ElVis: A System for the Accurate and Interactive Visualization of High-Order Finite Element Solutions

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

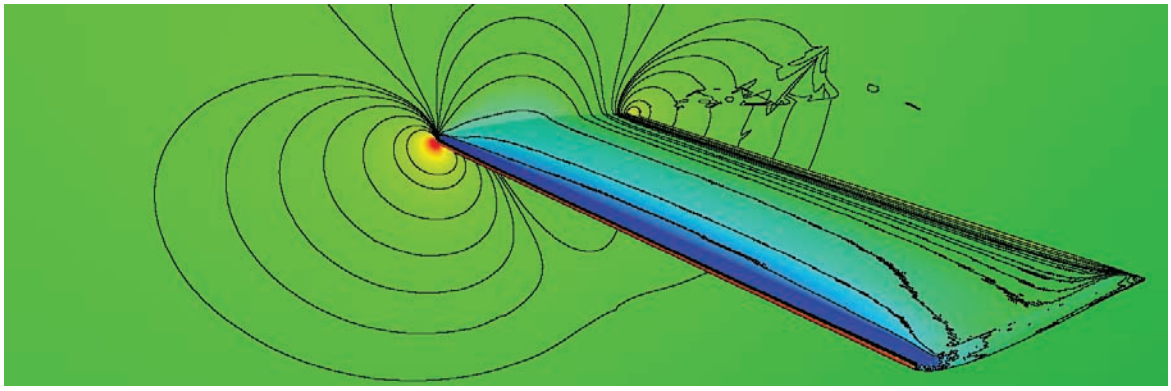


Fig. 1. Pressure field on the ONERA M6 Wing (Section 5.1.2), rendered using ElVis and illustrating the application of color maps and contour lines on curved and planar surfaces.

Abstract—This paper presents the Element Visualizer (ElVis), a new, open-source scientific visualization system for use with high-order finite element solutions to PDEs in three dimensions. This system is designed to minimize visualization errors of these types of fields by querying the underlying finite element basis functions (e.g., high-order polynomials) directly, leading to pixel-exact representations of solutions and geometry. The system interacts with simulation data through runtime plugins, which only require users to implement a handful of operations fundamental to finite element solvers. The data in turn can be visualized through the use of cut surfaces, contours, isosurfaces, and volume rendering. These visualization algorithms are implemented using NVIDIA's OptiX GPU-based ray-tracing engine, which provides accelerated ray traversal of the high-order geometry, and CUDA, which allows for effective parallel evaluation of the visualization algorithms. The direct interface between ElVis and the underlying data differentiates it from existing visualization tools. Current tools assume the underlying data is composed of linear primitives; high-order data must be interpolated with linear functions as a result. In this work, examples drawn from aerodynamic simulations–high-order discontinuous Galerkin finite element solutions of aerodynamic flows in particular–will demonstrate the superiority of ElVis' pixel-exact approach when compared with traditional linear-interpolation methods. Such methods can introduce a number of inaccuracies in the resulting visualization, making it unclear if visual artifacts are genuine to the solution data or if these artifacts are the result of interpolation errors. Linear methods additionally cannot properly visualize curved geometries (elements or boundaries) which can greatly inhibit developers' debugging efforts. As we will show, pixel-exact visualization exhibits none of these issues, removing the visualization scheme as a source of uncertainty for engineers using ElVis.

Index Terms—High-order finite elements, spectral/hp elements, discontinuous Galerkin, fluid flow simulation, cut surface extraction, contours, isosurfaces.

1 Introduction

High-order finite element methods (a variant of which are the spectral/hp element methods considered in this work [12]) have advanced to the point that they are now commonly applied to many real-world engineering problems, such as those found in fluid mechanics, solid mechanics and electromagnetics [25, 11]. An attractive feature of these methods is that convergence can be obtained by reducing the size of an element (h adaptivity), by increasing the polynomial order within an element (p adaptivity), or by combining these two approaches. Meshes built with high-order solutions in mind can obtain the same level of accuracy with far fewer degrees of freedom than their linear counterparts [36].

Engineers working with high-order simulation data encounter significant obstacles when attempting to generate accurate visualizations. The underlying finite element basis functions are represented in terms of nonlinear functions (e.g., the high-order polynomials considered in this work), yet most visualization methods assume that the basis functions are linear. Therefore, to generate a visualization, the simulation data must first be converted into a linear format. While linear interpolation enables the engineer to produce the desired image, it introduces error into the visualization. This leads to the common question: are the features and artifacts seen in the visualization part of the high-order data, or are they errors introduced through the visualization process? Unfortunately, visualization errors arising from linear interpolation look the same as genuine solution artifacts arising from problems such as insufficient mesh resolution. With traditional interpolation-based rendering techniques, engineers are hard-pressed to differentiate between the numerous potential causes of visual artifacts. The severity of these visualization errors can be mitigated by refining the linear

- *B. Nelson and R.M. Kirby are with the School of Computing and the Scientific Computing and Imaging Institute at the University of Utah, E-mail: bnelson,kirby@cs.utah.edu.*
- *E. Liu and R. Haimes are with the Department of Aeronautics and Astronautics, MIT, E-mail: ehliu,haimes@mit.edu.*

Manuscript received 31 March 2012; accepted 1 August 2012; posted online 14 October 2012; mailed on 5 October 2012.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

approximations, but this approach does not scale well, requiring too much computational time and storage to be practical [27].

A variety of visualization algorithms developed in recent years address interpolation-based errors by using the high-order data directly. However, these techniques are scattered across multiple software packages and tools; as a result, the barrier to entry is high for users. Additionally, existing methods assume knowledge of the underlying basis functions used, requiring that the data first be converted into the appropriate format for each tool used.

To address these issues, we need an integrated visualization system that is designed specifically for high-order finite element solutions. Specifically, such a system must have the following features:

- **Extensible Architecture:** To support data originating from any high-order simulation, the system's visualization algorithms should be decoupled from the data representation, allowing them to change independently of each other. The advantage of this approach is that the visualization algorithms can be improved as new techniques and algorithms are developed, while engineers are free to choose whatever basis functions are most appropriate for the scenarios under investigation. This architecture enables the system to support methods currently in use, as well as methods that have not yet been developed.

- **Accurate Visualization:** To avoid introducing error into the visualization, the high-order system must work with the high-order data directly. Specifically, the system must be able to evaluate the solution at arbitrary locations in the domain to machine precision. The system must also support visualization methods that have been developed based on the *a priori* knowledge that the data was produced by a high-order finite element simulation. These methods will ideally make use of the smoothness properties of the high-order field on the interior of each elements, while respecting the breaks in continuity that may occur at element boundaries.

- **Interactive Performance:** In terms of computational resources required, using the high-order data directly carries significantly higher costs than using simpler linear approximations. While a high-order system is not expected to provide the same level of performance as its linear counterparts, it should provide an interactive experience on a standard desktop workstation (i.e., it should not require expensive, special purpose hardware).

In this paper we describe the Element Visualizer (ElVis), a new high-order finite element visualization system that meets the requirements listed above. We demonstrate ElVis' utility by using it to visualize finite element simulations produced by ProjectX, which is a general-purpose PDE solver with an emphasis on aerospace applications [7, 21, 6, 36, 35]. Specifically, we will consider the visualizations necessary during the debugging and verification processes of model generation.

2 Related Work

In recent years, there has been a growing awareness of the errors caused by using linear methods to visualize high-order data. This has lead to the development of many new algorithms that have been designed to accurately render high-order fields. In this section, we provide a brief overview of these algorithms.

A popular visualization technique is the use of color maps and contours on surfaces (element and geometry boundaries) and cut-surfaces. These techniques allow the engineer to gain detailed information about specific locations in the data set. There are several schemes that apply color maps to planar data. In one approach, the color map is generated by what is called a polynomial basis texture [8]. Each basis function used in the high-order field is sampled onto a triangular texture map. The colors in the triangle are not generated by linear interpolation, but instead by the linear combination of the appropriate textures, based on the triangle's order. In this way, a set of basis textures can be generated in a pre-processing step, and then, assuming there is sufficient resolution in the texture, accurate images can be generated for all high-order triangles. Another method uses an OpenGL fragment shader to calculate the field's value at each fragment's location, resulting in more accurate lookup into the color map [4]. Finally, another method analytically calculates the intersection of a plane and quadratic tetrahedra, then uses a ray tracer to apply the color map to the new primitive [32].

