
Combining Size-Preserving and Smoothing Procedures for Adaptive Quadrilateral Mesh Generation

E. Ruiz-Gironés¹, X. Roca², and J. Sarrate¹

¹ Laboratori de Càlcul Numèric (LaCàN),
Departament de Matemàtica Aplicada III (MA III),
Universitat Politècnica de Catalunya (UPC),
Campus Nord UPC, 08034 Barcelona, Spain
{[eloi.ruiz](mailto:eloi.ruiz@upc.edu), [jose.sarrate](mailto:jose.sarrate@upc.edu)}@upc.edu

² Aerospace Computational Design Laboratory,
Department of Aeronautics and Astronautics,
Massachusetts Institute of Technology, Cambridge, MA 02139, USA
xeviroca@mit.edu

Summary. It is well known that the variations of the element size have to be controlled in order to generate a high-quality mesh. Hence, several techniques have been developed to limit the gradient of the element size. However, the obtained discretizations do not always reproduce the prescribed size function. This is of the major importance for quadrilateral meshes, since they are much more constrained than triangular ones. To solve this issue, we first define a quantitative criterion to assess when an element reproduces the prescribed size function. Then, using this criterion, we develop the novel size-preserving technique to create a new size function that ensures a high-quality mesh where all the elements are of the correct size. Two direct applications are presented. First, the size-preserving approach allows to generate quadrilateral meshes that correctly preserve the prescribed element size, without coarsen or refine the mesh. Second, we show that we can reduce the number of iterations to converge an adaptive process, since we do not need additional iterations to generate a mesh that correctly reproduces the size function. A smoother is applied in order to further increase the element quality while preserving the element size. Only when combining the size-preserving technique and the smoother, a high-quality mesh that correctly reproduces the size function is obtained.

Keywords: Mesh size function, background mesh, adaptive process, quadrilateral mesh, gradient-limiting, smoothing.

1 Introduction

In several numerical simulations, such as adaptive processes [1, 2], it is of the major importance to generate a mesh that correctly preserves the prescribed element size. On the one hand, the element size is directly related to the error of the final results. On the other hand, the element size influences the computational cost to obtain the final solution. One of the most used techniques to prescribe an isotropic element size consists on assigning scalar values at the nodes of a background mesh and then interpolate these values over the whole domain. For instance, this technique is used in adaptive simulations, where starting with an initial mesh, a size function is deduced from the computed solution via an error estimate. Then, this mesh is used as a background mesh to generate a new spatial discretization.

The size function has to verify certain requirements in order to generate a high-quality mesh that reproduces the size function. Different mesh generation algorithms need different properties on the size function. For instance, triangular and tetrahedral algorithms [1, 2, 3] can easily follow the variation of the size function, since these kind of meshes can be coarsened or refined where needed without generating low-quality elements. However, this is not true for quadrilateral meshing algorithms [4, 5], where coarsen or refine the mesh is a difficult task which may lead to low-quality elements. For this reason, quadrilateral mesh generation algorithms benefit from size functions with *good* properties. Therefore, special effort has been focused on generating element size functions that facilitates the generation of the desired mesh, [6, 7, 8]. Current techniques, such as the gradient-limiting algorithms [9, 10, 11, 12], modify a given size function by limiting its gradient. Thus, ratio of neighboring element size is bounded and it is easier for any mesh generator to provide a high-quality mesh with smooth variation of the element size. However, the final mesh may still not correctly reproduce the initial size function. That is, the mesh could contain elements that are bigger than the requested size. Thus, it is still necessary to develop new techniques that, given a prescribed element size function, modify it in such a way that a mesh generation algorithm can provide a discretization that verifies the initial size function.

The main contribution of this work is to develop the novel concept of size-preserving size function. To this end, we first define a quantitative criterion to assess when an element reproduces a size function. Then, using this criterion, we compute a new size function by solving a non-linear and implicit equation. The new size-preserving size function ensures that all the elements are smaller or equal to the prescribed size function and, at the same time, its gradient is limited. In addition, we propose an implementation to solve the implicit, non-linear equation that states the size-preserving size function.

