Engineering with Computers (2006) 21: 247–258
DOI 10.1007/s00366-005-0009-3

ORIGINAL ARTICLE

Frédéric Alauzet · Xiangrong Li · E. Seegyoung Seol
Mark S. Shephard

Parallel anisotropic 3D mesh adaptation by mesh modification

Received: 1 July 2005 / Accepted: 1 August 2005 / Published online: 20 January 2006

Abstract Improvements to a local modification-based anisotropic mesh adaptation procedure are presented. The first improvement focuses on control of the local operations that modify the mesh to satisfy the given anisotropic mesh metric field. The second is the parallelization of the mesh modification procedures to support effective parallel adaptive analysis. The resulting procedures are demonstrated on general curved 3D domains where the anisotropic mesh size field is defined by either an analytic expression or by an adaptive correction indicator as part of a flow solution process.

1 Introduction

Unstructured mesh adaptation has proven its efficiency for improving the accuracy of numerical solutions used to model the behavior of physical phenomena. Adaptive methods dramatically reduce the number of degrees of freedom needed to obtain a given level of accuracy. In order to solve large scale problems, the mesh adaptation process has to be parallelized. Papers have been published on mesh adaptation for numerical simulations in computational solid and fluid mechanics (see reference [1] for a survey). Recently, a number of papers have addressed the creation of 3D unstructured adaptive anisotropic meshes [2–12].

Mesh adaptation can be carried out through two alternative mechanisms, by global remeshing of the entire domain based on meshing techniques, or by local mesh modifications. This paper gives the recent improvements of our approach to generate unstructured anisotropic meshes using local mesh modification [7, 13] that improve the ability of the mesh to respect of the given prescribed anisotropic size field. The improvements focus on the importance of carefully selected local mesh operators, such as coarsening and swapping, during the refinement procedure to increase the quality of the mesh to improve the efficiency of this stage. The paper also describes the approach taken to have 3D mesh adaptation procedures run on distributed meshes partitioned on parallel computers. Performing mesh adaptation in parallel requires parallelization of each individual modification procedure and local mesh migration algorithm to treat mesh entities on or near by the common boundary between several partitions [14–16].

Section 2 describes the anisotropic mesh adaptation procedure by local mesh modifications. Section 3 presents the parallelization of the mesh modification operators. In Sect. 4, examples of analytical metric fields on a curved geometry are provided to illustrate the efficiency of the proposed approach. Finally, Sect. 5 demonstrates the application of the adaptive platform to a 3D example in CFD employing the discontinuous Galerkin finite element method.

F. Alauzet · X. Li · E. S. Seol (✉) · M. S. Shephard
Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, USA
E-mail: seole@scorec.rpi.edu

2 Anisotropic mesh adaptation by mesh modification

In many engineering applications, it is desirable to create adapted meshes representing anisotropic features (stretc- hed elements in arbitrary directions). This section provides recent improvements of our approach to generate unstructured anisotropic meshes using mesh modification.

2.1 Metric related issues

The first step of anisotropic mesh adaptation constructs an appropriate metric tensor describing the size and anis- otropy of the mesh over the domain.

2.1.1 Mesh metric field

Mesh metric fields are used to specify the desired size of elements in each direction. At each point of the space, the metric field is defined by a $d \times d$ symmetric definite positive tensor, denoted $\mathscr{M}$, where d is the dimension of the domain being meshed. This tensor is diagonalizable as $\mathscr{M} = \mathscr{R}\Lambda\mathscr{R}^{-1}$, where $\mathscr{R}$ is the eigenvectors matrix and Λ is the diagonal matrix of the eigenvalues. The eigenvectors and eigenvalues matrix can be considered, respectively, as the combination of rotation matrix that gives the principal directions of the metric space and the size in the corresponding directions. A metric tensor $\mathscr{M}$ can be represented geometrically by its unit ball that is mapped to an ellipsoid in the space of the mesh. The principal axes are given by the eigenvectors of $\mathscr{M}$ and the length of each axe by the square roots of the inverse of the corresponding eigenvalues. For implementation purposes, a discrete approximation of the metric field is defined at the vertices of the mesh with metric. Interpolation schemes are used to obtain a continuous field over the domain.

2.1.2 Unit mesh

The generation of an adapted mesh is based on the specification of a mesh metric field to compute the edge lengths and directions with respect to this metric. For the sake of simplicity, it is possible to define the metric tensor to prescribe a unit edge length in the metric space. The standard Euclidean scalar product is then modified using a metric tensor field. Let P be a vertex and let $\mathscr{M}(P)$ be the metric at P. The desired edge PX must have a length close to one with respect to $\mathscr{M}(P)$:

$$l_{\mathscr{M}(P)}(\overrightarrow{PX}) = \sqrt{{}^t\overrightarrow{PX}\,\mathscr{M}(P)\overrightarrow{PX}} = 1.$$

As the metric varies in the domain (it is not constant in an element), we need to consider the metrics at the edge endpoints as well as all intermediate metrics along the edge. To this end, we introduce the average length of PX as:

$$l_{\mathscr{M}}(\overrightarrow{PX}) = \int_0^1 \sqrt{{}^t\overrightarrow{PX}\,\mathscr{M}(P + t\overrightarrow{PX})\overrightarrow{PX}} = 1.$$

The desired adapted mesh is then a *unit mesh*, i.e., a mesh such that for each edge $\mathbf{e} \in E_K$, $l_{\mathscr{M}}(\mathbf{e}) \approx 1$ and each element is regular tetrahedra. Since it is not possible to construct conforming meshes that cover space and exactly match the metric field, we consider a mesh with its edge lengths in the metric space that are between $[L_{\text{low}}, L_{\text{up}}]$, where $L_{\text{low}} = \sqrt{2}/2$ and $L_{\text{up}} = \sqrt{2}$, and the elements have acceptable quality in the metric space [7] as an acceptable mesh.

2.1.3 Element quality

During the mesh adaptation procedure, we control the edge length of the mesh as well as the mesh quality. It is possible to create elements which satisfy the length criterion but with a small volume, called *sliver elements*, which have a poor quality. To this end, to evaluate the quality of each element K of the mesh, the cubic of the mean ratio function in the metric space is used:

$$Q_{\mathscr{M}}(K) = \frac{15552 V_{\mathscr{M}}(K)^2}{\left(\sum_{i=1}^{6} l^2_{\mathscr{M}}(e_i)\right)^3},$$

where $V_{\mathscr{M}}(K) = \det(\mathscr{M})^{1/2} V(K)$ is the volume of K in the metric and $l_{\mathscr{M}}(e_i)$ is the length in the metric of the ith edge of the element. The value $Q_{\mathscr{M}}$ has been normalized to the interval [0, 1] with 0 for the worst quality (a zero volume tetrahedra) and 1 for the best quality (a regular tetrahedra in the metric space).

