INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng (in press)
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nme.1787

Efficient visualization of high-order finite elements

Jean-François Remacle[1,*,†], Nicolas Chevaugeon[1], Émilie Marchandise[1,3] and Christophe Geuzaine[2,3]

[1]*Département de Génie Civil et Environnemental, Université Catholique de Louvain, Bâtiment Vinci, Place du Levant 1, B-1348 Louvain-la-Neuve, Belgium*
[2]*Department of Applied Mathematics, California Institute of Technology, Pasadena, CA, U.S.A.*
[3]*Fonds National de la Recherche Scientifique, rue d'Egmond, Bruxelles*

SUMMARY

A general method for the post-processing treatment of high-order finite element fields is presented. The method applies to general polynomial fields, including discontinuous finite element fields. The technique uses error estimation and *h*-refinement to provide an optimal visualization grid. Some filtering is added to the algorithm in order to focus the refinement on a visualization plane or on the computation of one single iso-zero surface. 2D and 3D examples are provided that illustrate the power of the technique. In addition, schemes and algorithms that are discussed in the paper are readily available as part of an open source project that is developed by the authors, namely Gmsh. Copyright © 2006 John Wiley & Sons, Ltd.

Received 8 July 2005; Revised 16 February 2006; Accepted 25 April 2006

KEY WORDS: high-order finite elements; discontinuous Galerkin; scientific visualization

1. INTRODUCTION

In the recent years, a large research effort has been devoted to the development of high-order finite element discretization techniques: spectral element methods [1], high-order finite elements [2, 3] or high-order discontinuous Galerkin methods (DGM) [4–6]. Our focus is high-order DGM. Those are now used extensively for solving transient aero-acoustics problems [7, 8], electromagnetic problems [9], dynamics of compressible fluids [4, 10], Korteweg-de Vries (KdV) equations [11]

*Correspondence to: Jean-François Remacle, Département de Génie Civil et Environnemental, Université Catholique de Louvain, Bâtiment Vinci, Place du Levant 1, B-1348 Louvain-la-Neuve, Belgium.
†E-mail: remacle@gce.ucl.ac.be

Contract/grant sponsor: European Community through Messiaen
Contract/grant sponsor: Belgian National Fund for Scientific Research (FNRS)

and many other relevant physical problems. Using a quadrature free approach together with a very careful BLAS3 implementation [12], our experience shows that, with fifth or sixth-order polynomials on a tetrahedral mesh, we can run a DGM code for which we use about 80 per cent of the peek performance of an off-the-shelf desktop computer. This makes high-order DGM a very competitive approach.

In contrast, most of the finite element post-processing tools are unable to provide accurate high-order visualizations. Typically, quadratic elements are the highest available order for visualization. In this paper, we propose a robust and efficient methodology for the visualization of high-order finite element solutions. This work includes the issues of contouring, iso-surfacing and cutting. The methodology is inspired by h-adaptive finite element techniques [4, 10]. Note that our aim is not to compete with classical iso-surfacing techniques like marching cubes [13]. The marching cubes algorithm takes as input a data set (voxels or cells) and renders iso-surfaces efficiently. The method that we propose here is aimed at producing an optimized visualization data set from a high-order finite element solution. Then, marching cubes or any other iso-surfacing algorithm can be used against this optimized data set.

In all what follows, computer efficiency is always a central concern. When plots and images of 2D and 3D results will be shown, we will always indicate the computation time and the number of rendered polygons.

Another concern that we have is compatibility. We think that it is the post-processor that has to adapt to the solver and not the inverse. In our developments, the solver thus dictates how fields are interpolated and how elements are mapped. This means that we should be able to visualize any kind of finite element solutions, continuous or not, using any sort of shape functions.

Finally, providing some source code is essential in this kind of work. An implementation of our methodology is readily available as an open source code. Gmsh [14] is an automatic 3D finite element grid generator (primarily Delaunay) with a build-in CAD engine and post-processor. Its design goal is to provide a simple meshing tool for academic problems with parametric input and up to date visualization capabilities. Gmsh is copyright ©1997–2005 by C. Geuzaine and J.-F. Remacle and is distributed under the terms of the GNU General Public License (GPL). All the illustrations shown below have been done with version Gmsh 1.61.

2. HIGH-ORDER DISCONTINUOUS FINITE ELEMENTS

Finite element methods (FEMs) involve a double discretization. First, the physical domain Ω is discretized into a collection of $\mathcal{N}$ elements

$$\mathcal{T} = \bigcup_{e=1}^{\mathcal{N}} e \tag{1}$$

called a mesh. This first step is the one of *geometrical discretization*. Then, the continuous function spaces (infinite dimensional) are replaced by finite dimensional expansions. The difference between the DGM and classical FEMs is that the solution is approximated in each element separately: no *a priori* continuity requirements are needed. The discrete solution may then be discontinuous at inter-element boundaries. Figure 1 shows a typical situation of three elements e_1, e_2 and e_3. The approximated field u is smooth in each element but may be discontinuous at inter-element boundaries.

DOI: 10.1002/nme

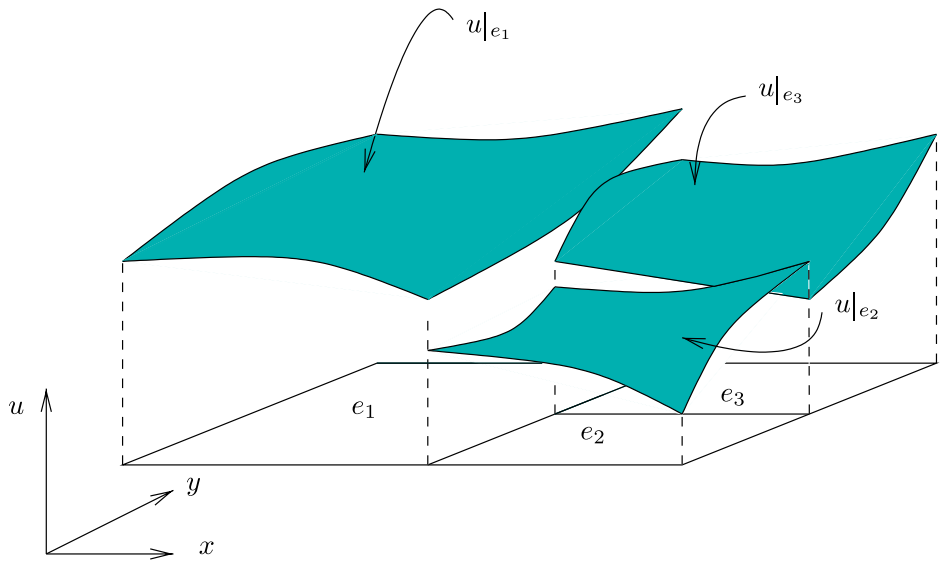

Figure 1. Three elements e_1, e_2 and e_3 and the piecewise discontinuous solution u.

Consider a scalar field $u(x, y, z)$. In each element, polynomial spaces are usually used for approximating u. The approximation of u over element e, noted u^e, is written as

$$u^e = \sum_{j=1}^{d} \phi_j(\xi, \eta, \zeta) U(e, j)$$

where the $U(e, j)$ are the coefficients of the approximation or degrees of freedom, where d is the size of the discrete function space and where $\phi_j(\xi, \eta, \zeta)$ is the jth approximation function defined in a parametric space ξ, η, ζ. In each element, we have d coefficients, and, because we consider (this is the most general case) that all approximations are disconnected, $\mathcal{N} \times d$ coefficients are required for describing u. In our implementation, we organize this data in a $\mathcal{N} \times d$ matrix that we call U. U_{ij} is therefore the jth coefficient of the approximation of u^i of u in element i. We use the GNU scientific library (GSL [15]) for manipulating matrices and vectors. The GSL has the advantage to provide direct and easy bindings to the basic linear algebra subroutines (BLAS).

The element e may be geometrically high order. We define the geometrical mapping as the mapping that transforms a reference element into the real element:

$$x^e = x^e(\xi, \eta, \zeta), \quad y^e = y^e(\xi, \eta, \zeta), \quad z^e = z^e(\xi, \eta, \zeta)$$

with

$$x^e = \sum_{j=1}^{m} \psi_j(\xi, \eta, \zeta) X(e, j), \quad y^e = \sum_{j=1}^{m} \psi_j(\xi, \eta, \zeta) Y(e, j), \quad z^e = \sum_{j=1}^{m} \psi_j(\xi, \eta, \zeta) Z(e, j)$$

where m is the number of geometrical shape functions. Typically, for geometrically first-order elements, m is the number of vertices. Matrices X, Y and Z are of size $m \times 3$.

