On the Treatment of Field Quantities and Elemental Continuity in FEM Solutions

Ashok Jallepalli, Julia Docampo, Jennifer K. Ryan, R. Haimes and Robert M. Kirby

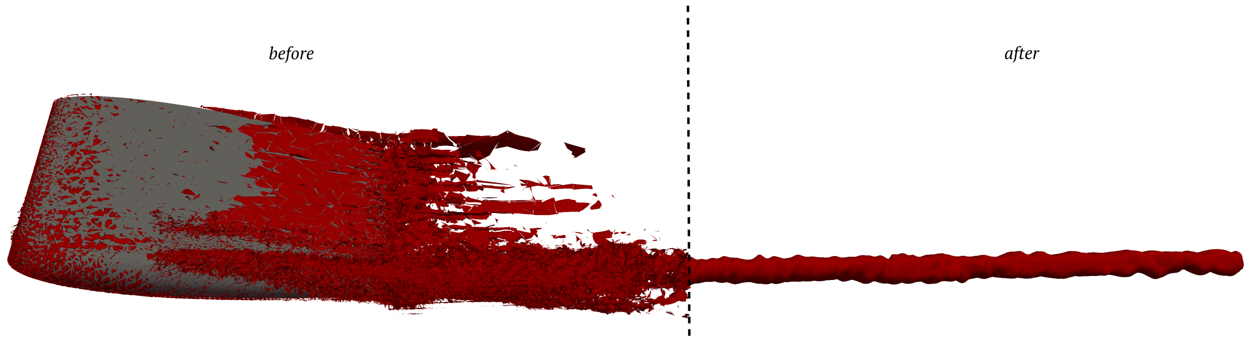

Fig. 1. 3D NACA wing example of vorticity before (left of the dashed line) and after (right) the application of our proposed approach.

Abstract—As the finite element method (FEM) and the finite volume method (FVM), both traditional and high-order variants, continue their proliferation into various applied engineering disciplines, it is important that the visualization techniques and corresponding data analysis tools that act on the results produced by these methods faithfully represent the underlying data. To state this in another way: the interpretation of data generated by simulation needs to be consistent with the numerical schemes that underpin the specific solver technology. As the verifiable visualization literature has demonstrated: visual artifacts produced by the introduction of either explicit or implicit data transformations, such as data resampling, can sometimes distort or even obfuscate key scientific features in the data. In this paper, we focus on the handling of elemental continuity, which is often only C^0 continuous or piecewise discontinuous, when visualizing primary or derived fields from FEM or FVM simulations. We demonstrate that traditional data handling and visualization of these fields introduce visual errors. In addition, we show how the use of the recently proposed line-SIAC filter provides a way of handling elemental continuity issues in an accuracy-conserving manner with the added benefit of casting the data in a smooth context even if the representation is element discontinuous.

Index Terms—Flow Visualization, discontinuous Galerkin (dG) methods, continuous Galerkin (cG) methods, finite element methods (FEM), finite volume methods, filtering techniques, Scalar Field Data, Irregular and Unstructured Grids, Extraction of Surfaces(Isosurfaces).

1 Introduction

The use of unstructured finite element method (FEM) and finite volume method (FVM) technology continues to expand within both the academic research world and industrial applications. Both the traditional and high-order variants of these methods are now being used in areas ranging from Formula-1 race car design [16] to bioengineering applications [2]. Simulation tools using these technologies are not used in isolation, but are often part of a design pipeline including modeling, meshing, simulation, and visualization. In light of this observation, Kirby and Silva [15] called for "Verifiable Visualizations" – that is, for visualizations that consider both the errors in the individual visualization components within the design pipeline and also the interaction between and interpretation of the accumulated errors generated in the computational design pipeline, including the visualization and analysis components. In the case of unstructured FEM and FVM data, several challenges exist when applying the verifiable visualization philosophy, ranging from the handling of unstructured data to the levels of field continuity available or assumed. The focus of this paper is on the handling of FEM and FVM data in the verifiable visualization sense; that is, in a way that we can be assured of the impact on the error budget when employed within an engineering design pipeline. In particular we will focus on the impact of inter-elemental continuity on the post-processing (analysis and visualization) of principle and derived fields from FEM and FVM methods.

To begin, let us review the relevant basic building blocks upon which FEM and FVM technologies are based. In both methodologies, the domain of interest as defined by the engineering application is decomposed into a collection of non-overlapping elements, the union of which fills the domain. In two-dimensions, triangles and quadrilaterals are the frequently used element types; in three-dimensions, tetrahedra, hexahedra and prisms are most often used (with pyramids allowing for transitioning within hybrid meshes). Over each element an approximating function is built, either using information solely at the vertices of an element or in the case of high-order elements, via additional degrees of freedom added to the element (as either nodal points within the element or as modal information associated with the element). In an elemental sense, evaluation and derived quantities (such as derivative information) can be exactly computed once values at the degrees of freedom on an element are specified. It is at this stage that FEM and

- *Ashok Jallepalli is with SCI Institute, University of Utah. E-mail: ashokj@sci.utah.edu*
- *Julia Docampo is with School of Mathematics, University of East Anglia. E-mail: J.Docampo@uea.ac.uk*
- *Jennifer K. Ryan is with School of Mathematics, University of East Anglia. E-mail: jennifer.ryan@uea.ac.uk*
- *R. Haimes is with Department of Aeronautics, MIT. E-mail: haimes@mit.edu*
- *Robert M. Kirby is with SCI Institute, University of Utah. E-mail: kirby@sci.utah.edu*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

FVM bifurcate: traditional and high-order FEM fields are constructed as the C^0 continuous assembly of this elemental information, while FVM and their high-order variant, discontinuous Galerkin (dG) methods, operate on piecewise discontinuous fields. In either case, both methodologies at best enforce only C^0 continuity at element interfaces for the primitive variables within the simulation; derivative quantities in both methodologies exist only as piecewise discontinuous fields. The definition of continuity at the interfaces of elements informs us, in some sense, how many times a derivative can be taken over the field before the results become piecewise discontinuous (the derivative of a C^1 field is C^0, where the derivative of a C^0 field becomes a piecewise discontinuous field).

The lack of strong continuity requirements at element interfaces is exploited in the mathematical and algorithmic developments of both FEM and FVM simulations; however, as will be demonstrated in this paper, it is a major challenge when attempting to analyze and visualization both primitive and derived fields in a verifiable manner that is true to the original data. When visualizing FEM and FVM results, elemental continuity (or the lack thereof) can produce visual artifacts which in the best case are only a distraction to the investigator but in the worst case can either obfuscate what truly being represented within the data or inhibit further analysis. Within the simulation community, these issues are often handled through a combination of resampling and explicit data sampling.

In this paper, we examine the traditional simulation-to-visualization pipeline in light of how FEM and FVM data is handled: in particular, focusing on the impact of inter-element continuity when analyzing and visualizing simulation data. In the simulation-to-visualization pipeline, simulation data is generated in a way consistent with the underlying mathematical representations of the data used by the simulation; it is then transformed into a representation which can be inputted into the visualization system. This transformation from simulation output to visualization input is in the best case an identity mapping; in these situations, as demonstrated in [21–23], the underlying simulation representations can be used to generate pixel-exact visualizations that reduce in a verifiable way or eliminate any errors introduced in the visualization process. However, to accommodate the use of general visualization tools, many simulation packages export data transformed into standardized low-order structured (lattice) or unstructured (tessellation) formats with data values stored at the vertices. Mathematically, such a transformation is considered an interpolation projection from the original simulation data space to the representation space of the visualization tool. As we will demonstrate, such transformations may introduce visual artifacts or unquantified errors. In addition, any further derived quantities computed within the visualization tool may not be as accurate as natively computed quantities as the simulation representational metadata has been lost. Transforming from native simulation data to representations that can be read by a large number of visualization tools is not without its merits, which leads us to investigate alternative means of transforming the data (beyond interpolation projections) in a way consistent with the simulations.

