

Efficient Visualization of High Order Finite Elements

Jean-François Remacle^{1*}, Nicolas Chevaugeon¹, Émilie Marchandise^{1,3}, 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*

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 some optimal visualization grid. Some filtering is added to the algorithm in order to focus the refinement on a visualization plane, one on the computation of one single iso-zero surface. Some 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 that is developed by the authors, namely Gmsh. Copyright © 2005 John Wiley & Sons, Ltd.

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 [6], high order finite elements [3, 2] or high order Discontinuous Galerkin Methods [12, 1, 5]. Our focus is high order Discontinuous Galerkin Methods (DGM). Those are now used extensively for solving transient aeroacoustics problems [10], electromagnetic problems [14], dynamics of compressible fluids [13, 12], Korteweg-de Vries (KdV) equations [15] and many other relevant physical problems. Using a quadrature free approach together with a very careful BLAS3 implementation, our experience show that, with 5th or 6th order polynomials on a tetrahedral mesh, we can run a DGM code for which we use about 80 percent of the peak 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

*Correspondence to: J.F. Remacle, Département de Génie Civil et Environnemental, Université Catholique de Louvain, Bâtiment Vinci, Place du Levant 1, B-1348 Louvain-la-Neuve, Belgium.

†Email: remacle@gce.ucl.ac.be

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 [12, 13].

In all what follows, computer efficiency is always a central concern and we will be always careful when it comes to cpu time issues. When plots and images of 2D and 3D results will be shown, we will always indicate computation time and 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, it is the solver that 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, we think that providing some kind of source code is essential in this kind of business. An implementation of our methodology is readily available as an open source code. Gmsh[8] 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 copyright ©1997-2004 by C. Geuzaine and J.-F. Remacle and is distributed under the terms of the GNU General Public License (GPL). The high order visualizations techniques described in this paper have been added to Gmsh. All the illustrations shown below have been done with Gmsh.

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}_e elements

$$\mathcal{T}_e = \bigcup_{e=1}^{\mathcal{N}_e} e \quad (1)$$

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 Finite Element Methods (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.

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

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

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 j th approximation function defined in a common parametric space ξ, η, ζ . In each element, we have d coefficients, and,

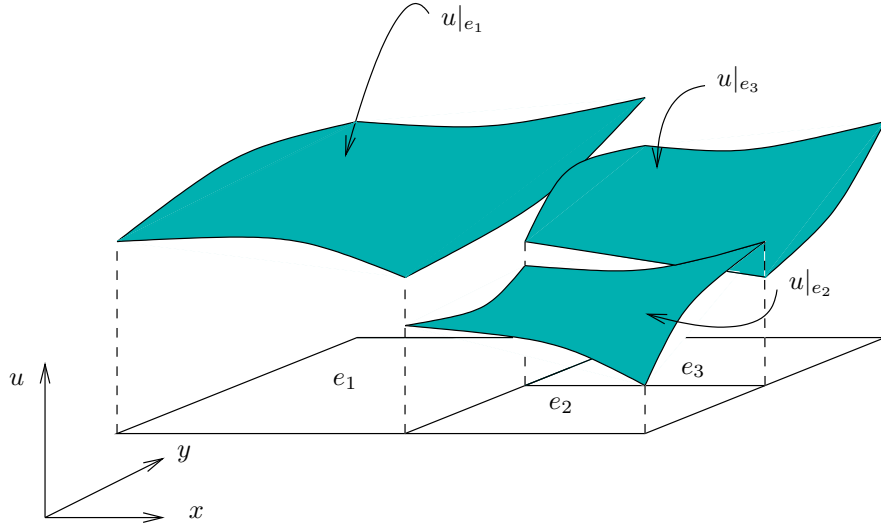


Figure 1. Three elements e_1 , e_2 and e_3 and the piecewise discontinuous solution u .

because we consider (this is the most general case) that all approximations are disconnected, $\mathcal{N}_e \times d$ coefficients are required for describing u . In our implementation, we organize those data in a $\mathcal{N}_e \times d$ matrix that we call U . U_{ij} is therefore the j th component of u^i . We use the GNU scientific library (GSL) for manipulating matrices and vectors. The GSL have the advantage to provides direct and easy bindings to the Basic Linear Algebra Subroutines (BLAS).

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

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

with

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

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

$$z^e = \sum_{j=1}^m \psi_j(\xi, \eta, \zeta) Z(e, j).$$

Matrices X , Y and Z are of size $m \times 3$.

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 represent of discontinuous fields,
- distinguish geometric and functional discretization,

- be able to use any set of polynomial interpolation function.

The fact that we have disconnected interpolations by defining as much coefficients as $\mathcal{N}_e \times d$ allow 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 the 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 allow the definition of a canonical polynomial basis:

$$\mathcal{P} = \{p_1(\xi, \eta, \zeta), \dots, p_d(\xi, \eta, \zeta)\}$$

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, the a bilinear isoparametric quadrilateral element, we have

$$P = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}, \Phi = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}, \mathcal{P} = \{1, \xi, \eta, \xi\eta\} \text{ and}$$

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

This way of building the basis allow to define any polynomial basis, complete or not, of any dimension. The advantage is that any FEM code can use the tool, eventually 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 allow the use of exotic polynomial approximations and are, therefore, a good support for testing our visualization methodology.