2.1.4 Efficiency index

To quantify how well a mesh edge satisfies the mesh metric, a metric satisfaction measure is used. For an edge e_i, it is defined as:

$$Q_l(e_i) = \begin{cases} l_{\mathscr{M}}(e_i) & \text{if } l_{\mathscr{M}}(e_i) \leq 1 \\ 1/l_{\mathscr{M}}(e_i) & \text{otherwise.} \end{cases}$$

To quickly estimate the quality of the mesh $\mathscr{H}$ with respect to a mesh metric field, the mesh efficiency index is introduced as Ref. [17]:

$$\tau_{\mathscr{H}} = \exp \frac{1}{\text{ne}} \left(\sum_{i=1}^{\text{ne}} (Q_l(e_i) - 1) \right)$$

where ne is the number of edges in the mesh. The efficiency index varies between 0 and 1, 1 being for a mesh with all its edges of unit length in the metric.

2.2 Mesh modification procedure

Given a geometric domain, a current mesh and a desired mesh metric field defined over that mesh, a series of controlled mesh modification steps are applied to obtain a new mesh that satisfies the given mesh metric field [7].

The mesh modification algorithm is composed of three stages:

(i) Mesh coarsening to eliminate the initial short edges
(ii) Intelligent mesh refinement that includes operations for ensuring the proper geometric approximation of the mesh to the geometric domain and proper shape to match the metric field, and
(iii) Mesh optimization to improve the quality of the final mesh

2.2.1 Coarsening stage

The first stage eliminates the edges in the initial mesh that are shorter than L_{low} with coarsening operations.

The local mesh operations used for this process include edge collapse, compound operators that are the combination of swaps, collapses, and vertex relocation. The algorithm operates by identifying all the short edges to be eliminated and then removes them, one at a time, trying the possible modifications in an efficient order. If the coarsening operation would yield an invalid mesh (e.g., collapsing an edge with vertices classified on two different model faces), coasening is not allowed. This is acceptable since the consequence is the mesh which is locally finer than it needs to be. An important consideration in this process where a substantial number of edges in a region are to be coarsened is to be sure those collapsed are well distributed. This is accomplished by being sure not to follow the collapse of one edge by collapsing an immediate neighbor [14].

At the beginning and the end of this stage, a quality improvement procedure that performs mainly the swap operation is applied to improve the mesh quality. The quality improvement procedure enhances the efficiency of the coarsening stage and produces a better result in term of the mesh quality.

2.2.2 *Refinement stage*

The refinement stage reduces the maximal mesh edge length of the current mesh in the transformed space until it is less than L_{up}. More precisely, the length reduction in the transformed space is performed through multiple iterations using the full set of edge-based refinement templates [14]. To achieve intelligent refinement patterns, a refinement criterion, refining a set of long mesh edges in the transformed space in each iteration, is adopted and the maximal transformed length of the mesh is reduced in an incremental manner [7]. At each iteration, the procedure refines the edges with length l of value max $(\alpha L_{\max}, L_{\text{up}}) \leq l \leq L_{\max}$, where $0 \leq \alpha \leq 1$ (usually $\alpha \geq \sqrt{2}/2$), and $L_{\max}$ denotes the largest edge of the current mesh in the metric. The process repeats until $L_{\max} \leq L_{\text{up}}$.

Since the refinement process can create new mesh vertices that are classified on curved boundary model entities, it is necessary to move those vertices onto an appropriate location on the boundary. In general, this process can cause connected elements to become invalid. In those cases, a specific process that includes mesh modification and possibly a local cavity triangulation is applied [13]. Nevertheless, the process of placing the vertices on the appropriate curved boundaries will change the length of mesh edges. Then, appropriate mesh modifications are applied to satisfy the mesh metric.

The edge-based refinement templates can introduce new edges that are locally shorter than required by the mesh metric field, thus producing meshes that are finer than needed and providing an undesirable edge length distribution. Furthermore, if the mesh quality is not taken into account during the refinement procedure, poor quality elements can also be created. It is difficult to remove such small edges and poorly shaped elements after the refinement stage, thus it is crucial to treat such feature during this process. When they would be created. To improve the efficiency of this process while still allowing the use of efficient refinement templates, coarsening is performed at each iteration of the stage to remove the small elements (edges shorter than L_{low}) created during the iteration.

This stage also optimizes the mesh to improve the quality of the mesh and the efficiency of the coarsening by applying swap operations using the shape correction algorithm described in Ref. [7], which determines the mesh modification operations most likely to be successful in correcting the poorly shaped element through consideration of the configuration of the unacceptably shaped elements. By projecting one vertex of a tetrahedra on the plane defined by the other three, it is possible to identify the mesh entity most appropriate for elimination as well as the most appropriate operations to eliminate that entity, thus leading to the suppression of the poorly shaped element.

It has been observed that the result obtained shows that the control of the mesh quality and the suppression of the short edges during the refinement stage (at each iteration) improve the efficiency of the processor considerably, and have a substantial impact on the final mesh quality as well as edge length distribution in the metric.

2.2.3 *Optimization stage*

The last step performs optimization procedure to improve the quality over the entire mesh using swap operation and vertex relocation.

3 Parallelization

This section describes the parallelization of the mesh modification procedures on parallel computers. To technically describe mesh entities and the adjacency between them, the following are nomenclature used throughout this section [18]:$\{M\{M^d\}\}$: a set of topological entities of dimension d in meshM_i^d: the ith mesh entity of dimension d. $d=0$ for a vertex, $d=1$ for an edge, $d=2$ for a face, and$d=3$ for a region.$\{M_i^d\{M^q\}\}$: a set of mesh entities of dimension q that areadjacent to M_i^d.

3.1 Distributed mesh data representation

The execution of parallel mesh adaptation is supported by meshes distributed to partitions on the parallel computers. A partition P_i consists of a set of mesh entities assigned to the ith processor. When a mesh is

distributed into partitions, a set of mesh entities that form partition boundaries common to multiple partitions must be duplicated on multiple partitions. The Flexible distributed Mesh DataBase (FMDB) [15, 16] has been used to support the needed mesh-based parallel operations on distributed meshes including communications between mesh partitions, migrating mesh entities between the partitions and the dynamic mesh load balancing.

For the mesh entities duplicated on partition boundaries, their *remote copies* (the memory location of duplicated copies on non-self partitions, called *remote partitions*) are maintained to provide inter-partition communication links. Figure 1 illustrates a mesh that is distributed on three partitions. Vertex M_1^0 is common to the three partitions and a copy exists on each partition. Several mesh edges like M_j^1 are common to two partitions. The dashed lines are *partition boundaries* that consist of mesh vertices and edges duplicated on multiple partitions. Each partition is treated as a serial mesh on a processor with the entities on the partition boundaries where the copies on each partition and the links between the copies are maintained.