DOI: 10.1002/nme

Our goal is to be able to treat (i.e. visualize) any piecewise polynomial approximation. For that purpose, our visualization methodology has to

- enable the representation of discontinuous fields,
- distinguish geometric and functional discretizations,
- be able to use any set of polynomial interpolation functions.

The fact that we have disconnected interpolations by defining as much coefficients as $\mathcal{N} \times d$ allows naturally to represent discontinuous fields. The fact that we consider the general case of non-isoparametric elements, which means that $d \neq m$ and $\phi_i \neq \psi_i$, enables us to distinguish geometric and functional discretizations.

Here, we only consider polynomial approximations. Usual FEM have limited choices for the ϕ_i's due to the *a priori* continuity requirements of the approximation. Nodal, hierarchical, Serendip or non-conforming basis are among the usual basis for classical FEMs. In the general case of a DGM, there are no limitations for the choice of the ϕ_i's. For giving a general form to our ϕ_i's, we define the two following matrices. The first matrix, named P, is of size $d \times 3$. Matrix P is used for the definition of a canonical polynomial basis:

$$\mathscr{P} = \{p_1(\xi, \eta, \zeta), \ldots, p_d(\xi, \eta, \zeta)\}$$

where

$$p_i = \xi^{P_{i1}} \eta^{P_{i2}} \zeta^{P_{i3}}$$

This basis is then used for building the ϕ_i's. For that, we define the coefficient matrix Φ of size $d \times d$ and we write

$$\phi_i = \sum_{k=1}^{d} p_k \Phi_{ik}$$

For example, for the bilinear isoparametric quadrilateral element, we have

$$P = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}, \quad \Phi = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}, \quad \mathscr{P} = \{1, \xi, \eta, \xi\eta\}$$

and

$$\phi = \tfrac{1}{4}\{(1+\xi)(1+\eta), (1+\xi)(1-\eta), (1-\xi)(1-\eta), (1-\xi)(1+\eta)\}$$

This way of building the basis allows to define any polynomial basis, complete or not, of any dimension. The advantage is that any FEM code can use the tool, for instance Gmsh, without having to change its own representation to the one of the post-processor.

The same technique is used for describing geometrical mappings in the general form. A high-order finite element visualization structure is described in Algorithm 1. The DGM allows the use of

DOI: 10.1002/nme

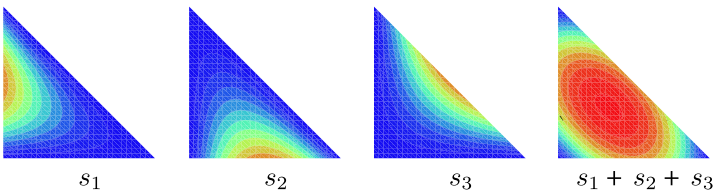

Figure 2. Illustration of the visualization technique developed in Reference [19] based on high-order textures.

exotic polynomial approximations and are, therefore, a good support for testing our visualization methodology.

Algorithm 1
A general structure for the definition of high-order finite element solution.

```
struct _high_order_fem {
  // coefficients for the element mappings
  GSL_Matrix X,Y,Z;
  // coefficients for the function
  GSL_Matrix U;
  // Canonical matrices for U and for the mapping
  GSL_Matrix PU,PM;
  // Interpolation matrices
  GSL_Matrix PHIU,PHIM;
};
```

3. CONTOURING HIGH-ORDER FIELDS

Efficient techniques for volume rendering have been known in the computer graphics community for a number of years. These techniques fall into two broad categories: direct volume rendering (DVR) algorithms and surface fitting (SF) algorithms [16].

Among SF algorithms, marching cubes [13, 17] is certainly the most popular, and it provides a simple method for extracting iso-surfaces from discrete three-dimensional data. However, it does not guarantee the surface to be topologically consistent with the data, and it creates triangulations which contain many triangles of poor aspect ratio. Marching tetrahedra is a variation of marching cubes, which overcomes this topological problem [18]. The originality of the method proposed in this paper is that it produces an *optimized* collection of data cells from high-order finite element solutions. Marching cubes, marching tetrahedra or any other SF algorithm can then be used to extract iso-surfaces from this optimized set of cells.

Another interesting approach to high-order contouring can be found in Reference [19]. This DVR approach is based on the definition of an algebra of high-order textures. Each basis function ϕ_i corresponds to one basis texture and the final visualization object is constructed by linear combination of basis textures. Figure 2 shows an example of this approach. Quadratic textures

DOI: 10.1002/nme

s_1, s_2 and s_3 corresponding to hierarchical shape functions associated to the edges of a triangle are shown. They are subsequently summed and the result $s = s_1 + s_2 + s_3$ is computed as the graphic sum of the basis textures. Disappointingly, this technique relies heavily on the graphic hardware and very few off-the-shelf graphic cards allow to perform those kind of combinations efficiently. Also, common graphical APIs like OpenGL support blending equations for textures but this process is essentially bi-dimensional and our final aim is to do efficient 3D visualizations.

More important, this approach has aspects that are way too restrictive for finite element visualizations. The number of iso-contours, for example, cannot be changed easily because basis textures contain a fixed number of colours. Changing to a different scale, a logarithmic scale for example, is difficult because the logarithm of the sum is not the sum of the logarithms. Also, extracting iso-surfaces or plane cuts out of 3D data is not straightforward with this approach and generating resolution-independant vector output is impossible.

Here, we focus on what we believe to be a more convenient approach. Contouring high-order FEM fields is difficult because of the topological complexity of high-order curves and surfaces. If we consider seventh or ninth-order approximations u^e on a triangle, the shape of one iso-contour of $u^e = C$ may be very complex (it could even be not connected). Only piecewise linear ($p = 1$) interpolations allow straightforward representation: one iso-contour is a straight line, one iso-surface is a polygon. We assume here that our visualization tool is able to deal with piecewise linear simplices: lines, triangles and tetrahedra. It is also able to draw bi- and tri-linear fields on quadrangles and hexahedra. In Gmsh, this is done by splitting quadrangles and hexahedra into simplices and doing linear visualization.

One robust approach of high-order visualization consists therefore in dividing the elements into sub-elements and doing a linear visual approximation on every sub-element. The main problem here is 'how far do we have to divide in order to capture the complexity of the high-order field?'. For addressing that issue, we use h-refinement. The problem is made simpler here by the fact that we know the exact visualization error, i.e. the difference between u^e and its 'drawable' piecewise linear representation.

4. VISUALIZATION MESH

The technique we use is based on classical automatic mesh refinement (AMR) methodologies, see for example References [4, 20]. For each element type (triangle, quadrangle, tetrahedron, hexahedron), we define a template for dividing the element itself recursively into sub-elements of the same type. As an example, the refinement template is shown for the triangle in Figure 3 at different recursion levels r. The element subdivision pattern is performed in the reference system of co-ordinates ξ, η, ζ. For a given maximal recursion level r, we obtain an array of N_r visualization points ξ_i, η_i and ζ_i with

$$N_r = \tfrac{1}{2}(2^r + 1)(2^r + 2)$$

for triangles,

$$N_r = (2^r + 1)^2$$

for quadrangles,

$$N_r = \tfrac{1}{6}(2^r + 1)(2^r + 2)(2^r + 3)$$

DOI: 10.1002/nme

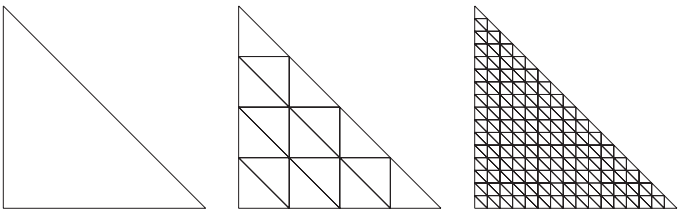

Figure 3. Some AMR refinement templates for triangles at different resolution levels.

for tetrahedra and

$$N_r = (2^r + 1)^3$$

for hexahedra. Using previous definitions, we construct the following interpolation matrix:

$$I_{ij} = \phi_j(\xi_i, \eta_i, \zeta_i)$$

of size $N_r \times d$. Matrix I is computed once and stored. The value of u^e at visualization points is computed efficiently using a simple matrix–vector product

$$u^e(\xi_i, \eta_i, \zeta_i) = I_{ij} U(e, j)$$

This product can be done efficiently using BLAS2 routines [12]. Typically, around one GigaFlop can be obtained for $p>4$ and $r>4$ on a 2.4 GHz Pentium IV Xeon. Our experience has shown that it was faster to compute a large amount of points using fast linear algebra techniques than only computing values when needed.