3. Contouring high order fields

One interesting approach to high order contouring can be found in [9]. This approach is based on the definition of some kind of algebra of high order textures. Each basis function ϕ_i correspond to one basis texture and the final visualization object is constructed by linear combination of basis textures. Figure 2 show an example of this approach. Quadratic textures s_1 , s_2 and s_3 corresponding to hierarchical shape functions associated to the edges of a triangle

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;
};

```

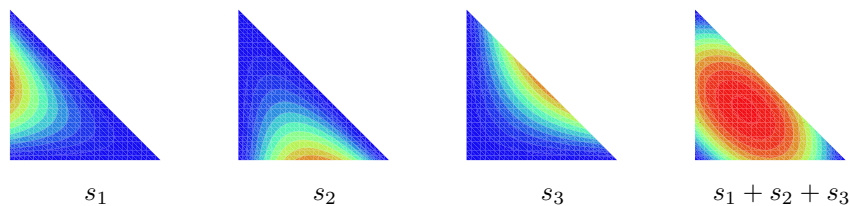


Figure 2. Illustration of the visualization technique developed in [9] based on high order textures.

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, the most common graphical API, i.e. OpenGL, does not fully support linear combination of textures.

More important, this approach has aspects that are way too restrictive for finite element visualizations. The number of isocontours, for example, cannot be changed easily because basis textures contain a fixed number of colors. 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.

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 7th or 9th 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 consist 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 here 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 here is based on classical AMR (Automatic Mesh Refinement) methodologies, see for example [4, 12]. For each element type (triangle, quad, 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 on Figure

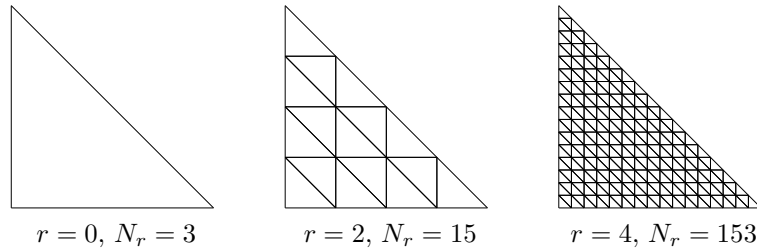


Figure 3. Some AMR refinement templates for triangles at different resolution levels.

3 at different recursion levels r . The element sub-division pattern is performed in the reference system of coordinates 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

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

for triangles,

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

for quadrangles,

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

for tetrahedra and

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

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

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

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 [7]. 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

