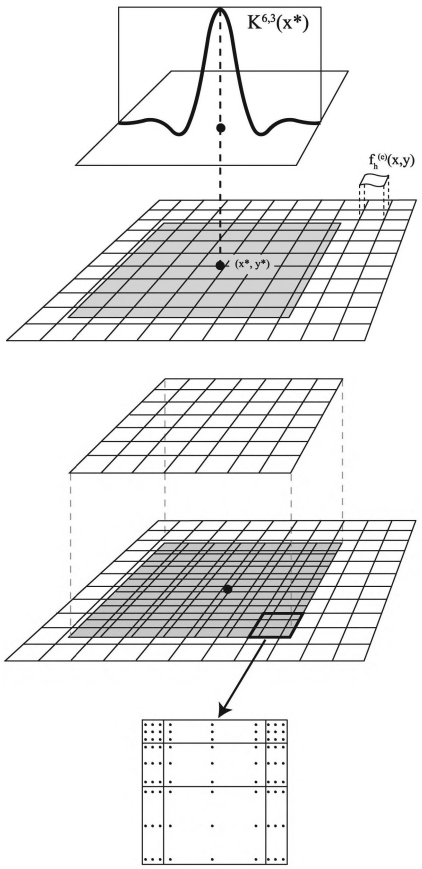

Fig. 1. The convolution kernel for a 2D quadratic approximation when element $I_{i,j} = [x_i - \frac{h_x}{2}, x_i - \frac{h_x}{2}] \times [y_j - \frac{h_y}{2}, y_j - \frac{h_y}{2}]$ is postprocessed. The B-spline mesh is shown (shaded area), as well as the approximation mesh (in white). The bottom figure represents one quadrilateral element and the points at which the postprocessor is evaluated.

$c_\gamma^{2k+1,2k}$ [7]. Similarly, a partially left one-sided postprocessor is obtained by changing the bounds of the summation. In each case, the use of $2k+1$ B-splines remains consistent. The right one-sided postprocessed solution is a mirror image of the left.

The 2D symmetric kernel is a product of the 1D kernels with the form

$$K^{2(k+1),k+1}(x^*, y^*) = \sum_{\gamma_x=-k}^{k} c_{\gamma_x}^{2(k+1),k+1} \psi^{(k+1)}(x^* - \gamma_x) \cdot \sum_{\gamma_y=-k}^{k} c_{\gamma_y}^{2(k+1),k+1} \psi^{(k+1)}(y^* - \gamma_y), \tag{11}$$

[5]. The kernel is suitably scaled by the mesh size in both the x- and y-directions. The same coefficients, $c_\gamma^{2(k+1),k+1}$, as well as B-splines, are used for both directions. This easily extends to three dimensions as well (the kernel will be a tensor product of three B-splines).

4.1 Putting It Together

Putting this filtering process together with the streamline algorithm for a 2D quadratic approximation, we would execute the following algorithm:

- Obtain the approximation over a given quadrilateral or hexagonal mesh using a finite element or finite volume scheme (Fig. 1).
- Form the postprocessing kernel using a linear combination of B-splines, scaled by the mesh size. This will form the postprocessing mesh (Fig. 1, shaded area). The intersection of these two meshes forms more subelements (Fig. 1, bottom).
  - Obtain the B-spline from $\psi^{(3)} = \delta_0 * \delta_0 * \delta_0$.
  - Obtain the appropriate coefficients from $K * p = p$ for $p = 1, x, x^2, x^3, x^4$.
- Given the kernel for this quadratic function, use RK-4 (or your favorite ODE solver) to integrate the streamline, where the right-hand-side function is given by

$$f^*(x^*, y^*) = \frac{1}{h_x h_y} \int_{\mathbb{R}^2} \sum_{\gamma_x=-3}^{3} c_{\gamma_x}^{6,3} \psi^{(3)}\left(\frac{z_1 - x^*}{h_x} - \gamma_x\right) \cdot \sum_{\gamma_y=-3}^{3} c_{\gamma_y}^{6,3} \psi^{(3)}\left(\frac{z_2 - y^*}{h_y} - \gamma_y\right) f_h(z_1, z_2)\, dz_1 dz_2. \tag{12}$$

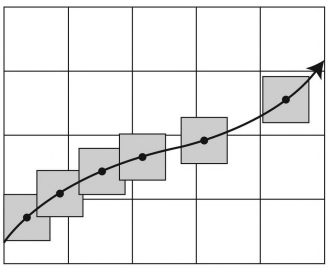

Fig. 2. Diagram showing the time integration along a streamline as it crosses element boundaries.

In Fig. 2, the time integration of a streamline as it crosses element boundaries is shown. By using this postprocessor as part of the time integration step, we have the ability to impose a type of interelement continuity at the evaluation points of the integration (denoted by the circles) by using the solution information in the neighborhood of the point (denoted by the shaded regions—that is, by using the information in the footprint of the kernel) to regain convergence.