The number of triangles or quadrangles at recursion level r is 4^r and the number of tetrahedron or hexahedron is 8^r. All sub-elements should not always be visible: some kind of error analysis (or participation analysis) should be done in order to decide whether a given triangle is visible or not. This decision has to be goal oriented, i.e. it should depend on what has to be visualized:

- A 2D colourmap of the field in linear or logarithmic scale.
- 3D iso-surfaces.
- One given iso-contour or iso-surface.
- The area delimited by the positive region of a levelset surface.
- A zoom of a given area.

As an example, we represent one ninth-order Lagrange shape function in the reference triangle. In one case, we want to visualize the function using filled (coloured) iso-values. In the other case, we want to represent one given iso-value $u^e = C$. Figure 4 shows the two different adapted

DOI: 10.1002/nme

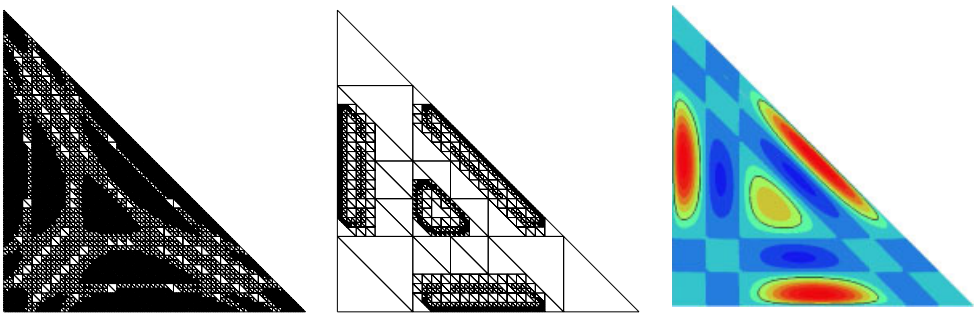

Figure 4. Visualization of one ninth-order Lagrange shape function. The left figure shows the visualization mesh for the goal $|u-u_h|<0.001$. The central figure shows the visualization mesh that has been optimized in order to capture accurately one given iso-contour $u=0.5$. The right figure shows the corresponding visualization results i.e. a colourmap and, in black, the iso-contour $u=0.5$.

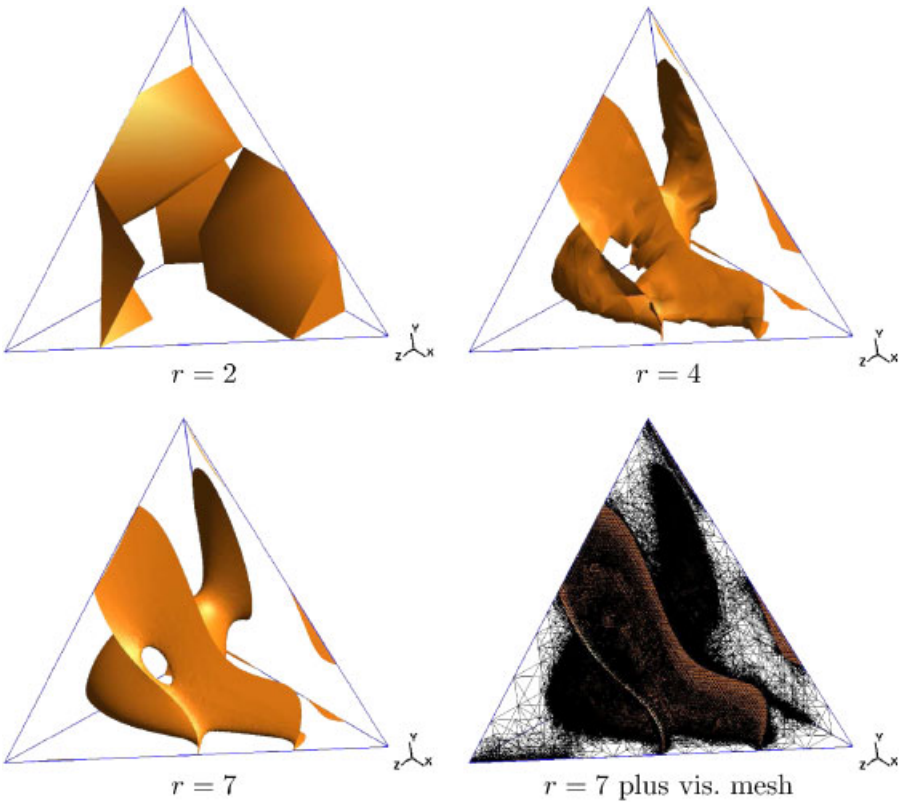

Figure 5. Visualization of one fourth-order function on one single tetrahedron. Figures show one iso-surface using different error thresholds. The last figure (bottom-right) show the visualization mesh together with the iso-surface.

DOI: 10.1002/nme

visualization grids relative to the two pre-defined visualization goals. We can do the same with 3D views on tetrahedral meshes and an example is shown in Figure 5.

5. VISUALIZATION ERROR

We aim to use classical h-refinement finite element techniques in order to optimize the visualization mesh. For that, we should be able to estimate the visualization error. In a classical *a posteriori* error analysis, the 'Quest of the Graal' is to find the local error. Indeed, if the error at a point is known, so is the exact solution. Usually, only estimations of L^2 or H^1 error norms are available, but in the particular case of our visualization problem, the *exact error* is known: the exact field, u^e, being the one defined by the high-order polynomial interpolation and the approximate one, u_h^e being the piecewise linear field defined on the visualization mesh. The exact local visualization error ε is therefore simply defined as $u^e - u_h^e$. In this, our aim is to build an adapted visualization mesh while only using values computed at the positions ξ_i, η_i, ζ_i defined by the interpolation matrix I_{ij}. Let us consider the triangular element depicted in Figure 6. We look if triangle e with nodal values u_1, u_2 and u_3 has to be subdivided. At mid-points, the exact solution u_{12}, u_{13} and u_{23} is available because it corresponds to the ξ_i, η_i, ζ_i of child subdivisions. At those midpoints, linear approximations would give average values $\frac{1}{2}(u_1 + u_2)$, $\frac{1}{2}(u_1 + u_3)$, and $\frac{1}{2}(u_2 + u_3)$. We define the following error indicator:

$$\begin{aligned}\varepsilon^e &= \max(|u_1 + u_2 - 2u_{12}|, |u_1 + u_3 - 2u_{13}|, |u_2 + u_3 - 2u_{23}|)\\ &= \max(\varepsilon_{12}^e, \varepsilon_{13}^e, \varepsilon_{23}^e)\end{aligned}$$

which is similar to what can be found in Reference [21]. Note that, for example,

$$\varepsilon_{12}^e = |u_1 + u_2 - 2u_{12}| = h_\eta^2 \left| \frac{\partial^2 u}{\partial \eta^2} \right| + \mathcal{O}(h_\eta^3)$$

so that the error indicator ε^e is a measure of the maximal second derivative of the exact field. We also see that, if we choose a maximal recursion level of r, then, element sizes are reduced by a factor 2^r and the visualization error is reduced by 2^{2r}. We proceed in the same way for other element types. Elements are subdivided recursively until $\varepsilon^e < \bar{\varepsilon}$ where $\bar{\varepsilon}$ is a threshold value.