Most of the work dealing with the generation of contour lines deals only with 2D high-order elements. A common theme is to generate the contours in an element's reference space (which we will define in Section 3) and then transform them into global (world) space for display. One approach [15] creates contour lines in an element's reference space by subdividing the domain and using linear interpolation within these sub-domains to create a piecewise linear contour. Another approach steps along a direction orthogonal to the field's gradient [3], where each step is controlled by a user-defined step size. A method for generating contour lines over quadrilateral elements by determining the shape of the contour in reference space and then generating a polyline to approximate it was developed in [28] and later extended to linear and quadratic triangles in [29].

The only 3D contouring algorithm [13] generates contour lines on cut-planes through finite element volumes. The procedure first locates a seed point for the contour line along the element's boundary. It then steps in a direction orthogonal to the field's gradient, using a user-controlled step size, to generate a polyline representing the contour. It differs from the previously described methods in that the plane is a three-dimensional entity. At each step, the contour can, and often does, move off of the cut-surface. The method introduces a correction term to fix this problem and keep the contour on the cut-plane. As with the other object-space contour methods described, the step size is useful to determine how accurately the polyline represents the contour in world space, but is not as useful in expressing how accurate the final image is, as it can be accurate from one view but have large error in another.

Several approaches have been developed for volume rendering high-order fields. An analytic solution to the volume rendering integral was developed in [34] for linear and quadratic tetrahedra. Numerical, point-based solutions for high-order tetrahedra were presented in [37], and solutions for arbitrary elements and order have been developed as well [30]. Approaches for isosurface rendering have been developed for quadratic tetrahedra using analytical calculation of the isosurface in reference space [33] and through ray-tracing approaches [32]. Other approaches include using a ray tracer for arbitrary elements of arbitrary order [19], a point-based approach that uses particles that actively seek and distribute themselves on the isosurface [16] and a hybrid system that combines isosurface-seeking particles and ray-tracing [23]. Finally, new approaches for creating line-type features in scalar and vector fields [22] have recently been developed that are optimized for high-order fields.

3 High-Order Finite Element Methods

A finite element volume is represented by the triangulation T_H of an open, bounded domain Ω into a mesh of N_e non-overlapping elements κ_i. Ω exists in what is called the *world space*. The elements κ_i are such that $\Omega = \bigcup_{i \leq N_e} \kappa_i$ and $\kappa_i \bigcap \kappa_j = \emptyset, \forall i \neq j$. The four basic element shapes for 3D finite elements are the tetrahedron, hexahedron, (triangular) prism, and (square-base) pyramid.

3.1 Element Reference Spaces

The elements of the triangulation can come in a multitude of sizes and shapes in the world space; treating each as its own unique entity is cumbersome. Thus, it is common practice to standardize a *reference space* for each element shape. For example, one choice for the reference space of a tetrahedron is the tetrahedron with corners at $(0,0,0)$, $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$. Then a function (or composition of functions) $\Phi : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ provides the continuous, bijective mapping from the reference space to the world space. Additionally, the Jacobian determinant (henceforth simply called the Jacobian) of this transformation must be positive to guarantee that the inverse exists and that

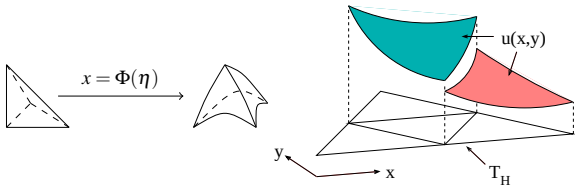


Fig. 2. (Left) - Illustration of the mapping between reference and world space for a tetrahedron. Reference points are denoted by η, and world space points by x. (Right) - Degree p polynomials interpolate the solution within elements, but discontinuities are allowed at element boundaries; 2D example.

the desired orientation is maintained. In the linear case, this mapping can be conceptualized as a combination of translation, rotation, and scaling operations. In the high-order setting, the mapping is nonlinear. A diagrammatic example of these mappings for a tetrahedron is shown in Figure 2.

Since Φ is a bijection with positive Jacobian, it can be inverted to compute the reference point corresponding to a particular world point. Unfortunately, Φ^{-1} generally does not have a closed-form. As a result, a more general root-finding scheme must be used. For high-order elements, Φ^{-1} is also nonlinear and a scheme like the Newton-Raphson Method is required.

3.2 Basis Functions

A standard finite element practice is to describe the function Φ in terms of element-wise basis functions defined on the reference space. The space spanned by these functions is finite dimensional. In this work, examples will apply a basis that is polynomial in the reference coordinates, but other choices (e.g., Fourier basis) are possible as well. For example, consider a function $F(\xi) \in \mathscr{P}^{p_1,p_2,p_3}$ with respect to the reference element, where p_1, p_2, p_3 denote (possibly) different polynomial orders in the three principle directions and ξ is a reference coordinate. This basis choice is associated with at most $(p_1+1)(p_2+1)(p_3+1)$ degrees of freedom.

3.3 Solution Representation under Continuous and Discontinuous Finite Elements

Similar to the geometry, the solution representation is also commonly written in terms of basis functions. The solution basis can be different than the geometry basis; in this work, polynomials are used again. In visualization applications, it is important to distinguish between continuous and discontinuous finite elements. In continuous finite elements, the solution is defined to be continuous across element boundaries. As a result, the value of the solution at every point in the domain is uniquely defined. With discontinuous elements, this is not the case: discontinuities can exist at element boundaries and in general, solutions are multi-valued along such boundaries; see Figure 2.

The discontinuous setting raises some challenges for visualization. For example, contour "lines" may have discontinuities at element boundaries since the underlying solution value could "jump." Similarly, isosurfaces may have holes in them. Any derived, continuous solution representation used within the visualization system is only guaranteed to be valid within individual elements.

4 The Element Visualizer (ELVIS)

ElVis is a system that has been developed to implement the features necessary for high-order visualization, as described in Section 1: namely, visualization accuracy, interactive performance, and extensible support for arbitrary high-order finite element systems.

ElVis is designed to provide visualization tools that are broadly applicable to any high-order finite element solution. ElVis' implementation is generic and aims to decouple the implementation of the visualization from the implementation of the high-order basis functions. ElVis achieves this goal through the use of plugins, which provide a simplified interface to the high-order data by exposing the minimal amount of functionality required to generate a visualization. In this way, it is broadly applicable to a wide variety of simulation products, and gives each product wide latitude on how it behaves behind the scenes. We discuss plugins in more detail in Section 4.1.

Once the data is accessible to ElVis through a plugin, ElVis can perform the required visualizations without knowledge of the details of the underlying simulation. ElVis' visualization algorithms focus particular attention on the two often competing goals of image accuracy and interactive performance. Image accuracy is obtained by devising high-order specific versions of common visualization strategies (cut-surfaces, isosurfaces, and volume rendering). Performance is achieved by careful implementation of these algorithms as parallel algorithms on a NVIDIA GPU, using the OptiX [24] ray-tracing engine and Cuda [1] as the framework. We present more details about ElVis' visualization capabilities in Section 4.2.

4.1 Extensibility Module

One of the fundamental challenges of creating a general-purpose visualization system for high-order finite element simulations is that there is no single set of basis functions that is appropriate in all simulation settings. Therefore, each simulation system chooses the basis that is most suited for the problems at hand. This means that ElVis cannot be implemented in terms of any specific basis and expect to be used with arbitrary simulation systems. The Extensibility Module addresses this issue by providing a plugin interface that acts as a bridge between the visualization system and the simulation package. The module accepts plugins written in one of two ways, each providing different trade-offs as described below. The first type of plugin is the *data conversion plugin*, which is used to convert a data set from the format used by the simulation package to the format used by ElVis' default plugin, the Nektar++ extension [2]. The second plugin type is the *runtime plugin*, which provides an interface for ElVis to interactively query the simulation data on both the CPU and GPU.

