2023

Optimized Implementation of Recursive Sub-Division Technique for Higher-Order Finite-Element Isosurface and Streamline Visualization

VISUALIZING HIGHER-ORDER ELEMENTS

Scott T. Imlay Yves-Marie Lefevbre, Scott Fowler,
Michael Saunders, and John Goetz
Tecplot Inc., Bellevue, WA, 98006

Optimized Implementation of Recursive Sub-Division Technique for Higher-Order Finite-Element Isosurface and Streamline Visualization

Scott T. Imlay[1], Yves-Marie Lefebvre[2], Scott Fowler[3], Michael Saunders[4], and John Goetz[5]
Tecplot Inc., Bellevue, WA, 98006

Higher-Order finite-element CFD methods have the potential to reduce the computational cost to achieve a desired solution error. These techniques have been an area of research for many years and are becoming more widely available in popular CFD codes. CFD visualization software is lagging behind the development of higher-order CFD analysis codes. This paper discusses a technique for visualizing isosurfaces and streamlines in higher-order element solutions with reduced memory usage. The technique recursively subdivides higher-order elements into smaller linear sub-elements where the isosurface can be extracted using standard marching-tets techniques. Memory usage is minimized by discarding unneeded sub-elements. In a previous paper this technique was demonstrated with higher-order quadratic hexahedra, tetrahedra, prism, and pyramid elements with Lagrangian polynomial basis functions. In this paper, the technique is extended to cubic elements and streamlines, and performance optimization are discussed. The results are compared to other techniques for visualization of higher-order element isosurfaces.

[1] Advisory Software Development Engineer, P.O. Box 52708, Bellevue, WA, Senior Member AIAA.
[2] Chief Technology Officer, P.O. Box 52708, Bellevue, WA.
[3] Tecplot Product Manager, P.O. Box 52708, Bellevue, WA.
[4] Advisory Software Development Engineer, P.O. Box 52708, Bellevue, WA.
[5] Senior Software Development Engineer, P.O. Box 52708, Bellevue, WA.

I. Introduction

The use of higher-order (greater than second order) computational fluid dynamics (CFD) methods is increasing. Popular government and academic CFD codes such as FUN3D, KESTREL, and SU2 have released, or are planning to release, versions that include higher-order methods. This is because higher-order accurate methods offer the potential for better accuracy and stability[1], especially for unsteady flows. This trend is likely to continue.

Commercial visual analysis codes are not yet providing full support for higher-order solutions. The CFD 2030 vision states "…higher-order methods will likely increase in utilization during this time frame, although currently the ability to visualize results from higher order simulations is highly inadequate. Thus, software and hardware methods to handle data input/output (I/O), memory, and storage for these simulations (including higher-order methods) on emerging HPC systems must improve. Likewise, effective CFD visualization software algorithms and innovative information presentation (e.g., virtual reality) are also lacking." The isosurface algorithm described in this paper is the first step toward improving higher-order element visualization in the commercial visualization code Tecplot 360.

Higher-order methods can be based on either finite-difference methods or finite-element methods. While some popular codes use higher-order finite-difference methods (OVERFLOW, for example), this paper will focus on higher-order finite-element techniques. Specifically, we will present a memory-efficient recursive subdivision algorithm for visualizing the isosurface of higher-order element solutions. In previous papers[5,6] we demonstrated this technique for quadratic tetrahedral, hexahedral, pyramid, and prism elements with Lagrangian polynomial basis functions. In this paper we discuss the integration of these techniques into the engine of the commercial visualization code Tecplot 360 and discuss speed optimizations. We also discuss the extension of the recursive subdivision algorithm to cubic tetrahedral and pyramid elements, and quartic tetrahedral elements. Finally, we discuss the extension of the recursive subdivision algorithm to the computation of streamlines.

II. Approach

A. Related Work

Nearly all isosurfacing techniques for higher-order finite-elements involve subdivision of the higher-order element into a number of sub-elements which are then processed with standard linear methods. There are a couple of variations on subdivision technique. The simplest is to subdivide a specified number of times, adding nodes via interpolation using the element's basis functions, until the isosurface through the set of linear sub-elements sufficiently approximates the isosurface through the higher-order element. One example of this approach is the work of Remacle et. al.[3] where the local refinement is terminated when the "visualization error" is below a desired threshold. A second approach is by Thompson and Pebay[2] who first add nodes at minima and maxima within the element and on the element faces, and then tesselates the resulting existing and new nodes to get a linear subdivision. This technique is guaranteed to give a topologically correct isosurface, but the error of the isosurface may still be high and the cost of finding minima and maxima is non-trivial.

We use recursive subdivision technique similar to Remacle et. al.[3] However, we further minimize the computational time by discarding all sub-elements that don't contain the isosurface. This paper describes the optimized implementation of the algorithm described by Imlay et. al.[5,6] in the engine of Tecplot 360, the extension of the algorithm to cubic and quartic elements, and the extension of the algorithm to the computation of streamlines.