It is important to point out that we propose a non-intrusive technique to obtain a size field that can be realized as a mesh. The proposed technique modifies an initial size field that is represented as a background mesh. The

main goal of the modified size field is to ensure that the final mesh preserves the prescribed size. Note that the resulting size field can be used in any mesh generation process that queries the element size using a background mesh such as the mesh generation or the smoothing processes. That is, the proposed size-preserving technique is non-intrusive. To obtain the results of this work, we have used an existing mesh generator [5], and we only modified the code that corresponds to the initialization of the size function.

The new size-preserving size function has several advantages. For instance, quadrilateral mesh generators benefit from this new size function since these types of meshes are more difficult to refine or coarse. In addition, the new size function can reduce the number of iterations needed to converge an adaptive process, since we do not need additional iterations to properly reproduce the size field.

In order to improve the quality of the final mesh and still obtain a mesh that reproduces the size function, we have to apply a smoothing process that takes into account the element quality and the element size, see [13] for details. We will show that when this smoother is applied to a mesh obtained using the size-preserving method, we obtain a high-quality mesh that correctly reproduces the initial size function. However, this is not the case when applying the smoother to a mesh obtained using the gradient-limiting technique. The final mesh contains elements that do not correctly reproduce the size function. It is necessary to apply the proposed size-preserving technique combined with the smoothing process to obtain a mesh that correctly preserves the prescribed size function.

The outline of this paper is the following. In Section 2, we introduce the size-preserving size functions. In Section 3, we show how to compute a size-preserving size-function. In Section 4, we present the smoothing procedure. Finally, in Section 5, several examples are presented to illustrate the capabilities of the size-preserving size function.

2 Size-Preserving Size Function

In this section, we first motivate the use of the size-preserving size functions and introduce the basic definitions using a one-dimensional example. Then, we deduce the size-preserving method and analyze its properties.

2.1 Motivation

To illustrate the most common problems arising from the use of a size function in a mesh generation process, we consider the following one-dimensional size function defined in the $[0, 1]$ interval, see Figure 1:

$$h(x) = \min\{0.5, 0.1 + 3|x - 0.75|\}, \quad (1)$$

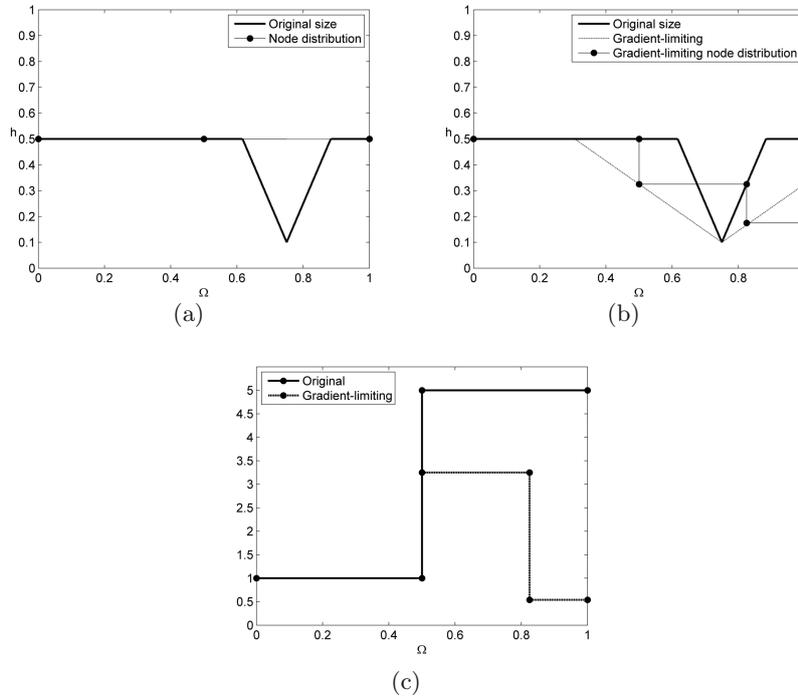


Fig. 1 Node distribution using: (a) the original size function, and (b) the gradient-limiting size function. (c) Ratio $R(e)$ for mesh computed using: the original size function (solid line) and the gradient-limiting size function (dashed line).

Function (1) is constant almost everywhere, with a valley at $x = 0.75$. The minimum size is 0.1 and the maximum size is 0.5. Figure 1(a) presents the node distribution generated using an advancing front method where the element size is obtained using the size function (1). Since the size prescribed at $x = 0$ is 0.5, only two elements are generated and the discretization does not preserve the prescribed element size. The main reason of this shortcoming is that the size function contains high gradients that the meshing algorithm cannot reproduce.