The purpose of the data conversion plugin is to convert fields and geometry from the format used by the simulation package into the native Nektar++ format used by ElVis. The Nektar++ data format is supported through a default runtime plugin (described below) that is distributed with ElVis as a reference implementation for the development of plugins for other simulation systems. Nektar++ uses a polynomial basis to represent its data, and the data conversion plugin is responsible for projecting the field from the simulation package onto the polynomial basis used by Nektar++. Projection of the data then occurs as follows. First, ElVis queries the plugin to obtain information about each element's type (e.g., hexahedron, tetrahedron) and the desired polynomial order of the converted data set. For simulations already using a polynomial basis, this can be chosen so that the projection introduces no error beyond floating-point rounding errors. For other bases, it can be set to the level needed to meet the desired accuracy requirements. ElVis then queries the plugin to determine if the resulting projection should be represented using functions that are continuous or discontinuous at element boundaries. Finally, ElVis samples the field at a collection of points determined by the choices made in the previous steps and creates the projected data set.

The advantages of this approach when compared to runtime plugins (described below) are that they will generally require less coding and, once the conversion is done, ElVis will have no runtime dependencies on the simulation package. Another advantage is that ElVis handles all of the details about file formats and data storage—the plugin is only responsible for sampling the solution. The downside is that the native internal format represents fields and geometry as the tensor product of one-dimensional polynomials [12]. Therefore, data sets from simulation packages where fields are represented by non-polynomial basis functions cannot be represented exactly, so this approach will introduce projection error into the visualization. Another disadvantage is that simulations using non-standard elements do not fit the input requirements described above and cannot be converted.

Runtime plugins are loaded into ElVis each time it is run; they provide access to a simulation's data during the rendering process. The data can be accessed directly in the format used by the simulation (e.g., polynomial or Fourier basis functions) without the need to convert formats first. However, implementing a runtime plugin re-

quires significantly more code than the data conversion approach, and it also requires a working knowledge of both OptiX and Cuda. All of ElVis' visualization algorithms are implemented on the GPU; therefore, all runtime plugins must provide a means to access data fields on the GPU.

A runtime plugin consists of three components:

- **Volume Representation Component:** This component is responsible for reading a volume on the CPU and then transferring it to the GPU. ElVis imposes only one restriction on the way in which the volume's data is represented and accessed on the GPU, leaving the choice of optimal implementation to the extension. The sole requirement is that the data be accessible to the OptiX-based ray-tracer through a specially named node in the ray-tracer's scene graph.

- **Volume Evaluation Component:** All of the visualization methods described in the next section require the ability to evaluate a high-order field and its gradient at arbitrary locations within an element. These functions must be implemented on a GPU and will be called a large number of times for each of the visualization methods discussed in the next section, so extra care must be taken to ensure that they are as efficient as possible.

- **Ray-Tracing Component:** Finally, the ray-tracing component connects the OptiX portion of ElVis to the simulation data. This component is not responsible for handling the high-level mechanics of the ray-tracer; however, it is responsible for providing some of the primitives used by ElVis to perform the ray-tracing. Examples include providing ray-element intersection tests, element and element face bounding box procedures, and tests for whether a point is located in an element.

4.2 Visualization

In this section, we discuss the visualization methods for scalar fields that are currently available in ElVis; namely, color maps and contours on cut-surfaces [20], isosurfaces [19], and volume rendering [18]. Each of these visualization methods is discussed in greater detail in the references provided. The scalar field restriction is not a fundamental limitation of the ElVis framework; rather, it is a reflection of the algorithms that have been developed for the initial release. Future releases will support visualization of high-order vector fields.

The visualization algorithms discussed below all require quick and efficient point location queries. ElVis uses a bounding volume hierarchy, which is supplied by OptiX, to accelerate all point location queries.

4.2.1 Surface Visualization

A surface visualization is where the scalar field is plotted on a surface using color maps, isocontours or both [20]. These types of visualizations, while conceptually simple, are very useful for the engineer when studying the simulation. Much of the interesting behavior occurs on or near certain domain boundaries (e.g., a wing), and it makes sense to be able to plot the behavior of the field on these surfaces accurately. An overview of the surface rendering algorithm is presented in Algorithm 1, with details in Nelson et al. [20].

ElVis supports the rendering of an arbitrary number of cut-planes and surfaces, of which the curved faces lying on domain boundaries are common choices. In Figure 1, we show an example of the pressure field on an ONERA M6 wing (see Section 5.1.2 for details) and on a plane cutting through the wing. A color map is applied and contour lines are plotted on the wing's curved surface and on the cut-plane's flat surface.

ElVis also has the ability to plot the intersection of the 3D mesh and a surface through an extension of the contouring algorithm discussed above. It is often useful to see the mesh on a surface to verify that the mesh has been generated correctly, as well as to aid in debugging. Oftentimes features may appear in the visualization that appear out of place, but turn out to be reasonable if they occur next to a mesh boundary. An example of such a feature is a discontinuous contour line in a DG field. In this scenario, a break in the contour is expected at element boundaries, but not in the interior of the element. Further examples that apply the meshing tool can be found in Section 5.

ALGORITHM 1: Surface Visualization

Step 1 - Sample the field on the surface.
if *Surface is a cell face* **then**
  Sample the scalar field using the basis functions defined for the face.
else
  Cast a secondary ray to find the enclosing element.
  Sample the scalar field using the element's basis functions.
end

Step 2 - Visualize the scalar samples.
if *Rendering Color Maps* **then**
  Use the scalar value as a lookup into a color table.
else if *Rendering Contours* **then**
  Use the scalar value to determine if the contour crosses the pixel, using the crossing tests from [20].
end

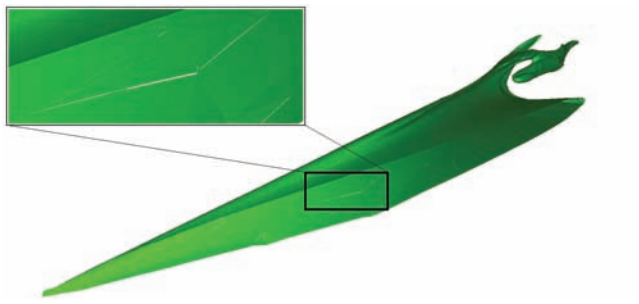


Fig. 3. Isosurface of Mach number 0.1919 for the delta wing simulation (Section 5.1.1), showing the development and roll up of the vortex structures along the leading edge and downstream of the wing. Note the crack in the surface arises because the underlying solution is from a DG method.

4.2.2 Isosurfaces

Isosurfaces in ElVis [16, 19] take advantage of the smoothness properties of high-order finite element solutions to project the field along a ray onto a polynomial. Once the polynomial is created, accurately finding the location of an isosurface is a root-finding problem [19]. We give an overview of this procedure in Algorithm 2. We note that this procedure is only valid on the interior of elements, where the solution is guaranteed to be smooth, so our isosurface algorithm operates on element-partitioned segments along the ray.

An advantage this method has over existing object-space methods is that it respects the features of the high-order data. In particular, unless care is taken, object-space isosurfacing methods such as marching cubes can miss valid features of DG simulations, such as discontinuities across element boundaries that can cause cracks in the isosurface, and isosurfaces that exist entirely within an element. An example of this can be seen in Figure 3, where we plot an isosurface of the Mach number for the delta-wing simulation described in Section 5.1.1.

ALGORITHM 2: Isosurface Algorithm

Input: A ray $R(t)$ and a list of all elements E traversed by the ray, ordered by intersection distance, and desired isovalue ρ.

foreach *Element* $E_i \in E$ **do**
  Determine ray entrance t_a and exit t_b for element E_i.
  Evaluate the field on the ray segment $[t_a, t_b]$ using interval arithmetic to obtain field bounds $[f_{min}, f_{max}]$.
  if $\rho \in [f_{min}, f_{max}]$ **then**
    Perform root-finding procedure to identify the location of the isosurface.
  end
end

4.2.3 Volume Rendering

In some scenarios, isosurfaces can be noisy and difficult to interpret. In these cases, it can be useful to use volume rendering to represent the isosurface. When applied to high-order solutions, ElVis uses an algorithm that takes advantage of the structure of the field to provide better convergence properties than those available through existing methods [18]. By doing this, ElVis is able to display more accurate images for a given amount of time.