B. Higher-Order Tetrahedra and Pyramid Basis Functions

The Lagrangian basis functions for quadratic hexahedral, tetrahedral, prism, and pyramid element types were covered in a previous papers[5,6]. This paper describes the cubic Lagrangian basis functions for the tetrahedra and pyramid element types.

Cubic Tetrahedral Basis Functions

The cubic trahedra has 20 nodes as shown in figure 1: fours nodes at the corners and two non-corner nodes on each of the six edges. Figure 1 also shows the local coordinate system used to define the basis functions.

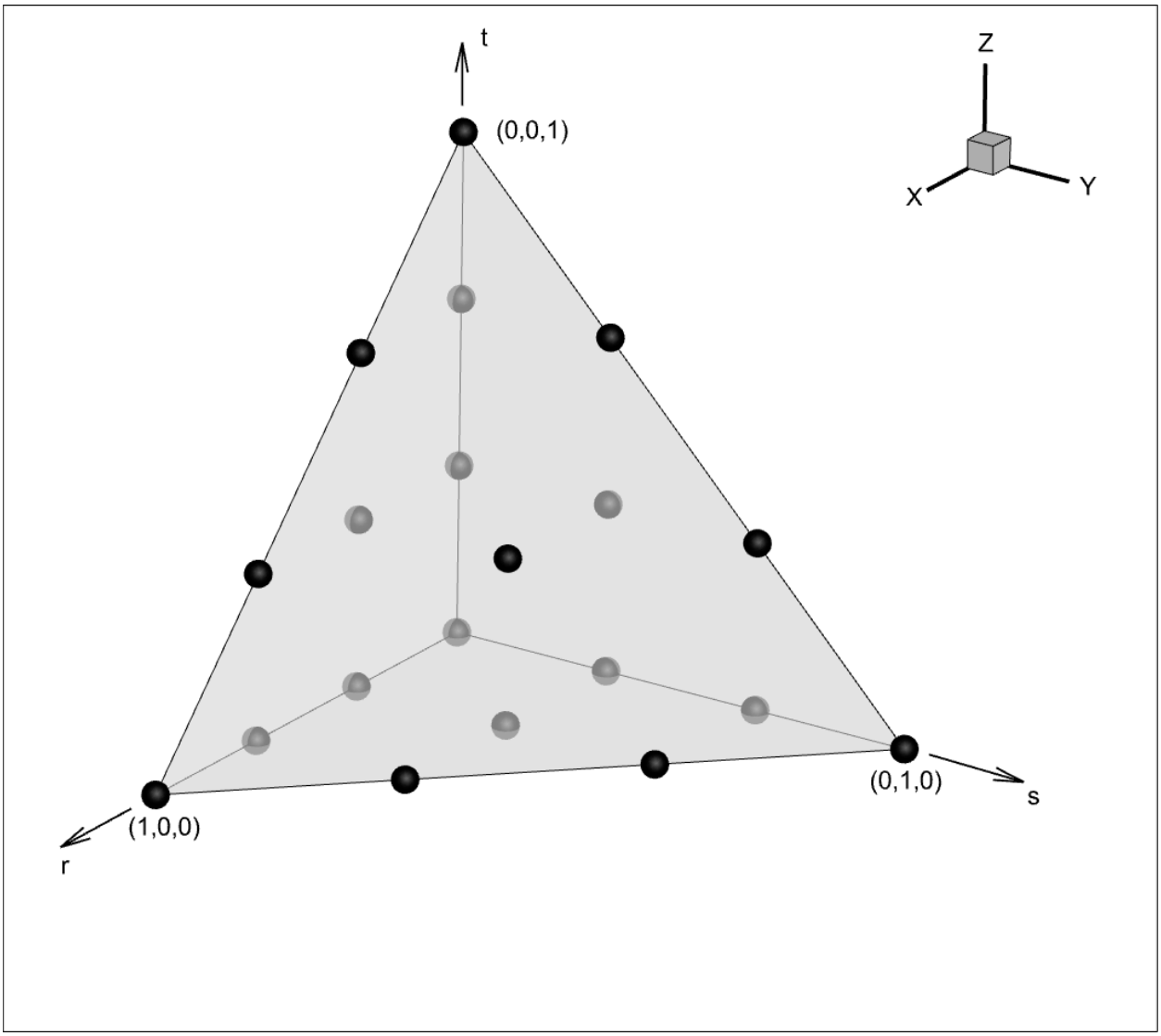

Figure 1. Cubic tetrahedra nodes and natural coordinates.