Parallelizing the mesh modification procedures simply requires support of the *check*/*set*/*retrieve* operators, listed below, regardless of the specific details of distributed mesh data representations as discussed in Ref. [15]:

- *Check* if an entity (vertex/edge/face) is on partition boundary.
- *Set* an entity onto interior or partition boundary.
- For any entity on partition boundary, *retrieve*/*set* its remote copies.

3.2 Parallelizing mesh modification procedures

The key technical issues to be considered in parallelizing mesh modification procedures are: (a) evaluating (predicting the validity or quality of the result mesh) and executing mesh modifications on or near by partition boundaries, (b) effective message packing, and (c) the dynamic mesh load balancing.

Applying mesh modification operations on partition boundary requires inter-partition communications for its evaluation and may break inter-partition links during execution. Therefore, a technique is required to prevent the operation from breaking links, or to repair the broken links following an operation. Since the inter-partition communications are costly for distributed memory systems, it is important to reduce the number of such communications by packing small messages from many mesh modifications together into larger messages. Message passing interface (MPI) [20] and parallel message packing library, the Autopack [21], are used for efficient parallel communications between processors.

The mesh load balancing is also critical for the parallel procedure to achieve desirable performance. The Zoltan library [22] is used to make partition assignment during the dynamic load balancing.

This section discusses the parallel mesh coarsening (Sect. 3.2.1), the parallel mesh refinement and projecting new boundary vertices onto model boundaries (Sect. 3.2.2), and the dynamic load balancing (Sect. 3.2.3). Parallel shape correction procedure is not discussed since it uses the same parallel techniques as mesh coarsening. The overall approach used for parallelizing the mesh modifications is consistent with those described in Ref. [14].

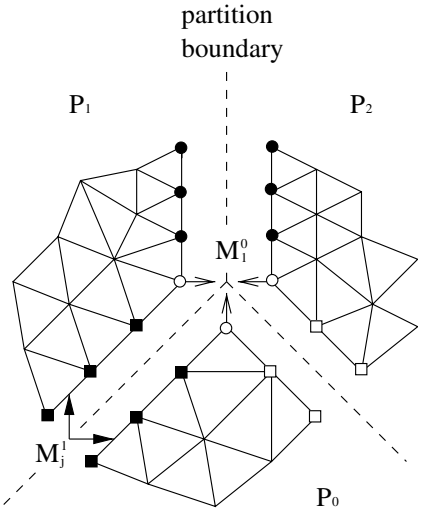

Fig. 1 2D example of distributed mesh data paradigm [19]

3.2.1 Coarsening

Mesh coarsening relies on repeated evaluation and execution of a few local mesh modification operators that change both mesh topology and geometry. Since the direct evaluation or execution of these operators on partition boundaries is complicated as well as inefficient, a mesh migration-based parallelization approach has been used. More precisely, whenever a mesh modification operation on partition boundaries is desired, we migrate the mesh entities within the polyhedron associated with (i.e., the mesh regions affected by) the operation into one partition and since then we apply the operation as in serial.

Table 1 lists the local mesh modifications used in mesh coarsening and the definition of their associated polyhedron. In general, the associated polyhedron are fully determined by the type and the key mesh entity of the mesh modification operation.

Table 1 Mesh operator and its associated polyhedron

Type	Key mesh entity	Associated polyhedron
Edge collapse	Edge M_i^1	$\{M_i^1\{M^0\}\{M^3\}\}$
Edge swap	Edge M_j^1	$\{M_j^1\{M^3\}\}$
Face swap	Face M_k^2	$\{M_k^2\{M^3\}\}$
Vertex repositioning	Vertex M_l^0	$\{M_l^0\{M^3\}\}$

Fig. 2 Pseudo code for the parallel mesh coarsening

```
1   initialize a local buffer on each partition
2   for each mesh vertex in a given dynamic list on each partition
3     determine a desired mesh modification operation to eliminate the current vertex
4     if the polyhedron of the desired operation is fully on a local partition
5       apply the operation, update the dynamic list and tag/untag vertices
6     else
7       put the desired mesh operation into the local buffer
8     end if
9   end for
10  determine all requests to migrate mesh regions in terms of desired operations in local buffers
11  perform mesh migration and update each local dynamic list
12  repeat 1 − 11 until vertices in all dynamic lists are tagged or all dynamic lists are empty
```

Figure 2 presents the pseudo code to perform parallel mesh coarsening. Given a dynamic vertex list to be coarsened on each partition, it is repeatedly traversed until the list becomes empty or all vertices in the list are tagged as one that cannot be moved (line 12). To be able to reduce the number of total mesh migrations, prior to each traversal, a buffer is initialized on each partition to book keep desired mesh modifications on partition boundaries so that requests for migrating mesh regions can be packed (line 1). During the traversal (line 2–9), desired edge collapse, vertex reposition or compound operations are determined to eliminate the current vertex. If all affected polyhedron of a desired mesh operation are on one partition, the operation is applied as in serial with the local dynamic list updated and neighboring vertices tagged or untagged. Otherwise, the desired operation is inserted into the local buffer. When all partitions are done traversing their dynamic vertex list, requests to migrate mesh regions are made in terms of what in the local buffers to make all polyhedron associated with a desired operation be on a partition (line 10–11). These requests then drive the application of one global mesh migration operation, in the meanwhile, update the dynamic vertex list on each partition in terms of what to migrate.

In case where the requests to migrate mesh regions conflict, the poorest partition has priority, where the poorest partition is the partition that has the least number of partition objects[1] [15]. With this scheme, mesh load balance is kept during adaptive mesh control simulations since the local mesh migration procedure performed during mesh adaptation always migrates entities to poor partitions improving the overall performance of the parallel simulation.

Figure 3 illustrates above approach in a simple 2D setting with three partitions (P_0, P_1 and P_2) and two desired edge collapse operations (collapsing edge M_0^1 and edge M_1^1). Since the polyhedron associated with these operations cross partition boundaries as indicated by dots and circles, in the first traversal, both collapse operations cannot be evaluated and two requests are made to migrate mesh entities in each polyhedra into one partition. It can be seen that the two requests conflict since their polyhedron intersect. The right picture shows the distributed mesh after one mesh migration (the lowest P_i has priority), in which the polyhedron associated with collapsing edge M_0^1 becomes fully on partition P_0. Due to the conflict, one more mesh migration is needed to make collapsing M_1^1 applicable.

[1]The basic unit to assign the destination partition id in the mesh migration procedure which is any mesh entity not on the boundary of any higher order entities.