Volume rendering proceeds in a manner similar to isosurface generation. The ray traverses the volume and as it encounters each element, it categorizes the field along the ray in the element, then applies an optimized volume rendering algorithm.

4.2.4 Combining Visualizations

A constant challenge for high-order visualization is the computational cost of each of the algorithms described above. This comes from the cost of sampling a high-order field (primarily the cost of converting world points to reference points and evaluating the field) and the cost of traversing the volume with a ray (where ray-element intersections can be of high-order and require expensive, iterative root finding procedures). Several of the visualization algorithms above have been shown to be interactive while maintaining accuracy when applied in isolation; but combining these visualization methods into a single image without taking into account their interactions leads to slow performance.

Instead, when multiple visualization methods are used in a single image, we create a system of communication and sharing between modules to minimize the cost. The first optimization is sample sharing, where samples taken for one module can be reused in another. For example, when a surface color map is generated, the samples can also be used to generate the contours. The second optimization is occlusion sharing. We render the surface modules first, since they are the cheapest, and use the depth information obtained by those modules to terminate the mesh traversal portion of the isosurface and volume rendering algorithms once the occluding structure has been reached. Finally, the isosurface and volume rendering modules share the mesh traversal algorithm, reducing the number of ray-element intersection tests that are performed.

5 Results

This section demonstrates the capabilities and advantages of ElVis through several examples. We focus our attention on cut-surface and isosurface visualizations, as ProjectX users do not currently use volume rendering in their engineering analysis. Our examples are taken largely from engineering problems solved or being worked on with ProjectX; some examples were synthetic problems designed to demonstrate specific capabilities. The following subsection describes the sources of our examples, enumerated as series of cases. Then we show results and comparisons with current visualization "best-practices" used by ProjectX developers.

5.1 Simulation Examples

We preface this subsection by noting that the upcoming cases are all characterized by quadratic geometry representations and cubic solution representations. These are not limitations of ProjectX nor ElVis; rather, both pieces of software can handle different order geometry and solution representations in different elements. However, they are representative of cases being examined by ProjectX engineers at present. Additionally, these relatively low polynomial orders present a "best case" for Visual3 in the comparisons to ElVis presented in this section. The differences that exist could only become more pronounced at higher polynomial orders.

5.1.1 Case 1

Case 1 is an isolated half delta-wing geometry with a symmetry plane running down the center chord-line of the wing. The case was originally proposed by [14] to demonstrate the efficacy of their adaptation strategy. Delta-wings are common geometries for CFD testing due to their relatively simple geometry and the complexities involved in the vortex formation along the leading edge of the wing and the subsequent roll-up of those vortices. The equations being solved are the compressible Navier-Stokes equations. The flow conditions are $M_\infty = 0.3$, $Re_c = 4000$, and $\alpha = 12.5$. The solution was obtained through ProjectX, using an output-adaptive automated solution strategy [35].

The delta-wing geometry is linear. The computational mesh consists of 5032 linear, tetrahedral elements with 10434 total faces (linear triangles). The solution was computed with cubic polynomials.

5.1.2 Case 2

Case 2 is an isolated ONERA M6 wing again with a symmetry plane running down the center chord-line [26]. We are presenting a subsonic, turbulent flow over the same geometry (transonic was not available). The flow conditions are $Re_c = 11.72 \times 10^6$, $M_\infty = 0.3$, and $\alpha = 3.06$. The flow is fully turbulent; the RANS equations are being solved with the Spalart-Allmaras model for closure. As before, the solution was obtained using ProjectX.

The computational mesh consists of 70631 quadratic elements with 146221 faces (quadratic triangles); meshing limitations restrict us to a quadratic geometry representation. The solution polynomials are cubic.

5.1.3 Case 3

Case 3 is synthetic example meant to demonstrate the mesh-plotting capability of ElVis. The geometry is a hemisphere. The mesh is composed of 443 quadratic tetrahedra with 1037 faces (quadratic triangles). Case 2 also uses curved elements, but the curvature is almost unnoticeable except on boundary faces. This mesh has noticeably curved elements away from the geometry as well.

5.1.4 Case 4

Case 4 is another synthetic example designed to show how ElVis can display negative Jacobians naturally. "Real" computational meshes with negative Jacobians are not usable; as a result these are discarded. Thus it was simpler to create a one element mesh with very obvious negative Jacobian issues. The mesh consists of one quadratic tetrahedron with four faces (quadratic triangles). The tetrahedron has corners at (0,0,0), (1,0,0), (0,1,0), and (0,0,1). The mesh was created by first placing quadratic nodes at their locations on the linear element. Then the quadratic node at (0.5,0,0) was moved to (0.5,0,0.6), causing the element to intersect itself and leading to negative Jacobians.

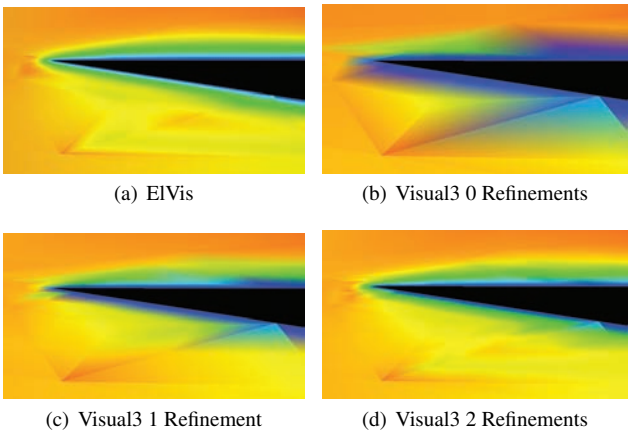
5.2 ProjectX

As mentioned above, all results generated in this section are from solutions produced by the ProjectX software. The results considered in this paper arise from solutions of the compressible Navier-Stokes and RANS equations. ProjectX implements a high-order Discontinuous Galerkin (DG) finite element method; DG features relevant to visualization were discussed in Section 3. It also strives to increase the level of solution automation in modern CFD by taking the engineer "out of the loop" through estimation and control of errors in outputs of interest (e.g., lift, drag) [7, 36]. Solution automation is accomplished through an iterative mesh optimization process, which is driven by error estimates based on the adjoints of outputs of interest [35].

We have worked closely with ProjectX engineers to endow ElVis with visualization and interface features that they would find useful. Visualization has great potential to aid ProjectX developers' ability to understand and analyze their solutions. It has perhaps even greater application in the realm of software debugging, where visual accuracy is of the utmost importance since it is often difficult to discern visual artifacts from genuine or erroneous (i.e., a result of a software bug) solution features. As we will discuss below, to date, visual inaccuracies in their current software have often inhibited ProjectX engineers' analysis and debugging efforts.

5.3 Comparison Visualization Software

ProjectX developers currently use Visual3 [10, 9] to examine and attempt to understand their solution data. The reasons are simple: Visual3 is freely available, can deal with general (linear) element types,



(a) ElVis (b) Visual3 0 Refinements

(c) Visual3 1 Refinement (d) Visual3 2 Refinements

Fig. 4. Plotting Mach Number at the leading edge of the delta wing (Case 1) at the symmetry plane with ElVis (a) and Visual3 using 0 (b), 1 (c), and 2 (d) levels of refinement.

is extendable and there is local knowledge of this software. Visual3 supports a number of usage modes; e.g., cut-planes, surface rendering, surface contours, isosurfaces and streamlines/streaklines. ProjectX developers use Visual3 to explore their solution data and to support debugging of all aspects of their solver.

At present, Visual3 is rather dated and more modern methods (e.g., those discussed in Section 2) exist. Visual3 is a complete, well-tested, and thoroughly-documented application with a functional GUI and a clearly-specified API. Cutting-edge visualization software often lack these features. Moreover, until recently, native high-order visualization on commodity hardware was not possible. Ultimately, ProjectX developers are not members of the visualization community; they do not have the time or expertise to dedicate towards turning prototype software and technology demonstrators into full-fledged visualization systems. As a result, ProjectX developers continue to use Visual3 since it is a robust and familiar tool.