The tetrahedra basis functions are polynomials in the natural coordinates (ξ, η, ς) as shown in Figure 1. The cubic (p=3) basis functions $N_t^3(\xi, \eta, \varsigma)$, are defined in terms of the linear (p=1) basis functions, $N_t^1(\xi, \eta, \varsigma)$, which will be described first. The linear basis functions for each node vary linearly from one at that node to zero at the other two nodes. So, for node 1, N_{t1}^1 is one at node 1 and zero at nodes 2, 3, 4. Mathematically:

$$N_{t1}^1 = 1 - \xi - \eta - \varsigma$$
$$N_{t2}^1 = \xi$$
$$N_{t3}^1 = \eta$$
$$N_{t4}^1 = \varsigma$$

The same rules apply for the cubic basis function: the basis function for a node is one at the node and zero at all other nodes. This is satisfied by the equations:

$$N_{t1}^3 = {}^9/_2\, N_{t1}^1(N_{t1}^1 - {}^1/_3)(N_{t1}^1 - {}^2/_3)$$
$$N_{t2}^3 = {}^9/_2\, N_{t2}^1(N_{t2}^1 - {}^1/_3)(N_{t2}^1 - {}^2/_3)$$
$$N_{t3}^3 = {}^9/_2\, N_{t3}^1(N_{t3}^1 - {}^1/_3)(N_{t3}^1 - {}^2/_3)$$
$$N_{t4}^3 = {}^9/_2\, N_{t4}^1(N_{t4}^1 - {}^1/_3)(N_{t4}^1 - {}^2/_3)$$
$$N_{t5}^3 = {}^{27}/_2\, N_{t1}^1 N_{t2}^1(N_{t1}^1 - {}^1/_3)$$
$$N_{t6}^3 = {}^{27}/_2\, N_{t1}^1 N_{t2}^1(N_{t2}^1 - {}^1/_3)$$
$$N_{t7}^3 = {}^{27}/_2\, N_{t2}^1 N_{t3}^1(N_{t2}^1 - {}^1/_3)$$
$$N_{t8}^3 = {}^{27}/_2\, N_{t2}^1 N_{t3}^1(N_{t3}^1 - {}^1/_3)$$
$$N_{t9}^3 = {}^{27}/_2\, N_{t3}^1 N_{t1}^1(N_{t3}^1 - {}^1/_3)$$
$$N_{t10}^3 = {}^{27}/_2\, N_{t3}^1 N_{t1}^1(N_{t1}^1 - {}^1/_3)$$
$$N_{t11}^3 = {}^{27}/_2\, N_{t4}^1 N_{t1}^1(N_{t1}^1 - {}^1/_3)$$
$$N_{t12}^3 = {}^{27}/_2\, N_{t4}^1 N_{t1}^1(N_{t4}^1 - {}^1/_3)$$
$$N_{t13}^3 = {}^{27}/_2\, N_{t4}^1 N_{t2}^1(N_{t2}^1 - {}^1/_3)$$
$$N_{t14}^3 = {}^{27}/_2\, N_{t4}^1 N_{t2}^1(N_{t4}^1 - {}^1/_3)$$
$$N_{t15}^3 = {}^{27}/_2\, N_{t4}^1 N_{t3}^1(N_{t3}^1 - {}^1/_3)$$
$$N_{t16}^3 = {}^{27}/_2\, N_{t4}^1 N_{t3}^1(N_{t4}^1 - {}^1/_3)$$
$$N_{t17}^3 = 27 N_{t1}^1 N_{t2}^1 N_{t3}^1$$
$$N_{t18}^3 = 27 N_{t1}^1 N_{t2}^1 N_{t4}^1$$
$$N_{t19}^3 = 27 N_{t2}^1 N_{t3}^1 N_{t4}^1$$
$$N_{t20}^3 = 27 N_{t1}^1 N_{t3}^1 N_{t4}^1$$

Cubic Pyramid Basis Functions

The basis functions for the pyramid are more complicated than for the other element types. This is because the variation of the solution along the triangular faces must match the distribution along the faces of adjacent tetrahedra and, at the same time, the variation of the solution along the quadrilateral face must match the distribution along the face of an adjacent hexahedron. This can't be done with simple orthogonal polynomial basis functions, so we use rational polynomials for the pyramid basis functions. We use the basis functions of Chan and Warburton[4].

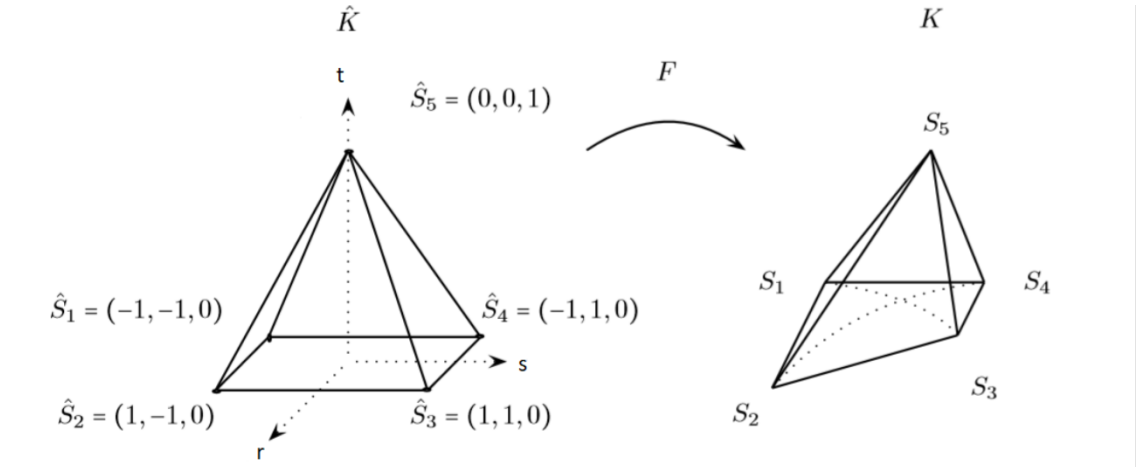

Figure 2. Pyramid Local Coordinates.