5 Implementation of Smoothness-Increasing Accuracy-Conserving Filters

When computing more than one streamline in a given field, it is often more efficient to postprocess the entire field ahead of time. This allows us to simplify the implementation of the postprocessing kernel given in (5) by exploiting the symmetry of the postprocessor and the B-spline coefficients [5], [7], [36]. There is an additional advantage in the local behavior of the postprocessor, specifically that the kernel needs information only from its nearest neighbors and can be implemented by doing small matrix-vector multiplications [5]. The matrix-vector format in the modal basis in one dimension is found from evaluating

$$f^*(x^*) = \frac{1}{h}\sum_{j=-k'}^{k'}\int_{I_{i+j}} K^{2(k+1),k+1}\left(\frac{y-x^*}{h}\right) \cdot \sum_{l=0}^{k} f_{i+j}^{(l)}\phi_{i+j}^{(l)}(y)\,dy \qquad (13)$$
$$= \sum_{j=-k'}^{k'}\sum_{l=0}^{k} f_{i+j}^{(l)} C(j,l,k,x^*),$$

where

$$C(j,l,k,x^*) = \frac{1}{h}\sum_{\gamma=-k}^{k} c_\gamma^{2(k+1),k+1}\int_{I_{i+j}} \psi^{(k+1)}(\eta)\phi_{i+j}^{(l)}(y)\,dy,$$

$\eta = \frac{y-x^*}{h} - \gamma$, and $\phi_i^{(l)}$ are the basis functions of the projected function on cell $I_i = (x_i - \frac{h}{2}, x_i + \frac{h}{2})$. The modes on element I_i are given by $f_i^{(l)}$, $l = 0, \cdots, k$. We further note that the compact support of the $2k+1$ B-splines reduces the area of the integral, which leads to a total support of $2k'+1$ elements, where $k' = \lceil \frac{3k+1}{2} \rceil$.

Furthermore, to provide ease of computation, we can choose to compute the postprocessed solution at specific points within an element. This allows us to calculate the postprocessing matrix coefficients ahead of time and store the values for future calculations. It also simplifies the application in two and three dimensions, which are just tensor products of the 1D postprocessor.

Consider a 2D application that uses four Gauss points per element (two in each direction, denoted z_n, $n = 1, 2$). The matrix-vector representation to postprocess element $I_{i,j} = [x_i - \frac{h_x}{2}, x_i - \frac{h_x}{2}] \times [y_j - \frac{h_y}{2}, y_j - \frac{h_y}{2}]$ would be

$$f^*(z_n(x), z_n(y)) = \sum_{m_x=-k'}^{k'}\sum_{m_y=-k'}^{k'}\sum_{l_x=0}^{k}\sum_{l_y=0}^{k} f_{i+m_x,j+m_y}^{(l_x,l_y)} \cdot C(m_x,l_x,k,z_n(x)) \cdot C(m_y,l_y,k,z_n(y)). \qquad (14)$$

Notice that $C(m_x, l_x, k, z_n(x))$ and $C(m_y, l_y, k, z_n(y))$ contain the same matrix values. The only new information plugged into calculating the postprocessed solution is the data provided on element $I_{i,j}$ and surrounding elements. That is, the value of the data $f_{i,j}$.

We rely on a 2D quadratic approximation over the element $I_{i,j}$ for further explanation. In this case, the approximation is given by

$$f_h(x,y) = f^{(0,0)} + f^{(1,0)}\phi_i^{(1)}(x) + f^{(0,1)}\phi_j^{(1)}(y) + f^{(1,1)}\phi_i^{(1)}(x)\phi_j^{(1)}(y) + f^{(2,0)}\phi_i^{(2)}(x) + f^{(0,2)}\phi_j^{(2)}(y), \qquad (15)$$

where the basis of the approximation is denoted with $\phi_i(x)$, $\phi_j(y)$. In many DG simulations, this basis is chosen to be the Legendre basis, although mathematically it is not required to be. Let us denote by $f_{i,j}$ the vector $[f_{i,j}^{(0,0)}, f_{i,j}^{(0,1)}, f_{i,j}^{(0,2)}, f_{i,j}^{(1,0)}, f_{i,j}^{(1,1)}, f_{i,j}^{(2,0)}]^T$. We then proceed as follows:

- Obtain the five coefficients and the quadratic B-splines outlined in Section 4.
- For $n = 1, 2$, evaluate the integral

$$C(j,l,2,x_n) = \frac{1}{h}\sum_{\gamma=-2}^{2} c_\gamma^{6,3}\int_{I_{i+j}} \psi^{(3)}(\eta)\phi_{i+j}^{(l)}(y)\,dy,$$

at the Gauss points, x_n. Denote this matrix C.

- For each element in our 2D mesh, the postprocessed solution at the Gauss points would be the matrix-vector multiplication:

$$f^*(x_n, y_n) = \sum_{m_x=-3}^{3}\sum_{m_y=-3}^{3} f_{i+m_x,j+m_y} C_x C_y. \qquad (16)$$

We note that it is only necessary to compute the postprocessing matrix, C, once, even though we apply it in each direction and on every element. This is the advantage of the directionally uniform mesh assumption. As is shown in [6], the postprocessing takes $\mathcal{O}(N)$ operations to filter the entire field, where N is the total number of elements. Each field presented in this paper was postprocessed in less than 5 seconds. Once the postprocessing is accomplished, the only cost difference between the original and the postprocessed field is the cost of the field evaluation. From our analysis, the cost function for evaluation of the field goes as $C_1 P^d + C_2 P$, where P is the polynomial degree of the field, d is the dimension of the field, and C_1 and C_2 are constants independent of the degree. The first term relates to the cost of the linear combination; the second term relates to the cost of evaluating the polynomials. It is worth noting that the postprocessed field has twice the number of modes per dimension as the unprocessed field, and thus, the cost of each postprocessed field evaluation is asymptotically 2^d times the cost of an unprocessed field evaluation.

6 Results

To demonstrate the efficacy of the proposed filtering methodology when used in conjunction with classic streamlining techniques, we provide the following tests on a set of 2D vector fields. We provide these illustrative results on 2D quadrilateral examples because it is easy to see the ramifications of the filtering (as exact solutions are available by construction and hence allow us to do convergence tests); everything we have discussed and will demonstrate holds true for 3D hexahedral examples also.