To obtain a better discretization, a gradient-limiting technique [9, 10, 11, 12] can be applied. Figure 1(b) presents the gradient-limiting size function obtained from (1), and the node distribution obtained using the same meshing algorithm. The maximum gradient of the new size function is imposed to be $\varepsilon = 1$. Note that this node distribution does not correctly reproduce the prescribed element size. Although the element size is smaller around the valley of the size function, the minimum element size is not captured in the final

mesh. To compare these discretizations we have to measure how accurately a mesh reproduces a prescribed size function. To this end, we introduce the following definitions.

Definition 1. A mesh \mathcal{M} reproduces a prescribed size function if it verifies

$$\mu(e) \leq \beta \min_{x \in e} h(x), \quad \forall e \in \mathcal{M}, \quad (2)$$

where $\mu(e)$ is the size of element e , and β is a scaling factor.

By introducing the ratio

$$R(e) = \frac{\mu(e)}{\beta \min_{x \in e} h(x)}, \quad (3)$$

a valid mesh has to verify

$$R(e) \leq 1 \quad \forall e \in \mathcal{M}. \quad (4)$$

Figure 1(c) plots function (3) for the discretizations generated using the original size function and the gradient-limiting size function with $\beta = 1$. Since $R(e) \geq 1$ for all the elements of the mesh, these meshes do not reproduce the size function.

Remark 1. In several applications it is required to preserve different size functions. For instance, it can be required to preserve a size field defined by the geometric features of the domain, and a size function defined by an error estimation. Specifically, given the size fields $h_i(x)$, $i = 1, \dots, n$, we can obtain a new size function as:

$$h(x) = \min_{i=1, \dots, n} h_i(x).$$

The resulting size function ensures that all the initial size functions are preserved. For this reason, in this work we only consider the modification of one size function. In the case where several size functions have to be addressed, we can consider the minimum of all them.

2.2 Size-Preserving Size Function

In order to generate a mesh that correctly preserves the prescribed size function, we introduce the new concept of *size-preserving size function*. Given the original size function, $h(x)$, we will deduce an alternative size function, called size-preserving size function and denoted by $h^*(x)$, such that it allows reproducing the size function according to Equation (2).

In fact, the new size function, $h^*(x)$, can be written in terms of the original one, $h(x)$. To this end, we consider the one-dimensional example presented in Figure 2(a). To obtain a mesh that correctly reproduces the size function, we

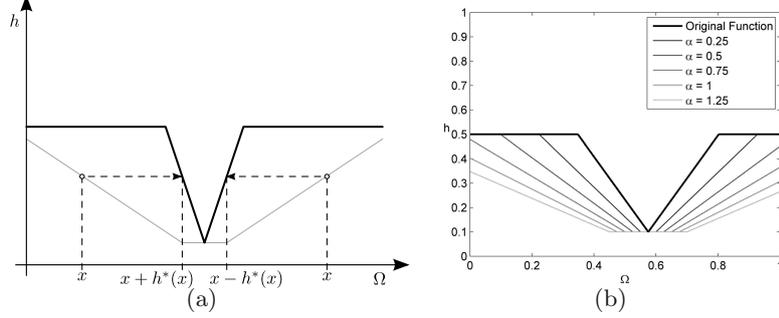


Fig. 2 (a) Original size function (thick black line) and size-preserving size function (grey line), and (b) one-dimensional size function defined in the $[0, 1]$ interval (thick black line) and several size-preserving size functions for different values of α (grey lines)

assume that the new element size around a point $x \in \Omega$ has to be $h^*(x)$. Then, the new node is created at position $x \pm h^*(x)$, depending on the advancing direction of the meshing algorithm, see Figure 2(a). For this reason, the size of the new element, e , is $\mu(e) = h^*(x)$. Taking into account condition (2), we deduce that the following equation has to be verified:

$$h^*(x) = \mu(e) \leq \beta \min_{y \in [x-h^*(x), x+h^*(x)]} h(y).$$

If we want $h^*(x)$ as big as possible to generate the minimum amount of elements, we have that

$$h^*(x) = \beta \min_{y \in [x-h^*(x), x+h^*(x)]} h(y).$$

To add more flexibility to our method, we include a parameter α that determines the trial interval (i.e. the interval in which the minimum of the original size function is computed):