us 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 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 colormap 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 9th order Lagrange shape function in the reference triangle. In one case, we want to visualize the function using filled (colored) iso-values. In the other case, we want to represent one given iso-value $u^e = C$. Figure 4 show the two different adapted visualization grids relative to the two pre-defined visualization goals. We can do the same with

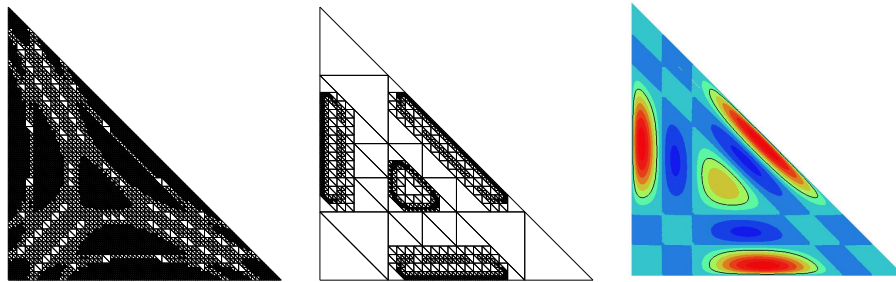


Figure 4. Visualization of one 9th order Lagrange shape function. On left, we see the visualization mesh for the goal $|u - u_h| < 0.001$. On center, we see the visualization mesh that has been optimized in order to capture accurately one given iso-contour $u = 0.5$. On right, we see the corresponding visualization results i.e a colormap and, in black, the iso-contour $u = 0.5$.

3D views on tetrahedral meshes and an example is shown at Figure 5.

5. Visualization error

We aim here 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 posteriori error analysis, the “Quest of the Graal” is to find local error. Indeed, if the error at a point is known, so is the exact solution. It is usually estimations of L^2 or H^1 that are available. In this case, and contrary to the case of finite element *a posteriori* error estimation, 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

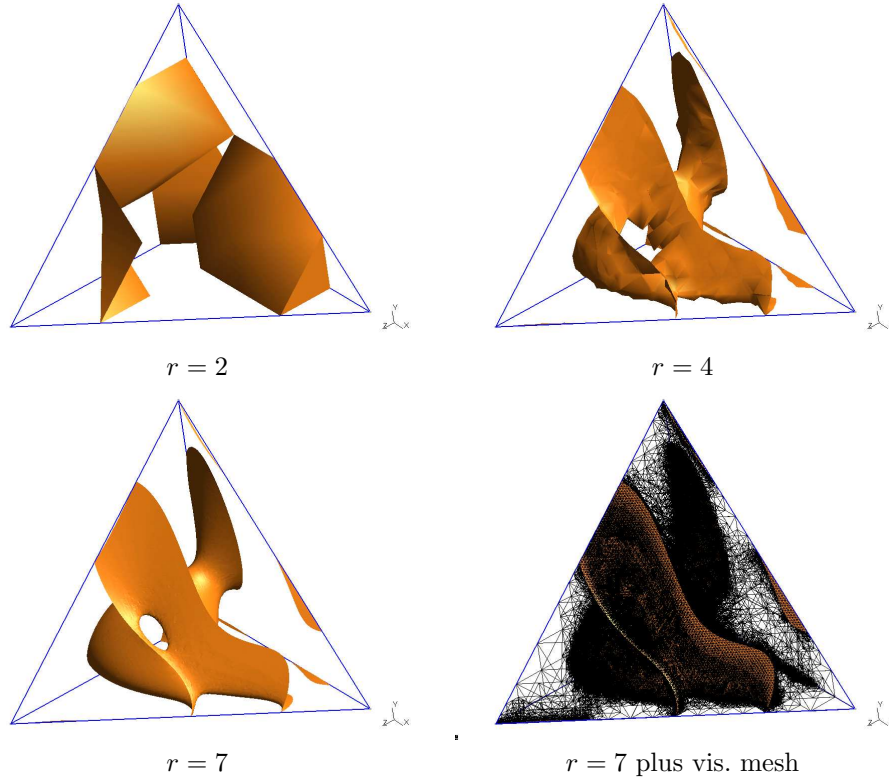


Figure 5. Visualization of one 4th order function on one single tetrahedron. Figures show one iso-surface using different error thresholds. Last figure (bottom-right) show the visualization mesh together with the iso-surface.

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 correspond 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

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

which is similar to what can be found in [11]. Note that, for example,

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

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

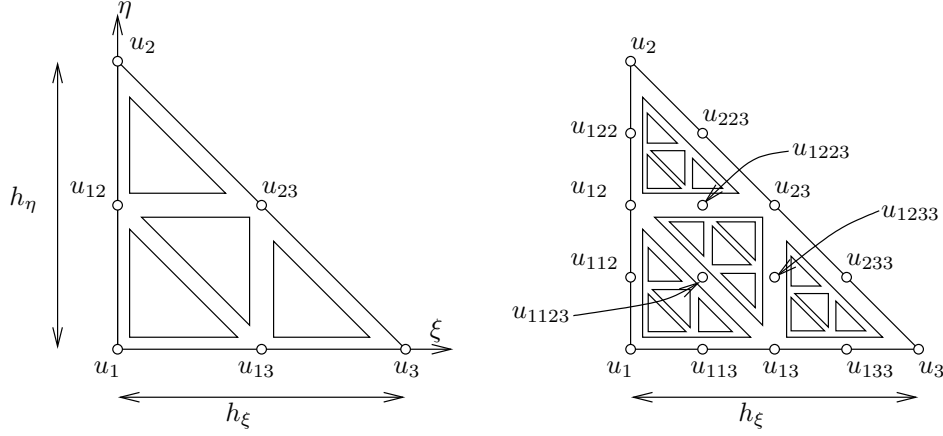


Figure 6. One triangular element and its subdivision.

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 $\epsilon^e < \bar{\epsilon}$ where $\bar{\epsilon}$ 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 some few arithmetic operations. Hence, 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

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

with

$$\begin{aligned}\epsilon_{ij}^e &= |2u_{ij} - u_i - u_j|, \\ \epsilon_{ijj}^e &= \left| 2u_{ijj} - \frac{1}{2}u_i - \frac{3}{2}u_j \right|\end{aligned}$$

and

$$\epsilon_{ijjk}^e = \left| 2u_{ijjk} - u_j - \frac{1}{2}(u_i + u_j) \right|.$$

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 4 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

Algorithm 2 Visualization element class.