The three vector field examples $(u, v)^T = \vec{F}(x, y)$ presented in this paper can be replicated using the set of equations given below; they were obtained from [51]. The domain of interest in all three cases is $[-1, 1] \times [-1, 1]$, and the range is a subspace of $\mathbb{R}^2$. Each field is then projected (using an L^2 Galerkin projection) onto an evenly spaced quadrilateral high-order finite volume mesh—a procedure that mathematically mimics the results of a DG simulation. For details as to the mathematical properties and numerical issues of such projections, we refer the reader to [49].

Once projected to a (high-order) finite volume field, we can then apply the postprocessor in an attempt to regain smoothness at element interfaces and to increase accuracy (in the sense of the minimization of the L^2 norm of the difference between the true and the approximate solution). For simplification reasons, the comparisons in this paper

TABLE 1
The L^2 and L^∞ Errors for the U and V Components of Field 1

U component

	L^2 error		L^∞ error	
N	BEFORE	AFTER	BEFORE	AFTER
	$\mathbb{P}^1$			
20	1.2642E-02	4.8779E-04	1.3028E-01	2.0830E-03
40	4.4291E-03	3.8597E-05	4.8341E-02	1.7929E-04
80	1.3054E-03	2.7114E-06	1.7165E-02	1.3033E-05
	$\mathbb{P}^2$			
20	2.2576E-04	6.8329E-06	1.8986E-03	1.3061E-05
40	5.0880E-05	1.4086E-07	5.4698E-04	2.6435E-07
80	8.4056E-06	2.4689E-09	9.9905E-05	4.6007E-09

V component

	L^2 error		L^∞ error	
N	BEFORE	AFTER	BEFORE	AFTER
	$\mathbb{P}^1$			
20	1.8593E-02	8.2343E-04	3.5945E-01	3.5262E-03
40	6.8577E-03	6.4599E-05	1.2345E-01	2.9945E-04
80	2.0597E-03	4.4989E-06	4.2493E-02	2.1573E-05
	$\mathbb{P}^2$			
20	2.5455E-04	1.1907E-05	4.0543E-03	2.1461E-05
40	6.5659E-05	2.0750E-07	1.1096E-03	3.9563E-07
80	1.1284E-05	3.4297E-09	1.9817E-04	6.6518E-09

Results are shown for before and after postprocessing.

TABLE 3
The L^2 and L^∞ Errors for the U and V Components of Field 3

U component

	L^2 error		L^∞ error	
N	BEFORE	AFTER	BEFORE	AFTER
	$\mathbb{P}^1$			
20	5.8599E-03	2.0096E-04	5.8993E-02	6.5475E-04
40	2.0326E-03	1.4980E-05	2.1084E-02	5.1405E-05
80	5.9359E-04	1.0187E-06	6.2881E-03	3.5671E-06
	$\mathbb{P}^2$			
20	9.3092E-05	4.8121E-06	1.0252E-03	4.8288E-06
40	1.9834E-05	7.5189E-08	2.1987E-04	7.5451E-08
80	3.1548E-06	1.1748E-09	3.4352E-05	1.1789E-09

V component

	L^2 error		L^∞ error	
N	BEFORE	AFTER	BEFORE	AFTER
	$\mathbb{P}^1$			
20	7.3409E-03	7.3190E-05	8.9810E-02	2.5865E-04
40	2.6545E-03	5.0130E-06	3.4032E-02	1.8700E-05
80	7.8884E-04	3.2750E-07	1.1066E-02	1.2361E-06
	$\mathbb{P}^2$			
20	1.1326E-04	4.5073E-16	1.5893E-03	7.9936E-15
40	2.6867E-05	1.4891E-15	3.6759E-04	2.3981E-14
80	4.4509E-06	2.7195E-15	5.8067E-05	4.7962E-14

Results are shown for before and after postprocessing.

were limited to those regions of $[-1,1]^2$ for which the symmetric postprocessing kernel remains within the domain. In general, the entire domain can (and should) be postprocessed using a combination of the symmetric and one-sided kernels as described in [7].

In Tables 1, 2, and 3, we present errors measured in the L^2 norm and a discrete approximation of the L^∞ norm (sampled on a 1,000 $\times$ 1,000 uniform sampling of the domain) for both the projected and postprocessed solutions. The errors are calculated by comparing the projected and postprocessed solutions against the analytic fields. This error calculation is performed for various numbers of uniform element partitions (N in each dimension) and polynomial orders ($\mathbb{P}^k$). Both u and v components are provided individually.

The three analytic fields used as examples are all of the form:

$$z = x + \imath y,$$
$$u = Re(r),$$
$$v = -Im(r).$$

Example Field 1

$$r = (z-(0.74+0.35\imath))(z-(0.68-0.59\imath))(z-(-0.11-0.72\imath))(\bar{z}-(-0.58+0.64\imath))(\bar{z}-(0.51-0.27\imath))(\bar{z}-(-0.12+0.84\imath))^2.$$

Example Field 2

$$r = (z-(0.74+0.35\imath))(\bar{z}+(-0.18-0.19\imath))(z-(-0.11-0.72\imath))(\bar{z}-(-0.58+0.64\imath))(\bar{z}-(0.51-0.27\imath)).$$

Example Field 3

$$r = -(z-(0.74+0.35\imath))(z-(0.11-0.11\imath))^2(z-(-0.11+0.72\imath))(z-(-0.58+0.64\imath))(\bar{z}-(0.51-0.27*\imath)).$$