3.2.2 *Refinement and projecting new boundary vertices*

Mesh refinement relies on tagging mesh edges and applying edge, face and region subdivision templates, which changes mesh topology but not geometry. Parallelizing subdivision templates consists of three steps. First, marked mesh edges and their neighboring mesh faces on partition boundaries are subdivided using the four templates, given in Fig. 4, regardless of breaking inter-partition links between *child* mesh entities. In the meanwhile, ordered *child–parent* messages, as listed in Table 2, are stored in a corresponding local buffer for each subdivision so that broken links can be effectively set later. This step ensures consistent subdivision of the replicated faces across partition boundaries identically since replicated faces have their bounding edges and vertices in the same order and the four templates in Fig. 4 are not ambiguous. The second step performs one inter-partition communication that sends/receives all messages in local buffers, then repairs the broken inter-partition links between *child* mesh entities by matching local ordered *child–parent* relationship with what received. Finally, mesh regions are subdivided using exactly the same templates as in serial without any inter-partition communication. The pseudo code of this parallel refinement algorithm is given in Fig. 5.

Figure 6 demonstrates the above procedure in a 2D example. Figure 6a depicts the original distributed mesh, in which mesh edge M_0^1, M_1^1 and M_2^1 are marked for refinement where M_0^1 on P_0 is the remote copy of M_1^1 on P_1. Figure 6b shows the mesh after the first step, where new mesh entities M_1^0, M_3^1 and M_4^1 (resp. M_2^0, M_5^1 and M_6^1) are created on P_0 (resp. P_1) with message $\{P_1, M_1^1, M_1^0, M_3^1, M_4^1\}$ (resp. $\{P_0, M_0^1, M_2^0, M_5^1, M_6^1\}$) stored in local buffer. When P_1 receives the message, it compares the contents of the message with the ordered local *child–parent* relationship, the links from P_1 to P_0, *i.e.*, $M_5^1 \rightarrow$

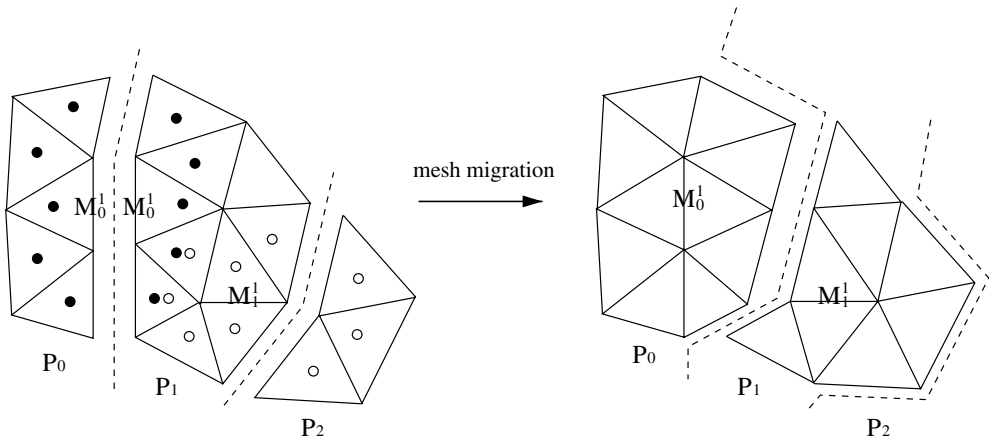

Fig. 3 2D example of parallel mesh coarsening based on mesh migrations. *Dashed lines* indicate partition boundaries. *Solid dots* indicate the polyhedron associated with collapsing edge M_0^1. *Circles* indicate the polyhedron for collapsing edge M_1^1

M_3^1, $M_6^1 \rightarrow M_4^1$ and $M_2^0 \rightarrow M_1^0$. Similar are the links from P_0 to P_1. Figure 6c shows the distributed mesh where the broken links have been repaired and all subdivisions are completed.

Once refinement is completed, each partition holds a list of mesh vertices that are classified onto curved geometries and need to be projected onto the model boundaries. The parallel vertex snapping algorithm uses two methods to snap a vertex on partition boundaries. For the majority of the vertices, repositioning is directly applied on partition boundaries after it is ensured that on all partitions no mesh region will be inverted or flattened. For the remaining vertices, mesh modifications or local cavity remeshing techniques used in the parallel mesh coarsening are applied to maintain a valid mesh. Figure 7 gives the pseudo code for parallel vertex snapping algorithm that properly combines these two techniques. First, for each vertex to be snapped, referred to as M_i^0 in Fig. 7, the algorithm distinguishes three situations:

- Evaluate the direct repositioning on local partition if M_i^0 is on partition boundaries. Insert a message into local buffer B_0 if the repositioning will invert or flatten any local mesh region so that its remote copy (or copies) can be informed.
- Snap as many vertices not on partition boundaries as possible by repositioning or local mesh modifications.
- If M_i^0 cannot be snapped in the second situation but possibly can be snapped by modifying a partition boundary mesh entity M_j^k (i.e., M_i^0 is near partition boundary), insert a request into local buffer B_1 to migrate mesh regions in $\{\{M_i^0\ \{M^3\}\},\ \{M_j^k\{M^3\}\}\}$ onto one partition.

The algorithm performs one round of communication that sends/receives messages in buffer B_0, and loops over partition boundary vertices to be snapped on each partition again. For the vertex that does not receive any message (i.e., not inverting or flattening any mesh regions on all partitions), snap the vertex. Otherwise, insert a request into local buffer B_1 to migrate all regions adjacent to the vertex onto one partition. Finally, the algorithm performs one mesh migration in terms of requests in local buffer B_1. Above processes repeat until the list of vertices to be snapped on all partitions becomes empty.

3.2.3 *Dynamic load balancing*

Having adapted distributed meshes, the partitions will become imbalanced, in which case the dynamic load balancing phase needs to be applied to maximize utilization of all partitions by minimizing idling of partitions due to load imbalance and inter-partition communication.

The Zoltan library [22] is a collection of data management services for parallel, unstructured, adaptive, and dynamic applications. It includes a suite of

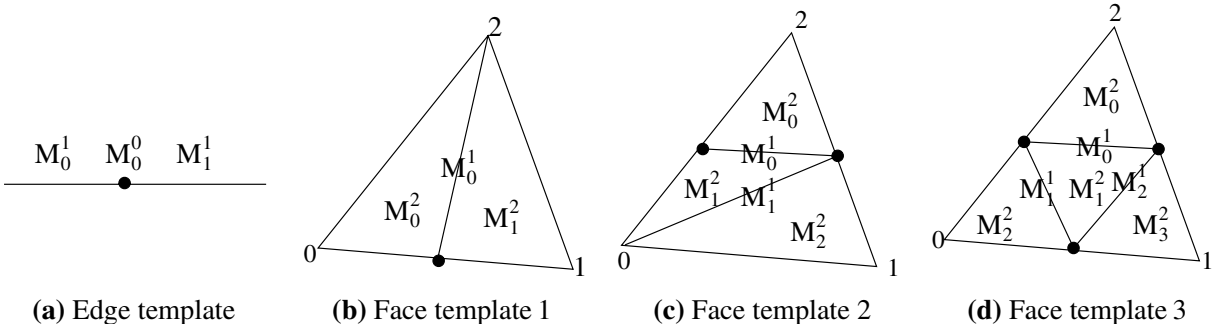

Fig. 4 Subdivision templates on partition boundaries

Table 2 Packed messages in distributed element subdivisions where pair(i) denotes the ith remote *pid* and remote copy pair of the parent mesh entity and N the number of remote copies of the parent mesh entity