$$h^*(x) = \beta \min_{y \in [x-\alpha h^*(x), x+\alpha h^*(x)]} h(y).$$

Note that taking $\alpha > 1$ we enlarge the trial interval. Thus, the size-preserving size function can achieve smaller values. On the contrary, taking $\alpha < 1$ we reduce the trial interval and the size-preserving size function can achieve larger values. The previous equation can be expressed in any dimension as:

$$h^*(\mathbf{x}) = \beta \min_{\mathbf{y} \in \mathcal{B}_{\alpha h^*(\mathbf{x})}(\mathbf{x})} h(\mathbf{y}), \quad (5)$$

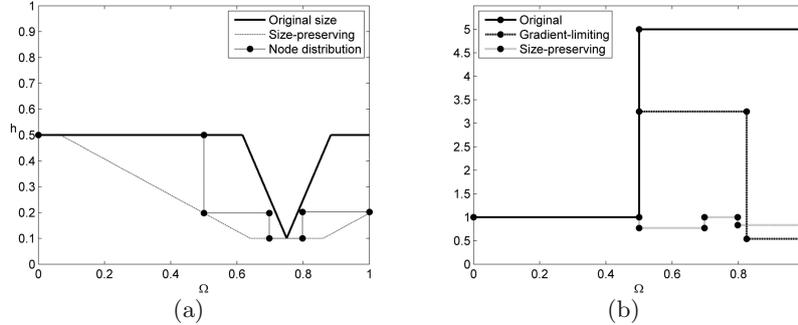


Fig. 3 (a) Node distribution using the size-preserving function. (b) Ratio $R(e)$ for mesh computed using: the original size function (solid line), the gradient-limiting size function (dashed line) and the size-preserving size function (dotted line).

where $B_r(\mathbf{x})$ is the set of points at distance at most r from \mathbf{x} . Note that β can be interpreted as a parameter that controls the maximum ratio $R(e)$ accepted in the final mesh. It is important to point out that Equation (5) is an implicit and non-linear definition of the size-preserving size function. Figure 2(b) plots several size-preserving functions for different values of parameter α . This parameter plays an important role in Equation (5). In fact, it controls:

- (i) *The measure of local minima in the element size function.* This ensures that small element sizes prescribed at local minima can be correctly reproduced. That is, an element can be held at each local minima.
- (ii) *The maximum gradient allowed.* This ensures that a high-quality mesh can be generated. Although we do not have a formal proof, experience has shown that the maximum gradient of the size-preserving size function is limited to β/α .

Although in Equation (5) parameters α and β are arbitrary, it is recommended to select $\beta \leq 1$ and $\alpha \geq \beta$ in order to obtain a mesh that satisfies Equation (2). In addition, we use $\beta = 1$ through the rest of the paper if it is not explicitly stated.

Figure 3(a) presents the original size function stated in Equation (1), the corresponding size-preserving size function, and the node distribution of the generated 1D mesh. The size-preserving size function is computed according to Equation (5) with $\alpha = 1$. Note that the region around the minimum is automatically enlarged in order to held an element of the requested size. In addition, the gradient of the size-preserving size function has been automatically limited.

Figure 3(b) presents the ratio $R(e)$, see Equation 3, for the meshes generated using: the original, the gradient-limiting and the size-preserving size

functions. Note that for the size-preserving size function the distribution of nodes correctly preserves the original size function.

3 Size-Preserving Size Function Generation

In this section, we present an algorithm to compute a size-preserving size function, $h^*(\mathbf{x})$, from an original size function, $h(\mathbf{x})$. This process is performed for each node of the background mesh. That is, at each node of the background mesh we compute $h^*(\mathbf{x})$ such that Equation (5) is verified.

Given a node, n_0 , of the background mesh, located at $\mathbf{x}_0 \in \Omega$, the main idea of the algorithm is to shrink a ball centered at \mathbf{x}_0 and, at the same time, compute the minimum value of $h(\mathbf{y})$ in the ball. The ball radius, r , is decreased until the following equation is verified:

$$r = \alpha \min_{\mathbf{y} \in B_r(\mathbf{x}_0)} h(\mathbf{y}). \quad (6)$$

Thus, by construction, the size-preserving size function is less or equal to the original one. To compute the nodes that belong to the ball, we store the nodes using a min-heap. A min-heap is a data structure such that its values are stored in ascending order. In this case, the nodes in the min-heap are sorted according to the distance to the center of the ball. Algorithm 1 details the calculation of $h^*(\mathbf{x})$ for a given node n_0 . In this algorithm, the distance between an arbitrary node n of the background mesh and node n_0 is computed as the distance along the edges. The distance to node n_0 is initialized to infinity (for instance, the maximum value for an object of type double).