The local coordinates for the pyramid are shown in figure 2. Given this, the cubic (N=3) basis functions are

$$P_{ijk}(r,s,t) = P_i^{0,0}\left(\frac{r}{1-t}\right) P_j^{0,0}\left(\frac{s}{1-t}\right)(1-t)^c P_k^{2(c+1),0}(2t-1),$$

where

$$c = \max(i,j), \quad 0 \le i,j \le N, \quad 0 \le k \le N-c.$$

and $P_n^{a,b}(x)$ is the Jacobi polynomial of order n, orthogonal with respect to the weight $(1-x)^a(1+x)^b$. For a given N, the above procedure produces N_p distinct basis functions, where

$$N_p = \frac{(N+1)(N+2)(2N+3)}{6}$$

We may order these orthogonal functions arbitrarily as $\phi_j(r,s,t)$ from $j = 1,\ldots,N_p$.

For cubic (N=3) this results in thirty functions. The above are modal basis functions, meaning that the coefficients of the polynomial basis functions are not the solution at the nodes. We wish to have a nodal basis function.

Using the above basis functions, we may define the generalized Vandermonde matrix

$$V_{ij} = \phi_j(r_i,s_i,t_i), \quad i = 1,2,\ldots,N_p$$

where (r_i,s_i,t_i) are a set of nodal points contained inside or on the boundary of the reference pyramid. Then, a nodal basis may be constructed using elements of the inverse of the Vandermonde matrix

$$\ell_i(r,s,t) = \sum_{j=1}^{N_p}(V^{-1})_{ij}\phi_j(r,s,t).$$

C. Subdivision into Linear Sub-Elements

The subdivision for each higher-order pyramid or prism element is as follows:

- Break it into the logical set of linear sub-tetrahedra.
- If the error too large, create new edge nodes on each of the sub-tetrahedra and subdivide into 8 sub-sub-tetrahedra as shown in figure 3. Interpolate the solution to these new nodes using the original hexahedron, tetrahedron, prism or pyramid basis functions.
  - Repeat until the desired accuracy is obtained.

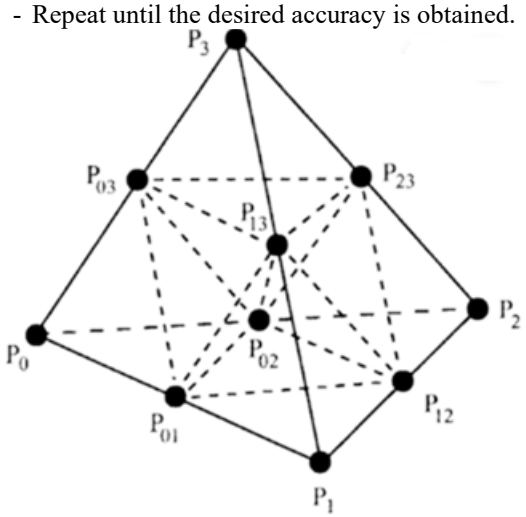

Figure 3. Subdivision of Quadratic Tetrahedron

While the process is the same for all element shapes, the initial subdivision is different.

Cubic Tetrahedron Subdivision:

The subdivision of the quadratic tetrahedron was shown in figure 3.