As predicted by the theory, we see a decrease in the L^2 error for all applications of the postprocessor. In addition, we observe that when the postprocessing order matches the original polynomial order of the analytical solution, we regain the solution to machine precision (as predicted by the theory). We note that this represents the "ideal" case—when the filter is able to regain the exact solution. We also observe a benefit in the L^∞ norm—something not explicitly given

TABLE 2
The L^2 and L^∞ Errors for the U and V Components of Field 2

U component

	L^2 error		L^∞ error	
N	BEFORE	AFTER	BEFORE	AFTER
	$\mathbb{P}^1$			
20	2.7581E-03	7.8928E-05	2.4325E-02	1.4777E-04
40	9.2500E-04	5.5474E-06	7.9316E-03	1.0683E-05
80	2.6559E-04	3.6589E-07	2.5096E-03	7.1213E-07
	$\mathbb{P}^2$			
20	4.6335E-05	1.7024E-16	2.8118E-04	2.6645E-15
40	9.2945E-06	4.4493E-16	6.2435E-05	8.4377E-15
80	1.4251E-06	9.8346E-16	9.9100E-06	2.0428E-14

V component

	L^2 error		L^∞ error	
N	BEFORE	AFTER	BEFORE	AFTER
	$\mathbb{P}^1$			
20	3.2476E-03	8.4842E-05	3.9655E-02	1.7589E-04
40	1.0626E-03	5.8785E-06	1.2347E-02	1.2440E-05
80	3.0191E-04	3.8556E-07	3.8142E-03	8.2192E-07
	$\mathbb{P}^2$			
20	4.8614E-05	3.9956E-16	4.2340E-04	9.3259E-15
40	9.6142E-06	8.1733E-16	8.6459E-05	1.6875E-14
80	1.4665E-06	1.9159E-15	1.3251E-05	3.9080E-14

Results are shown for before and after postprocessing.

TABLE 4
Error Comparison in Critical Point Location

Field 1				
	N = 20		N = 40	
Critical Point	Before	After	Before	After
(0.74, 0.35)	0.008387	0.000995	0.003549	0.000052
(0.68, -0.59)	0.00268	0.000384	0.000802	0.000027
(-0.11, -0.72)	1.977954	0.006531	0.003356	0.000428
(-0.58, -0.64)	0.001955	0.000894	0.001471	0.000065
(0.51, 0.27)	0.002401	0.000938	0.001026	0.000062
Field 2				
	N = 20		N = 40	
Critical Point	Before	After	Before	After
(0.74, 0.35)	0.008305	0.000392	0.003321	0.000021
(0.18, -0.19)	0.000198	0.000093	0.000270	0.000006
(-0.11, -0.72)	0.001504	0.000197	0.000589	0.000014
(0.58, 0.64)	0.001049	0.000134	0.000772	0.000009
(0.51, 0.27)	0.002616	0.000481	0.000970	0.000032

Results are shown for before and after postprocessing.

by the theory but something upon which many of the theoretical papers have commented is likely and has been observed empirically [4], [5].

One of the nice consequences of the reduction in the L^∞ error of the fields is that the computation of field "features" such as critical points are not hampered. As an example, we computed a sampling of critical point locations for two of the previously mentioned fields. Critical points were computed using a Newton-Raphson algorithm with a finite differencing of the field values for creating the Jacobian matrix [48]. As the exact position of the critical points are known, we can compare in the standard euclidean norm the distance between the exact and computed critical point location. In Table 4, we present the results of a collection of computed critical point locations based upon the projected and the postprocessed field. Note that in general, the postprocessed field does no worse than the original field.

In Fig. 3, we present three vector field visualizations produced by projecting the functions above over a 40×40 uniform mesh on the interval $[-1,1] \times [-1,1]$. The field approximations are linear in both the x- and y-directions. The "true-solution" streamlines (denoted as black lines in all three images) are calculated by performing RK-4/5 on the analytical function. The blue streamlines represent the streamlines calculated from the L^2-projection of the field. The red streamlines represent the streamlines calculated from the postprocessed projection of the field. All streamlines were calculated using RK-4/5 with an error tolerance of 10^{-6}.

Note that in the cases presented, the streamline based upon the postprocessed data more closely follows the true solution. In these cases, we also observe that in regions where bifurcations occur, the postprocessed solution follows the true solution instead of diverging away like the non-postprocessed solution. This behavior can be attributed to the projection accuracy obtained due to the use of the postprocessor.

To ameliorate the accuracy and smoothness issues induced by jumps at element interfaces in the L_2-projected field, the Runge-Kutta-Fehlberg adaptive-error-controlled RK-4/5 time-stepping method [52] was used with an error tolerance of 10^{-6} and no min/max step sizes. This tolerance was chosen as a representative of what would be selected if one wanted streamlines that were accurate to single-precision machine zero. The smoothness of the analytical function and the

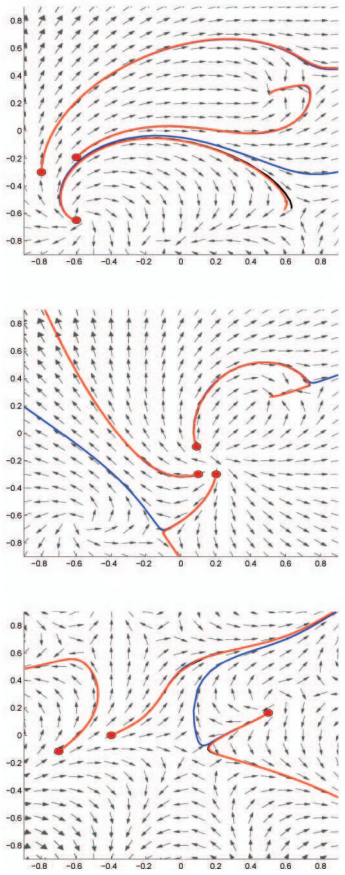

Fig. 3. Three streamline integration examples based upon vector fields generated using the methodology presented in [51]. Black streamlines denote "exact" solutions, blue streamlines were created based upon integration on an L_2-projected field, and red streamlines were created based upon integration on a filtered field. In cases where the black streamline is not visible, it is because the red streamline lines obfuscates it. The streamline seed points are denoted by circles. Specific details concerning the projected fields are given in the text.