First, some variables are initialized, Lines 2–6. The initial node, n_0 , located at distance 0 from the center of the ball, is inserted in the min-heap container. The radius of the ball is initialized as $r = \alpha h(\mathbf{x}_0)$. The main loop of the algorithm begins at this point. Each node, n , of the min-heap is removed from the container and then it is processed. We denote by d the distance between this node and node n_0 . In addition, we denote by \mathbf{x}_n the location of node n and we define an auxiliary variable $r' = \alpha h(\mathbf{x}_n)$ that stores the radius of a ball centered at \mathbf{x}_0 and computed according to the prescribed size at node n . Then, the algorithm updates the value of the ball radius, r , (and thus the value of the size-preserving size function at \mathbf{x}_0) according to the values of r , r' and d . Five cases are considered:

- (i) $r' < r$ and $d \leq r'$, Lines 14–16

In this case, the radius defined by the current node, r' , is less than the previous value, r . In addition, this node belongs to $B_{r'}(\mathbf{x}_0)$, since $d \leq r'$. For this reason, the value of r is updated to $r = r'$. Then, we update the distance of the adjacent nodes to node n according to Algorithm 2.

Algorithm 1. Computation of $h^*(x)$ **Ensure:** $h^*(x)$

```

1: function sizePreserving(BackgroundMesh  $\mathcal{M}$ , Node  $n_0$ , Real  $\alpha$ )
2:   NodeMinHeap  $\mathcal{N}$ 
3:   setDistance( $n_0, 0$ )
4:   insert( $n_0, \mathcal{N}$ )
5:   Real  $v_0 \leftarrow h(\mathbf{x}_0)$ 
6:   Real  $r \leftarrow \alpha \cdot v_0$ 
7:   while getSize( $\mathcal{N}$ ) > 0 do
8:     Node  $n \leftarrow$  firstNode( $\mathcal{N}$ )
9:     removeNode( $n, \mathcal{N}$ )
10:    Real  $d \leftarrow$  getDistance( $n$ )
11:    Real  $v \leftarrow h(\mathbf{x}_n)$ 
12:    Real  $r' \leftarrow \alpha \cdot v$ 
13:    if  $r' < r$  then
14:      if  $d \leq r'$  then
15:         $r \leftarrow r'$ 
16:        updateAdjacentNodesDistance( $n, r, \mathcal{N}$ )
17:      else if  $d \leq r$  then
18:         $r \leftarrow r \frac{r-d}{r-r'} + r' \frac{d-r'}{r-r'}$ 
19:      end if
20:    else if  $r' \geq r$  and  $d \leq r$  then
21:      updateAdjacentNodesDistance( $n, r, \mathcal{N}$ )
22:    end if
23:  end while
24:   $h^*(\mathbf{x}_0) \leftarrow \beta r / \alpha$ 
25: end function

```

(ii) $r' < r$ and $r' < d \leq r$, Lines 17–18

In this case, the radius defined by the current node, r' , is also less than the previous value, r . However, the node does not belong to $B_{r'}(\mathbf{x}_0)$ although it belongs to $B_r(\mathbf{x}_0)$. We update the value of the radius, r , according to Line 18. Since the node is outside of $B_{r'}(\mathbf{x}_0)$, we do not need to update the distance of the adjacent nodes.

(iii) $r' < r$ and $r < d$.

In this case, no actions have to be taken because the node is outside of the ball $B_r(\mathbf{x}_0)$.

(iv) $r' \geq r$ and $d \leq r$, Lines 20–21

The radius, r , is not updated, because $r' \geq r$. However, the node belongs to $B_r(\mathbf{x}_0)$, since $d \leq r$. For this reason, the distance of the adjacent nodes has to be updated.

(v) $r' \geq r$ and $r < d$.

In this case, no actions have to be taken, because the node is outside of the ball $B_r(\mathbf{x}_0)$.