The main advantage of this approach is its simplicity. Nodal values are computed once for all and the error indicator only involves a few arithmetic operations. However, the approach has a major drawback: if a feature has a characteristic size that is much smaller than the size of e, it can be missed by the procedure. This is especially true at low recursion levels. So, at low recursion levels, more points are considered to compute the error indicator. The right part of Figure 6 shows the 2 recursion levels configuration. The error in element e is still computed as the maximal numerical second derivatives at all interior points

$$\varepsilon^e = \max(\varepsilon_{12}^e, \varepsilon_{13}^e, \varepsilon_{23}^e, \varepsilon_{112}^e, \varepsilon_{122}^e, \varepsilon_{113}^e, \varepsilon_{133}^e, \varepsilon_{223}^e, \varepsilon_{233}^e, \varepsilon_{1123}^e, \varepsilon_{1223}^e, \varepsilon_{1233}^e)$$

DOI: 10.1002/nme

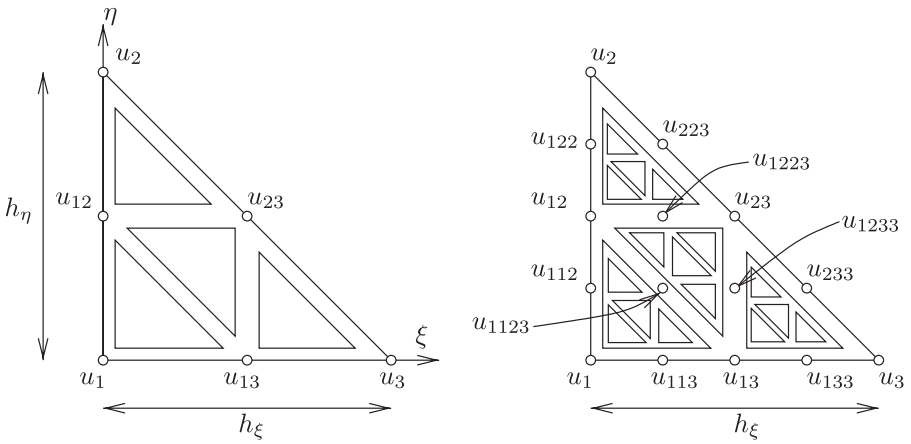

Figure 6. One triangular element and its subdivision.

with

$$\varepsilon^e_{ij} = |2u_{ij} - u_i - u_j|$$

$$\varepsilon^e_{ijj} = |2u_{ijj} - \tfrac{1}{2}u_i - \tfrac{3}{2}u_j|$$

and

$$\varepsilon^e_{ijjk} = |2u_{ijjk} - u_j - \tfrac{1}{2}(u_i + u_j)|$$

6. VISUALIZATION MESH ADAPTATION

We now describe how we choose adaptively the sub-elements that are visible. The sub-element data structure is described on Algorithm 2. We apply a recursive pattern for doing the mesh refinement and the `_subelement` data structure is build like a multi-dimensional tree. For example, one triangular `_subelement` has four children while a hexahedron `_subelement` has 8. There is only one instance of the `_subelement` data structure: nodal values are updated for each element using one BLAS2 matrix–vector multiplication. We call the `_root` the initial unrefined element who is, in fact, the ancestor of all sub-elements. The mesh adaptation is done in one recursion pass. On the way down of the recursion process, we compute the exact error and decide whether it is useful to go to a lower level. On the way up of the recursion, we may apply some filtering, i.e. eliminate useless parts of the view. The filter may for example only allow sub-elements whose nodal values are in a prescribed range (the case of the capture of one given iso-surface) or only allows to draw sub-elements that intersect a given plane (the case of a planar cut in a 3D view). Algorithm 3 presents C++ code that is used to determine which sub-elements of the `_root`

DOI: 10.1002/nme

element are visible. Note that, starting from a continuous finite element field, our non-conforming refinement procedure may produce discontinuous visualization data. It is possible to ensure that continuous approximations remain strictly continuous by ensuring a conforming refinement, i.e. by introducing more complex refinement templates than the ones proposed here.

Algorithm 2
Visualization element class.

```
struct _subelement {
  static  int nbChilderen;
  static  int nbNodes;
  bool   visible;
  int    *node_numbers;
  _element *child;
};
```

Algorithm 3
Recursive algorithm that determines the set of visible sub-elements. The algorithm is first called with the `_root` element as first argument. It goes down in the sub-element layers, stopping when the error is below a given threshold `eps_0`. If the algorithm goes down to the deepest layer of the `_subelement` tree, then the return value of the function `recur_visible` is `false`. More layers can be computed in order to ensure that the required accuracy is reached.

```
bool recur_visible ( _subelement *e , double eps_0 ){
   if(!e->child[0])
   {
    // Max recursion has been attained without having
    // reached desired accuracy
    e->visible = true;
    return false;
   }
   else
    // compute error using the 1- or the 2-level formula
    double eps = compute_error ( e );
    if (eps > eps_0){
     e->visible=false;
     accuracy_reached = true;
     for (i=0;i<e->nbChilderen;++i)
      accuracy_reached &= recur_visible (e->child[i],eps_0);
     return accuracy_reached;
    }
    else{
     e->visible=true;
     return true;
    }
  }
 };
```


DOI: 10.1002/nme

7. TWO-DIMENSIONAL EXAMPLES

7.1. Vortex-in-a-box

This example is usually chosen in order to test the ability of an advection scheme to accurately resolve thin filaments on the scale of the mesh which can occur in stretching and tearing flows. We consider the following stream function:

$$\psi = \frac{1}{\pi} \sin^2(\pi x) \sin^2(\pi y) \tag{2}$$

that defines the following velocity field:

$$\begin{aligned} v_x &= \partial_y \psi = \cos(2\pi y) \sin^2(\pi x) \\ v_y &= -\partial_x \psi = -\cos(2\pi x) \sin^2(\pi y) \end{aligned} \tag{3}$$

We consider a disk of radius 0.15 placed at (0.5, 0.75) and the distance function $u_0^2 = (x-0.5)^2 + (y-0.75)^2 - 0.15^2$. We aim to solve the following hyperbolic problem:

$$\partial_t u + v_x \partial_x u + v_y \partial_y u = 0$$

on a square domain of size [1, 1] and with $u = u_0$ as initial conditions. The flow satisfies $v_x = v_y = 0$ on the boundaries of the unit square. The resulting velocity field stretches out the circle into long and thin filaments. For solving this problem, we use a 32×32 quadrilateral grid with a fourth-order ($p = 4$) DGM for space discretization. A fifth-order explicit Runge–Kutta is used for time stepping. Some visualization results are shown at time step 1000.

Figures 7 and 8 show an adaptive visualization. Figures 7(a) and (b) show the unrefined mesh (1024 quadrangles) and the unrefined visualization. Figures 7(c) and (d) show results for a visualization error of $\bar{\varepsilon} = 10^{-2}$: the visualization mesh 7(c) is made of 4021 quadrangles and the maximal recursion level required for obtaining the target error everywhere is $r = 4$. The computation time for generating 7(d) was 0.02 s. Figures 7(e) and (f) show results for a visualization error of $\bar{\varepsilon} = 10^{-3}$: the visualization mesh 7(e) is made of 46 747 quadrangles and the maximal recursion level required for obtaining the target error everywhere is $r = 6$. The computation time for generating 7(f) was 0.24 s.

Figure 8 shows an adaptive visualization that targets to capture accurately only the iso-zero of function u. Figures 8(a) and (b) show the unrefined mesh (1024 quadrangles) and the unrefined visualization. Figures 8(c) and (d) show results for a visualization error of $\bar{\varepsilon} = 10^{-2}$: the visualization mesh 8(c) is made of 2032 quadrangles. The computation time for generating 8(d) was 0.03 s. Figures 8(e) and (f) show results for a visualization error of $\bar{\varepsilon} = 10^{-3}$: the visualization mesh 8(e) is made of 6454 quadrangles and the computation time for generating 8(f) was 0.08 s.

In the case $\bar{\varepsilon} = 10^{-3}$, the maximal recursion level required for obtaining the desired accuracy was $r = 6$. The equivalent uniformly refined mesh contains therefore 1024×4^6 quadrangles, i.e. a little more than four million quadrangles. Our procedure allows to reduce this by a factor 650 in the case of the iso-zero computation.