postprocessed field would allow for efficient integration using a standard RK-4 method; however, for comparison, integration on all fields was performed using the same adaptive RK-4/5 method with the same tolerances. Table 5 shows the number of accepted RK-4/5 steps and total number of RK-4/5 steps (accepted plus rejected) required to compute the streamlines in Fig. 3. In Table 6, we provide a timing comparison based upon our nonoptimized Matlab implementation of the streamline algorithms running on an Intel Pentium 4 3.2-GHz machine. Note that the ratio of filtered to nonfiltered time per integration step is much less than the ratio of the total number of integration steps required. That is, even though the cost per integration step is greater on the postprocessed field, the reduction in the total number of integration steps required more than compensates for this difference.

For most streamlines, the total number of RK-4/5 steps[1] required for the postprocessed field is comparable to the number of steps required for the analytical function. For this error tolerance, the total number of steps required for the L_2-projected field is asymptotically four times greater than

1. Here, "step" does not refer to a time step—the advancement of time—but rather the execution of all the stages of the RK algorithm and error estimation.

TABLE 5
Number of RK-4/5 Steps Required to Calculate Different Streamlines on the Three Different Fields with the Error Tolerance Set to 10^{-6}

Streamline	No. of Accepted Steps			Total No. of Steps		
	Exact	L_2	Post	Exact	L_2	Post
Field 1						
1	36	177	33	43	523	42
2	159	132	47	211	358	54
3	52	722	65	64	2476	93
Field 2						
1	26	118	25	34	398	35
2	86	65	41	106	135	45
3	27	89	27	46	263	49
Field 3						
1	34	65	32	54	206	58
2	31	137	31	42	499	44
3	29	50	28	43	129	46

TABLE 7
Number of RK-3/4 Steps Required to Calculate Different Streamlines on the Three Different Fields with the Error Tolerance Set to 10^{-6}

Streamline	No. of Accepted Steps			Total No. of Steps		
	Exact	L_2	Post	Exact	L_2	Post
Field 1						
1	230	1897	229	237	3310	239
2	267	1562	264	277	2657	276
3	424	2070	560	429	3156	575
Field 2						
1	123	1609	125	125	2869	133
2	208	1190	198	216	2046	206
3	290	1208	288	291	2389	293
Field 3						
1	253	1675	249	255	3385	256
2	288	1075	289	288	1862	293
3	242	1337	243	242	2685	248

the number required for the postprocessed field (recall from Section 5 that the postprocessed solution is asymptotically four times more expensive to evaluate; hence, a computational win is attained when the postprocessed solution takes less than four times the number of steps. In Table 6, we see that we are not in the asymptotic worst case as the cost of evaluating the postprocessed solution is only twice that of the unfiltered field). This discrepancy is again due to the RK-4/5 method drastically reducing the time step and rejecting many steps to find discontinuities in the underlying field. When tighter error tolerances are used or when coarser discretizations are examined (which cause larger jumps in the solution at element boundaries), the discrepancy between the number of RK-4/5 steps required grows, and likewise, with looser error tolerances or refined meshes, the discrepancy decreases. To compare with another integration method, Table 7 shows the results for the same tests with the RK-3/4 method used by Stalling and Hege [19]. The difference between the total number of steps required for the L_2-projected field and the postprocessed field is much greater with RK-3/4 than with RK-4/5.

To further illustrate how discontinuities at element boundaries affects the performance of RK-4/5, Fig. 4 shows the cumulative number of RK-4/5 steps required for a portion of one streamline.

TABLE 6
Number of Steps, Time per Integration Step (in Seconds), and Ratio of Filtered to Nonfiltered Time per Step Required to Calculate Different Streamlines on the Three Different Fields with the Error Tolerance Set to 10^{-6}

Streamline	L_2		Post-Processed		Ratio
	Steps	Time/Step (sec)	Steps	Time/Step (sec)	
Field 1					
1	523	0.00374	42	0.00693	1.853
2	358	0.00400	54	0.00705	1.763
3	2476	0.00372	93	0.00708	1.903
Field 2					
1	398	0.00370	35	0.00719	1.943
2	135	0.00370	45	0.00710	1.919
3	263	0.00365	49	0.00707	1.937
Field 3					
1	206	0.00364	58	0.00718	1.973
2	499	0.00365	44	0.00709	1.942
3	129	0.00364	46	0.00708	1.945

This study shows that there exist regimes based on things such as the tolerance of the desired streamline and the size of the jump discontinuities in which postprocessing the solution prior to streamline integration provides a computational benefit. The improved smoothness greatly reduces the need for adaptive time-stepping schemes to adapt the time step to account for discontinuities, reduces the number of rejected steps, and allows for much larger step sizes for the same error tolerance.