Algorithm 2. Update the distance of adjacent nodes

```

1: function updateAdjacentNodesDistance(Node  $n$ , Real  $r$ , NodeMinHeap  $\mathcal{N}$ )
2:   Real  $d \leftarrow$  getDistance( $n$ )
3:   for all Edge  $e$  adjacent to  $n$  do
4:     Real  $l_e \leftarrow$  length( $e$ )
5:     Node  $n_e \leftarrow$  oppositeNode( $e, n$ )
6:     Real  $d_e \leftarrow$  getDistance( $n_e$ )
7:     Real  $d'_e \leftarrow d + l_e$ 
8:     if ( $d'_e < d_e$ ) and ( $d'_e < r$ ) then
9:       setDistance( $n_e, d'_e$ )
10:      updateHeap( $n_e, \mathcal{N}$ )     $\triangleright$  Since the distance of the node has changed
11:    end if
12:  end for
13: end function

```

When the min-heap is empty, the process is finished and the value the size-preserving size function is computed as $h^*(x) = \beta r / \alpha$.

Given a node, n , Algorithm 2 updates the distance from its adjacent nodes to the center of the ball $B_r(\mathbf{x}_0)$. Since this information is transmitted from the nodes with smaller values to the nodes with larger values, the node that holds the smaller value contains the correct value of the distance. Recall that this node, n , is the first node of the min-heap. The new distance, d'_e , of a node, n_e , adjacent to n through edge e is computed as:

$$d'_e = \min\{d_e, d + l_e\},$$

where d and d_e are the current computed distance of node n and n_e , respectively, and l_e is the length of edge e . If the new distance, d'_e is less than r , the current radius of the ball, node n_e is inserted in the min-heap with distance d'_e .

4 Smoothing Process

In order to improve the mesh quality, after generating the mesh, we have to apply a smoothing technique. However, the smoothing technique has to be aware of the prescribed size function to obtain a high-quality mesh that correctly reproduces the element size. To this end, we have applied the smoothing technique presented in [13]. The main idea of the smoothing process is to define a quality for each element that depends on the position of its nodes. Then, a minimization process is performed in which the optimal position of the nodes are computed. In [13], the authors propose an element quality that is a combination of the shape quality [14] and the target size for each element. The result is a quality function that improves the element quality and, at the same time, the prescribed element size is preserved.

Let S be the matrix that transform the ideal element, e_I , to the physical element, e_P . Then, the element quality function is:

$$\eta(e_P) = \eta_{sh}(e_P)\eta_{si}(e_P), \quad (7)$$

where $\eta_{sh}(e_P)$ and $\eta_{si}(e_P)$ correspond to the shape quality and the size quality of element e_P , respectively. These qualities are defined as

$$\eta_{sh}(e_P) = \frac{\|S\|^2}{2|\sigma|}, \quad \eta_{si}(e_P) = \frac{1}{\mu(\sigma)},$$

where $\|S\|$ and σ are the Frobenius norm and the determinant of matrix S , and $\mu(\sigma)$ is

$$\frac{e}{2} \left(\sigma e^{-\sigma} + \sigma^{-1} e^{-\sigma^{-1}} \right).$$

The η_{sh} function presents asymptotes when $\sigma = 0$. For this reason, in reference [15] modifies this size function in the following way:

$$\eta_{sh}^*(e_P) = \frac{\|S\|^2}{2|h(\sigma)|}, \quad h(\sigma) = \frac{1}{2}(\sigma + \sqrt{\sigma^2 + 4\delta^2})$$

where δ is a small parameter.

Using the element quality function (7), a continuous minimization problem is stated, and the optimum position of the nodes is computed. For more details on minimizing function (7), see reference [13].

5 Examples

This section presents two examples in order to illustrate the behavior of the size-preserving size function and compare it with the gradient-limiting method. The size-preserving size function has been successfully implemented in the ez4u meshing environment [16, 17, 18]. The first example shows the advantages of using the size-preserving size function in an adaptive process. The second example shows the mesh generated for a complex 2D size function defined using an MRI image. All the meshes have been generated using the quadrilateral algorithm presented in [5]. The smoothing process detailed in [13] has been applied in order to improve the quality of the mesh while preserving the prescribed size of the elements. In both examples we have used the shape quality to measure the quality of the mesh. Finally, we have used comparable parameters for the gradient-limiting and size-preserving approaches in order to better compare the resulting meshes. That is, we use $\varepsilon = 1/\alpha$.