Subdivision template	Messages to be sent/received
Edge template	$\{\text{pair}(i),\ M_0^0,\ M_0^1,\ M_1^1\}$ $(i=1,2,...,N)$
Face template 1	$\{\text{pair}(i),\ M_0^1,\ M_0^2,\ M_1^2\}$ $(i=1,2,...,N)$
Face template 2	$\{\text{pair}(i),\ M_0^1,\ M_1^1,\ M_0^2,\ M_1^2,\ M_2^2\}$ $(i=1,2,...,N)$
Face template 3	$\{\text{pair}(i),\ M_0^1,\ M_1^1,\ M_2^1,\ M_0^2,\ M_1^2,\ M_2^2,\ M_3^2\}$ $(i=1,2,...,N)$

parallel algorithms for dynamically partitioning problems over sets of partitions. The load balance procedure interfaces with the Zoltan to obtain redistribution information of the mesh. The load balancing procedure computes the input to the Zoltan which is a representation of the distributed mesh, usually a weighted graph, then, Zoltan's re-partitioning algorithm returns a vector containing the redistribution information for the mesh entities that will restore the load balance.

The design of distributed meshes in the FMDB allows a straightforward algorithm to perform load balancing using the mesh migration procedure. With the redistribution information from Zoltan at hand, the dynamic re-partitioning step is completed by calling the mesh migration procedure that moves the appropriate entities from one partition to another during parallel mesh adaptation to maintain mesh load balance. Figure 8 illustrates an example of 2D mesh load balancing. In the left, all mesh faces are tagged with their destination partition ids. The final balanced mesh is given in the right. Reference [15] provides more discussions on the mesh load balancing procedure.

4 Analytical metric fields on a curved geometry

In this section, the mesh adaptation procedure is applied on analytical examples in serial and on an analytical example in parallel. As the mesh size fields are analytically defined, the metric at a new vertex location is defined *via* the analytical function rather than interpolated.

4.1 Three analytical fields on a single processor

We present the results obtained by the mesh adaptation procedure on three analytical size fields for the same curved geometry representing a torus with four cylindrical holes (bounded by $[-1.5, 1.5]\times[-1.5, 1.5]\times[-0.5, 0.5]$) to illustrate that: (a) the geometry is preserved during the mesh adaptation process, (b) the given size field is enforced, and (c) the mesh quality is controlled.

The three analytical size fields considered represent a planar shock, a cylindrical shock and two spherical

```
1   initialize a buffer on each partition for message packing
2   for each edge (or face) to be subdivided on partition boundaries
3     subdivide it using the unambiguous templates depicted in Figure 4
4     for each remote copy of the current parent edge (or face)
5       pack its current copy with its child mesh entities (see Table 2)
6       insert the piece of packed message into the local buffer
7     end for
8   end for
9   send out messages in local buffers
10  repair the broken links between child mesh entities in terms of what received
11  subdivide mesh regions as in serial
```

Fig. 5 Pseudo code for the parallel mesh refinement

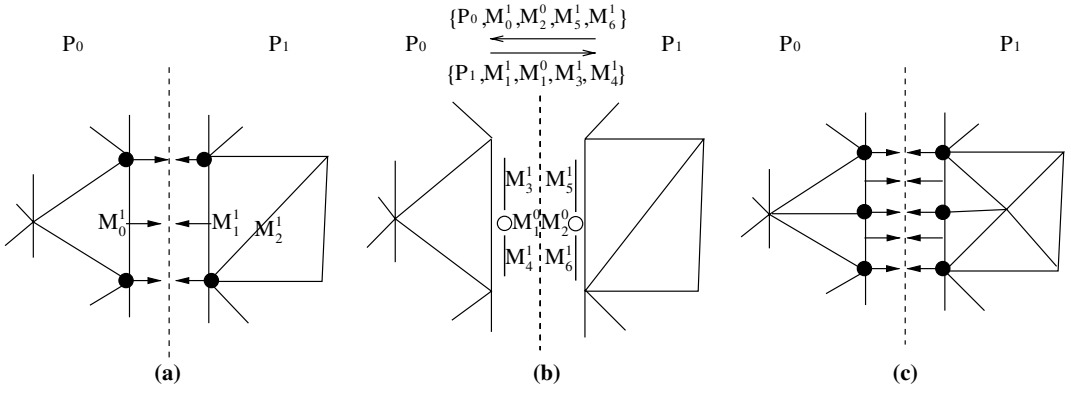

Fig. 6 2D example of distributed element subdivision operation

Fig. 7 Pseudo code for the parallel vertex snapping algorithm

```
1   initialize buffer, B0 and B1, on each partition for message packing
2   loop over the vertex list to be snapped on each partition
3     let M_i^0 be the current to be snapped
4     if M_i^0 is on a partition boundary
5       if repositioning M_i^0 will invert/flatten any mesh region on current partition
6         insert M_i^0's remote partition(s) and remote copy address(es) into buffer B0
7       end if
8     else
9       snap M_i^0 as in serial
10      if modifying partition boundary mesh entity M_j^k is desired to snap M_i^0
11        put the request to migrate {{M_i^0{M^3}}, {M_j^k{M^3}}} together in buffer B1
12      end if
13    end if
14  end loop
15  perform one communication in terms of messages in buffer B0
16  for each partition boundary vertex, referred to as M_j^0
17    if it does not receive any message (i.e., not invert/flatten a mesh region in all partitions)
18      snap M_j^0 and remove it from the vertex list
19    else
20      put the request to migrate {M_i^0{M^3}} together into buffer B1
21    end if
22  end for
23  perform a mesh migration in terms of requests in B1
24  update the vertex lists on each partition in terms of vertices migrated
25  repeat 1 – 24 until the vertex lists to be snapped are empty in all partitions
```

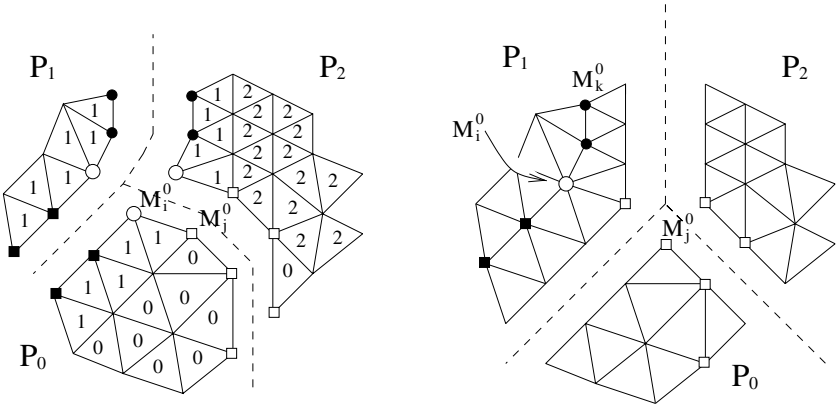

Fig. 8 Example of 2D mesh load balancing: (*left*) partition objects are tagged with their destination pids (*right*) mesh after load balancing [15]

shocks. These size fields are more precisely described in Appendix. In each case, the prescribed size field has almost an 100 anisotropy aspect ratio. The initial coarse mesh is shown in Fig. 9 (top), which consists of 4,351 vertices, 2,954 triangles, and 20,067 tetrahedron. Table 3 gives the statistics of final adapted meshes in terms of the number of entities and the total number of iterations performed to reach convergence in each case.