Smoothness-Increasing Accuracy-Conserving (SIAC) filtering [3] has been proposed within the FEM and FVM literature as a way of transforming data to increase the level of inter-element smoothness while conserving the accuracy of the original data. This is accomplished by using the mathematical tools upon which FEM and FVM methods are built, and hence provides a verifiable way (in the verifiable visualization sense) of inserting a transformation into the simulation-to-visualization pipeline in a mathematically consistent and quantifiable manner. The term "filter", when used in the context of the SIAC methodology, is not filtering in the signal processing sense. In signal processing, "filtering" has the connotation of changing the data stream. SIAC processing does not degrade the solution, but in fact can enhance the order of accuracy. SIAC filtering has already been demonstrated to be effective for the visualization of streamlines through high-order FEM and FVM data [28, 30]. There are, however, two downsides to the general SIAC approach used thus far: 1) to construct filtering operators in 2D and 3D, tensor-products of one-dimensional filters are used. SIAC filtering requires a combination of finding the geometric intersection between the filtering lattice and the underlying mesh to generate a super-mesh on which the solution is smooth and numerical quadrature on each of the super-mesh element can be accomplished, traditional tensor-product-based SIAC filtering is not practical in three-dimensions. 2) When moving to general domains representing complex geometries, it is often impossible to filter at all points within the domain using the traditional SIAC filter due to its filter width overlap requirements generated by the tensor-product footprint. These two limitations of traditional SIAC filtering motivated the development of the line-SIAC (LSIAC) filter [5], a family of one-dimensional SIAC filters (symmetric and one-sided) that can filter along a segment in any direction within an FEM or FVM field. In this paper, we argue that the LSIAC filter allows us to transform arbitrary, unstructured high-order data over complex geometries in a way that is consistent with the underlying simulation data (and hence in a verifiable way) while remaining computationally tractable.

In Section 2, we first present a brief review of the relevant literature in this area. In Section 3, we then present the LSIAC filter. We start with a description of the traditional one-dimensional SIAC filter upon which both the LSIAC filter and the previously used tensor-product-based SIAC filter is based; we then summarize the mathematical foundations of the LSIAC filter; and we then provide a detailed explanation of how one can implement the LSIAC filter as used in this paper. In Section 4, we discuss several two-dimensional and three-dimensional examples highlighting the problems with lack of inter-element continuity causes, common ways by which the simulation community tries to get around these issues when analyzing such data and its deficiencies, and then lastly also shows how the LSIAC approach significantly improves the analysis and visualization results of FEM and FVM fields. We conclude in Section 5 with a summary of our work and comments on future directions.

2 Previous Work

The previous work that addresses the issues raised in this paper is sparse. A literature review in this area shows that from the vantage point of visualization systems, these issues do not arise as the input formats to a particular system is well defined, and the system is designed to act in a manner consistent with that data. For example, with respect to unstructured high-order data, there has been clear work on refinement-based approaches [27] and ray-traced solutions [21–23].

The problems discussed in this paper present themselves when taken from the vantage point of the simulation-to-visualization pipeline. From this perspective, although not directly applicable, the body of work on verifiable visualization [9–11, 15] is relevant as it highlights the need for understanding the relationships between simulation output and visualization input in a quantifiable way. From the simulation perspective, the transformation of data to allow it to be inputted into a visualization system is a process/pipeline problem: it is acknowledged that such transformations might be necessary, but the details as to what has been done is often not fully documented. The work accomplished in the area of reconstruction filters using splines such as [7, 8, 12] can introduce C^1 and C^2 (and higher) continuity. These reconstruction filters have been proven to work well for discrete data (as obtained in image processing) and volumetric rendering; however, they cannot be used for our purposes since they are not proven to respect the accuracy of underlying simulation data. In the results section of this paper we have summarized several of the canonical ways by which simulation scientists transform their data for analysis and visualization. These methods are used as the baseline for our comparisons.

3 Methods

In this section we first review the family of one-dimensional SIAC filters, and we then present the basic building blocks of the line-SIAC (LSIAC) filter. We present this in two parts: a discussion of the basic mathematics of the LSIAC family and then a discussion of how we implemented the LSIAC filters over arbitrary elements.

3.1 The One-Dimensional SIAC Family of Filters

Given field data, the SIAC filtering technique is defined by the convolution of the input field with a carefully constructed kernel function, with the purpose of reducing the oscillations in the error and increasing the order of accuracy of the original field. To compute the result of the SIAC methodology acting on a piecewise C^0 or discontinuous approximation of degree k, we apply a filter containing a linear combination of B-splines designed so that the filtering operation does not destroy the accuracy of the original approximation. Traditionally, the filter employed $2k+1$ B-splines of order $k+1$ (*i.e.*, degree k) since this configuration can raise the convergence order of the field up to $2k+1$ [3]. However, alternative configurations can still lead to error reduction and smoothness recovery, even when building kernels using fewer B-splines and of lower order [20]. Here, we present the kernel in a general form and provide a brief introduction to this post-processing technique in order to understand its extension to the LSIAC family. For a more detailed discussion of the post-processor and its variants, see [13, 18, 19, 24, 29].

For the one-dimensional case, the SIAC post-processor is defined by:

$$u^{\star}(x) = \frac{1}{H}\int_{-\infty}^{\infty} K^{(r+1,n+1)}\left(\frac{y-x}{H}\right) u_h(y)dy, \tag{1}$$

where $u^{\star}$ is the post-processed solution, H is the characteristic length of the filter often taken as a function of the input mesh h, i.e., $H = m \cdot h$. In general, h is defined as maximum edge length in the neighborhood, and

$$K^{(r+1,n+1)}(x) = \sum_{\gamma=0}^{r} c_{\gamma}^{(r+1,n+1)}\, \psi^{(n+1)}(x-\bar{x}_{\gamma}). \tag{2}$$

The functions $\psi^{(n+1)}$ denote B-splines of order $n+1$ (degree n) and $\bar{x}_{\gamma}$ denotes offset points (which in the symmetric case default to the knot points of the B-splines). The knot positions of the B-splines within the kernel superimposed against the elemental mesh generates the super-mesh over which numerical integration is then performed. The resulting field is then a piecewise polynomial function having the same mesh macro-structure, for which the elemental polynomial degree is increased and the inter-element continuity is increased.

The kernel coefficients, $c_{\gamma}^{(r+1,n+1)}$, are determined by the following property:

$$K \star x^p = x^p, \quad p = 0,\ldots, r, \tag{3}$$

i.e., the kernel must reproduce monomials up to degree r (which is equivalent to maintaining consistency as well as moments up to degree r). Assuming that the input data is a polynomial of degree k, imposing $r+1 \geq k$ ensures that the filter preserves the accuracy of the original data. Hence, for visualization applications, we build kernels using $r+1 = k$ splines which improve the computational costs while not fundamentally affecting any field features. Additionally, versions of the filters in which the kernel coefficients have been modified to post-process derivative quantities exist [25]. The *derivative* kernel is built using higher-order B-splines ($n+1 \geq k$) so that when computing field derivatives together with a SIAC filter, the resulting approximation does not become a lower order polynomial. The convolution is defined in a similar way:

$$K^{(r+1,n+1+\alpha)} \star D^{\alpha}u = D^{\alpha}K^{(r+1,n+1+\alpha)} \star u, \tag{4}$$

where u is the original field and D^{α} denotes the α-derivative. We note that, although mathematically Equation 4 holds, [25] showed that computationally, these operators do not necessarily commute. This fact was taken into account during the experiments carried out in this paper. Figure 2 illustrates a *standard* SIAC kernel and its derivative ($\alpha = 1$) version.

3.2 The Line SIAC (LSIAC) Family of Filters

As was mentioned earlier, the LSIAC family of filters was motivated by the need for a flexible and yet computationally tractable SIAC filter in multi-dimensions. The authors in [5] proposed that one-dimensional SIAC filtering need not necessarily only be done in a way that is aligned with the coordinate axes, but rather it could be implemented as

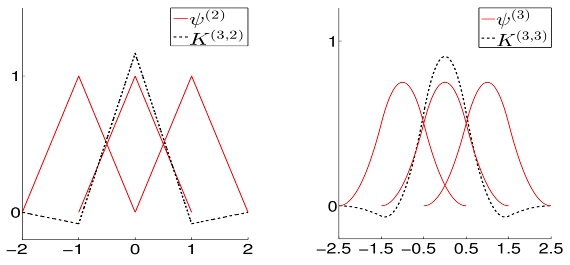

Fig. 2. A symmetric SIAC kernel (left) for post-processing the solutoin of a p=1 approximation and the corresponding derivative kernel (right).

a rotationally-dependent one-dimensional filter for higher-dimensional spaces. In their work they consider two-dimensional fields in which a single angle is needed to specify the orientation of the filter; in this work, we extend their ideas to three-dimensions by allowing the user to choose an orientation vector that specifies the direction in which the filter acts. Like the traditional SIAC filter, the LSIAC family of filters contains both symmetric and one-sided variants [26]. The fluid results presented in this paper employed only symmetric LSIAC kernels. Furthermore, derivative versions of the LSIAC filter also exist and will be used as part of this paper since it allows for computing field quantities, such as vorticity, without loosing approximation polynomial degrees by taking derivatives.