```

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

```

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 weather it is useful to go to a downer level. On the way up of the recursion, we may apply some filtering i.e. eliminate useless parts of the view. The filter may only allow sub-elements whose nodal values are in a prescribed range (the case of the capture of one given iso-surface). Another useful filter 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` element are visible.

7. Two Dimensional Examples

7.1. Vortex-in-a-box

This example is usually chosen in order to test the ability of a numerical 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

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

that defines the following velocity field

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

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 compute 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 a very 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.

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 = 1;
        return false;
    }
    else
        // compute error using the 1- or the 2-level formula
        double eps = compute_error ( e );
        if (eps > eps_0){
            e->visible=0;
            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=1;
            return true;
        }
    }
};

```

Figures 7 and 8 show an adaptive visualization. Figures 7-(a) and 7-(b) show the unrefined mesh (1024 quads) and the unrefined visualization. Figures 7-(c) and 7-(d) show results for a visualization error of $\bar{\epsilon} = 10^{-2}$: the visualization mesh 7-(c) is made of 4021 quads 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 seconds. Figures 7-(e) and 7-(f) show results for a visualization error of $\bar{\epsilon} = 10^{-3}$: the visualization mesh 7-(e) is made of 46747 quads 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 seconds.

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

the visualization mesh 8 (e) is made of 6454 quads and the computation time for generating 8-(f) was 0.08 seconds.

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

8. Three Dimensional Examples

8.1. Deformation of a sphere

A sphere of radius .15 is placed within a unit computational domain at (.35, .35, .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:

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

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 32787 elements. Some results of visualization are presented in Figure 9. Left figures were computed using a target error of $\bar{\epsilon} = 10^{-3}$ while $\bar{\epsilon} = 10^{-4}$ was used for the three right sub-figures of Figure 9. The hexahedral refined mesh for the left and right figures is composed of 66172 and 405784 elements respectively. Some 48426 triangles and 20585 quadrangles were used to draw the iso-zero in the $\bar{\epsilon} = 10^{-3}$ case. About 500000 triangles and 227000 quadrangles were used in the $\bar{\epsilon} = 10^{-4}$ case. Note that a maximal recursion level of $r = 6$ was necessary to obtain the desired accuracy of $\bar{\epsilon} = 10^{-4}$. An element that has reached this level of recursion has been cut into $8^6 = 262144$ sub-elements. Only a fraction of these sub-elements are visible. A fully refined visualization mesh would have required to process $32787 \times 262144 \simeq 8.5 \cdot 10^9$ elements. Here, 405784 visualization cells correspond to about 21000 times less than the equivalent fully refined mesh. Figure 10 shows visualization results at different time steps for a target error of $\bar{\epsilon} = 10^{-4}$. More than 4 million polygons are used at time step 2500. For that specific time step, only 12 seconds of cpu time were necessary to build up the polygons, including the adaptive algorithm and the cutting of the 1084850 resulting hexahedra. Note that the visible irregularities of the iso-surface 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 a duct.

We consider the problem of the propagation of acoustic modes in a quarter of a engine duct. The geometry of the problem as well as the discretization mesh are represented in Figure

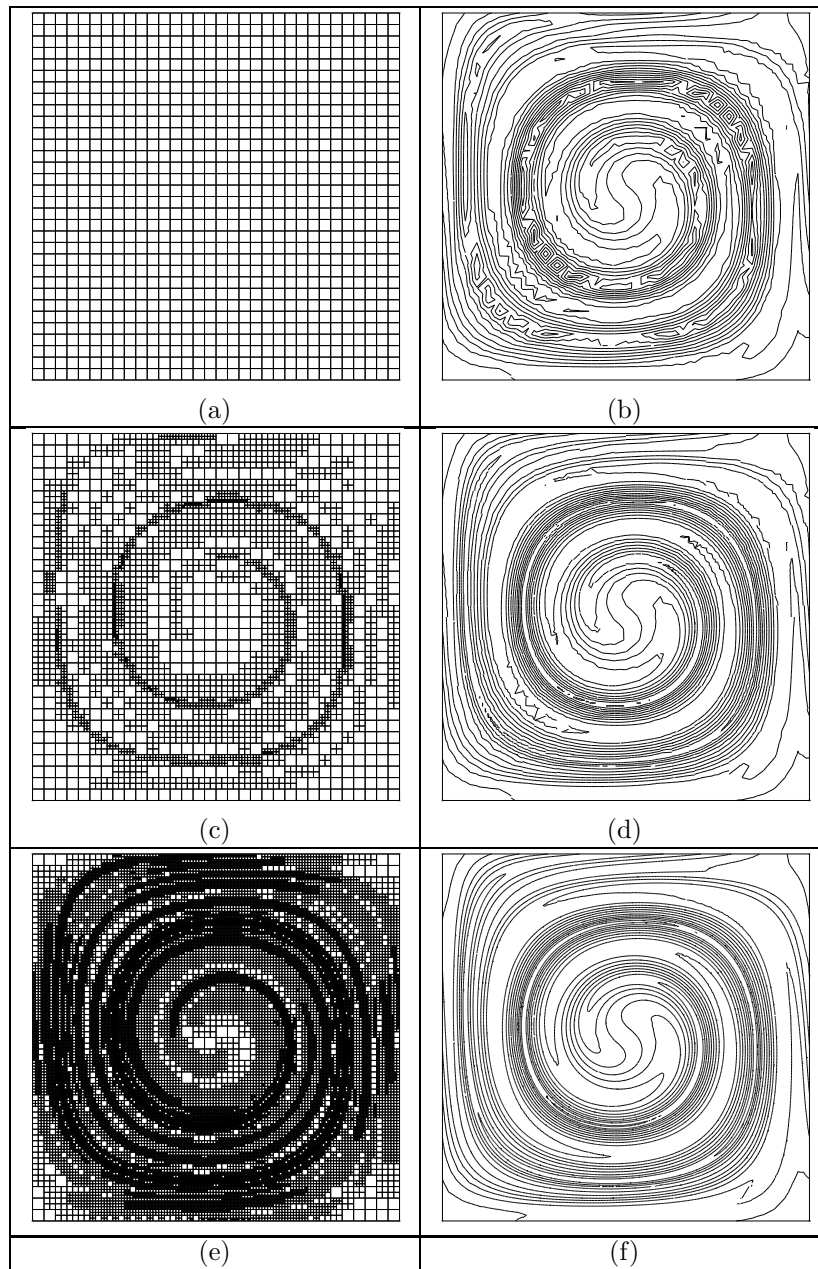


Figure 7. Visualization of function u at time step 1000 for the Vortex In a Box problem.