7 Summary and Future Work

Adaptive error control through error prediction and time-step refinement is a powerful concept that allows one to maintain a consistent error tolerance. When adaptive error control is used for streamline integration through discontinuous fields, the computational cost of the procedure is primarily determined by the number and size of the discontinuities. Hence, when one is integrating streamlines through a high-order finite element or finite volume data set, most of the adaptation occurs at the element boundaries. There has been a recent interest in the mathematical community in the development of postprocessing filters that increase the smoothness of the computational solution without destroying the formal accuracy of the field; we have referred to these filters as SIAC filters. In this paper, we have presented a demonstration of a complementary approach to classic error control—application of SIAC filters to discontinuous data prior to streamline integration. These filters are specifically designed to be consistent with the discretization method from which the data of interest was generated and hence can be subjected to the verification process [53]. The objective of this work was to understand the computational trade-offs between the application of error control on discontinuous data and the filtering of the data prior to integration.

If one neglects the cost of the filtering step as being a fixed "preprocessing" step to streamline integration (such as if one expects to integrate many streamlines through the same field), then the trade-off that arises can be expressed succinctly as follows: does one take many adaptive integration steps (due to the presence of discontinuities) through the original field, or does one take fewer adaptive steps through a more expensive field to evaluate (that is, the postprocessed field)? Through our empirical investigation, we find that when the error tolerance required for streamline integration

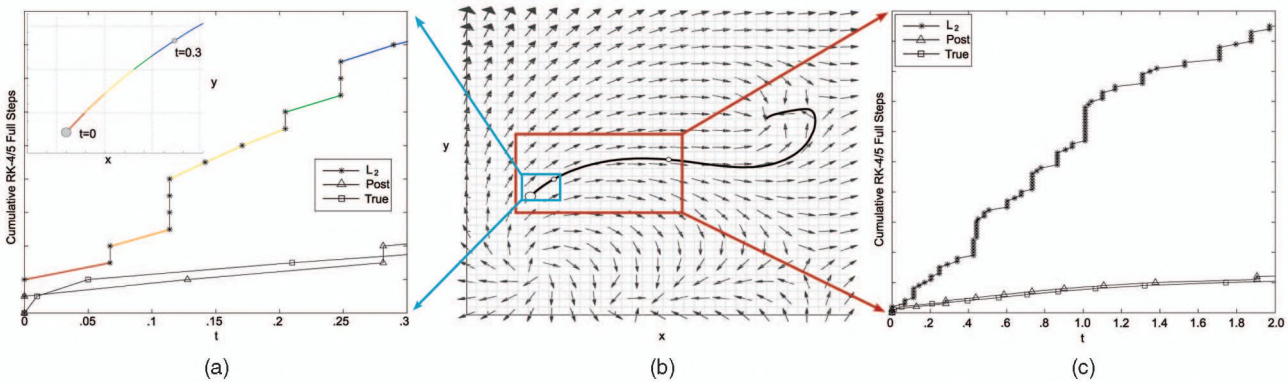

Fig. 4. The center graph shows Streamline 2 on the L_2 projected Field 1 integrated using RK-4/5. The left graph shows the streamline between $t = 0$ and $t = 0.3$ and the cumulative number of RK-4/5 steps (including rejects) required for integration. Vertical lines on this graph represent multiple rejected steps occurring when the streamline crosses element boundaries. The right graph shows the cumulative number of RK-4/5 steps required for integration to $t = 2.0$.

is low or when the jump discontinuities in the field are very high, the strategy advocated in this paper provides a computational win over merely using adaptive error control on the original field (that is, the total computational work can be greatly reduced). We do not advocate that the filtering as presented here replace adaptive error control but rather that it augments current streamline strategies by providing a means of increasing the smoothness of finite element/volume fields without accuracy loss. In doing so, it allows the visualization scientist to balance the trade-offs presented here for minimizing the computational cost of streamline integration through discontinuous fields.

As future work, we seek to extend the filtering techniques presented herein to general discretizations (for example, triangles and tetrahedra), as well as to gradients of fields. We also will seek to understand the relationship between postprocessing vector fields as a collection of single fields versus developing postprocessing techniques that preserve vector properties (such as divergence-free or curl-free conditions). If this were solved, these postprocessing techniques could possibly be of great value as a preprocessing stage prior to other visualization techniques such as feature extraction or isosurface visualization.

ACKNOWLEDGMENTS

The first author would like to acknowledge support from the US Department of Energy through the Center for the Simulation of Accidental Fires and Explosions (C-SAFE) under Grant W-7405-ENG-48. The second author would like to acknowledge the NSF REU support provided as part of NSF CAREER Award NSF-CCF0347791. The third author would like to acknowledge the support from ARO under Grant W911NF-05-1-0395 and would like to thank Bob Haimes (MIT) for the helpful discussions. The authors thank Nathan Galli of the SCI Institute for his assistance in producing the diagrams found in the text. S. Curtis was with the School of Computing, University of Utah, Salt Lake City, Utah.

REFERENCES

[1] D.H. Laidlaw, R.M. Kirby, C.D. Jackson, J.S. Davidson, T.S. Miller, M. da Silva, W.H. Warren, and M.J. Tarr, "Comparing 2D Vector Field Visualization Methods: A User Study," *IEEE Trans. Visualization and Computer Graphics*, vol. 11, no. 1, pp. 59-70, Jan./Feb. 2005.

[2] D. Weiskopf and G. Erlebacher, "Overview of Flow Visualization," *The Visualization Handbook*, C.D. Hansen and C.R. Johnson, eds., Elsevier, 2005.

[3] C.W. Gear, "Solving Ordinary Differential Equations with Discontinuities," *ACM Trans. Math. Software*, vol. 10, no. 1, pp. 23-44, 1984.