There are two major challenges associated with the optimal design of the LSIAC filter: choosing the kernel characteristic length and the rotation angle. Although the characteristic length choice is obvious for uniform meshes ($H =$ mesh size), there has been ongoing work on finding the optimal value for non-uniform and unstructured meshes [4, 17, 26]. In addition, introducing a rotation in the kernel leads to the problem of determining the optimal orientation. The authors in [5] link the rotation angle directly to the mesh, hence the optimal rotation problem inherits all the limitations encountered by the optimal characteristic length problem. In this paper, we demonstrate the impact of the characteristic length and rotation choices and provide a framework that, although not necessarily optimal, leads to satisfactory results. It should be noted that the streamline extraction of [30] is, in a sense, the precursor to LSIAC where one dimensional SIAC filters were used to extract smooth and continuous traces from 3D dG data, where the local streamline tangent was used as the filter direction.

The properties of the SIAC filter that are of importance to this work that we inherit when moving to LSIAC are as follows [5]:

- LSIAC maintains all the (provable) error characteristics of the tensor-product SIAC filter at significantly reduced computational cost, and
- LSIAC increases the smoothness of the solution in all directions provided the filter parameters are selected appropriately.

3.3 Implementation of LSIAC

We now discuss the implementation of LSIAC filtering to post-process any position within in a vector field. The input choices for the implementation are the FEM or FVM fields to be filtered, the point P in the field at which to post-process, the choice of a symmetric or derivative variant of the filter, the direction $\vec{r}$ along which to post-process, and the characteristic length of the filter H needed to define the post-processing kernel.

At a high level, we need to process the LSIAC filter with given field, which is done by following the three steps below:

- Construct the LSIAC filter: We always choose to apply the symmetric LSIAC filter unless we are unable to do so due to the LSIAC filter width impinging on a boundary of the simulation mesh. For a more detailed explanation on setting up knots for B-Splines and finding the coeffients of the kernel, we refer the reader to [19, 20]. First, we compute the width of the symmetric

LSIAC filter by considering both the characteristic length of the filter H and the footprint of the linear combination of B-splines that make up the kernel; we then shift the center of LSIAC kernel to the evaluation position P, and then rotate the LSIAC filter by the given direction $\vec{r}$. If the line segment does not cross any domain domain boundaries, we proceed to the next step.

- Dividing LSIAC into integrable segments: In order to perform the convolution, the elements overlapped by the line segment of the filter must be respected to isolate regions over which polynomial integration can be made exact (to machine precision). Given that the LSIAC filter is constructed via a linear combination of B-splines, which are piecewise polynomials at their knot values, these relative positions along the segment can be marked. The input field also consists of piecewise polynomials in between element boundaries, so the intersection of the element boundaries and the filter segment are also marked. Next, all marked positions in the parametric space of the LSIAC filter are sorted and the LSIAC's filter is divided into line segments.
- Integrate each segment: Perform the convolution by sampling each line segment on LSIAC's kernel against the field at the quadrature points in the line segment; that is, use quadrature rules to integrate the product of the LSIAC kernel and field values for each line segment. In the final step, perform the summation over all the line segments to set the updated value of the field at the point.

We have implemented the above algorithm in C++ and have used high-order FEM and FVM fields generated by Nektar++ [1] in our examples using simulation data. On a structured mesh, as there is no need to search for adjacent elements: the complexity of the above algorithm depends on computing the LSIAC coefficients ($LSIAC_{Coeffs}$), dividing LSIAC into line segments, and convolving the filter against the elements. Hence, if K is the maximum number of line segments, E is the complexity of evaluating the field at any given point and P is the combination of the LSIAC order and the order of the given field, the complexity of the algorithm on a structured mesh to evaluate a given point is $O(LSIAC_{Coeffs} + K \cdot P \cdot E)$. In the case of an unstructured mesh, there is an additional computation required for finding the adjacent elements (unless this information is available in the mesh or from the solver). We utilize R-trees for finding the intersection of line-segments with the unstructured mesh. The complexity of finding these intersections are $O(\log(N))$, where N is the total number of elements. Hence in the case of an unstructured mesh, our implementation complexity is $O(\log(N) + LSIAC_{Coeffs} + K \cdot P \cdot E)$. When we are evaluating multiple points, we optimize by saving the coefficients of symmetric LSIAC. Hence, the complexity is, in general, no more than $O(\log(N) + K \cdot P \cdot E)$.

To give insight into the performance of our methodology, we compare the cost of evaluating the traditional tensor-product SIAC as used in [28] and LSIAC as used in this work. The cost of 2D SIAC applied over a field of N elements is $(C_1 P^d + C_2 P)$, versus $C_3 P$, which is the complexity of applying LSIAC over a field on N elements. Here P is the polynomial degree of field, d is the dimension and C_1, C_2, C_3 are constants independent of degree. Additionally, in the case of SIAC filtering, the intersection between filter and meshes needs to be handled differently for every type of mesh element, which is not the case with LSIAC.

4 Results and Discussion

In this section, we examine how different sampling schemes, used for both visualization and data analysis, effect the interpretation of FEM and FVM field data. The sampling schemes include what has traditionally been done to examine both piecewise C^0 continuous (FEM) and piecewise discontinuous (FVM) results and are contrasted with LSIAC filtered results. The cases presented herein are discussed both qualitatively through imagery and quantitatively where metrics exist.

4.1 2D Vortical flow over a cylinder

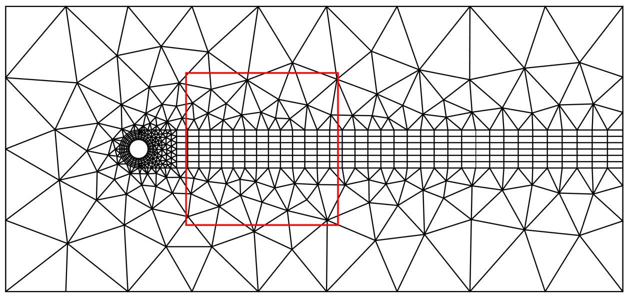

Fig. 3. The two-dimensional hybrid (triangle and quadrilateral) mesh used for all the cylinder flow examples herein.

The Nektar++ [1] solver suite, and in particular the incompressible Navier-Stokes solver, was used to generate the fluid flow results that are visualized throughout. Flow past a circular cylinder at the viscosity examined is a transient problem, but only a single snapshot (in time) is shown. The mesh used for this simulation is shown in Fig. 3, which contains polynomial degree two (P2) elements: 500 triangles and 330 quadrilaterals found primarily in the wake region behind the circle. A continuous Galerkin (FEM) methodology was used which provides C^2 (P2 polynomials) within the element and C^0 at the element interfaces. The Reynolds number was set at $Re = 500$, and the flow solver was run until shedding behind the cylinder generates consistently shaped vortices. These vortices are displayed using different techniques as described below.

Note that the boxed region (in red) in Fig. 3 will be used for the subsequent contour images shown for this case. In the text that follows and in keeping with the mathematical methodology terminology, we will use the term cG (continuous Galerkin) as shorthand for piecewise C^0 FEM fields and dG (discontinuous Galerkin) as shorthand for piecewise discontinuous FVM fields.

4.1.1 Raw dG vorticity

Though the simulation is cG, when taking the derivative of the velocity field (to get the vorticity field), the continuity is reduced by one degree, resulting in a dG field. Vorticity is displayed in Fig. 4.

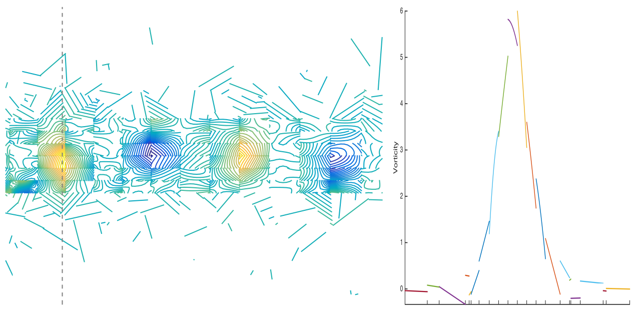

Fig. 4. Raw dG Vorticity Case: Contours of vorticity in the wake of the 2D cylinder (left); line plot of the vorticity extracted along the dashed line shown on the contour image (right). Elements in the one-dimensional plot are colored to help the reader see the continuity between elements.