Algorithm 3. Adaptive process

Ensure: Mesh \mathcal{M}

```

1: function AdaptiveProcess
2:   Mesh  $\mathcal{M} \leftarrow$  createUniformMesh
3:   BackgroundMesh  $bm \leftarrow$  estimateNewElementSize( $\mathcal{M}, sizeFunction$ )
4:   processBackgroundMesh( $bm$ ) ▷ grad-lim, size-pres.
5:   Boolean  $converged \leftarrow$  checkConvergence( $\mathcal{M}, bm$ )
6:   while not  $converged$  do
7:      $\mathcal{M} \leftarrow$  createNewMesh( $\mathcal{M}, bm$ )
8:     BackgroundMesh  $bm \leftarrow$  estimateNewElementSize( $\mathcal{M}, sizeFunction$ )
9:     processBackgroundMesh( $bm$ ) ▷ grad-lim, size-pres.
10:    Boolean  $converged \leftarrow$  checkConvergence( $\mathcal{M}, bm$ )
11:   end while
12: end function

```

5.1 Accelerating a Two Dimensional Adaptive Process

The objective of this example is to show that the proposed size-preserving approach can decrease the number of iterations to converge an adaptive process. To this end, we present two different executions of the adaptive process presented in Algorithm 3. We define, for all the executions of the adaptive process, the same analytical element size function:

$$h = \min \{0.25, |d - 0.2575| + 0.25 \cdot 10^{-3}\},$$

where $d = \sqrt{(x - 0.5)^2 + (y - 0.5)^2}$, and the spatial domain is defined as the $[0, 1] \times [0, 1]$ square.

First, we create a uniform mesh composed of 80 elements per side. From the analytical size function, we will compute a background mesh for each iteration of the adaptive process. The new background mesh is constructed from the mesh of the previous iteration. When the new background mesh is obtained, we generate the next mesh. This process is iterated until the mesh reproduces the analytical size function with a relative error below 0.1. That is, we accept all the elements whose size is, at most, 10% above the prescribed size of the analytical size function.

In the first execution, we process the size field using a gradient-limiting technique with parameter $\varepsilon = 0.5$. That is, the maximum gradient in the processed size function is 0.5. The process is not able to converge using 50 iterations. Figure 4(a) presents the generated mesh at the last iteration, while Figure 4(b) shows a detailed view. Figure 4(c) and 4(d) show the evolution of the ratio $R(e)$ computed against the background mesh of the current iteration and the analytic function, respectively. Note that the ratio $R(e)$ computed against the background mesh is always above 1.28 and, for this reason, the background mesh is not correctly captured. Thus, the adaptive process is not able to generate a mesh that correctly preserves the analytic size function.