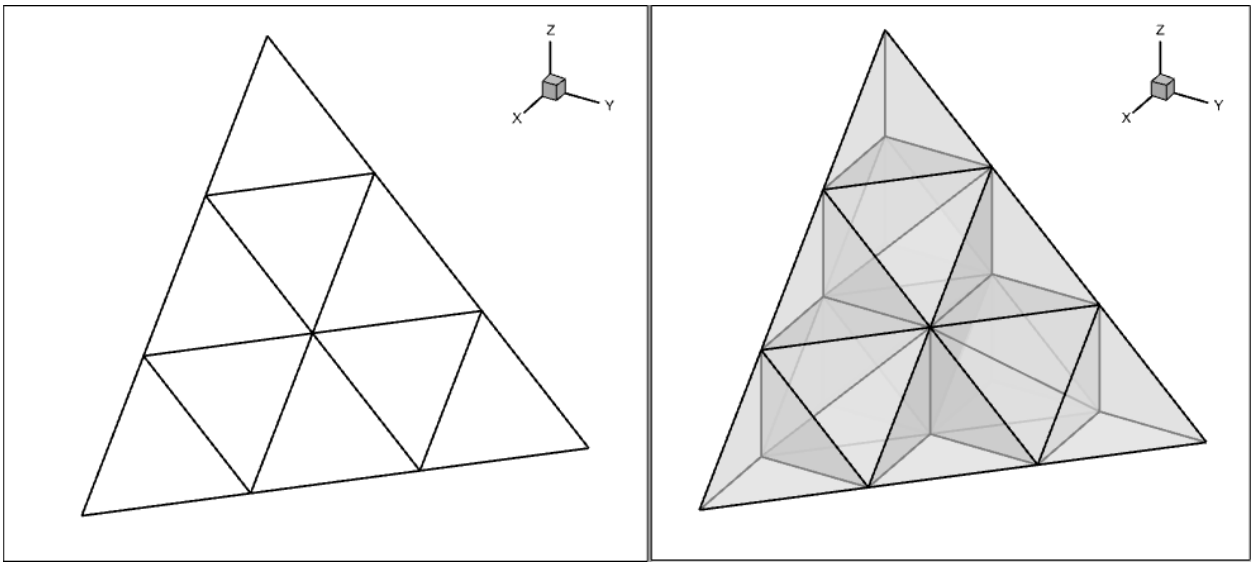

Figure 4. Cubic Tetrahedron Natural Subdivision into Linear Tetrahedra

The cubic Lagrangian tetrahedron has 20 nodes: 4 at the corners, two on each of the six edges, and one on each face as shown in figure 4. The subdivision is chosen so that subcell edges will always align between adjacent elements. For further subdivision, nodes are added to the mid-point of each sub-element edge and they are subdivided like quadratic tetrahedra.

Cubic Pyramid Subdivision:

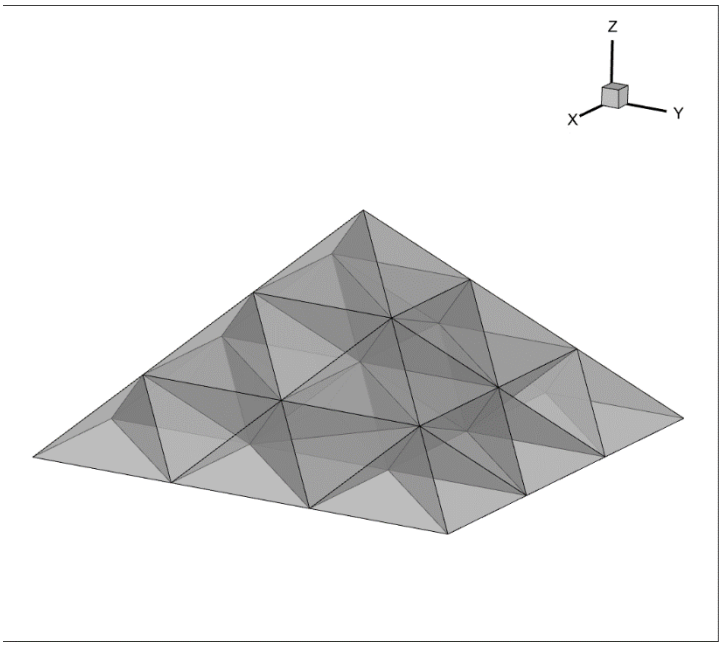

Figure 5. Subdivision of cubic Pyramid into Linear Tetrahedra

As shown in figure 5, the natural subdivision of the cubic pyramid results in 54 sub-tetrahedra. Unfortunately, unlike the cubic tetrahedron, the cubic pyramid has a quadrilateral face that where the triangulation may not match the triangulation in the adjacent element (the edges may cross). This can result in small gaps in isosurfaces at that face.

III. Streamline Algorithm

The streamline algorithm uses a recursive subdivision technique, similar to that used for the isosurface, to find the subcell containing the current location along the streamline and compute the velocities at that location using linear interpolation within the subcell. The streamline is then extended a short distance (step size) in the direction of the velocity and the process repeats. The step size is a fraction of the cell size and the algorithm adjusts the step-size to, in general, take multiple steps across each cell. The streamline computation takes advantage of the same optimizations used by the isosurface algorithm.

As the polynomial order of the cell basis function increases, the default step size must decrease to maintain the same level of accuracy. This process has not currently been automated, but the step size can be adjusted in the user interface.

IV. Optimization

The recursive subdivision is implemented in a way that minimized the amount of computation required.

First, at each level of subdivision, those cells that clearly cannot contain the isosurface (or the x,y,z point, in the case of probing or a step of a streamline) are eliminated before the next level of subdivision is done. This reduces the amount of computation and memory usage.

Second, much of the computation of higher-order interpolation weights is done once upfront. This is possible because the nodes added in the subdivision process are always at the same local coordinate (r,s,t) locations for each cell of a specific shape and polynomial order. The savings from doing this are substantial. For example, for the quadratic pyramid, the $l_i(r_i, s_i, t_i)$ weights ware computed just once for each level of subdivision. Then computing the value of a variable at any node at any subdivision level requires just 14 multiplications and 13 additions for 27 floating point operations (flops). For comparison, the computation of the weights requires 588 Jacobi polynomial evaluations (roughly 20 flops each), 3920 other flops, and 2200 flops for the matrix inversion, for a total of roughly 17,900 flops. If the weights were recomputed every time an interpolation was required, it would be 660 times more computationally expensive. The tradeoff is that the weights must be stored in memory but, since they are only stored once for each combination of cell shape, polynomial and subdivision level, the additional memory requirement is fairly small.