The contour image seen on the left-hand side of Fig. 4 is calculated on a per-element basis with the value of vorticity computed from the dG field directly. There are two important features to note: 1) the contour lines are discontinuous at the element boundaries (that is, all lines of constant color do not continue across the element boundaries) and 2) the contour lines are linear. The first feature highlights the jumps (the word "jumps" as used in the fluid's community refers to a field discontinuity) seen at the element interfaces. The linear contours are a result of the field now being P1 (due to a derivative being taken). It should also be noted that the contour colors used in this figure represent the same values for all of the figures for this 2D vorticity case.

The line plot on the right-hand-side of Fig. 4 depicts a vertical slice through the data and further highlights the discontinuity at element interfaces. The dG vorticity field is sampled along the dashed line (bottom to top) of the contour image with 8000 points. The different colors indicate a line passing through different elements and the tick marks along the X-axis depict the element interfaces crossed.

4.1.2 Vorticity sampled on a regular lattice

Vorticity is calculated on a lattice from the velocity field that is a regular sampling of the original data. This is consistent with how higher-order data is transformed so that it can be inputted into traditional visualization systems and is usually used when the mesh is relatively isotropic. Vorticity is then calculated from velocity using centered finite differences on the lattice. The lattice spacing is calculated by taking the length of the smallest side of the smallest element and dividing it by $P+2$ where P denotes the element degree (i.e., $smallestEdgeLength/4 = 0.1$ for this mesh). We sample the velocity component in x (i.e. the u component of the velocity) and the velocity component in y (i.e. the v component of the velocity) uniformly with this finite interval. We then use finite differencing of u and v in the y and x directions respectively to calculate $v_x - u_y$.

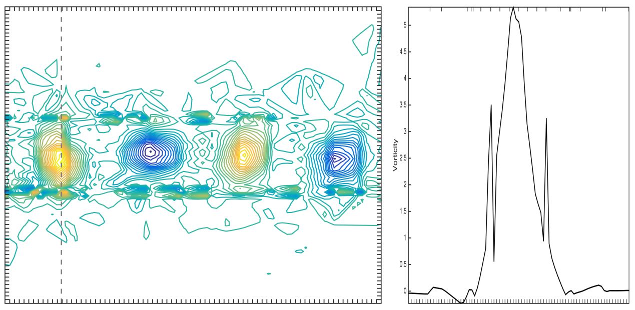

Fig. 5. Lattice-based Vorticity Case: Contours of vorticity in the wake of the 2D cylinder (left); line plot of the vorticity extracted along the dashed line shown on the contour image (right).

The left-hand-side of Fig. 5 displays the contours resulting from the lattice sampling. The ticks along the boundary of the image indicate the lattice spacing. The *graphical aliasing* due to the lattice aligning with the structured portion of the computational mesh is evident, but the jumps have been removed. This is an artifact of the regular sampling and the linear interpolation performed between lattice points.

The line plot seen on the right of Fig. 5 represents the same slice through the data as seen on the right-hand-side of Fig. 4. The tick marks on the bottom axis display the lattice resolution. The tick marks along the top of the plot show the mesh boundary intersections, where the value of vorticity is seen on the plot's Y-axis.

The line plot of vorticity is now continuous but not smooth. The aliasing at (what were) the jumps is evident by the smaller peaks along the flanks of the vortex center.

4.1.3 Vorticity calculated using velocity at collocation point

Another common technique used to map higher-order FEM results to field data consistent with most visualization systems is to breakup the elements into subelements where linear interpolation is then performed. This produces a finer (denser) mesh for visualization and is more appropriate for complex geometries and when the elements tend to be multi-scale or anisotropic. Figure 6 depicts the mesh used for this sampling method where the collocation points interior to the elements are used in the subdivision. (Note: the collocation points and quadrature points are co-located in the Nektar++ software implementation). The subdivision is performed while preserving self-replicating structure inside the element.

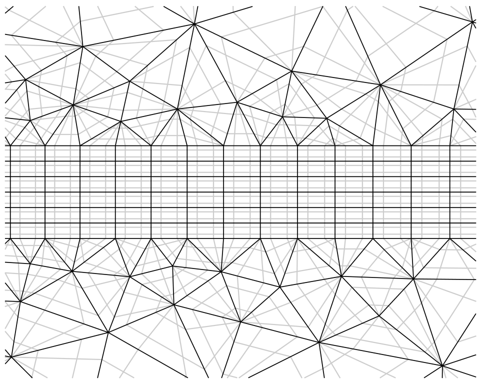

Fig. 6. Figure showing the element subdivision mesh based upon the collocation points.

In order to compute vorticity in a manner consistent with a traditional visualization system where linear interpolation has been assumed throughout, the following procedure is used:

1. Place the velocity at all of the vertices (element bounds and interior) as seen in Figure 6.
2. The velocity derivative components (u_x, u_y, v_x, v_y) at mesh vertices are computed as:
   (a) The velocity derivatives are calculated for each triangular subelement (note that quadrilaterals are further split into triangles). The result will be a constant (i.e., P0 dG) for each triangle.
   (b) To move the velocity derivatives back to the vertices, area weighted sums are accumulated at each vertex that supports the triangle, along with the total area touching the vertex.
   (c) After all triangles are sampled the vertex-based velocity derivative sums are divided by the areas accumulated.
3. Vorticity is then computed as $v_x - u_y$ at every collocation/vertex point.

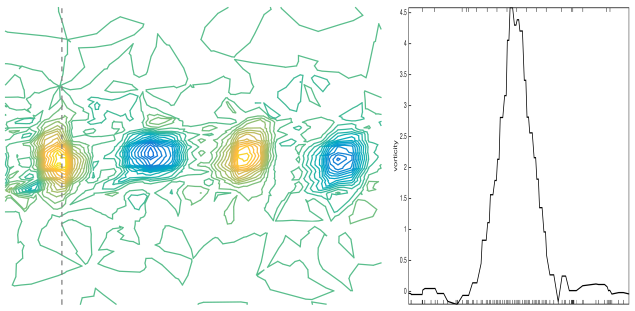

Fig. 7. Elemental Subdivision-based Vorticity Case 1: Contours of vorticity in the wake of the 2D cylinder computed within the visualization system based upon velocity information at the vertices (left); line plot of the vorticity extracted along the dashed line shown on the contour image (right).

Because the velocities are originally cG and the derivatives are taken in a way that produces a continuous (though area averaged) result, no jumps are evident in Fig. 7. The *boxy* nature of the contour image and the *stair step* vortex seen on the right-hand-line plot indicates that the result is not smooth. The tick marks seen on the bottom of the line plot indicate the intersection of the line with the subelements where the ticks on the top show the boundaries of the original elements.

4.1.4 Vorticity at the element collocation points

If the dG velocity derivatives are available, they can be used on the subdivided mesh (as seen in Fig 6). This scheme is included for completeness and to contrast with the method seen in Section 4.1.3. A visualization system would need to be able to deal with the native FEM results directly or the solver provide the derivative field at the collocation points.

It should be noted that there will be multiple values of vorticity at the bounds of the elements (due to the C^0 nature of the velocity field at these vertices). These are simply averaged in order to generate a continuous vorticity field.

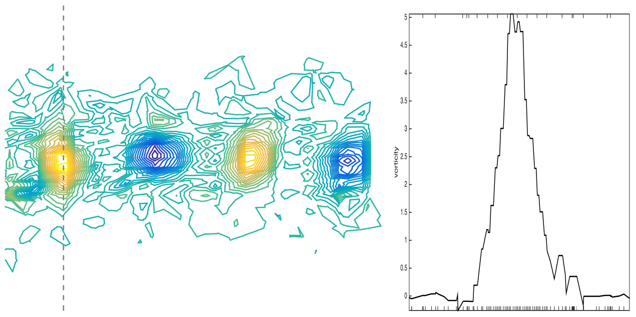

Fig. 8. Elemental Subdivision-based Vorticity Case 2: Contours of vorticity in the wake of the 2D cylinder computed directed within the simulation system and outputted to the visualization system (left); line plot of the vorticity extracted along the dashed line shown on the contour image (right).