Table 3 Statistics of the final adapted meshes for the three analytical size fields

Case	Number of vertices	Number of tetrahedron	Run time (s)	Number of iterations
Planar	5,904	26,901	105 (99)	22
Cylindrical	16,759	84,678	330 (307)	28
Two spherical	25,849	137,857	561 (516)	35

Table 4 provides the mesh quality in size and shape for each final adapted mesh. In each case, the mesh adaptation procedure matches well to the size field (even if the mesh is not a unit mesh) and a good efficiency index is obtained. For all meshes, the quality in shape is also satisfactory. It has been observed that for the two spherical shock example, 78.53% of the elements have a quality better than 0.037. These adapted meshes are shown in Fig. 9. Figure 10 shows that the anisotropy of the size field has been well captured by the adaptation procedure.

4.2 A parallel analytical example

The parallel mesh adaptation procedure is applied on an analytical mesh size field representing two spherical

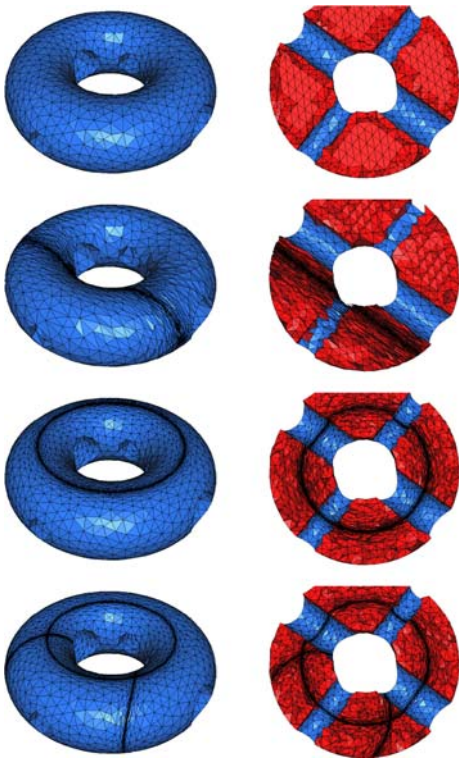

Fig. 9 From the *top to the bottom*, the initial mesh and the final adapted meshes for the planar shock, the cylindrical shock and the two spherical shocks

shocks (assume to be harder to solve than the previous one, see Appendix). The geometry and the initial mesh used in Sect. 4.1 are used. The mesh adaptations have been carried out on an IBM x335 cluster.

Table 5 gives the statistics of the final adapted mesh, and the compliance with the size field with 1–32 processors. In the example, the speedup[2] of the parallel procedure is between 1.2 and 2, and the mesh size field is scaled as the number of processors is increased.

is a purely hydro dynamical test and involves strong shocks. The Euler equations in their non-dimensional conservative form read:

$$\begin{cases} \frac{\partial \rho}{\partial t} + \nabla.(\rho \mathbf{U}) = 0 \\ \frac{\partial (\rho \mathbf{U})}{\partial t} + \nabla.(\rho \mathbf{U} \otimes \mathbf{U}) + \nabla p = 0 \\ \frac{\partial (\rho E)}{\partial t} + \nabla.((\rho E + p)\mathbf{U}) = 0, \end{cases}$$

where ρ, $\mathbf{U}$, E, p denote, respectively, the density, the velocity vector, the total energy, the pressure.

Flow solver The Euler system is solved by means of a flow solver based on the discontinuous Galerkin finite elements method [23]. The functional discretization is based on a L^2-orthogonal basis that yields to a diagonal mass matrix, thus the explicit time integration scheme does not necessitate any lumping or inversion of the mass matrix. The solution is approximated at order 1 on each element and an exact Riemann solver is used to compute the flux at the boundary of each element. A barth slope limiter [24] is employed and coupled with a shock detection algorithm [25] to be applied only in shock regions.

Metric construction based on an a posteriori error estimate The anisotropic a posteriori error estimate used in these examples is based on the control of the interpolation error. The aim is to generate an adapted mesh such that the interpolation error is equidistributed in all directions. For all mesh elements K, we have the following anisotropic error interpolation bound [4] based on the second derivatives of the variable u:

$$\|u - \Pi_h u\|_{\infty,K} \leq c_d \max_{x \in K} \max_{e \in E_K} \langle \mathbf{e}, |H_u(x)|\mathbf{e} \rangle = \varepsilon_K, \qquad (1)$$

where c_d is a constant depending on the dimension, E_K is

Table 4 Mesh quality in size and shape of the final adapted meshes for the three analytical size fields

Case	$\sqrt{2}/2 \leq l_{\mathcal{M}} \leq \sqrt{2}$ (%)	Efficiency	$0.125 < Q < 1$ (%)	Q_{worst}	Q_{average}
Planar	91.64	0.8396	96.04	$4.2e^{-5}$	0.347
Cylindrical	87.67	0.836	72.17	$1.8e^{-5}$	0.167
Two spherical	82.29	0.8214	43.22	$1.3e^{-5}$	0.069

5 A double Sedov explosion simulation

A parallel mesh adaptation procedure is applied to a double sedov explosion, modeled by the Euler equations, in a 3D geometry that simulates the evolution of two spherical blasts in a homogeneous medium. This problem

[2]The scalability of a parallel program running on p processors against the program on one processor.

the set of the element edges of K and $|H_u| = \mathscr{R}|\Lambda|\mathscr{R}^{-1}$ is the absolute value of the Hessian of the variable u, where $\mathscr{R}$ is the matrix of eigenvectors and $|\Lambda| = \text{diag}\ (|\lambda_i|)$ is the absolute value of the matrix of eigenvalues.