7.2. 2D aeroacoustics

We consider here the solution of the 2D linearized Euler equations that are modelling noise propagation. The geometry that we consider is a plane engine intake. A modal excitation at a

DOI: 10.1002/nme

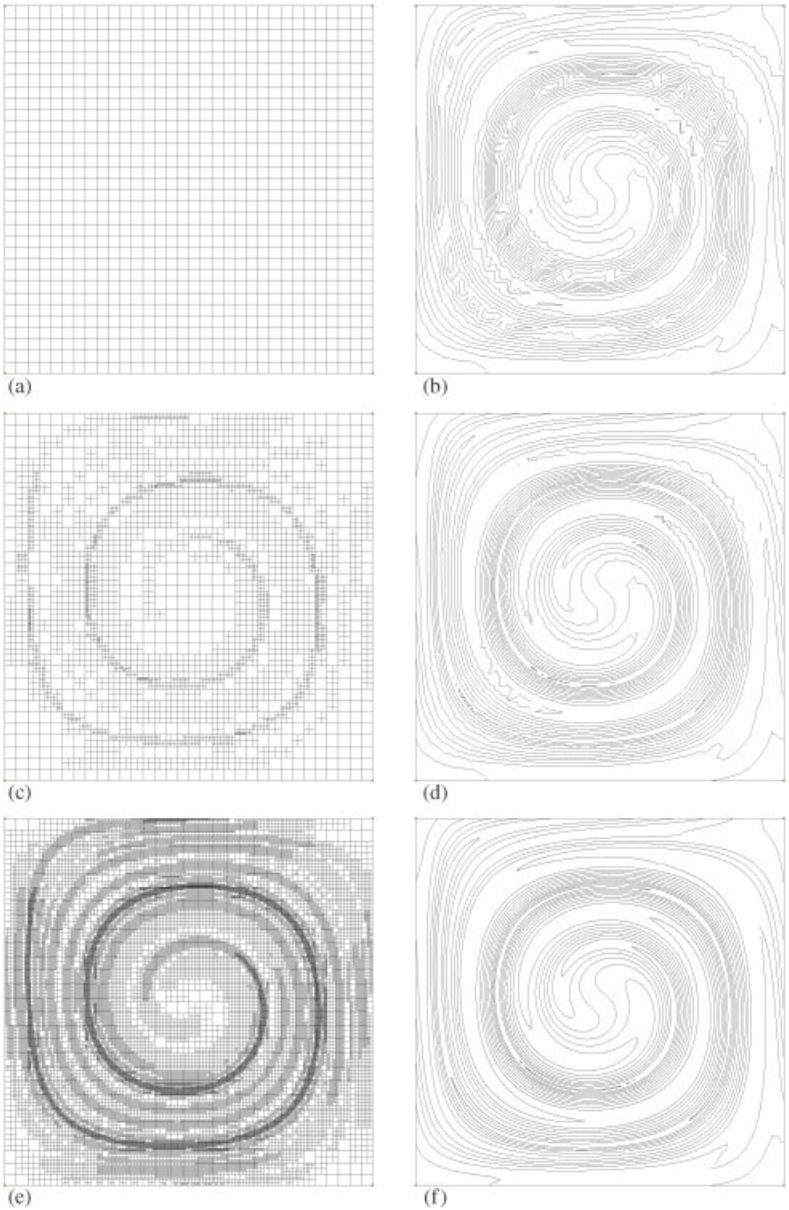

Figure 7. Visualization of function u at time step 1000 for the vortex-in-a-box problem.

frequency of 1500 Hz is applied on the plane representing the engine fan. We have solved the problem in the time domain using the DGM on a triangular grid (Figure 9) of 1635 triangles and using polynomial order $p = 8$ everywhere. As an illustration, we show pressure contours for different values of $\bar{\varepsilon}$ at a physical time of 1.10^{-2} s after the beginning of the simulation, when the wave front is about to reach the boundary of the domain 9. For a visualization error

DOI: 10.1002/nme

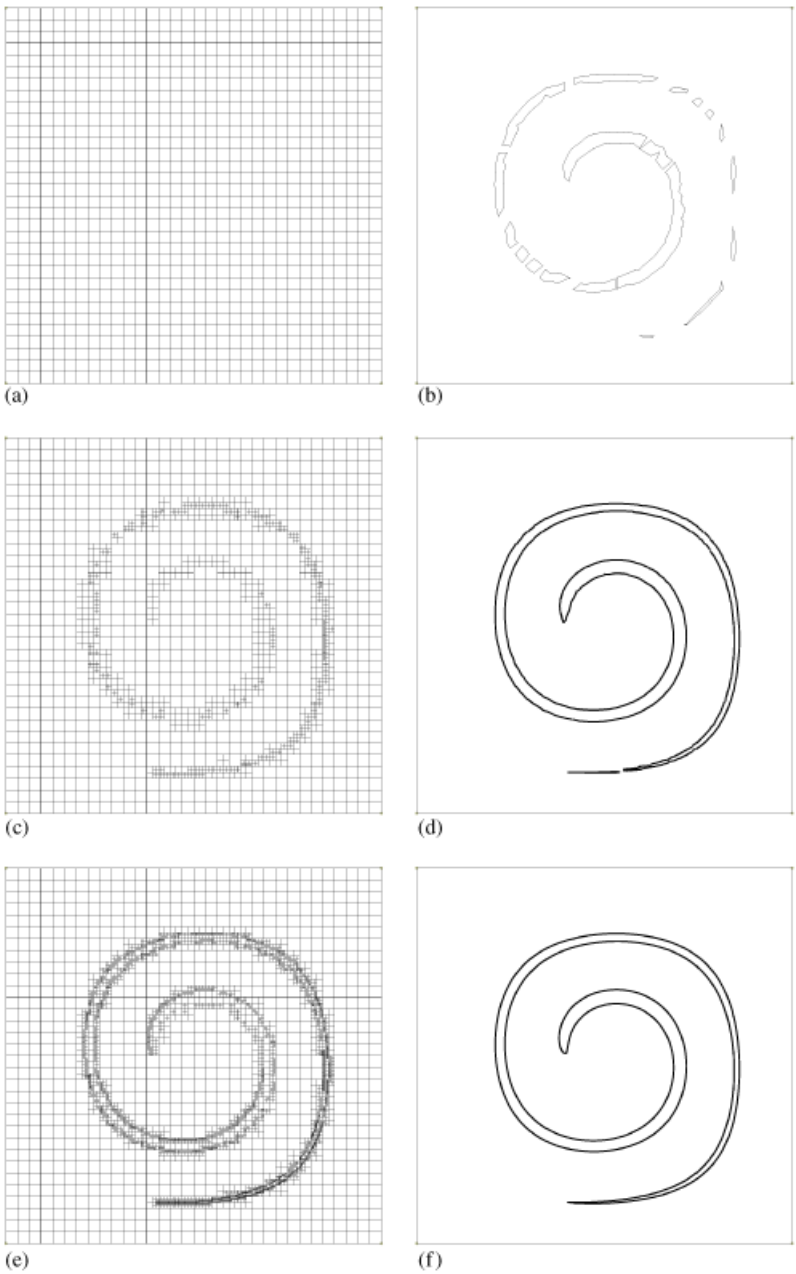

Figure 8. Visualization of the iso-zero of function u at time step 1000 for the vortex-in-a-box problem.

of $\bar{\varepsilon} = 2 \times 10^{-03}$, a maximal recursion level of $r = 6$ was used for obtaining the desired accuracy. Only 0.23 s of CPU time were necessary to do the adaptation. Visualization meshes are also shown in Figure 9.