The contour plot seen on the left-hand-side of Fig. 8 is similar to the one seen in Fig. 7. The primary differences are that the vorticity minima and maxima are larger here. Also note that there is less noise away from the vortices. The line plot seen in Fig. 8 is a result of the intersection the indicated dashed line in the contour plot. Again this is similar to the line plot of Fig. 7.

It should be noted that the data manipulations due to the sampling and averaging used in this technique (and the sampling techniques seen in Sections 4.1.2 and 4.1.3) do not ensure maintenance of certain fluid properties that result from the solution of either the Euler or Navier-Stokes equations. For example mass will most-likely not be conserved when dealing with the primitive field quantity *density*.

4.1.5 Vorticity using LSIAC filtering

In order to display the results of using LSIAC filtering, the box seen in Fig. 3 is sampled using a mesh consisting of 80×80 quadrilateral elements. Vorticity is computed at all the points by using the derivative variant of the LSIAC filter employing a 0 and 90 degree angle from the x-coordinate axis to calculate u_y^* and v_x^* respectively. The derivative LSIAC filter parameters used are B-splines of order two ($D^1K(3,3)$) and a characteristic length of $H = 0.675$, which is the largest edge of rectangular mesh elements. To give some indication of performance: the time taken to compute the derivative LSIAC at a single position in space is 0.95 milliseconds measured on a machine with a 2.4GHz (Intel CPU E-7-4870) processor.

The results can be seen in the Figure 9 where linear interpolation between mesh points has been employed to make the contour plot, although we note that the actual data is of higher-order. From a quantitative perspective the contours of vorticity are what would be expected from this type of simulation. That is, the contours produced are concentric and are relatively smooth.

The right-hand portion of Fig. 9 was computed by sampling the data along the indicated line by employing 8000 points and vorticity was computed at these locations by post-processing using the LSIAC filter (unlike the other previous figures where the data was sampled from the *visualization mesh*, with the exception of Section 4.1.1). The line segments are colored from each element differently to highlight the

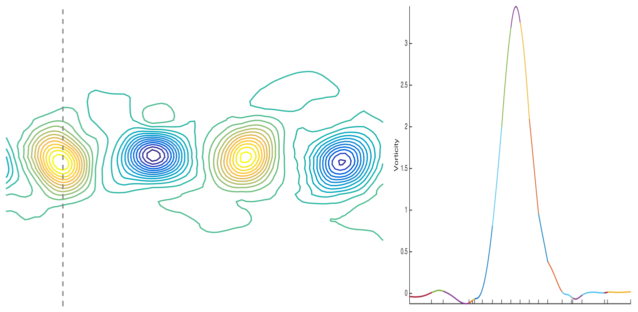

Fig. 9. LSIAC Vorticity Case: Contours of vorticity in the wake of the 2D cylinder (left); line plot of the vorticity extracted along the dashed line shown on the contour image (right). Elements in the one-dimensional plot are colored to help the reader see the continuity between elements.

removal of jumps. The ticks on the bottom indicate the intersection of mesh with sampled line to help with the visualization. This image is in stark contrast to that seen in Fig. 4 where the jumps are now gone and the plots smooth. There is also little *background noise* away from the vortices.

Unlike the traditional sampling schemes (seen in Sections 4.1.2 and 4.1.3) the results using the LSIAC filter, if properly utilized, can preserve fluid properties [5]. This is due to the explicit accuracy conserving nature of the post-processing scheme on cG and dG simulation data.

4.2 LSIAC Validation Study

Designing a LSIAC filter implies a proper selection of several kernel parameters. Hence, for completeness, we present the methodology that we followed in order to produce the data presented in Section 4.1.5.

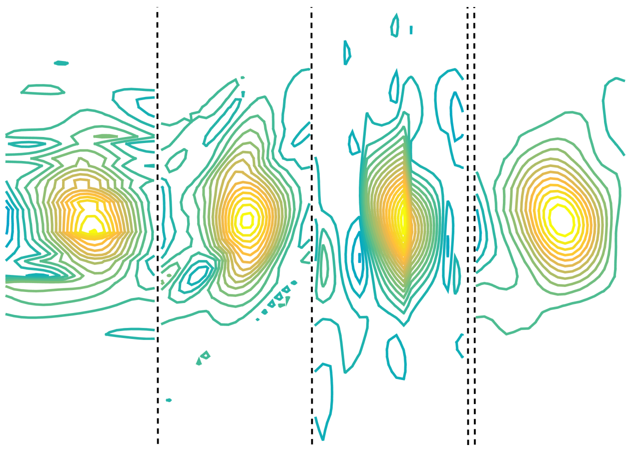

Fig. 10. Filter a particular field region by applying different kernels with rotation angles $\theta = 0^\circ$, 45°, 90° respectively (first three images) directly on the field vorticity. Apply a derivative filter (right-most image, after the double dotted line) prior to computing the field vorticity.

In Section 3 we discussed the challenges associated with the choices of kernel rotation and of characteristic length. Figure 10 shows the same vortex after applying several LSIAC filters. The first three images correspond to three different filter orientations using higher-order kernels once the field vorticity is computed (left-hand-side of Equation 4). On the other hand, the last image shows the result of applying a derivative kernel (right-hand-side of Equation 4) and calculating the vorticity afterwards. In the latter case, in order to obtain v_x and u_y, we have taken advantage of the directions in which the derivatives are computed and applied a zero and a ninety degree kernel respectively. The decomposition of the velocity field in the $x-$ and $y-$ directions suggested such filtering choice and enabled relatively easy computations, but we readily admit that alternatively, a higher order kernel with a particular direction applied directly on the field vorticity could lead to similar and even improved results. However, such an approach would require a theoretical orientation, and since it is not possible to determine the optimal one, throughout this paper we have applied derivative kernels in the direction of the field derivatives. This choice allows for recovering smoothness in all directions and produces satisfactory results.

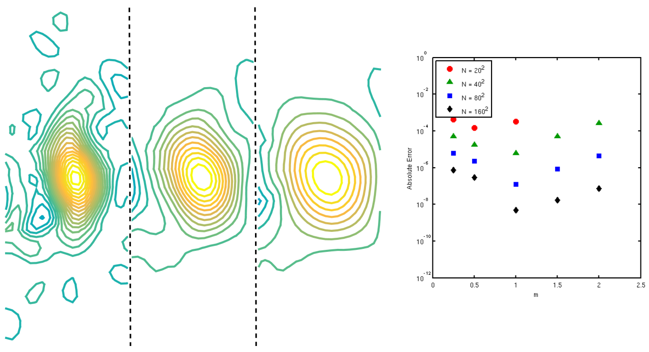

Fig. 11. Vorticity (first three images) after applying a derivative filter for three different kernel characteristic lengths corresponding to $H = 0.3375,\ 0.675,\ 1.0125$, respectively. The right-hand plot (taken from [14]) shows the global errors from a numerical experiment where a SIAC filter was applied for several characteristic lengths [14].

Regarding the kernel characteristic length, [14, 17] used the formula $H = m \cdot h$, with h being the largest element size, and performed global error analyses on known analytic fields for different values of m. They showed that asymptotically, the value $m = 1$ lead to optimal results. We performed a similar experiment and in Figure 11 we show the same filtered vortex after applying different characteristic lengths where one can appreciate that the value $H = 0.675$ ($m = 1$) gives the most satisfactory results. Observe that for the smallest characteristic length $H = 0.3375$ ($m = 0.5$), the filtered solution does not eliminate the noise. On the other hand, applying the largest characteristic length ($H = 1.0125$) results in smoothness recovery but reduced accuracy if compared to the intermediate value. For an examination of the accuracy conserving nature of LSIAC with regards to the characteristic length H, we refer the reader to Section 4.4

4.3 Preservation of critical point locations

A difficult task in filtering and sampling is the ability to maintain the position of critical points that may be in the field. A critical point is where the velocity vanishes in the vector field, so it is easy to upset the location if minima in the solution are effected by any transformations. We utilize a synthetic 2D analytical vector field from [28] and project the vector field onto a mesh of 20×20 P1 dG quadrilateral elements. Our objective is to quantitatively compute the error in the location of critical points introduced by post-processing the data through the different sampling schemes used for visualization.