Visual3 operates on a set of linear shape primitives (tetrahedron, hexahedron, pyramid, and prism). The solution data and computational mesh must be transferred onto these linear primitives (via interpolation) for visualization to occur. Since ProjectX produces high-order solutions on curved meshes, some error is introduced through the linear interpolation. To decrease visualization error, each individual element of the visualization mesh can be uniformly refined (in the reference space) a user-specified number of times. Due to compute time and memory constraints, ProjectX developers typically use 0 or 1 level of refinement. In this paper, 2 levels of refinement are performed in some cases for the sake of comparison. However there is usually insufficient memory for 3 levels of refinement on our typical workstations; additionally in the ever-larger simulations run by ProjectX developers and users, 2 levels of refinement is often infeasible as well.

5.4 Visualization Accuracy

As discussed above, the primary motivation behind this work is the ability to achieve accurate visualizations of high-order data. In this section, we demonstrate ElVis' accuracy, and describe how this enables ProjectX engineers to interpret and debug their simulation data more effectively.

5.4.1 Surface Rendering

We begin with visualizations of the leading edge of the delta wing (Section 5.1.1) at the symmetry plane, which we show in Figure 4. The black region is the airfoil at mid-span; the wing is not plotted so that it does not occlude any of the details of the boundary layer. This set of figures compares the pixel-exact rendering of ElVis to linearly-interpolated results from Visual3. For comparison, Visual3 results are posted with 0, 1, and 2 levels of uniform refinement.

As rendered by Visual3 with 0 refinements (Figure 4b), the characteristics of the solution are entirely unclear. The boundary layer

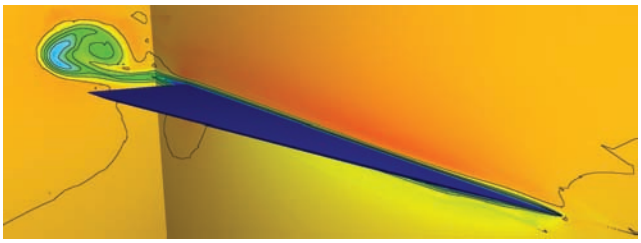


Fig. 5. A zoomed out ElVis generated image of the delta wing from Case 1 showing the location of the cut-plane used in Figure 6. The cut plane is located 0.2 chords behind the trailing edge of the wing.

appears wholly unresolved and severe mesh imprinting can be seen. It is not clear whether the apparent lack of resolution is due to a poor quality solution, bugs in the solver, or visualization errors. Even at one level of refinement (Figure 4c), the Visual3 results are still marred by visual errors. Again, mesh imprinting is substantial and the thickness of the boundary layer is not intelligible.

The Visual3 results continue to improve at 2 levels of refinement (Figure 4d) with the rough location of the boundary layer finally becoming apparent. But mesh imprinting is still a problem, and engineers could easily interpret this image as the result of a poor quality solution. Only the ElVis result (Figure 4a), clearly indicates the location of the boundary layer and clearly demonstrates where solution quality is locally poor due to insufficient mesh resolution. Although not shown here, distinct differences similar to the effects at the leading edge on the symmetry plane are also observed at the delta wing's trailing edge and along its entire leading edge.

ProjectX developers were genuinely surprised at the difference between between Figures 4d and 4a. In fact, since users had only generated and viewed Figures 4b and 4c previously, the common misconception was that resolution at the leading edge (and indeed in many other regions around the wing) was severely lacking.

However, had the ProjectX solver been subject to a software bug, engineers expressed that they would have been hard pressed to interpret Visual3 results to aid these efforts. Specifically, at 0 or 1 level of refinement, the visualization quality is so poor that developers are often unable to discern the precise source of solution artifacts. Unfortunately, a misdiagnosis can lead to a great deal of time wasted on a "wild goose chase." As a result, visualization has not played as large a role as it could in debugging practices.

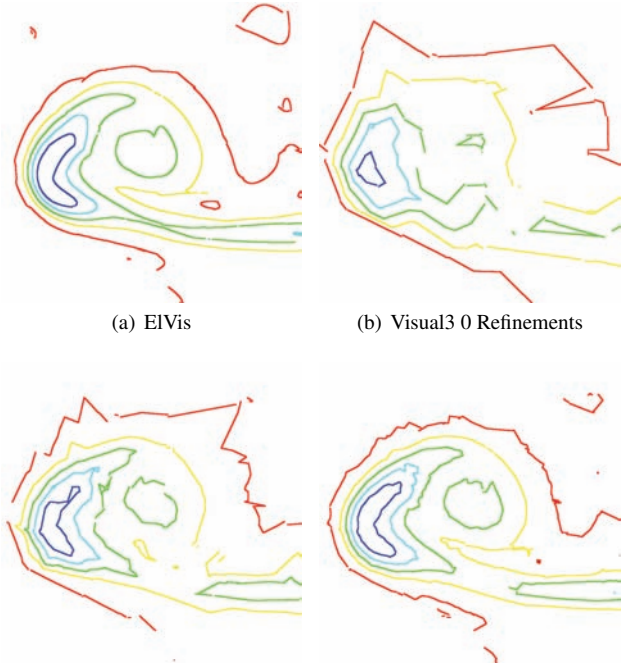
5.4.2 Contour Lines

For our next example, we illustrate the generation of contour lines. Contours are useful visualization primitives since they, unlike color plots, limit the amount of information being conveyed and allow for more detailed and targeted images. In particular, it can be difficult to interpret the magnitude and shape of a field's gradient through color maps; this is much easier with contours.

In Figure 6, we show a comparison of contours on a cut-plane behind the trailing edge of the delta wing (Section 5.1.1). The images were generated using ElVis and Visual3, once again using 0, 1, and 2 levels of refinement for the latter. Figure 5 provides a zoomed out view, showing the location of the cut-plane relative to the wing.

The images in Figure 6 reiterate the observations from the discussion of surface rendering. With 0 and 1 level of refinement (Figure 6b and c), most of the contour lines produced by Visual3 are extremely inaccurate. These images do little to illuminate the vortex structure they are trying to show. The situation improves somewhat with 2 levels of refinement in Visual3 (Figure 6d), but substantial errors remain.

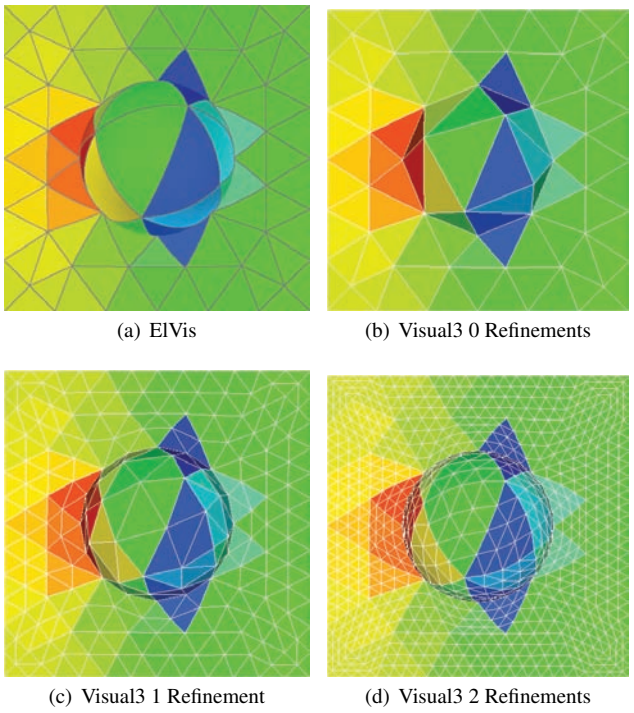
As before with surface rendering, the errors present in the Visual3 outputs at all tested levels of refinement are too great to properly support visualization as a debugging tool. The resolution of vortex structures like the one shown in Figures 5 and 6 is a prime candidate for the application of high-order methods, because vortexes are smooth flow features. They are also extremely common, arising as important features for lift and drag calculations in any 3D lifting flow, amongst



(a) ElVis (b) Visual3 0 Refinements

(c) Visual3 1 Refinement (d) Visual3 2 Refinements

Fig. 6. Plotting Mach number contours at the trailing edge of the delta wing (Case 1). An overview of this scenario is shown in Figure 5, with a detailed view of the contours on the trailing cut-plane generated by ElVis (a) and Visual3 using 0 (b), 1 (c), and 2 (d) levels of refinement. For visual clarity, we have modified the contour lines produced by both systems so they are thicker than the default one pixel width.