It should be noted that the pyramid has the most complicated basis function and therefore shows the most benefit from this optimization. However, for all 3D basis functions, the computational cost is reduced by at least and order of magnitude using this optimization. Details operation counts will be given for all basis functions in the final paper.

V. Results

The new isosurface algorithm has been applied to flow around the high-lift configuration of the Common Research Model (CRM). The data was computed by a high-order FR/CRM code[7,8] using a mixture of quadratic prisms, pyramids, and tetrahedra. The grid contained 84,549,975 nodes and 29,507,246 volume elements.

Figure 6 shows the effect of subdivision levels on the pressure isosurfaces in the region of the nacelle. The left image is one level of subdivision (the natural subdivision using existing higher-order nodes) and the right is two levels of subdivision. Note that, for these qualitative plots, it is difficult to see the difference between one and two levels of subdivision. It is visible if you look carefully, but not obvious.

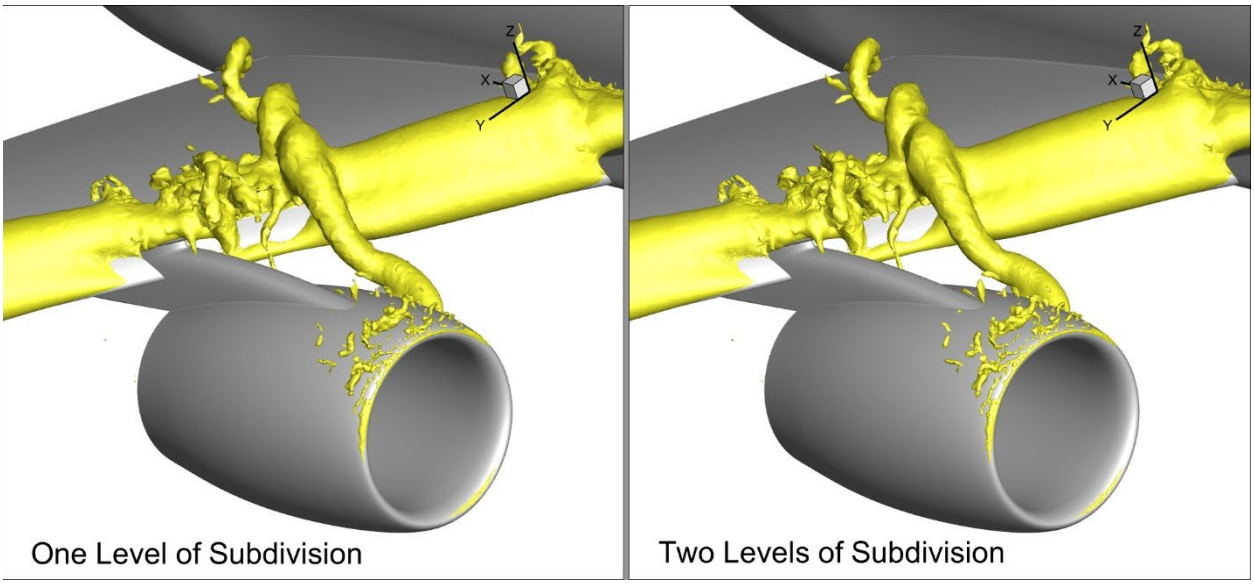

Figure 6. Pressure isosurface near nacelle of high-lift CRM configuration.

Figure 7 shows the effect of subdivision levels on boundary-layer velocity profiles for the same dataset. Note the velocity profiles are more sensitive to the number of subdivisions and at least three levels of subdivision are required at this location to accurately represent the high-order profile.