[4] J. Bramble and A. Schatz, "Higher Order Local Accuracy by Averaging in the Finite Element Method," *Math. Computation*, vol. 31, pp. 94-111, 1977.

[5] J. Ryan, C.-W. Shu, and H. Atkins, "Extension of a Post-Processing Technique for the Discontinuous Galerkin Method for Hyperbolic Equations with Application to an Aeroacoustic Problem," *SIAM J. Scientific Computing*, vol. 26, pp. 821-843, 2005.

[6] S. Curtis, R.M. Kirby, J.K. Ryan, and C.-W. Shu, "Post-Processing for the Discontinuous Galerkin Method over Non-Uniform Meshes," *SIAM J. Scientific Computing*, to be published.

[7] J. Ryan and C.-W. Shu, "On a One-Sided Post-Processing Technique for the Discontinuous Galerkin Methods," *Methods and Applications of Analysis*, vol. 10, pp. 295-307, 2003.

[8] E. Tufte, *The Visual Display of Quantitative Information*. Graphics Press, 1983.

[9] J.V. Wijk, "Spot Noise Texture Synthesis for Data Visualization," *Computer Graphics*, vol. 25, no. 4, pp. 309-318, 1991.

[10] B. Cabral and L. Leedom, "Imaging Vector Fields Using Line Integral Convolution," *Computer Graphics*, vol. 27, pp. 263-272, 1993.

[11] G. Turk, "Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion Textures," *Computer Graphics*, vol. 25, no. 4, pp. 289-298, 1991.

[12] A. Witkin and M. Kass, "Reaction-Diffusion Textures," *Computer Graphics*, vol. 25, no. 4, pp. 299-308, 1991.

[13] D. Kenwright and G. Mallinson, "A 3-D Streamline Tracking Algorithm Using Dual Stream Functions," *Proc. IEEE Conf. Visualization (VIS '92)*, pp. 62-68, 1992.

[14] D. Watanabe, X. Mao, K. Ono, and A. Imamiya, "Gaze-Directed Streamline Seeding," *ACM Int'l Conf. Proc. Series*, vol. 73, p. 170, 2004.

[15] G. Turk and D. Banks, "Image-Guided Streamline Placement," *Proc. ACM SIGGRAPH '96*, pp. 453-460, 1996.

[16] B. Jobard and W. Lefer, "Creating Evenly-Spaced Streamlines of Arbitrary Density," *Proc. Eighth Eurographics Workshop Visualization in Scientific Computing*, 1997.

[17] X. Ye, D. Kao, and A. Pang, "Strategy for Seeding 3D Streamlines," *Proc. IEEE Conf. Visualization (VIS)*, 2005.

[18] A. Mebarki, P. Alliez, and O. Devillers, "Farthest Point Seeding for Efficient Placement of Streamlines," *Proc. IEEE Conf. Visualization (VIS)*, 2005.

[19] D. Stalling and H.-C. Hege, "Fast and Resolution Independent Line Integral Convolution," *Proc. ACM SIGGRAPH '95*, pp. 249-256, 1995.

[20] C. Teitzel, R. Grosso, and T. Ertl, "Efficient and Reliable Integration Methods for Particle Tracing in Unsteady Flows on Discrete Meshes," *Proc. Eighth Eurographics Workshop Visualization in Scientific Computing*, pp. 49-56, citeseer.ist.psu.edu/teitzel97efficient.html, 1997.