Then, a discrete metric approximation that uses the mesh vertices as support is considered. Let us denote by $h_{\min}$ (resp. $h_{\max}$) the minimal (resp. maximal) mesh element size and ε the desired interpolation error. Then, according to the relation (Eq. 1), we define for each mesh vertex the anisotropic metric tensor as:

Table 5 Statistics related to the final adapted meshes for two spherical shock example with 1–32 processors

Number of processors	Number of vertices	Number of tetrahedron	$\sqrt{2}/2 \leq l \leq \sqrt{2}$
1	46,163	246,608	84.91
2	50,223	269,998	81.57%
4	49,886	267,906	82.00%
8	56,307	305,497	76.69%
16	63,825	350,376	71.88%
32	73,474	407,676	66.21%

$$\begin{aligned} \mathcal{M} &= \mathcal{R}\tilde{\Lambda}\mathcal{R}^{-1}, \text{ where} \\ \tilde{\Lambda} &= \text{diag}\,(\tilde{\lambda}_i), \\ \tilde{\lambda}_i &= \min\left(\max\left(\frac{c_d|\lambda_i|}{\varepsilon}, \frac{1}{h_{\max}^2}\right), \frac{1}{h_{\min}^2}\right) \end{aligned} \tag{2}$$

Introducing a minimal (resp. maximal) element size is a way to avoid unrealistic (impractical) metrics. It also controls the explicit time integration scheme of the discontinuous Galerkin method.

Nevertheless, inside shock regions the Hessian of the solution is not defined. In these regions, detected by the shock detection algorithm, a *discontinuity-based* metric is used where we imposed $h_{\min}$ in the direction orthogonal to the shock and $h_{\max}$ in the other two main directions [10].

The Sedov simulation A domain with a non-dimensional size of $[-1,1]\times[-1,1]\times[0,1]$ containing some curved obstacles is considered and filled with a homogeneous medium with the following initial conditions: $\rho_0 = 1$, $p_0 = 10^{-5}$ and $\mathbf{U_0} = (0,0,0)$.

To initialize the simulation, an energy e equal to 1 is applied onto two small hemisphere regions with a radius $\mathrm{dr} = 0.15$ and for centers $(-0.1, 0.2, 0)$, and $(0.1, 0.1, 0)$. Inside these regions, $\rho = 1$, $\mathbf{U} = (0, 0, 0)$, and the pressure is given by:

$$p = \frac{3(\gamma - 1)e}{(n+1)\pi \mathrm{dr}^n},$$

where $n = 3$ for a spherical explosion and $\gamma = 1.4$. The aim is to obtain the solution at non-dimensional time 0.1. The simulation executes 20 mesh adaptations and used $h_{\min} = 0.005$ and $h_{\max} = 0.1$ as the adaptation parameters.

The shock wave propagation (pressure isosurface) is illustrated Fig. 11 at various time steps. Figure 12 shows the anisotropic adapted meshes and the corresponding isopressure distributions at time 0.06 and 0.1. Table 6 gives the statistics of these adapted meshes, the adequation with the size field and the quality in shape. Note that the adapted meshes respect the metric quite well. The final mesh has an efficiency index of 0.8355. The mesh quality is also good, i.e., the final mesh has 96.28% of its elements with a quality better than 0.037 and the worst element quality is $8.4e^{-5}$.

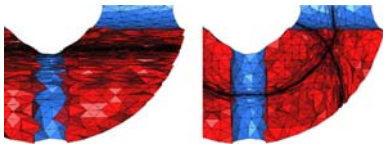

Fig. 10 Zoom on the final adapted meshes obtained for the planar shock (*left*) and the two spherical shocks (*right*)

The final mesh contains 638,677 tetrahedron that represent 2,554,708 degrees of freedom for each variable (thus, a total of 12,773,540 degrees of freedom). The simulation has been carried out on an IBM x335 cluster, each 2 GHz processor having 2 GB of memory, with 24 processors.

6 Closing remarks

In this paper, a parallel anisotropic mesh adaptation procedure by local mesh modification has been presented. Improvements to the mesh modification procedure have been introduced to improve the quality in shape and the compliance with the mesh size field. Parallelization of the mesh modification allows use of the procedure in large scale 3D simulations.

This approach has been applied in parallel on 3D analytical examples and a CFD simulation. We have

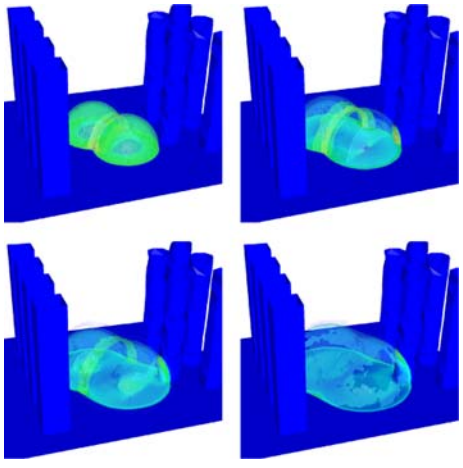

Fig. 11 Isosurface of pressure evolution in the domain at time 0.04, 0.06, 0.08 and 0.1 (from *left to right* and *top to bottom*)

Table 6 Statistics related to adapted meshes with the corresponding mesh quality in size and shape at various time steps

Time step	Number of vertices	Number of tetrahedron	$\sqrt{2}/2 \leq l \leq \sqrt{2}$ (%)	$0.125 < Q < 1$ (%)
0.06	83,898	475,314	87.92	67.86
0.1	112,713	638,677	87.98	67.76

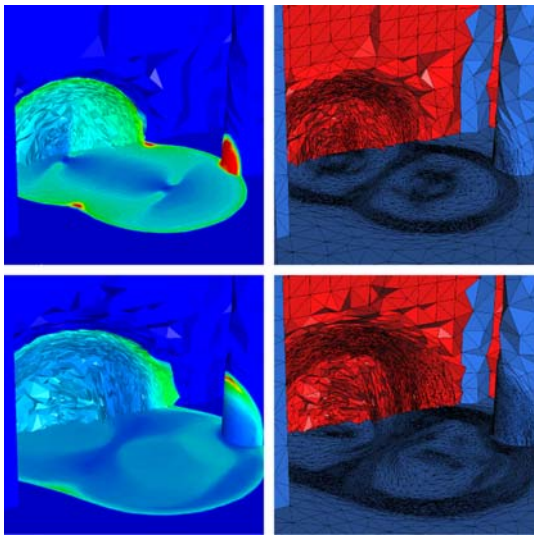

Fig. 12 Isopressure distribution (*left*) at time 0.06 and 0.1 and the associated anisotropic adapted meshes (*right*)

demonstrated that the new proposed algorithm allows to obtain adapted meshes with a good quality in shape and size. Nevertheless, the compliance of the adaptation procedure with the size field depends a lot on the mesh partitioning. Keeping one component for each mesh partition helps to obtain a good quality mesh.

7 Appendix: Analytical size field definition

In this section, the four analytical size fields used in Sect. 4 are described.