(a) ElVis (b) Visual3 0 Refinements

(c) Visual3 1 Refinement (d) Visual3 2 Refinements

Fig. 7. A top-down view of the hemisphere from Case 3. The mesh plotting tools of Visual3 and ElVis are enabled.

(a) (b)

Fig. 8. Views of Case 4 (negative Jacobians in a single tetrahedron) rendered with ElVis. Colors show Jacobian values from -0.5 to 0.0. Left is a view of the $x+y+z=1$ face being intersected by the $x-y$ face. Right is a view of the $x-z$ face, where the self-intersection effect of the quadratic node at (0.5,0,0.6) is apparent.

other things. Here, we are only showing a solution with cubic polynomials; with the even higher polynomial orders that could be applied to vortex flows, linear interpolation-based visualization methods will be even more inadequate.

5.5 Curved Mesh Visualization

Following on the footsteps of the previous section on accuracy, it is impossible to accurately visualize curved geometries with only linear interpolation techniques. Here we will examine a mesh where highly curved elements can be seen clearly: the hemisphere from Case 3. With 0 levels of refinement (Figure 7b), the Visual3 results are not obviously hemispherical at all. Figures 7c and 7d, showing 1 and 2 levels of refinement respectively, provide successively greater indication that the underlying geometry is in fact curved. However, without the color scheme which helps outline true element boundaries (as opposed to boundaries generated through refinement), typical Visual3 displays can make it difficult to discern which computational element contains to a particular point on the screen. This issue is further compounded by the fact that curved elements are linearized.

Figure 7a does not have any of these issues. Produced by ElVis, this representation accurately represents the curved surface. Engineers would be readily able to localize particular flow features or artifacts to specific elements for further investigation during debugging or analysis. ProjectX developers and users are given easy and direct access to curved geometries and curved elements, capabilities that were impossible using Visual3.

5.6 Negative Jacobian Visualization

Negative Jacobians can be extremely difficult to detect. In general, negative Jacobians manifest in elements whose reference to world space mapping is not invertible since it becomes multi-valued. As a result, their presence can lead to severe stability and convergence issues. In general, detecting negative Jacobians amounts to a multivariate root-finding problem of high-order polynomials. This procedure is very costly and it is not practical to search every element. Instead, finite element practitioners typically check for negative Jacobians at a specific set of sample points (usually related to the integration rules used); unfortunately sampling is not a sufficient condition for detection.

If negative Jacobians are suspected (e.g., through convergence failure of the solver), visualization of problematic elements is a potential path for deciding whether Jacobian issues played a role. At present, ProjectX developers have no way of directly visualizing negative Jacobians. Linear tetrahedral elements have constant Jacobians; thus the linear visualization mesh produced for Visual3 is of little use when it is used to visualize inverted elements. However, ElVis is not subject to such constraints since it handles curved elements naturally. Case 4 demonstrates our ability to directly visualize negative Jacobians as shown in Figure 8.

5.7 Distance Function Visualization

A distance function is a scalar field defined by

$$d(\vec{x}) = inf\{|\vec{x}-\vec{p}| \ : S(\vec{p}) = 0\} \quad (1)$$

where S is an implicit definition of a surface. In regard to ProjectX, the distance function is an important part of the Spalart-Allmaras turbulence model, which is used in Case 2. In fact, wall-distances are needed by most turbulence models. In aerospace applications, turbulent effects typically first arise due to very near-wall viscous interactions. The consequences of incorrect distance computations can vary

widely, from having no effect, to producing incorrect results, to reducing solver robustness or even preventing convergence all together.

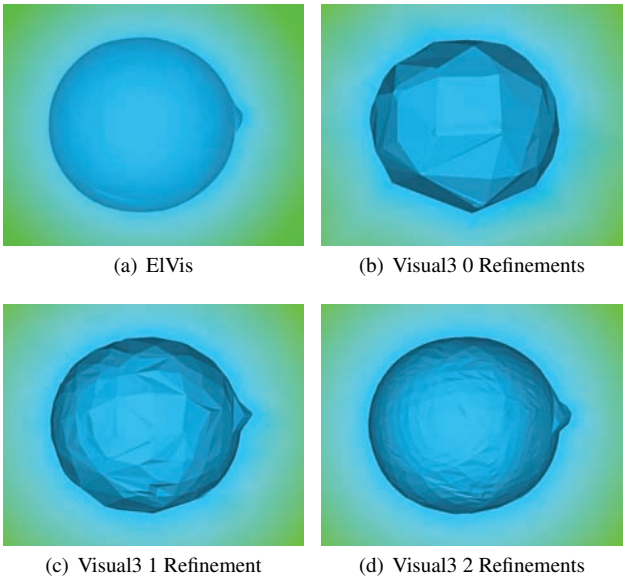
Visualization has the potential to be a valuable first attempt at diagnosing distance calculation errors. Developers can inspect the distance field for smoothness and make other qualitative judgments on the quality of the computation. Although visualization cannot guarantee that distance computations are correct, they can provide confidence and more importantly one would hope that visualization would make substantial errors in distance computations apparent.

Existing visualization packages introduce error into this visualization because they must interpolate high-order surfaces and interpolate the distance data, the latter of which is not even a polynomial field. Linear interpolation introduces a number of problems. It is impossible to judge the quality of distance calculations without at least being able to see the true shape of the underlying geometry. As a result, ProjectX developers typically find themselves unable to use visualization to aid in debugging distance computations; let us see why.

Figure 9 is meant to demonstrate the difficulty experienced by ProjectX developers when using Visual3 to help diagnose problems with distance function computations. In the 0 refinement case (Figure 9b), the visualization mesh is so coarse that almost nothing can be learned from this image, except possibly that the distance computation is not producing completely random numbers. The 1 refinement case Figure 9c) does little to improve the situation. Here, a large protrusion is visible on the right and a large recess is visible near the top of the isosurface. Other confusing-looking regions are also present. After 2 levels of refinement (Figure 9d), Visual3 gives strong evidence that a bug is present in the distance computation, but the linear interpolation is still preventing rendering of the expected smooth surface.

On the other hand, ElVis renders (Figure 9a) a smooth surface with one substantial protrusion (corresponding to the protruded region seen in Figures 9c and 9d). From the ElVis result, the fact that the distance computation is wrong is obvious. Additionally, the ElVis result was obtained in seconds, in stark contrast to performing 2 uniform refinements in Visual3.

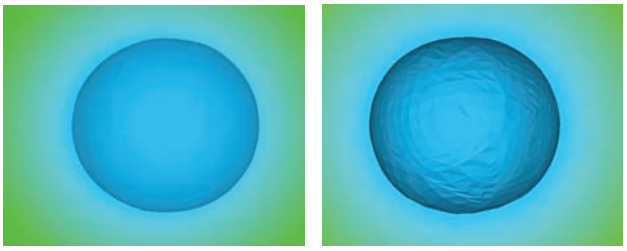
Figure 10 shows the result from plotting the correct distance function in Visual3 and in ElVis. The effect of the bug was very local (manifested in the large protrusion on the right side). The images from Figure 9 are largely unchanged, hence only the highest resolution Visual3 image was replicated. Indeed, the 0 refinement results from Visual3 with the bug fixed appears indistinguishable from Figure 9b, making



(a) ElVis (b) Visual3 0 Refinements

(c) Visual3 1 Refinement (d) Visual3 2 Refinements

Fig. 9. Plotting the isosurface for a distance of 6.2886 to the surface of the ONERA wing (Case 2) with ElVis (a) and Visual3 using 0 (b), 1 (c), and 2 (d) levels of refinement. Here, the underlying distance computation has a bug.