DOI: 10.1002/nme

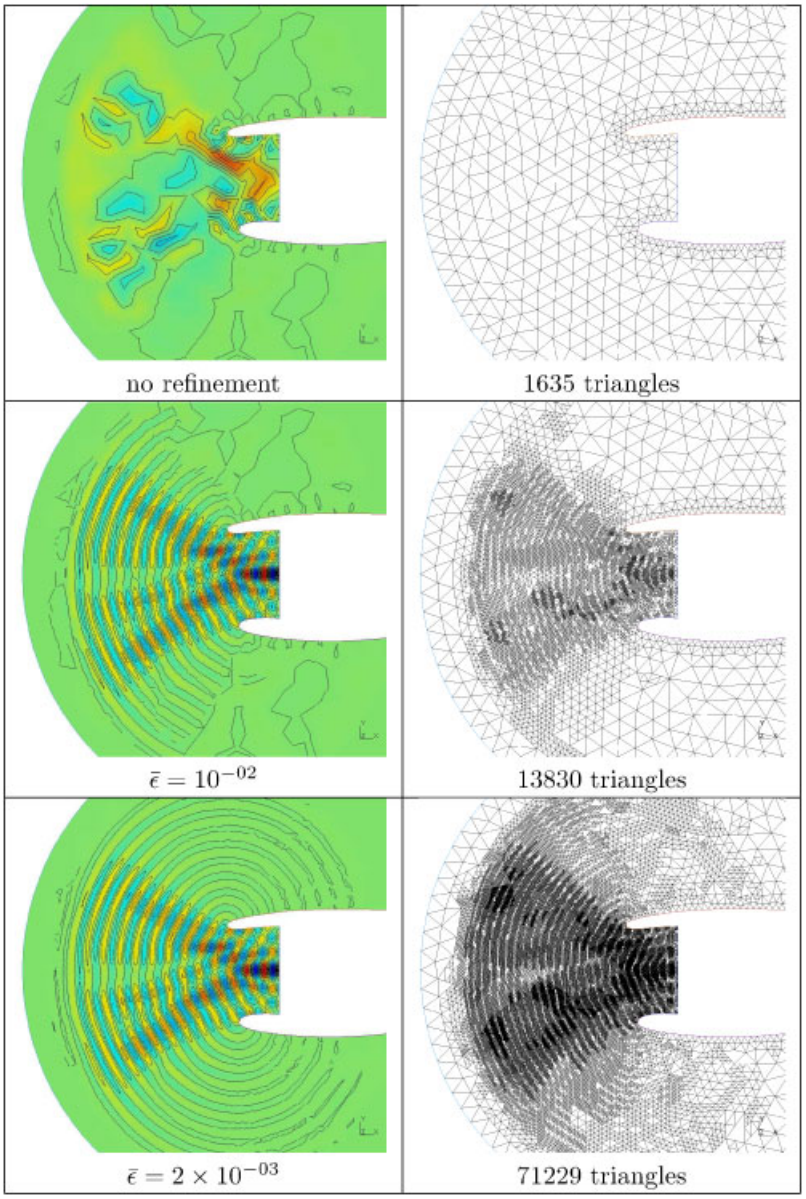

Figure 9. Pressure contours (left) and visualization meshes (right) for the 2D aeroacoustic problem at time $t = 1.10^{-2}$.

Pressure contours at different times are shown in Figure 10 for an error target of $\bar{\varepsilon} = 2 \times 10^{-04}$. At this level of refinement, about one 1.5 million visualization triangles were generated at iteration 2000 with a maximal recursion level of $r = 9$.

DOI: 10.1002/nme

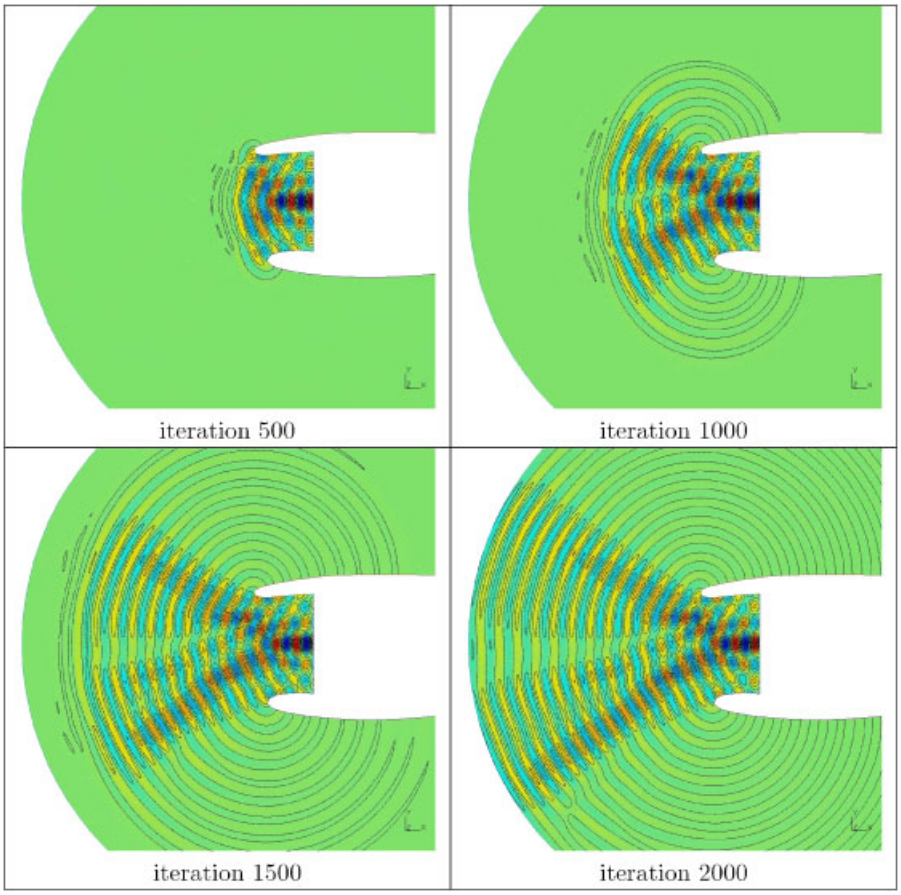

Figure 10. Pressure contours for the 2D aeroacoustic problem. A target error of $\bar{\varepsilon} = 10^{-4}$ was used.

8. THREE-DIMENSIONAL EXAMPLES

8.1. Deformation of a sphere

A sphere of radius 0.15 is placed within a unit computational domain at $(0.35, 0.35, 0.35)$ in a velocity field given by

$$\begin{aligned} v_x &= 2\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z)g(t) \\ v_y &= -\sin(2\pi x)\sin^2(\pi y)\sin(2\pi z)g(t) \\ v_z &= -\sin(2\pi x)\sin(2\pi y)\sin^2(\pi z)g(t) \end{aligned} \tag{4}$$

The time dependence $g(t)$ is given by

$$g(t) = \cos(\pi t/T)$$

DOI: 10.1002/nme

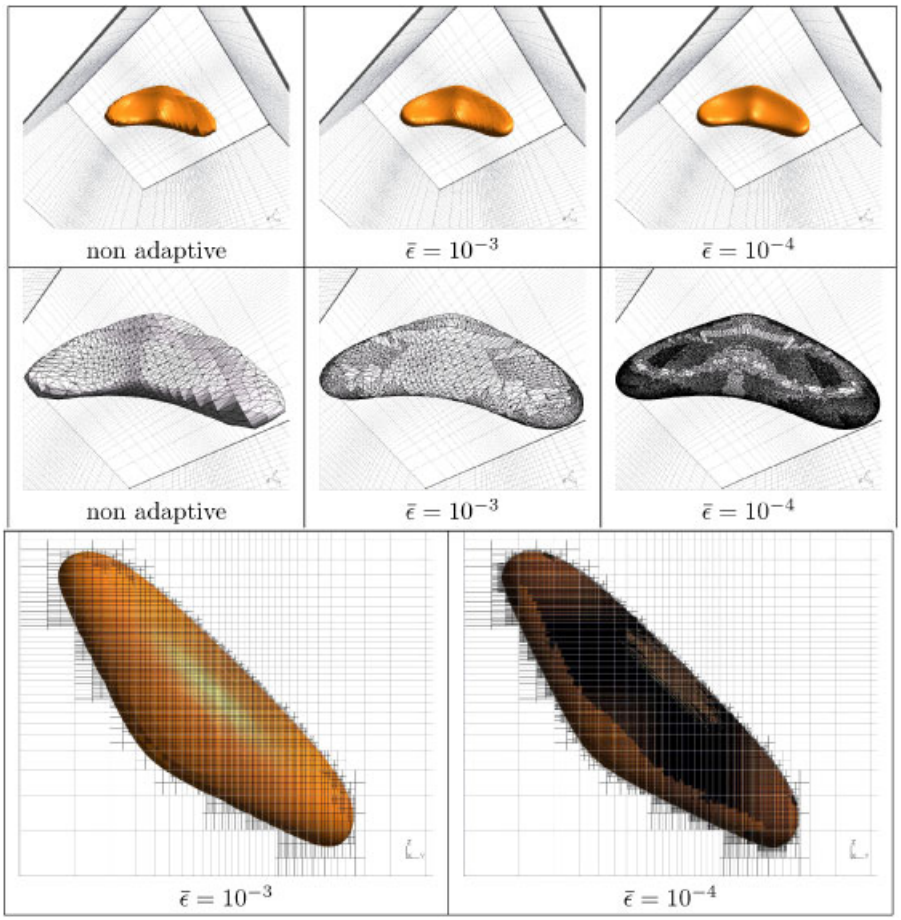

Figure 11. Visualization of the iso-zero for the problem of the deformation of the sphere. Left figures were computed using a target error of $\bar{\varepsilon} = 10^{-3}$ while $\bar{\varepsilon} = 10^{-4}$ was used for the three right sub-figures. Top figures show the iso-zero. Middle sub-figures show the visualization polygons. Bottom sub-figures show the volume visualization mesh.

where the reversal time period is chosen to be $T = 3$. This three-dimensional incompressible flow field combines a deformation in the x–y plane with a similar one in the x–z plane. The sphere is subsequently deformed by the flow.