The manufactured vector field can be see in Fig. 12 and is given by the following equation:

$$\begin{array}{rclrcl} z & = & x + \imath y & r & = & (z-(0.74+0.35\imath))(z-(0.68-0.59\imath)) \\ u & = & Re(r) & & & (z-(-0.11-0.72\imath))(\bar{z}-(-0.58+0.64\imath)) \\ v & = & -Im(r). & & & (\bar{z}-(0.51-0.27\imath))(\bar{z}-(-0.12+0.84\imath))^2 \end{array}$$

which results in the six critical points marked by the red dots in Fig. 12.

Most iterative schemes that find critical points need seed locations. It is difficult to balance the number of seed locations and where to seed given that, in general, one does not know where and how many points may be in the field. In this comparison, we only care about the accuracy of the location, so we will seed at the known critical point positions.

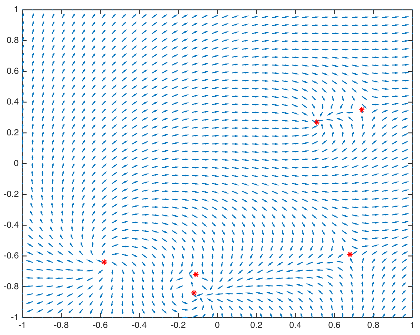

Fig. 12. Synthetic velocity field with known critical points (shown as red dots).

4.3.1 Finding critical points using FEM derivatives

If we can access the derivatives of the vector field directly, then a Newton-Raphson scheme can be used to find the zeros in the field. As can be seen in the *Derivative* column of Table 1, all points are found with relatively small errors. This is a best case scenario, in that the mesh is conceived in a contrived manner so that all critical points are in the interior of the elements. When the critical point is near an element interface, the derivatives tend to be inaccurate, which can send the iterative scheme abruptly away from the zero resulting in missing the location.

4.3.2 Finding critical points using lattice sampling

Again, to emulate what a traditional visualization system may do to find the critical point, we follow this procedure:

1. The 20×20 mesh is subsampled with on a regular lattice of 60×60 elements.
2. Evaluate the velocity at the vertices of the lattice from the dG solution.
3. Calculate u_x, u_y, v_x, v_y at the lattice points using central differences.
4. Employ the Newton-Raphson critical point finder seeded from the actual location.

As can be seen in the *lattice* column of Table 1, the error in the location is roughly the same as using the FEM derivatives directly.

4.3.3 Finding critical points using subdivided elements

In order to simulate finding critical points as a post-processing step within a traditional visualization system, we subdivide the elements as discussed in Section 4.1.3. The procedure used is:

1. Construct a subdivided subelement mesh from the dG elements using the collocation points as vertices. Divide all the subelements into triangles.
2. Compute u_x, u_y, v_x, v_y at mesh vertices by:
   (a) The velocity derivatives are calculated for each triangular subelement. The result will be a constant for each triangle.
   (b) To move the velocity derivatives back to the vertices, area weighted sums are accumulated at each vertex that supports the triangle, along with the total area touching the vertex.
   (c) After all triangles are sampled the vertex-based velocity derivative sums are divided by the areas
3. Employ the Newton-Raphson critical point finder seeded from the actual location.

In this case, there is a critical point that is not found and is marked without an error in Table 1 .

Position	Derivatives	Lattice	SubElement	LSIAC
$(0.74, 0.35)$	0.0081	0.0082	0.0095	0.0048
$(0.68, -0.59)$	0.0020	0.0033	0.0049	0.0006
$(-0.11, -0.72)$	0.1183	0.0137	0.0265	0.0095
$(-0.58, -0.64)$	0.0006	0.0006	0.0089	0.0021
$(0.51, 0.27)$	0.0024	0.0025	0.0038	0.0019
$(-0.12, -0.84)$	0.0103	0.0723	–	0.0317

Table 1. Errors in the critical point locations.

4.3.4 Finding critical points using LSIAC

The LSIAC filtering is performed on the base 20×20 dG velocity mesh. A LSIAC derivative filter is used on the velocity at 0 degrees from the x-coordinate axis to form u_x and v_x at all mesh vertices. Then another LSIAC derivative filter is used at 90 degrees from the x-coordinate axis to generate the velocity derivative components u_y and v_y through the entire mesh. The Newton-Raphson critical point finder is seeded from the actual location and the results can be seen in the *LSIAC* column of Table 1.

We note that errors produced by using the LSIAC filter are, in general, better than the other techniques. It should also be noted that we do not expect the error to be zero, because the process of projecting the analytic vector field onto the dG mesh introduces a projection error.

Though the LSIAC results are not substantially better than the other techniques, it does provide the following distinct advantages:

- As shown in [13], SIAC preserves the critical points near boundaries.
- Since the SIAC derivatives are continuous across elements, the Newton-Raphson technique can cross elements in search of critical points making seeding algorithms more efficient.

Additional studies of field features such as streamline visualization on analytical fields were performed in [6]. In these examples, it was shown that in cases where the dG approximation diverged from the exact streamline, the LSIAC post-processed dG solution converged to the expected solution.

4.4 Impact of characteristic length on accuracy

The effect of the characteristic length on the accuracy of the LSIAC filter is examined by comparing the vorticity computed using dG derivatives and LSIAC derivatives of different characteristic lengths through quantitative metrics. We project the synthetic 2D analytical vector field from [28], used to generate Figure 12 on both 20×20 and 40×40 P1 dG quadrilateral element meshes. For calculating vorticity using LSIAC, we obtain v_x and u_y using zero and ninety-degree derivative kernels ($D^1 K(3,3)$) respectively. The errors introduced in calculating vorticity using dG and LSIAC derivatives as compared to ground truth are shown in Table 2.

Mesh Res		20			40	
Errors		L^2	L^∞		L^2	L^∞
dG		5.996	31.04		2.504	15.93
LSIAC ($H = m \cdot h$)						
m	H	L^2	L^∞	H	L^2	L^∞
0.5	0.05	1.4175	6.8179	0.025	0.7006	3.4803
1.0	0.1	0.0979	0.7005	0.05	0.0249	0.1868
2.0	0.2	0.1202	0.7795	0.1	0.0253	0.1917
4.0	0.4	1.1471	2.0391	0.2	0.7551	0.2708
8.0	0.8	18.314	22.2474	0.4	1.1447	1.5369
16.0	1.6	293.04	347.86	0.8	18.3147	21.839

Table 2. Errors for calculating vorticity on a P1 dG field.

Note the following observations from the results in Table 2:

- The large dG errors are due to dG derivatives not using information beyond the element boundaries; this also results in large error at element boundaries. We can see this effect in Section 4.1.1.
- The minimum error occurs when LSIAC's characteristic length is equal to maximum edge length (i.e., $m = 1.0$) as stated in [14, 17].
- The vorticity computed using LSIAC for a kernel scaling between 0.5 and 4.0 have significantly lower errors compared to dG derivatives.
- LSIAC reduces the error in both the L^2 and L^∞ norms; this is an indication that there is a benefit "on average" (i.e., in the integral sense) and in a point-wise sense.

4.5 NACA 0012 wing simulation

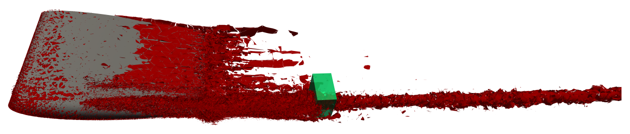

Fig. 13. 3D NACA wing case example of vorticity (before filtering). The box shown in the wake denotes the region within the field over which the results will be compared.

To qualitatively show the use of LSIAC post-processing in a realistically-sized 3D fluids setting, we will use another Nektar++ simulation example. In Lombard et al. [16] the authors perform Large-Eddy Simulation (LES) for the formation and evolution of a wingtip vortex. The wing is a 3D extrusion of a NACA 0012 airfoil section with a rounded wing tip and a blunt trailing edge. The simulation mesh contains 211180 polynomial degree five (P5) tetrahedra and 38680 P5 prisms, where the prism elements are stacked and graded away from the wing surface in order to best capture the boundary layer. A cG discretization was used and executed using a Reynolds number of 1.2×10^6 and the angle of attack α is 12 degrees for the case shown.

Figure 13 shows the wing and solution with the vortex generated from the wing tip and its convection downstream. The cube shown in the vortex, downstream from the tip, indicates the selected region that will be used to more closely examine the vorticity field. It should be noted that because this box is away from the wing, all elements are tetrahedra.