11. We have solved the Linearized Euler Equations (LEE) using 4th order polynomials and a Discontinuous Galerkin formulation. We have used our visualization algorithm for drawing

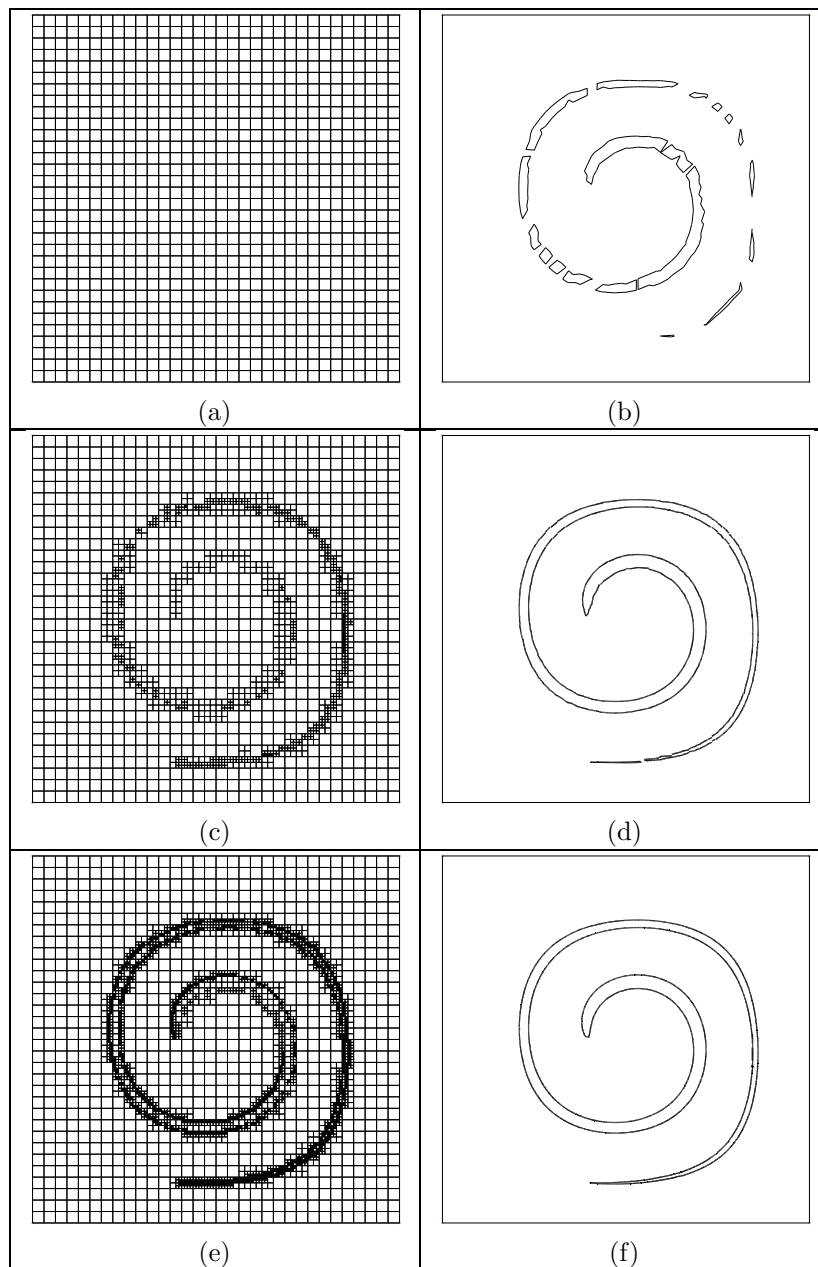


Figure 8. Visualization of the iso-zero of function u at time step 1000 for the Vortex In a Box problem.

contours of the acoustic pressure on both $y = 0$ and $z = 0$ planes with a target error of