We have computed this problem using a non-uniform hexahedral mesh composed of 32 787 elements. Some results of visualization are presented in Figure 11. Left figures were computed using a target error of $\bar{\varepsilon} = 10^{-3}$ while $\bar{\varepsilon} = 10^{-4}$ was used for the three right sub-figures of Figure 11. The hexahedral refined mesh for the left and right figures is composed of 66 172 and 405 784 elements, respectively. Some 48 426 triangles and 20 585 quadrangles were used to draw the iso-zero in the $\bar{\varepsilon} = 10^{-3}$ case. About 500 000 triangles and 227 000 quadrangles were used in the $\bar{\varepsilon} = 10^{-4}$ case. Note that a maximal recursion level of $r = 6$ was necessary to obtain the

DOI: 10.1002/nme

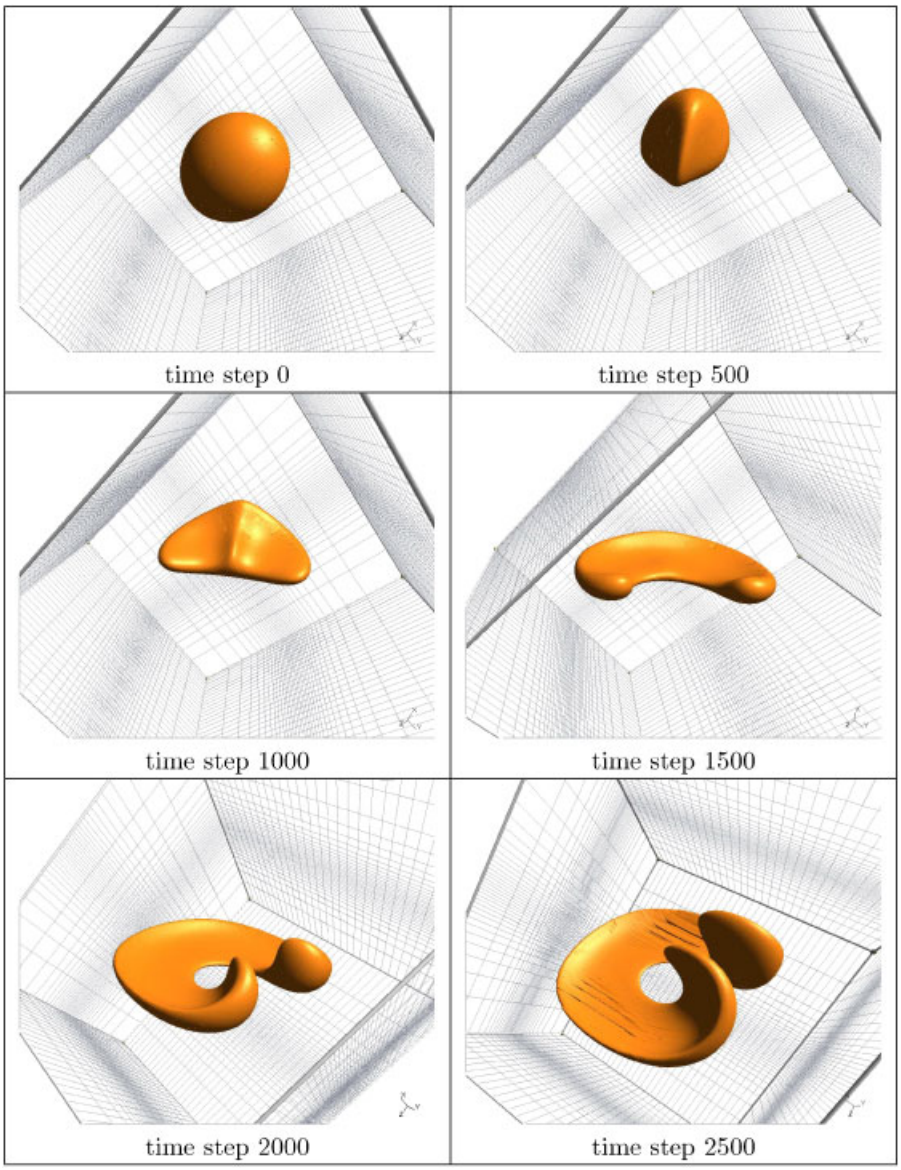

Figure 12. Adaptive visualization of the iso-zero for the problem of the deformation of the sphere at different times for a target error of $\bar{\varepsilon} = 10^{-4}$.

desired accuracy of $\bar{\varepsilon} = 10^{-4}$. An element that has reached this level of recursion has been cut into $8^6 = 262\,144$ sub-elements. Only a fraction of these sub-elements are visible. A fully refined visualization mesh would have required to process $32\,787 \times 262\,144 \simeq 8.5 \times 10^9$ elements. Here, 405 784 visualization cells correspond to about 21 000 times less than the equivalent fully refined

DOI: 10.1002/nme

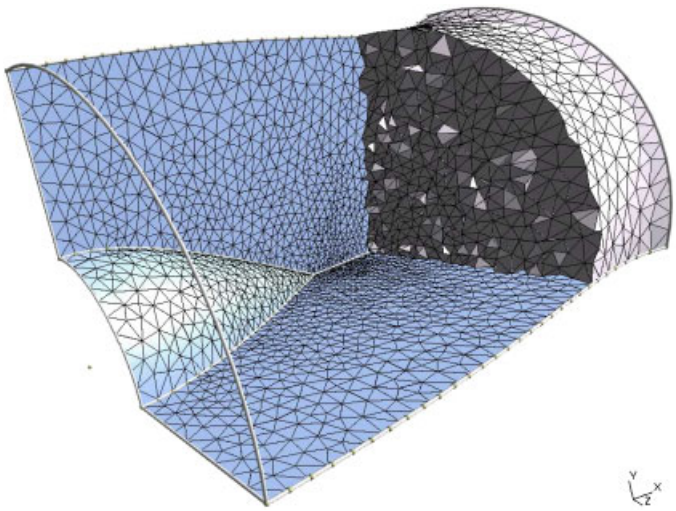

Figure 13. Geometry of the engine duct together with the computational mesh. The mesh is composed of 19 395 tetrahedra.

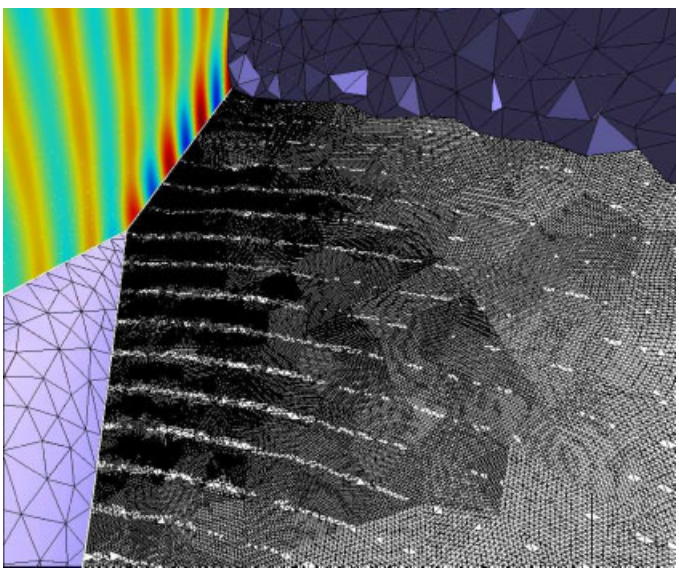

Figure 14. View of the adaptive visualization plane in the $z = 0$ plane.