(a) ElVis (b) Visual3 2 Refinements

Fig. 10. The same view as shown in Figure 9 but with the underlying distance computation fixed.

this level of Visual3 resolution useless for debugging. At 1 level of refinement, the right-side bump is gone, but there are so many other recesses and protrusions that ProjectX developers indicate they would have little confidence that the distance evaluation is correct. After 2 levels of refinement (Figure 9d), the Visual3 results look believable for a linear interpolation of the distance field. Nonetheless, ProjectX developers indicated that they would much rather have debugged the distance function with ElVis, even if Visual3 could perform more refinements without additional compute and storage overhead.

In fact, ProjectX developers used the distance computations shown in Figure 9 for a period of months before finding the bug that caused the large protrusion shown in the isosurfaces. They had checked the distance function using views similar to those shown in Figures 9b and 9c. However, due to the lack of clarity in those images and since the solver appeared to be performing reasonably, no issues were suspected.

When presented with the before and after views from Figures 9a and 10a, one ProjectX developer expressed great frustration that ElVis was not available during the development of the distance function. It would have saved him many hours of confusion, greatly accelerating the debugging process by providing clear and direct access to the underlying data.

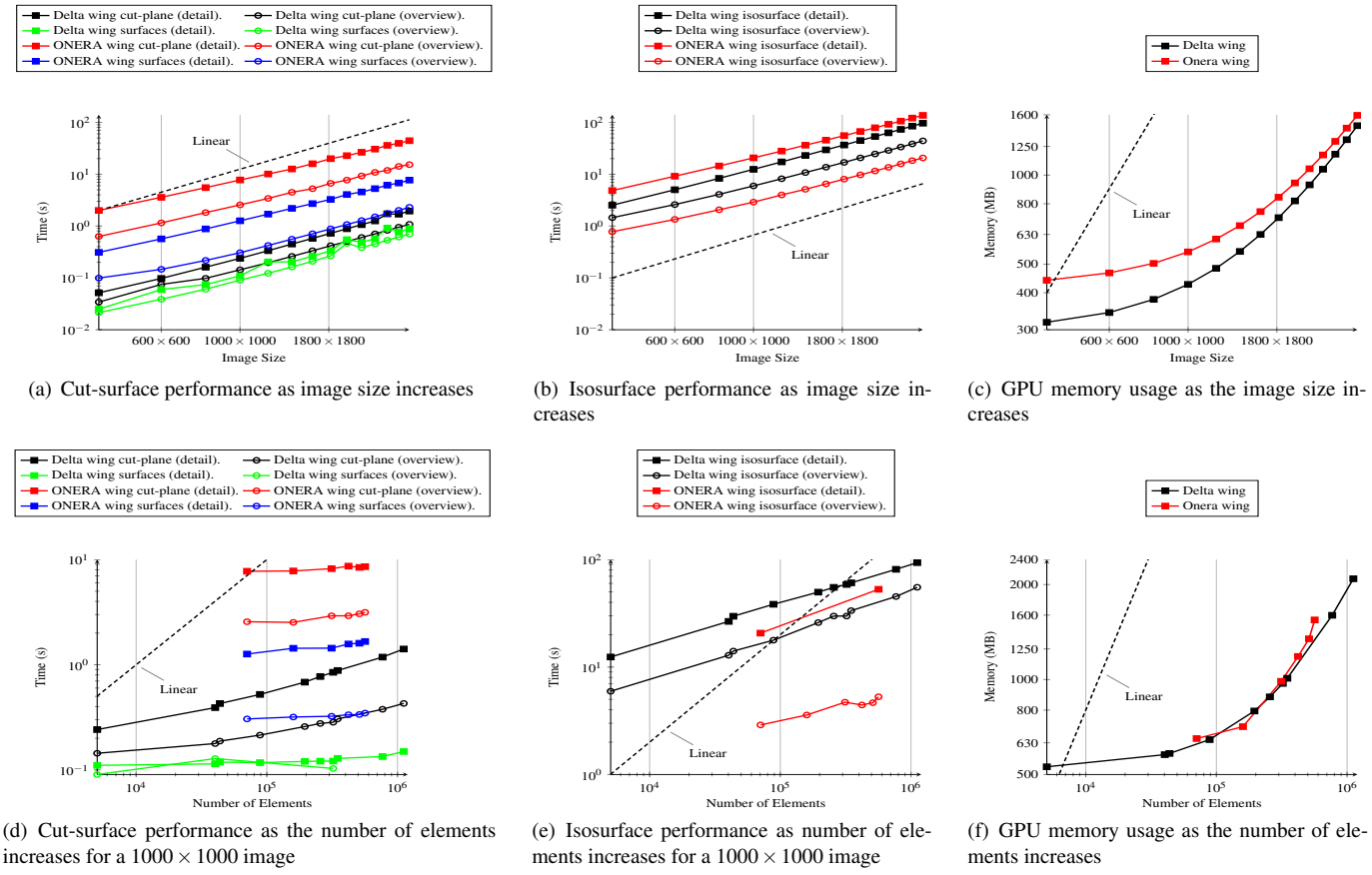
5.8 Performance

For ElVis to be useful, it must be able to provide images that are both accurate and interactive. To examine performance, we tested three different scenarios in both the delta and ONERA wing data sets. For test 1, we rendered a color map and ten contour lines on a cut-plane through each wing (similar to the view shown in Figure 5). Test 2 measures the performance of rendering color maps and ten contour lines on the element surfaces directly (as shown in Figure 1). The final test consists of an isosurface rendering in each data set. The results of these performance tests, along with the amount of memory used by the GPU during rendering, is shown in Figure 11.

We found that visualizations of regions with highly anisotropic elements performed worse than regions without. The root cause of this behavior is that these types of elements often produce bounding boxes that significantly overestimate the element's footprint, which reduces the effectiveness of ElVis' acceleration structures. To illustrate the practical impact, we rendered views with many anisotropic elements (denoted as "detail" in Figure 11) and views with few (denoted as "overview").

For all test cases, we measured performance and memory usage as the image size increases and as the number of elements in the data set increases. To create the data sets, the implied metric fields from Case 1 and Case 2 were scaled and new meshes were generated [17]. Tests were run on a desktop workstation equipped with an NVIDIA Tesla C2050 GPU and Intel Xeon W3520 quad-core processor running at 2.6 GHz. The Cuda and OptiX kernels were executed using double precision floating point numbers.

Figures 11(a) and 11(b) indicate that our method scales well with increasing image size. Figures 11(d) and 11(e) show that it also scales better than linear as the number of elements in the volume increases. This is important since, compared to traditional finite element methods which routinely use 100s of millions of elements, high-order solutions



(a) Cut-surface performance as image size increases

(b) Isosurface performance as image size increases

(c) GPU memory usage as the image size increases

(d) Cut-surface performance as the number of elements increases for a 1000×1000 image

(e) Isosurface performance as number of elements increases for a 1000×1000 image

(f) GPU memory usage as the number of elements increases

Fig. 11. Performance and memory usage for the delta and ONERA wing when varying the image size and number of elements.

rarely consist of volumes with as much as a million elements. In fact, the only real constraint on volume size is the memory required, shown in Figure 11(f), which limits the size of the volume that can be stored on the Tesla 2050 to approximately 1.5 million elements. The isosurface algorithm we are currently using is an adapted version of the algorithm shown by Nelson et al. [19] that has not yet been optimized for GPU execution, which results in the relatively poor performance of isosurfaces when compared to cut-surfaces. We plan on developing new, GPU-specific algorithms for isosurface generation to address this issue.

6 Conclusion

This paper presents ElVis, a new high-order finite element visualization system. ElVis was designed with an extensible architecture; it provides interactive performance; and it produces accurate, pixel-exact visualizations. The final point deserves further emphasis: ElVis makes few assumptions about the underlying data, nor does it use approximations when evaluating the solution. This degree of accuracy is a substantial step beyond what is possible in contemporary, linear interpolation-based methods. ElVis provides users with direct access to their finite element solution data either through conversion to known storage formats or by querying user-provided code directly. Direct data access is a capability previously unavailable to most finite element practitioners, and judging from the reactions of ProjectX developers, this capability is immensely valuable. We show that the software design elements behind ElVis reduce barriers to entry (in terms of coding effort) for new users. ElVis is also readily available as a full-fledged, ready-to-use application, making it a good choice for finite element practitioners seeking a native high-order visualization system.