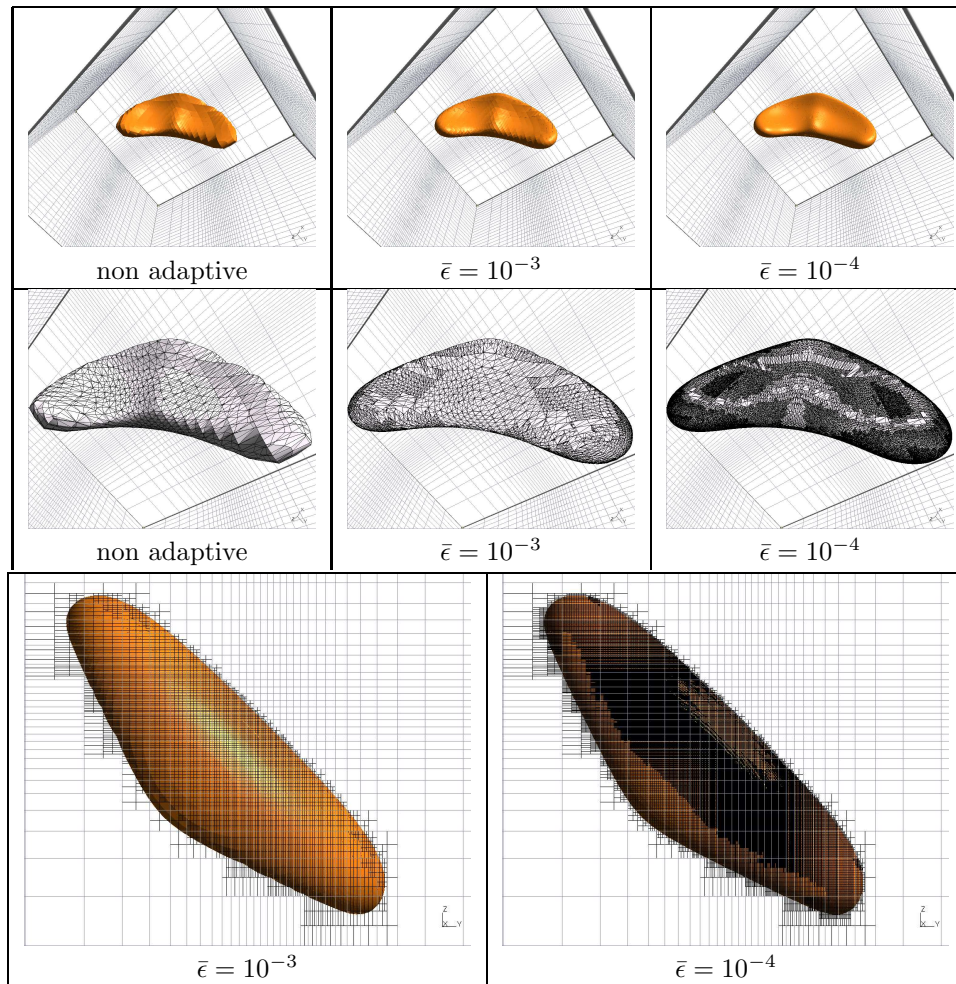


Figure 9. Visualization of the iso-zero for the problem of the deformation of the sphere. Left figures were computed using a target error of $\bar{\epsilon} = 10^{-3}$ while $\bar{\epsilon} = 10^{-4}$ was used for the three right sub-figures. Top figures shows the iso-zero. Middle sub-figures show the visualization polygons. Bottom sub-figures shows the volume visualization mesh.

$\bar{\epsilon} = 10^{-4}$. Figure 13 shows visualization results in both refined and unrefined cases. Clearly, the non-adaptive visualization strategy fails to obtain any relevant results. Figure 12 shows a zoom of the adaptive mesh in the $z = 0$ plane. Only 3 seconds of cpu time were needed to generate the adaptive results. There are about one million polygons in each visualization plane.