[21] Z. Liu and R.J. Moorhead, "Accelerated Unsteady Flow Line Integral Convolution," *IEEE Trans. Visualization and Computer Graphics*, vol. 11, no. 2, pp. 113-125, Mar./Apr. 2005.
[22] J. Parker, R. Kenyon, and D. Troxel, "Comparison of Interpolating Methods for Image Resampling," *IEEE Trans. Medical Imaging*, vol. 2, no. 1, pp. 31-39, 1983.
[23] H. Hou and H. Andrews, "Cubic Splines for Image Interpolation and Digital Filtering," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 26, pp. 508-517, 1978.
[24] A. Entezari, R. Dyer, and T. Möller, "Linear and Cubic Box Splines for the Body Centered Cubic Lattice," *Proc. IEEE Conf. Visualization (VIS '04)*, pp. 11-18, 2004.
[25] D. Mitchell and A. Netravali, "Reconstruction Filters in Computer-Graphics," *Proc. ACM SIGGRAPH '88*, pp. 221-228, 1988.
[26] G. Nürnberger, L. Slatexchumaker, and F. Zeilfelder, *Local Lagrange Interpolation by Bivariate C1 Cubic Splines*. Vanderbilt Univ., 2001.
[27] P. Sablonniere, "Positive Spline Operators and Orthogonal Splines," *J. Approximation Theory*, vol. 52, no. 1, pp. 28-42, 1988.
[28] Y. Tong, S. Lombeyda, A. Hirani, and M. Desbrun, "Discrete Multiscale Vector Field Decomposition," *ACM Trans. Graphics*, vol. 22, no. 3, pp. 445-452, 2003.
[29] I. Ihm, D. Cha, and B. Kang, "Controllable Local Monotonic Cubic Interpolation in Fluid Animations: Natural Phenomena and Special Effects," *Computer Animation and Virtual Worlds*, vol. 16, no. 3-4, pp. 365-375, 2005.
[30] T. Möller, R. Machiraju, K. Mueller, and R. Yagel, "Evaluation and Design of Filters Using a Taylor Series Expansion," *IEEE Trans. Visualization and Computer Graphics*, vol. 3, no. 2, pp. 184-199, June 1997.
[31] T. Möller, K. Mueller, Y. Kurzion, R. Machiraju, and R. Yagel, "Design of Accurate and Smooth Filters for Function and Derivative Reconstruction," *Proc. IEEE Symp. Volume Visualization (VVS '98)*, pp. 143-151, 1998.
[32] D. Stalling, "Fast Texture-Based Algorithms for Vector Field Visualization," PhD dissertation, Konrad-Zuse-Zentrum für Informationstechnik, 1998.
[33] D. Stalling and H.-C. Hege, "Fast and Resolution Independent Line Integral Convolution," *Proc. ACM SIGGRAPH '95*, pp. 249-256, 1995.
[34] V. Thomée, "High Order Local Approximations to Derivatives in the Finite Element Method," *Math. Computation*, vol. 31, pp. 652-660, 1977.
[35] I. Shoenberg, "Contributions to the Problem of Approximation of Equidistant Data by Analytic Functions, Parts A, B," *Quarterly Applied Math.*, vol. 4, pp. 45-99, 112-141, 1946.
[36] C.-W.S.B. Cockburn, M. Luskin, and E. Suli, "Enhanced Accuracy by Post-Processing for Finite Element Methods for Hyperbolic Equations," *Math. Computation*, vol. 72, pp. 577-606, 2003.
[37] M. Mock and P. Lax, "The Computation of Discontinuous Solutions of Linear Hyperbolic Equations," *Comm. Pure and Applied Math.*, vol. 18, pp. 423-430, 1978.
[38] D.G.W. Cai and C.-W. Shu, "On One-Sided Filters for Spectral Fourier Approximations of Discontinuous Functions," *SIAM J. Numerical Analysis*, vol. 29, pp. 905-916, 1992.
[39] P.G. Ciarlet, "The Finite Element Method for Elliptic Problems," *SIAM Classics in Applied Math.*, Soc. of Industrial and Applied Math., 2002.
[40] P.G. Ciarlet, "Interpolation Error Estimates for the Reduced Hsieh-Clough-Tocher Triangle," *Math. Computation*, vol. 32, no. 142, pp. 335-344, 1978.
[41] J.H. Bramble and M. Zlámal, "Triangular Elements in the Finite Element Method," *Math. Computation*, vol. 24, no. 112, pp. 809-820, 1970.
[42] P. Alfeld and L. Schumaker, "Smooth Finite Elements Based on Clough-Tocher Triangle Splits," *Numerishe Mathematik*, vol. 90, pp. 597-616, 2002.
[43] M. jun Lai and L. Schumaker, "Macro-Elements and Stable Local Bases for Splines on Clough-Tocher Triangulations," *Numerishe Mathematik*, vol. 88, pp. 105-119, 2001.
[44] O. Davydov and L.L. Schumaker, "On Stable Local Bases for Bivariate Polynomial Spline Spaces," *Constructive Approximations*, vol. 18, pp. 87-116, 2004.
[45] T.J.R. Hughes, *The Finite Element Method*. Prentice Hall, 1987.
[46] G.E. Karniadakis and S.J. Sherwin, *Spectral/HP Element Methods for CFD*. Oxford Univ. Press, 1999.
[47] E. Hairer, S.P. Norrsett, and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems*, second revised ed. Springer, 1993.
[48] R. Burden and J. Faires, *Numerical Analysis*. PWS, 1993.
[49] C. Canuto and A. Quarteroni, "Approximation Results for Orthogonal Polynomials in Sobolev Spaces," *Math. Computation*, vol. 38, no. 157, pp. 67-86, 1982.
[50] E. Murman and K. Powell, "Trajectory Integration in Vertical Flows," *AIAA J.*, vol. 27, no. 7, pp. 982-984, 1988.
[51] G. Scheuermann, X. Tricoche, and H. Hagen, "C1-Interpolation for Vector Field Topology Visualization," *Proc. IEEE Conf. Visualization (VIS '99)*, pp. 271-278, 1999.
[52] R.L. Burden and J.D. Faires, *Numerical Analysis*, fifth ed. PWS, 1993.
[53] I. Babuska and J.T. Oden, "Verification and Validation in Computational Engineering and Science: Basic Concepts," *Computer Methods in Applied Mechanics and Eng.*, vol. 193, no. 36-38, pp. 4057-4066, 2004.

Michael Steffen received the BA degree in computer science/mathematics and physics from Lewis & Clark College, Portland, Oregon. He then worked as a software engineer for the Space Computer Corporation developing real-time processors for airborne hyperspectral sensors. Currently, he is a PhD student in the School of Computing, University of Utah, Salt Lake City, working in the Scientific Computing and Imaging Institute.

Sean Curtis received the BA degree in German from Brigham Young University and the BS degree in computer science from the University of Utah. He is currently a PhD student in computer science in the Department of Computer Science, University of North Carolina, Chapel Hill, where he is pursuing research in graphics and simulation.

Robert M. Kirby received the ScM degrees in computer science and in applied mathematics and the PhD degree in applied mathematics from Brown University. He is an assistant professor of computer science in the School of Computing, University of Utah, Salt Lake City, and is a member of the Scientific Computing and Imaging Institute, University of Utah. His research interests lie in scientific computing and visualization. He is a member of the IEEE.

Jennifer K. Ryan received the MS degree in mathematics from the Courant Institute for Mathematical Sciences, New York University, in 1999 and the PhD degree in applied mathematics in 2003 from Brown University. She held the Householder postdoctoral fellowship at Oak Ridge National Laboratory and is currently an assistant professor in the Department of Mathematics, Virginia Polytechnic Institute and State University (Virginia Tech), Blacksburg.