To produce the upper image of Figure 14, the box is sampled and vorticity is computed at a resolution of $90 \times 90 \times 90$. This 3D lattice is then iso-surfaced to generate the vortex tube, which suffers from the same problems as seen in the 2D flow of Figure 4. Seen on the lower portion of the figure is the data from a 2D slice, which in this case is computed from the data in an $10^3 \times 10^3$ lattice. This has the same form as the left-hand-side of Figure 4 with abrupt jumps, but in this case the contour lines are curved (the vorticity field is P5 within each element).

We now calculate the vorticity of the field in same subset using the LSIAC methodology proposed in this work. All of the derivatives required, $u_y^*, u_z^*, v_x^*, v_z^*, w_x^*, w_y^*$, are computed using a LSIAC filter, where u, v, w are the components of velocity in the given dG vector field. The parameters used for the LSIAC filter are B-splines of order two ($D^1 K(3,3)$) and a characteristic length is $H = 0.05$. We calculate the magnitude of vorticity at all the sampled points and generate the iso-surface of vorticity at same value used for dG data (20 units) and draw the contours on the plane at the same location we selected for dG data. The results are shown in Figure 15. We notice all the jumps are completely removed; the contours are continuous and concentric as expected. To give some indication of performance: the time taken to compute the derivative LSIAC at a single position in space is 11.4 milliseconds measured on a machine with a 2.4GHz (Intel CPU E-7-4870) processor.

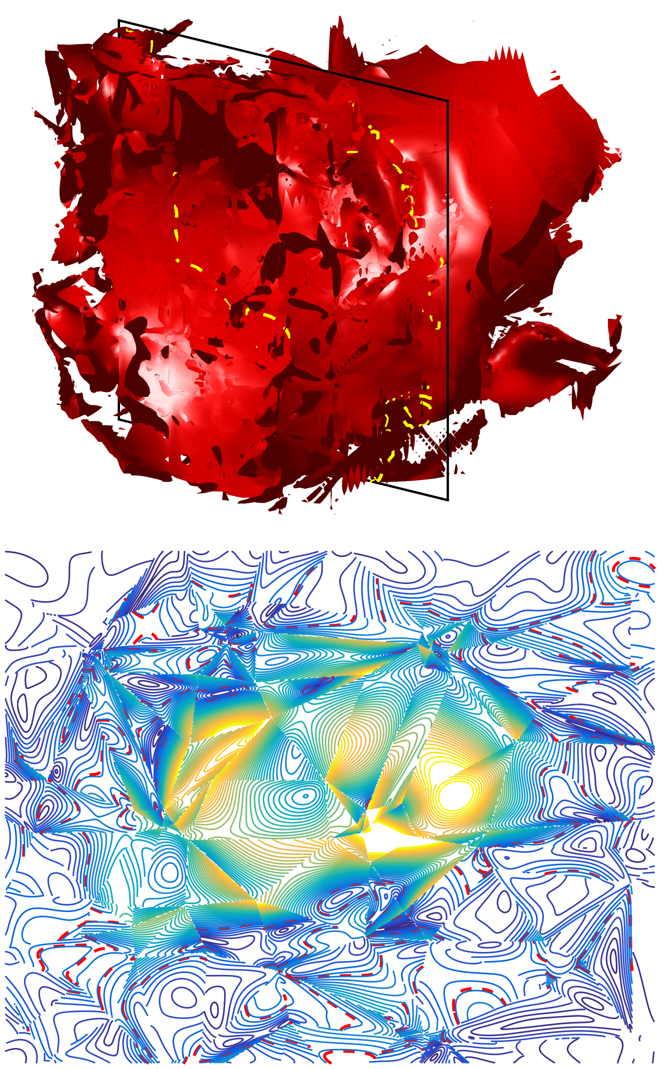

Fig. 14. 3D NACA Raw Data Case: Iso-contour of vorticity extracted from the field (top). Contour lines showing the vorticity field extracted over the plane shown in the upper figure (bottom).

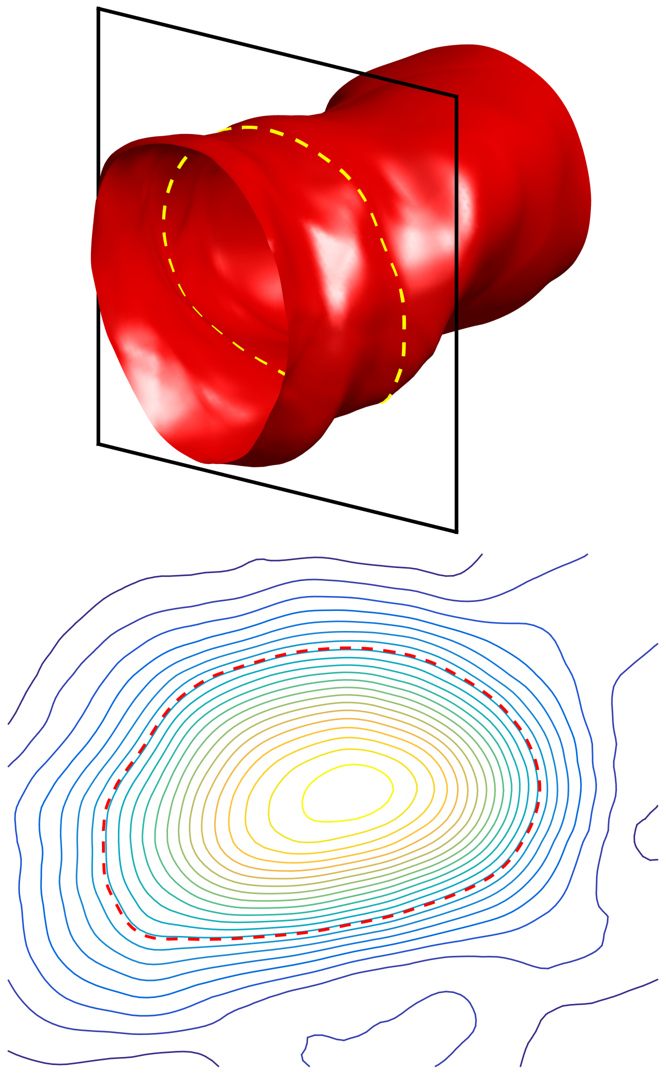

Fig. 15. 3D NACA LSIAC Case: Iso-contour of vorticity extracted from the field (top). Contour lines showing the vorticity field extracted over the plane shown in the upper figure (bottom).

5 Summary and Future Work

Our motivation in this work was to investigate the challenges arising from the analysis and visualization of unstructured (possibly high-order) FEM and FVM data; in particular, to focus on the issues arising from the weak imposition or lack of continuity at element interfaces. We have shown that the traditional simulation-to-visualization pipeline, and in particular interpolation projections to either lattices or refined unstructured data may appear to ameliorate these issues, but at the cost of maintaining important features of the data. In the past, the SIAC filtering methodology was proposed as an alternative solution. Unlike traditional filtering stemming from a signal or image processing perspective, the SIAC methodology is built upon the mathematical building blocks of the FEM and FVM technologies themselves, and delivers increased smoothness in the solution without loss of accuracy. As such, the SIAC methodology fits nicely into the verifiable visualization framework, and it acts as a data transformation that allows for analysis and visualization while still remaining faithful to the underlying numerical properties of the FEM and FVM schemes. In fact, in cases where the physics of the problem dictates that the solution is smooth and FEM and FVM methods relax this constraint for approximation reasons, the SIAC methodology removes these *artificial* numerical constraints (and their corresponding visual artifacts) without upsetting the accuracy of the solution data.

Investigators using FEM and FVM schemes that produce discontinuous results (either from derived and/or derivative fields) may be confused by imagery produced from the raw data. The physics may dictate smooth and continuous results but due to the numerics the data is not. The use of LSIAC post-processing removes the jumps found at element boundaries and recovers the expected continuous data. This means that the investigator need not be an expert in FEM discretizations and can better understand the results of the simulation without being encumbered by the errors generated by sampling or the artifacts seen in the raw simulation data.

As discussed in this paper, the principal challenge for the traditional SIAC filter in multi-dimensions made it infeasible for general use; these issues have been overcome by LSIAC. In this paper, we have demonstrated that the LSIAC approach for filtering two-dimensional and three-dimensional fields provides us the benefits of the SIAC methodology in a computationally tractable way. In particular LSIAC does not have the *curse of dimensionality* as experienced with SIAC post-processing (i.e, encountering boundaries is considerably simpler in the LSIAC case).