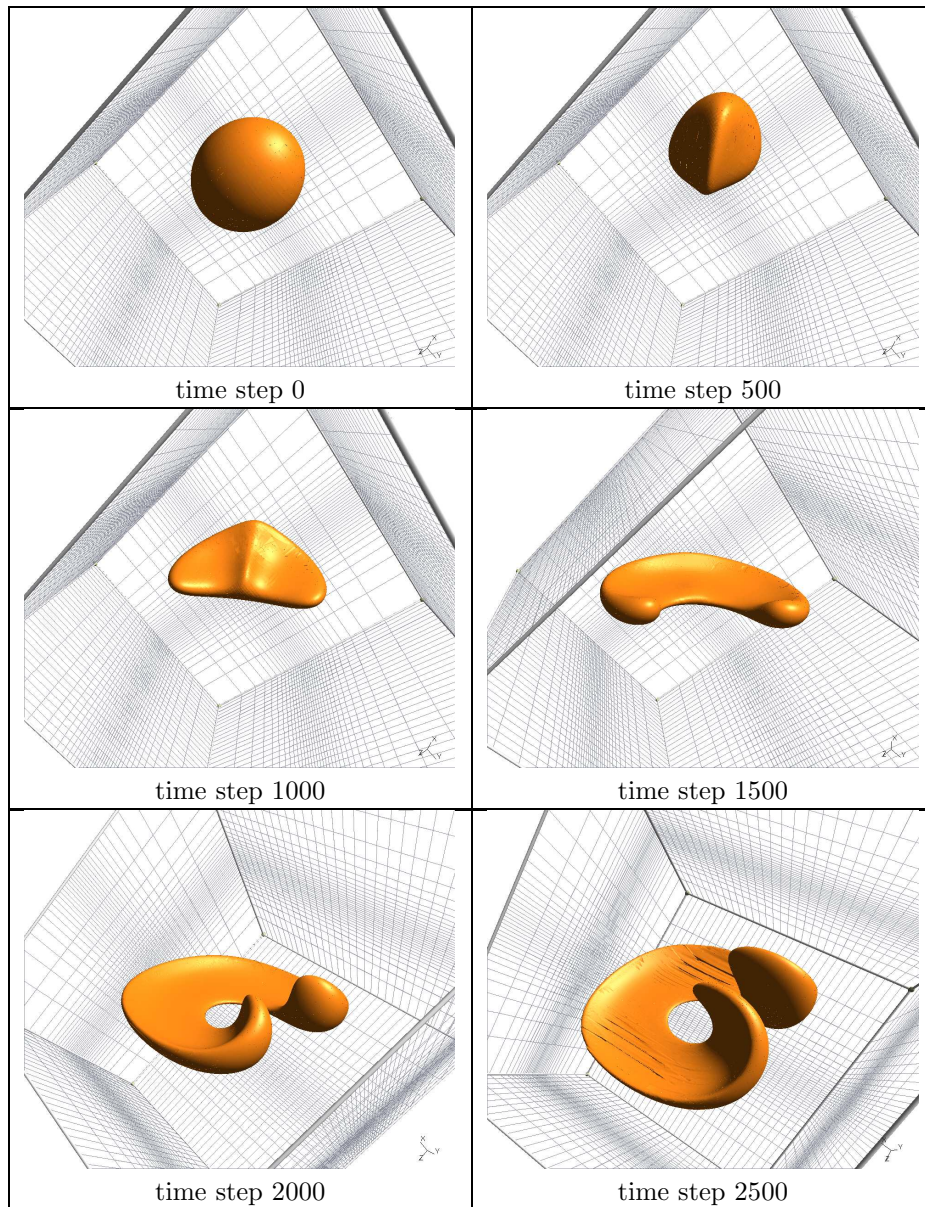


Figure 10. Adaptive visualization of the iso-zero for the problem of the deformation of the sphere at different times for a target error of $\bar{\epsilon} = 10^{-4}$.

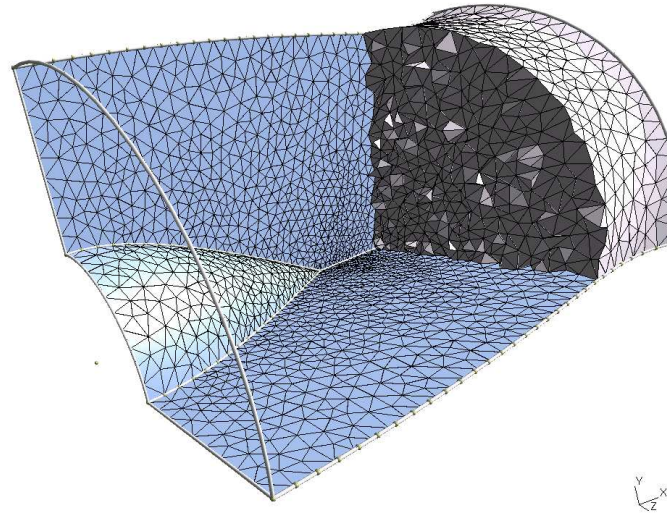


Figure 11. Geometry of the engine duct together with the computational mesh. The mesh is composed of 19395 tetrahedra.

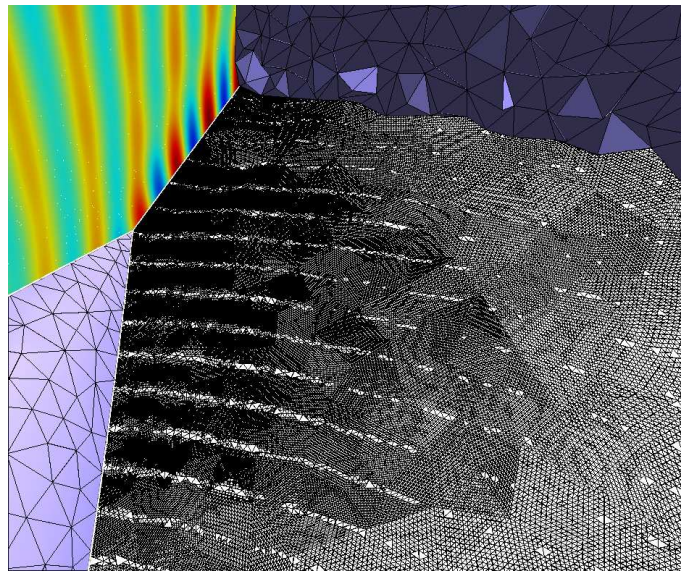


Figure 12. View of the adaptive visualization plane in the $z = 0$ plane.

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. Automatic mesh
Copyright © 2005 John Wiley & Sons, Ltd. *Int. J. Numer. Meth. Engng* 2005; **00**:1–6
Prepared using nmeauth.cls