mesh. Figure 12 shows visualization results at different time steps for a target error of $\bar{\varepsilon} = 10^{-4}$. More than 4 million polygons are used at time step 2500. For that specific time step, only 12 s of CPU time were necessary to build up the polygons, including the adaptive algorithm and the cutting of the 1 084 850 resulting hexahedra. Note that the visible irregularities of the iso-surface

DOI: 10.1002/nme

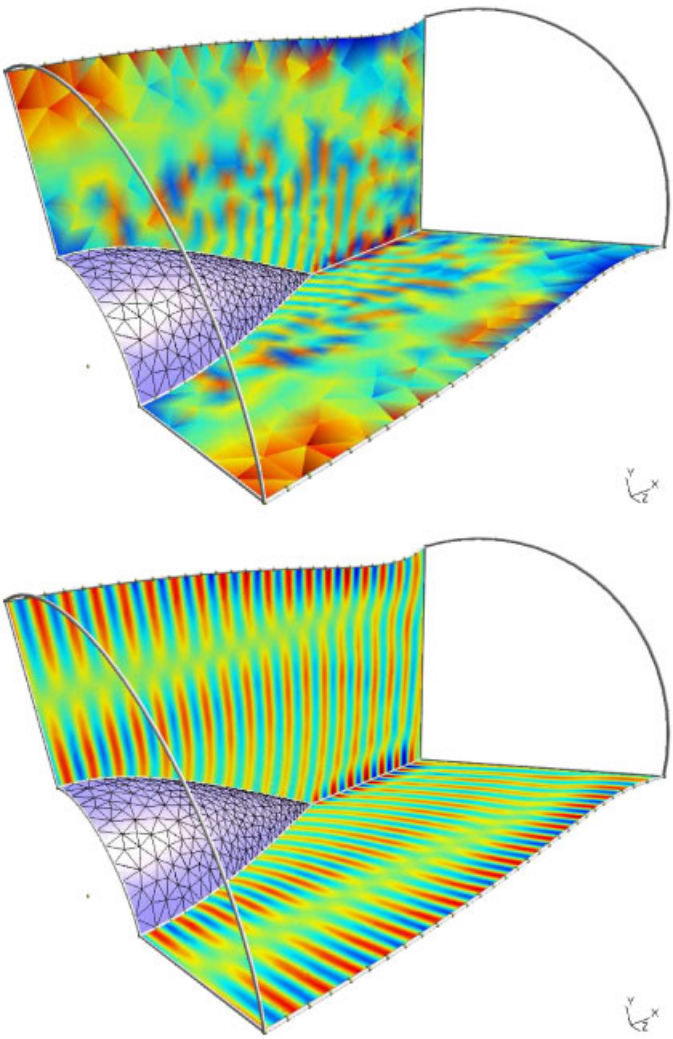

Figure 15. Visualization of the acoustic pressure field on both $y = 0$ and $z = 0$ planes. Top sub-figure shows the unrefined visualization results. Bottom sub-figure shows adaptive visualization results.

at time step 2500 are caused by the numerical scheme (the mesh is too coarse) and not to the accuracy of the visualization algorithm.

8.2. Propagation of acoustic modes in an exhaust duct

We consider the problem of the propagation of acoustic modes in a quarter of a engine exhaust duct. The geometry of the problem as well as the discretization mesh are represented in Figure 13. We have solved the linearized Euler equations (LEE) using fourth-order polynomials and a discontinuous Galerkin formulation. We have used our visualization algorithm for drawing contours

DOI: 10.1002/nme

of the acoustic pressure on both $y=0$ and $z=0$ planes with a target error of $\bar{\varepsilon}=10^{-4}$. Figure 15 shows visualization results in both refined and unrefined cases. Clearly, the non-adaptive visualization strategy fails to obtain any relevant results. Figure 14 shows a zoom of the adaptive mesh in the $z=0$ plane. Only 3 s of cpu time were needed to generate the adaptive results. There are about one million polygons in each visualization plane.

9. CONCLUSIONS

An adaptive technique for the visualization of high-order finite element fields has been developed. The technique is able to deal with general polynomial fields. AMR techniques have been used to generate optimal visualization grids. It has been shown that the method was able to provide visualization results using only a small fraction of the size that would have been required by an equivalent uniformly refined grid.

One of the main interests of the method developed here is its direct availability. An implementation is provided in Gmsh and this paper may be considered as a user's guide.

As a future work, we will consider extending the method to the visualization of non-polynomial fields with a focus on the extended finite element method (X-FEM). In X-FEM, non-polynomial enrichments are used for the representation of the solution of linear elasticity at the vicinity of the crack tip for example.

ACKNOWLEDGEMENTS

The various components of this work were supported by the European Community through Messiaen, a Specific Target Research Programme of the sixth framework programme and the Belgian National Fund for Scientific Research (FNRS).

REFERENCES

1. Deville M, Fisher PF, Mund E. *High-order Methods for Incompressible Fluid Flow*. Cambridge University Press: Cambridge, MA, 2002.
2. Babuska I, Suri M. The p and h–p versions of the finite element method, basic principles and properties. *SIAM Review* 1994; **36**(4):578–632.
3. Dunavant DA, Szabo BA. A posteriori error indicators for the p-version of the finite element method. *International Journal for Numerical Methods in Engineering* 2005; **19**(12):1851–1870.
4. Remacle J-F, Flaherty JE, Shephard MS. An adaptive discontinuous Galerkin technique with an orthogonal basis applied to compressible flow problems. *SIAM Review* 2003; **45**(1):53–72.
5. Ainsworth M. Dispersive and dissipative behavior of high order discontinuous Galerkin finite element methods. *Journal of Computational Physics* 2004; **198**(1):106–130.
6. Cockburn B, Shu C-W. TVB Runge–Kutta local projection discontinuous Galerkin methods for scalar conservation laws. II: General framework. *Mathematics of Computation* 1989; **52**:411–435.
7. Hu F, Atkins H. Eigensolution analysis of the discontinuous Galerkin method with nonuniform grids. I. One space dimension. *Journal of Computational Physics* 2002; **182**(2):516–545.
8. Chevaugeon N, Remacle J, Gallez, Ploumans P, Caro S. Efficient discontinuous Galerkin methods for solving acoustic problems. *11th AIAA/CEAS Aeroacoustics Conference*, 2005.
9. Warburton TC, Sherwin SJ, Karniadakis GE. Spectral basis functions for 2D hybrid hp elements. *SIAM Journal on Scientific Computing* 1999; **20**:1671–1695.
10. Remacle J-F, Li X, Flaherty JE, Shephard MS. Anisotropic adaptive simulation of transient flows using discontinuous Galerkin methods. *International Journal for Numerical Methods in Engineering* 2005; **62**(7):899–923.

DOI: 10.1002/nme

11. Yan J, Shu C-W. A local discontinuous Galerkin method for the kdv equations. *SIAM Journal on Numerical Analysis* 2002; **40**(2):769–791.
12. Dongarra JJ, Du Croz J, Duff IS, Hammarling S. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software* 1990; **16**:1–17.
13. Lorensen WE, Cline HE. Marching cubes in a high resolution 3d surface construction algorithm. *ACM Computer Graphics* 1987; **21**:163–169.
14. Geuzaine C, Remacle J-F. Gmsh web site. http://www.geuz.org/gmsh
15. Gough B (ed.). *GNU Scientific Library Reference Manual*.
16. Todd Elvins T. A survey of algorithms for volume visualization. *ACM Siggraph Computer Graphics* 1992; **26**(3):194–201.
17. Hansen CD, Hinker P. Massively parallel isosurface extraction. *VIS '92: Proceedings of the 3rd conference on Visualization '92*. IEEE Computer Society Press: Los Alamitos, CA, U.S.A., 1992; 77–83.
18. Treece GM, Prager RW, Gee AH. Regularised marching tetrahedra: improved iso-surface extraction. *Computers and Graphics* 1999; **23**(4):583–598.
19. Haasdonk B, Ohlberger M, Rumpf M, Schmidt A, Siebert KG. Multiresolution visualization of higher order adaptive finite element simulations. *Computing* 2003; **70**(3):181–204.
20. Berger MJ, Oliger J. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics* 1984; **53**:484–512.
21. Leone AO, Marzano P, Gobbetti E, Scateni R, Pedinotti S. Discontinuous finite element visualization. In *Proceedings of the 8th International Symposium on Flow Visualisation*, 1998; 1–6.

DOI: 10.1002/nme