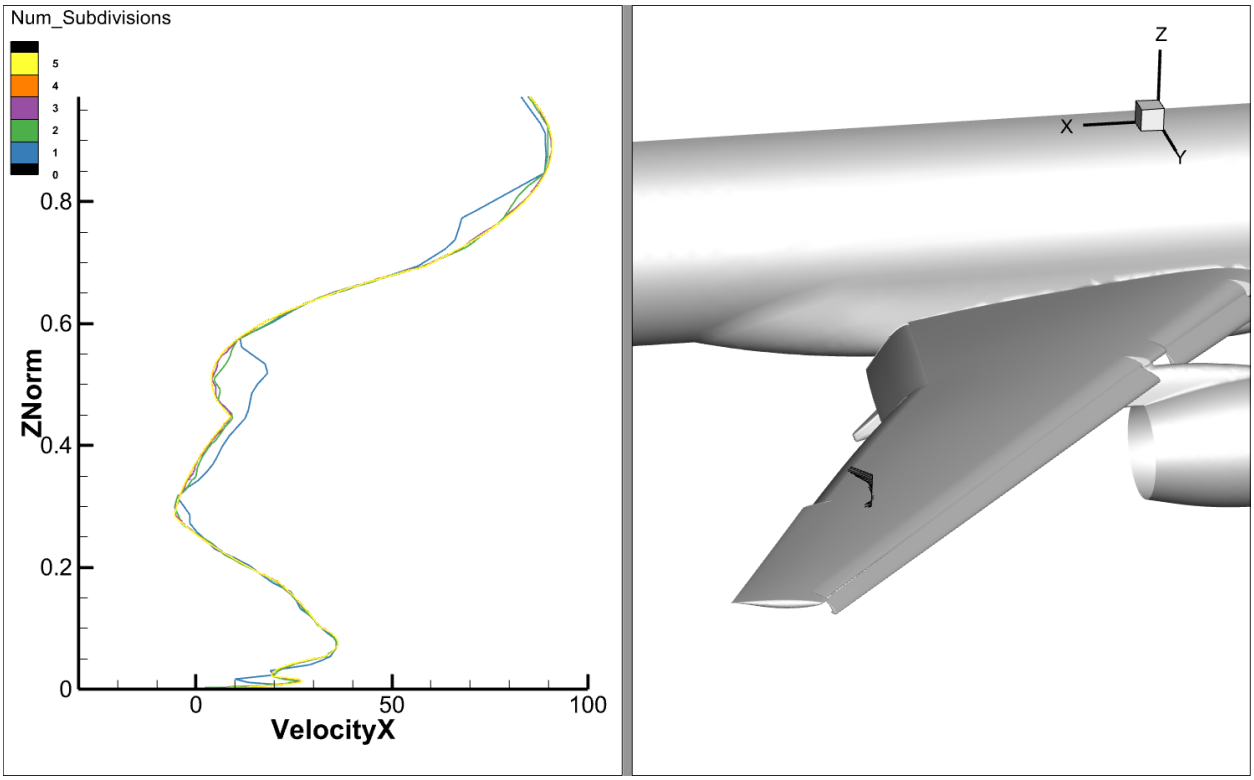

Figure7. Separated boundary layer velocity profile for CRM high-lift configuration.

The final CRM image, figure 8, shows the computed streamlines in the vicinity of the wing-tip vortex.