There is still future work in this area to be accomplished. We have not addressed the question of optimally selecting the LSIAC characteristic length H or the direction in which the filter is applied. In all our applications, we used default choices from the literature for the characteristic length (H) and we selected directions that were consistent with the derived quantities being investigated. A general strategy for selecting the filter direction based upon the field characteristics, mesh and polynomial degree used, and the quantity of interest being examined requires further investigation. Furthermore, we have only demonstrated the LSIAC methodology for the post-processing of primary and derivative-based derived fields; future work would be to incorporate the LSIAC work into vector field topology, feature detection and extraction algorithms as almost all such algorithms are built upon the computing of derivatives of the field.

REFERENCES

[1] C. D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R. M. Kirby, and S. J. Sherwin. Nektar plus plus : An open-source spectral/hp element framework. *Computer Physics Communications*, 192:205–219, Jul 2015. doi: 10.1016/j.cpc.2015.02.008

[2] K. Chooi, A. Comerford, S. Sherwin, and P. Weinberg. Intimal and medial contributions to the hydraulic resistance of the arterial wall at different pressures: a combined computational and experimental study. *Journal of The Royal Society Interface*, 13(119):20160234, 2016.

[3] B. Cockburn, M. Luskin, C.-W. Shu, and E. Süli. Post-processing of Galerkin methods for hyperbolic problems. In *Discontinuous Galerkin Methods*, pp. 291–300. Springer, 2000.

[4] S. Curtis, R. M. Kirby, J. K. Ryan, and C.-W. Shu. Postprocessing for the discontinuous Galerkin method over nonuniform meshes. *SIAM Journal on Scientific Computing*, 30(1):272–289, 2007.

[5] J. Docampo-Sánchez, J. K. Ryan, M. Mirzargar, and R. M. Kirby. Multidimensional filtering: Reducing the dimension through rotation. *SIAM Journal on Scientific Computing*, To appear, 2017.

[6] Docampo-Sánchez, Julia. *Smoothness-Increasing Accuracy-Conserving (SIAC) Line Filtering: Effective Rotation for Multidimensional Fields*. PhD Thesis in Applied Mathematics, University of East Anglia (UEA), November, 2016.

[7] A. Entezari, R. Dyer, and T. Moller. Linear and cubic box splines for the body centered cubic lattice. In *Proceedings of the conference on Visualization'04*, pp. 11–18. IEEE Computer Society, 2004.

[8] A. Entezari, D. Van De Ville, and T. Möeller. Practical box splines for reconstruction on the body centered cubic lattice. *IEEE Transactions on Visualization and Computer Graphics*, 14(2):313–328, 2008.

[9] T. Etiene, D. Jonsson, T. Ropinski, C. Scheidegger, J. L. Comba, L. G. Nonato, R. M. Kirby, A. Ynnerman, and C. T. Silva. Verifying Volume Rendering Using Discretization Error Analysis. *IEEE Transactions on Visualization and Computer Graphics*, 20(1):140–154, 2014. doi: 10.1109/TVCG.2013.90

[10] T. Etiene, L. Nonato, C. Scheidegger, J. Tienry, T. Peters, V. Pascucci, R. Kirby, and C. Silva. Topology verification for isosurface extraction. *IEEE Transactions on Visualization and Computer Graphics*, 18(6):952–965, June 2012. doi: 10.1109/TVCG.2011.109

[11] T. Etiene, C. Scheidegger, L. Nonato, R. Kirby, and C. Silva. Verifiable visualization for isosurface extraction. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1227–1234, Nov 2009. doi: 10.1109/TVCG.2009.194

[12] H. Hou and H. Andrews. Cubic splines for image interpolation and digital filtering. *IEEE Transactions on acoustics, speech, and signal processing*, 26(6):508–517, 1978.

[13] L. Ji, P. Van Slingerland, J. K. Ryan, and K. Vuik. Superconvergent error estimates for position-dependent smoothness-increasing accuracy-conserving (SIAC) post-processing of discontinuous Galerkin solutions. *Mathematics of Computation*, 83(289):2239–2262, 2014.

[14] J. King, H. Mirzaee, J. K. Ryan, and R. M. Kirby. Smoothness-increasing accuracy-conserving (SIAC) filtering for discontinuous Galerkin solutions: Improved errors versus higher-order accuracy. *Journal of Scientific Computing*, 53(1):129–149, 2012.

[15] R. Kirby and C. Silva. The Need for Verifiable Visualization. *IEEE Computer Graphics and Applications*, 28(5):78–83, September 2008. doi: 10.1109/MCG.2008.103

[16] J.-E. W. Lombard, D. Moxey, S. J. Sherwin, J. F. Hoessler, S. Dhandapani, and M. J. Taylor. Implicit large-eddy simulation of a wingtip vortex. *AIAA Journal*, 54(2):506–518, 2015.

[17] H. Mirzaee, J. King, J. K. Ryan, and R. M. Kirby. Smoothness-increasing accuracy-conserving filters for discontinuous Galerkin solutions over unstructured triangular meshes. *SIAM Journal on Scientific Computing*, 35(1):A212–A230, 2013. doi: 10.1137/120874059

[18] H. Mirzaee, J. K. Ryan, and R. M. Kirby. Quantification of errors introduced in the numerical approximation and implementation of smoothness-increasing accuracy conserving (SIAC) filtering of discontinuous Galerkin (dg) fields. *Journal of Scientific Computing*, 45(1-3):447–470, 2010.

[19] H. Mirzaee, J. K. Ryan, and R. M. Kirby. Efficient implementation of smoothness-increasing accuracy-conserving (SIAC) filters for discontinuous Galerkin solutions. *Journal of Scientific Computing*, 52(1):85–112, 2012.

[20] M. Mirzargar, J. K. Ryan, and R. M. Kirby. Smoothness-increasing accuracy-conserving (SIAC) filtering and quasi-interpolation: a unified view. *Journal of Scientific Computing*, 67(1):237–261, 2016.

[21] B. Nelson and R. M. Kirby. Ray-tracing polymorphic multidomain spectral/hp elements for isosurface rendering. *IEEE Transactions on Visualization and Computer Graphics*, 12:114–125, January 2006. doi: 10.1109/TVCG.2006.12

[22] B. Nelson, R. M. Kirby, and R. Haimes. Gpu-based interactive cut-surface extraction from high-order finite element fields. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1803–1811, Dec. 2011. doi: 10.1109/TVCG.2011.206

[23] B. Nelson, E. Liu, R. Haimes, and R. M. Kirby. Elvis: A system for the accurate and interactive visualization of high-order finite element solutions. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2325–2334, 2012.

[24] J. Peters. General spline filters for discontinuous Galerkin solutions. *Computers & Mathematics with Applications*, 70(5):1046–1050, 2015.

[25] J. K. Ryan and B. Cockburn. Local derivative post-processing for the discontinuous Galerkin method. *Journal of Computational Physics*, 228(23):8642–8664, 2009.

[26] J. K. Ryan, X. Li, R. M. Kirby, and K. Vuik. One-sided position-dependent smoothness-increasing accuracy-conserving (SIAC) filtering over uniform and non-uniform meshes. *Journal of Scientific Computing*, 64(3):773–817, 2015.

[27] 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. *Visualization and Computer Graphics, IEEE Transactions on*, 12(4):446 –460, july-aug. 2006. doi: 10.1109/TVCG.2006.74

[28] M. Steffen, S. Curtis, R. M. Kirby, and J. K. Ryan. Investigation of smoothness-increasing accuracy-conserving filters for improving streamline integration through discontinuous fields. *IEEE Transactions on Visualization and Computer Graphics*, 14(3):680–692, 2008.

[29] P. van Slingerland, J. K. Ryan, and C. Vuik. Position-dependent smoothness-increasing accuracy-conserving (SIAC) filtering for improving discontinuous Galerkin solutions. *SIAM Journal on Scientific Computing*, 33(2):802–825, 2011.

[30] D. Walfisch, J. K. Ryan, R. M. Kirby, and R. Haimes. One-sided smoothness-increasing accuracy-conserving filtering for enhanced streamline integration through discontinuous fields. *Journal of Scientific Computing*, 38(2):164–184, 2009.