The capabilities demonstrated here are the components of ElVis' initial release, which has been focused on scalar field visualization. This is not a fundamental limitation of the software; future releases will address additional visualization capabilities to address additional user requests. In particular, our next release will address vector fields. While techniques exist for generating streamlines in high-order fields [31, 5], it is not immediately apparent how this will extend to a GPU implementation; additional capabilities will need to be developed for ElVis to support these features. Additionally, the direct-access to solution data will allow ElVis to naturally handle solutions involving cut cells [7], a capability not available in any current visualization system.

Even as it stands now, every ProjectX developer wanted to know how to "get [ElVis] on my computer" or "when can I start using [ElVis]." ElVis fills a major gap that has existed in scientific visualization. While solvers are moving toward high-order methods, visualization systems continue to apply linear interpolations. ProjectX developers and users were hard-pressed to debug their solver and analyze the results it produced. High levels of visualization errors caused developers to misdiagnose bugs and arrive at erroneous conclusions about mesh resolution, amongst other issues. These problems could have been avoided with ElVis. Ultimately, everyone involved agrees that ElVis would be a welcome and valuable addition to their kit of development, debugging, and analysis tools.

Acknowledgments

The authors would like to thank the entire ProjectX team for their many contributions, including the solutions used in our examples and many insightful discussions over the course of this work. This work is supported under ARO W911NF-08-1-0517 (Program Manager Dr. Mike Coyle), Department of Energy (DOE NET DE-EE0004449), and the DOE Computational Science Graduate Fellowship (DOE DE-FG02-97ER25308). Infrastructure support provided through NSF-IIS-0751152.

References

[1] Cuda. developer.nvidia.com/category/zone/cudazone, 2012.

[2] Nektar++. http://www.nektar.info, 2012.

[3] J. Akin, W. Gray, and Q. Zhang. Colouring isoparametric contours. *Engineering Computations*, 1:36–41, 1984.

[4] M. Brasher and R. Haimes. Rendering planar cuts through quadratic and cubic finite elements. In *Proceedings of the conference on Visualization '04*, VIS '04, pages 409–416, Washington, DC, USA, 2004. IEEE Computer Society.

[5] G. Coppola, S. J. Sherwin, and J. Peiró. Nonlinear particle tracking for high-order elements. *J. Comput. Phys.*, 172:356–386, September 2001.

[6] L. T. Diosady and D. L. Darmofal. Massively parallel solution techniques for higher-order finite-element discretizations in CFD. In *Adaptive High-Order Methods in Computational Fluid Dynamics*. World Scientific, 2011.

[7] K. J. Fidkowski and D. L. Darmofal. A triangular cut-cell adaptive method for higher-order discretizations of the compressible Navier-Stokes equations. *J. Comput. Phys.*, 225:1653–1672, 2007.

[8] B. Haasdonk, M. Ohlberger, M. Rumpf, A. Schmidt, and K. G. Siebert. Multiresolution visualization of higher order adaptive finite element simulations. *Computing*, 70:181–204, July 2003.

[9] R. Haimes and D. Darmofal. Visualization in computational fluid dynamics: A case study. In *IEEE Computer Society, Visualization*, pages 392–397. IEEE Computer Society Press, 1991.

[10] R. Haimes and M. Giles. Visual3: Interactive unsteady unstructured 3d visualization. AIAA 91-0794, 1991.

[11] G. Karniadakis, E. Bullister, and A. Patera. A spectral element method for solution of two- and three-dimensional time dependent Navier-Stokes equations. In *Finite Element Methods for Nonlinear Problems, Springer-Verlag*, page 803, 1985.

[12] G. E. Karniadakis and S. J. Sherwin. *Spectral/hp element methods for CFD*. Oxford University Press, New-York, NY, USA, 1999.

[13] G. D. Kontopidis and D. E. Limbert. A predictor-corrector contouring algorithm for isoparametric 3d elements. *International Journal for Numerical Methods in Engineering*, 19(7):995–1004, 1983.

[14] T. Leicht and R. Hartmann. Error estimation and anisotropic mesh refinement for 3d laminar aerodynamic flow simulations. *J. Comput. Phys.*, 229:7344–7360, 2010.

[15] J. L. Meek and G. Beer. Contour plotting of data using isoparametric element representation. *International Journal for Numerical Methods in Engineering*, 10(4):954–957, 1976.

[16] M. Meyer, B. Nelson, R. Kirby, and R. Whitaker. Particle systems for efficient and accurate high-order finite element visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13:1015–1026, 2007.

[17] T. Michal and J. Krakos. Anisotropic mesh adaptation through edge primitive operations. AIAA 2012-159, 2012.

[18] B. Nelson. *Accurate and Interactive Visualization of High-Order Finite Element Fields*. PhD thesis, University of Utah, 2012.

[19] 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.

[20] 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.

[21] T. Oliver and D. Darmofal. Impact of turbulence model irregularity on high-order discretizations. AIAA 2009-953, 2009.

[22] C. Pagot, D. Osmari, F. Sadlo, D. Weiskopf, T. Ertl, and J. Comba. Efficient parallel vectors feature extraction from higher-order data. *Computer Graphics Forum*, 30(3):751–760, 2011.

[23] C. Pagot, J. Vollrath, F. Sadlo, D. Weiskopf, T. Ertl, and J. ao Luiz Dihl Comba. Interactive Isocontouring of High-Order Surfaces. In H. Hagen, editor, *Scientific Visualization: Interactions, Features, Metaphors*, volume 2 of *Dagstuhl Follow-Ups*, pages 276–291. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2011.

[24] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich. Optix: A general purpose ray tracing engine. *ACM Transactions on Graphics*, August 2010.

[25] A. Patera. A spectral method for fluid dynamics: Laminar flow in a channel expansion. *J. Comp. Phys.*, 54:468, 1984.

[26] V. Schmitt and F. Charpin. Pressure distributions on the onera-m6-wing at transonic mach numbers. AGARD AR 138, 1979.

[27] W. Schroeder, F. Bertel, M. Malaterre, D. Thompson, P. Pebay, R. O'Barall, and S. Tendulkar. Framework for visualizing higher-order basis functions. In *Visualization, 2005. VIS 05. IEEE*, pages 43 – 50, 2005.

[28] C. Singh and D. Sarkar. A simple and fast algorithm for the plotting of contours using quadrilateral meshes. *Finite Elem. Anal. Des.*, 7:217–228, December 1990.

[29] C. Singh and J. Singh. Accurate contour plotting using 6-node triangular elements in 2d. *Finite Elem. Anal. Des.*, 45:81–93, January 2009.

[30] M. Üffinger, S. Frey, and T. Ertl. Interactive high-quality visualization of higher-order finite elements. *Computer Graphics Forum*, 29(2):337–346, 2010.

[31] 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. *J. Sci. Comput.*, 38(2):164–184, Feb. 2009.

[32] D. F. Wiley, H. Childs, B. Hamann, and K. Joy. Ray casting curved-quadratic elements. In O. Deussen, C. D. Hansen, D. Keim, and D. Saupe, editors, *Data Visualization 2004*, pages 201–209. Eurographics/IEEE TCVG, ACM Siggraph, 2004.

[33] D. F. Wiley, H. R. Childs, B. F. Gregorski, B. Hamann, and K. I. Joy. Contouring curved quadratic elements. In *Proceedings of the symposium on Data visualisation 2003*, VISSYM '03, pages 167–176, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

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

[35] M. Yano and D. Darmofal. An optimization framework for anisotropic simplex mesh adaptation: application to aerodynamic flows. AIAA 2012–0079, Jan. 2012.

[36] M. Yano, J. M. Modisette, and D. Darmofal. The importance of mesh adaptation for higher-order discretizations of aerodynamic flows. AIAA 2011–3852, June 2011.

[37] Y. Zhou and M. Garland. Interactive point-based rendering of higher-order tetrahedral data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):2006, 2006.