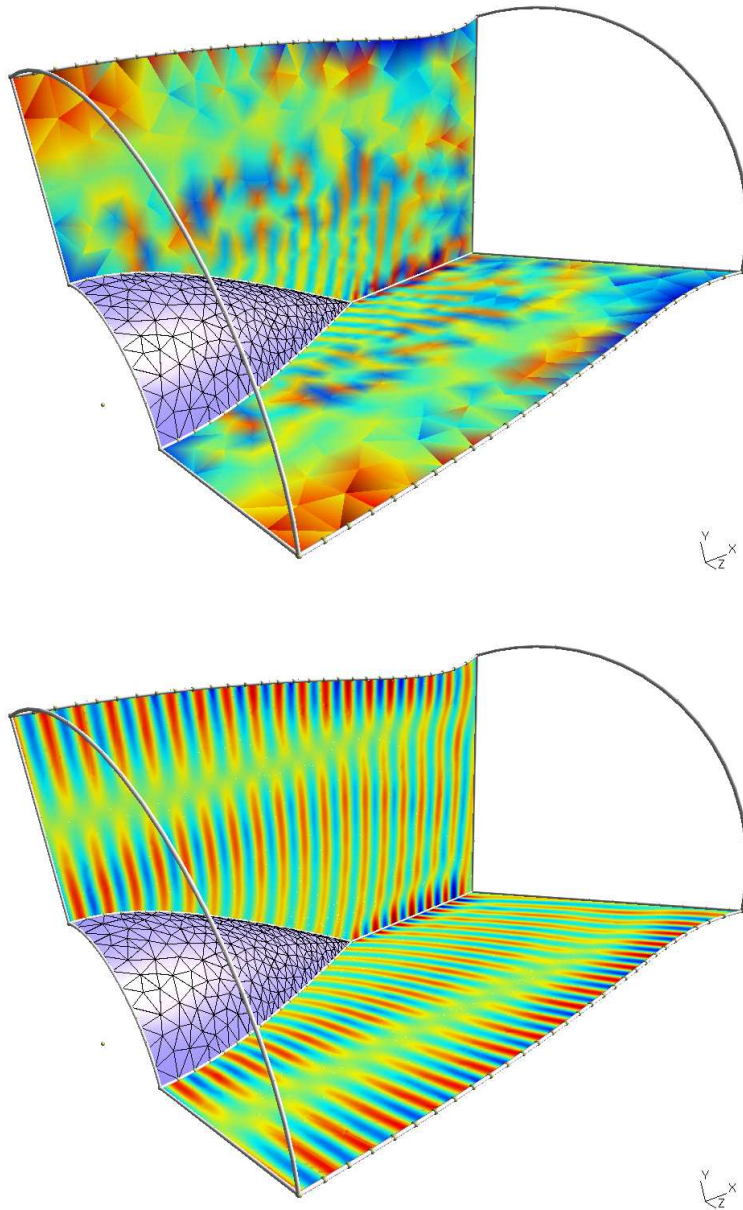


Figure 13. Visualization of the acoustic pressure field on both $y = 0$ and $z = 0$ planes. Top sub-figure shows the unrefined visualization results. Bottom sub-figure shows adaptive visualization results.

refinement (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 to extend the method 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 6th framework programme.

REFERENCES

1. M. Ainsworth. Dispersive and dissipative behavior of high order Discontinuous Galerkin finite element methods. *Journal of Computational Physics*, 198(1):106–130, 2004.
2. Szabo B. and Babuska I. *Finite Element Analysis*. John Wiley and sons, 1991.
3. Ivo Babuska and Manil Suri. The p and h-p versions of the finite element method, basic principles and properties. *SIAM Rev.*, 36(4):578–632, 1994.
4. M.J. Berger and J. Olinger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, 1984.
5. B. Cockburn and C.-W. Shu. TVB Runge-Kutta local projection discontinuous Galerkin methods for scalar conservation laws II: General framework. *Mathematics of Computation*, 52:411–435, 1989.
6. M. Deville, P. F. Fisher, and E. Mund. *High-order methods for incompressible fluid flow*. Cambridge University Press, 2002.
7. J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Soft.*, 16:1–17, 1990.
8. C. Geuzaine and J.-F. Remacle. Gmsh web site. <http://www.geuz.org/gmsh>.
9. B. Haasdonk, M. Ohlberger, M. Rumpf, A. Schmidt, and K. G. Siebert. Multiresolution visualization of higher order adaptive finite element simulations. *Computing*, 70(3):181–204, 2003.
10. F. Hu and H. Atkins. Eigensolution analysis of the Discontinuous Galerkin Method with Nonuniform Grids i One Space Dimension. *Journal of Computational Physics*, 182(2):516–545, 2002.
11. A. O. Leone, P. Marzano, E. Gobetti, R. Scateni, and S. Pedinotti. Discontinuous finite element visualization. In *In the proceedings of th 8th International Symposium on Flow Visualisation*, pages 1–6, 1998.
12. J.-F. Remacle, J.E. Flaherty, and M.S. Shephard. An adaptive discontinuous galerkin technique with an orthogonal basis applied to compressible flow problems. *SIAM Review*, 45(1):53–72, 2003.
13. J.-F. Remacle, X. Li, J.E. Flaherty, and M.S. Shephard. Anisotropic adaptive simulation of transient flows using discontinuous galerkin methods. *International Journal for Numerical Methods in Engineering*, 62(7):899–923, 2005.
14. T.C. Warburton, S.J. Sherwin, and G.E. Karniadakis. Spectral basis functions for 2D hybrid hp elements. *SIAM Journal on Scientific Computing*, 20:1671–1695, 1999.
15. Jue Yan and Chi-Wang Shu. A local discontinuous galerkin method for the kdv equations. *SIAM Journal of Numerical Analysis*, 40(2):769–791, 2002.