A planar shock We consider the following analytical size

$$h_1 = h_{\max}|1 - e^{-|x-0.5|}| + 0.003$$

with $h_{\max} = 0.2$ and the analytical metric is given by:

$$\mathscr{M} = \mathscr{R}\Lambda\mathscr{R}^{-1}, \quad \text{with } \Lambda = \begin{pmatrix} h_1^{-2} & 0 & 0 \\ 0 & h_{\max}^{-2} & 0 \\ 0 & 0 & h_{\max}^{-2} \end{pmatrix}$$
$$\text{and } \mathscr{R} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

A cylindrical shock The analytical size field representing a cylindrical shock of radius $R = 1$ and axis $z = 0$ is defined by:

$$h_1 = h_{\max}|1 - e^{-2|x^2+y^2-R^2|}| + 0.0015$$

with $h_{\max} = 0.14$ and the analytical metric is given by:

$$\mathscr{M} = \mathscr{R}\Lambda\mathscr{R}^{-1}, \quad \text{with } \Lambda = \begin{pmatrix} h_1^{-2} & 0 & 0 \\ 0 & h_{max}^{-2} & 0 \\ 0 & 0 & h_{max}^{-2} \end{pmatrix}$$
$$\text{and } \mathscr{R} = \begin{pmatrix} x/r & -y/r & 0 \\ y/r & x/r & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

where $r = \sqrt{x^2 + y^2}$.

Two spherical shocks We consider an analytical size field representing two spherical shocks of radius $R = 1$ and centers (0, 0, 0) and (1, 0, 0), defined by:

$$h_1 = h_{\max}|1 - e^{-3|x^2+y^2+z^2-R^2|}| + 0.0015,$$
$$h_2 = h_{\max}|1 - e^{-3|(x-1)^2+y^2+z^2-R^2|}| + 0.0015$$

with $h_{\max} = 0.14$ and the analytical metric is given by:

$$\mathscr{M} = \mathscr{R}\Lambda\mathscr{R}^{-1}, \quad \text{with } \Lambda = \begin{pmatrix} h_1^{-2} & 0 & 0 \\ 0 & h_2^{-2} & 0 \\ 0 & 0 & h_{max}^{-2} \end{pmatrix}$$
$$\text{and } \mathscr{R} = \begin{pmatrix} \vdots & \vdots & \vdots \\ v_1 & v_2 & v_3 \\ \vdots & \vdots & \vdots \end{pmatrix},$$

where $v_1 = (x/r_1,\ y/r_1,\ z/r_1)$ with $r_1 = \sqrt{x^2 + y^2 + z^2}$, $v_2 = ((x-1)/r_2,\ y/r_2,\ z/r_2)$ with $r_2 = \sqrt{(x-1)^2 + y^2 + z^2}$ and $v_3 = v_1 \wedge v_2$.

In parallel, we define the same metric field with the following size:

$$h_1 = h_{\max}|1 - e^{-3|x^2+y^2+z^2-R^2|}| + 0.004,$$
$$h_2 = h_{\max}|1 - e^{-3|(x-1)^2+y^2+z^2-R^2|}| + 0.001$$

with $h_{\max} = 0.14$.

References

1. Baker TJ (1997) Mesh adaptation strategies for problems in fluid dynamics. Finite Elem Anal Des 25(3–4):243–273
2. Apel T, Grosman S, Jimack PK, Meyer A (2004) A new methodology for anisotropic mesh refinement based upon error gradients. Appl Numerical Math 50(3–4):329–341
3. Bottasso CL (2004) Anisotropic mesh adaptation by metric-driven optimization. Int J Numer Meth Eng 60(3):597–639
4. Frey PJ, Alauzet F (2004) Anisotropic mesh adaptation for CFD simulation. Comput Methods Appl Mech Eng (accepted)
5. George PL, Hecht F (1999) Non isotropic grids. In: Thompson J, Soni BK, Weatherill NP (eds) CRC Handbook of Grid Generation. CRC Press Inc., Boca Raton 20.1–20.29
6. Kunert G (2002) Toward anisotropic mesh construction and error estimation in the finite element method. Numerical Meth Partial Differential Equations 18:625–648
7. Li X, Shephard MS, Beall MW (2005) 3D anisotropic mesh adaptation using mesh modifications. Comput Methods Appl Mech Eng (in press)
8. Pain CC, Umpleby AP, de Oliveira CRE, Goddard AJH (2001) Tetrahedral mesh optimisation and adaptivity for steady-state

and transient finite element calculations. Comput Methods Appl Mech Eng 190:3771–3796
9. Peraire J, Peiro J, Morgan K (1992) Adaptive remeshing for three dimensional compressible flow computation. J Comp Phys 103:269–285
10. Remacle JF, Li X, Shephard MS, Flaherty JE (2003) Anisotropic adaptive simulation of transient flows using discontinuous Galerkin methods. Int J Num Meth Eng (in press)
11. Tchon KF, Khachan M, Guibault F, Camarero R (2005) Three-dimensional anisotropic geometric metrics based on local domain curvature and thickness. CAD Comput Aided Des 37(2):173–187
12. Yamakaw S, Shimada K (2000) High quality anisotropic mesh generation via ellipsoial bubble packing. 9^{th} International Meshing Roundtable
13. Li X, Shephard MS, Beall MW (2003) Accounting for curved domains in mesh adaptation. Int J Numer Meth Eng 58:247–276
14. de Cougny HL, Shephard MS (1999) Parallel refinement and coarsening of tetrahedral meshes. Int J Numer Meth Eng 46:1101–1125
15. Seol ES, Shephard MS (2005) Efficient distributed mesh data structure for parallel automated adaptive analysis. Eng Comput (submitted)
16. Seol ES, Shephard MS, Musser DR (2005) FMDB: Flexible distributed mesh database. http://www.scorec.rpi.edu/FMDB
17. Frey PJ, George PL (2000) Mesh generation. Application to finite elements. Hermès Science Publishing, Paris, Oxford
18. Beall MW, Shephard MS (1997) A general topology-based mesh data structure. Int J Numer Meth Eng 40:1573–1596
19. Remacle JF, Klaas O, Flaherty JE, Shephard MS (2002) A parallel algorithm oriented mesh database. Eng Comput 18:274–284
20. Gropp B et al (2005) The Message passing interface (MPI) standard library. Argonne National Laboratory http://www-unix.mcs.anl.gov/mpi
21. Loy R (200) Autopack user manual. Science Division, Argonne National Laboratory
22. Boman E, Devine KD, Hendrickson B et al (2001) Zoltan: a dynamic load-balancing library for parallel applications, Sandia National Labs, Zoltan's user guide v1.23. http://www.cs.sandia.gov/zoltan
23. Remacle JF, Flaherty JE, Shephard MS (2003) An adaptive discontinuous Galerkin technique with an othogonal basis applied to compressible flow problems. SIAM Rev 45:55–73
24. Barth TJ, Jespersen DC (1989) The design and application of upwind schemes on unstructured meshes. AIAA Paper 89-0366
25. Krivodonova L, Xin J, Remacle JF, Chevaugeon N, Flaherty JE (2004) Shock detection and limiting with discontinuous Galerkin methods for hyperbolic conservation laws. Appl Numer Math 48:323–338