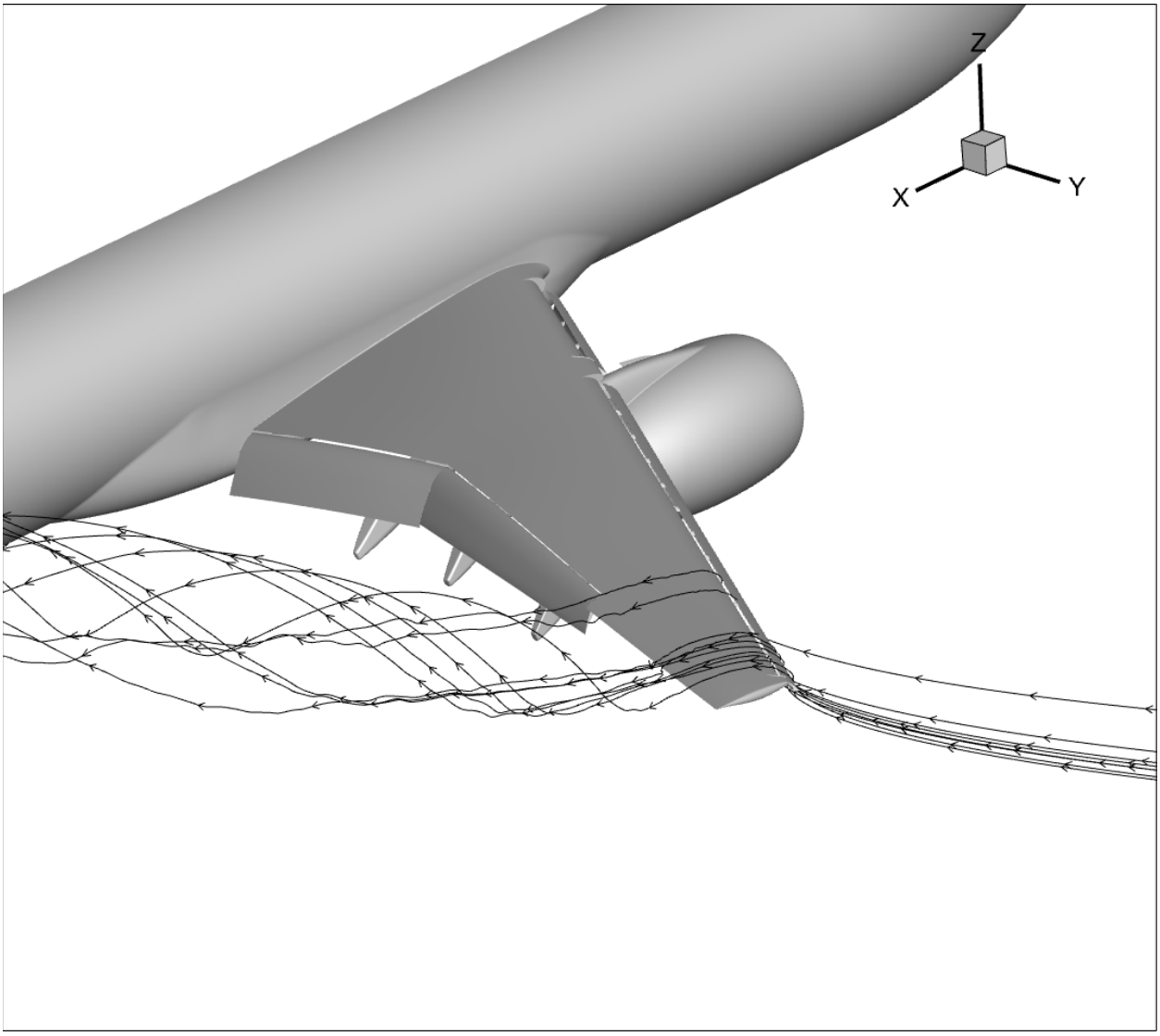

Figure 8. Streamlines near the wingtip vortex for CRM high-lift configuration.

Performance Comparison:

The time for the new algorithm to generate the pressure isosurfaces in figure 6 (without the airplane geometry) for two grid densities are shown in Table 1. Also shown are the time to generate the same isosurface in Paraview 5.11.0 on the same computer[9]. The test computer is a Dell Precision T7610 with 24 cores (12 physical + 12 hyperthread), 128GB RAM, an NVIDIA Quadro K4000, and Windows 10.

Num Nodes	Num Elements	Tecplot (CGNS)	Tecplot (SZL)	Paraview 5.11.0
2,454,672	627,413	5.3	2.5	232
84,549,975	29,507,246	117	32	More than 8 hrs

Table 1. Time, in seconds, to generate the isosurfaces in Figure 6.

Note, only the default (non-mpi) version of Paraview was used. There are, no doubt, versions of Paraview (perhaps the mpi version) that would generate the isosurfaces more quickly.

VI. Conclusions

A recently developed recursive subdivision algorithm[5,6] to compute isosurfaces for higher-order element solutions has been implemented in the engine of the commercial visualization code Tecplot 360. The algorithm minimizes memory usage by keeping only sub-elements that contain the isosurface. In the process, the algorithm has been optimized by precomputing the weights of the basis functions at added nodes of all element shapes, polynomial orders, and subdivision levels. The algorithm also uses multi-threading parallelism to efficiently use all CPU cores on a shared-memory workstation. The algorithm has been extended to work with selected cubic elements (tetrahedra and pyramids) and to accurately and efficiently compute streamlines through the higher-order elements.

The benefits of the algorithm have been demonstrated on quadratic solutions of the flow around the common research model high-lift configuration. Finally, the performance of the algorithm as implemented in Tecplot 360 has been shown to be a factor of 40 faster than the default download of a well-know open-source visualization code for an isosurface in the flow around an airplane configuration.

References

[1]Slotnick, J., Khodadoust, A., Alonso, J., Darmofal, D., Gropp, W., Lurie, E. and Mavriplis, D., "CFD Vision 2030 Study: A Path to Revolutionary Computational Data Science," *NASA/CR-2014-218178*, 2014.

[2]Thompson, D.C., and Pebay, P.P., "Visualizing Higher Order Finite Elements: Final Report," SAND2005-6999, Nov. 2005.

[3]Remacle, J.-F., Chevaugeon, N., Marchandise, E. and Geuzaine, C., "Efficient visualization of high-order finite elements," Int. J. Numer. Meth. Engng, Jul. 2006.

[4]Chan, J. and Warburton, T., "A Comparison of High-Order Interpolation Nodes for the Pyramid," SIAM J. Sci. Computing, Dec. 2014.

[5]Imlay, S., Taflin, D., and Mackey, C., "Recursive Sub-Division Technique for Higher-Order-Element Isosurface Visualization," AIAA 2020-3223, AIAA AVIATION Forum, Jun. 2020.

[6]Imlay, S., Taflin, D., and Mackey, C., "Recursive Sub-Division Technique for Higher-Order Pyramid and Prism Isosurface Visualization," AIAA 2021-1363, AIAA SciTech Forum, Jan. 2021.

[7]Wang, Z.J., Li, Y., Laskowski, G.M., Kopriva, J., Paliath, U., and Bhaskaran, R., "Toward Industrial Large Eddy Simulation Using the FR/CPR Method," Computers and Fluids, Vol. 156, 12, Oct. 2017, pp 579-589.

[8]Wang, Z.J., "Wall-Modeled Large Eddy Simulation of the NASA CRM High-Lift Configuration with the High-Order FR/CPR Method," AIAA Aviation, June 2022.

[9]http://www.kitware.com/products/paraview.html