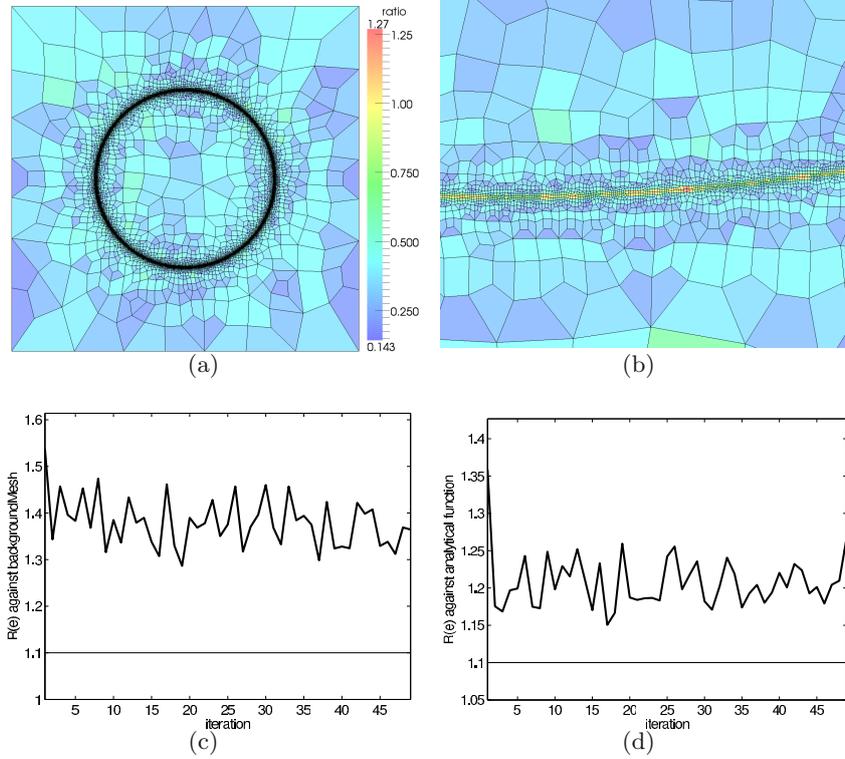


Fig. 4 Adaptive process, not converged in 50 iterations, using the gradient-limiting technique: (a) mesh at the last iteration; (b) detail of the mesh; (c) evolution of the ratio $R(e)$ against the current background mesh, and (d) evolution of the ratio $R(e)$ against the analytical function

In the second execution, we process the background mesh using the proposed size-preserving technique, $\alpha = 2$. In this case, the whole process has been converged using only four iterations. Figure 5(a) and Figure 5(b) show the mesh for the last iteration and a detailed view, respectively. Note that at each iteration, the background mesh is correctly captured, see Figure 5(c) and, for this reason, the process is able to generate a mesh that correctly preserves the analytical size function, see Figure 5(d).

Table 1 presents the statistics for the meshes of the adaptive process at the last iteration. The mesh generated using the size-preserving technique contains more elements than the mesh generated using the gradient-limiting method, since the size-preserving size function is smaller or equal than the gradient-limiting function. The percentage of correct faces using the gradient-limiting function is around 80%, while the percentage of correct faces using the proposed size-preserving approach is more than 99%. Note that the

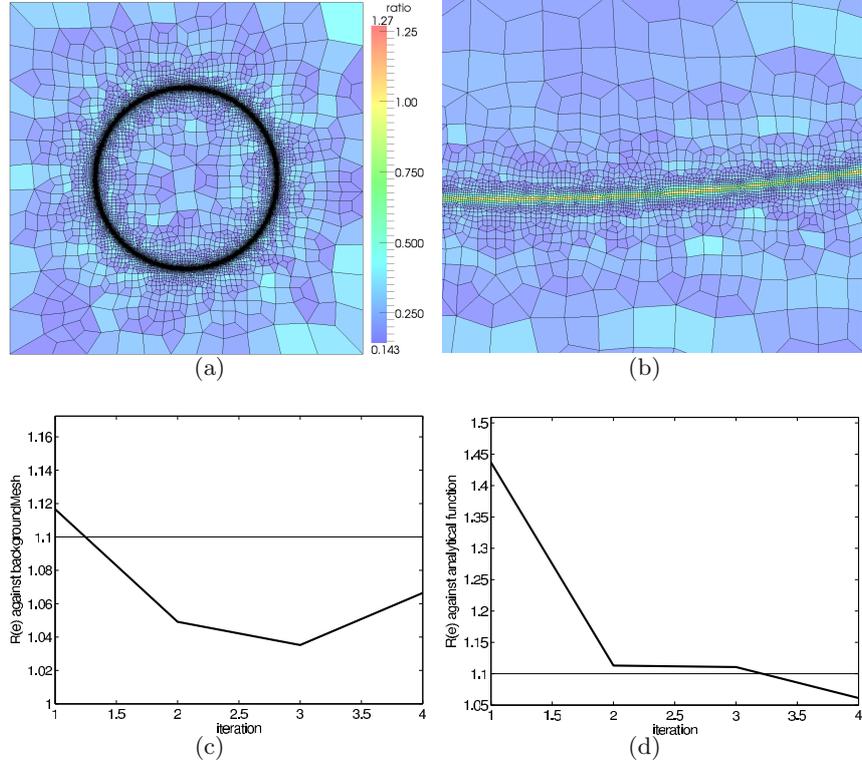


Fig. 5 Adaptive process, converged in 4 iterations, using the size-preserving technique: (a) mesh at the last iteration; (b) detail of the mesh; (c) evolution of the ratio $R(e)$ against the current background mesh, and (d) evolution of the ratio $R(e)$ against the analytical function

Table 1 Statistics for the meshes of the adaptive process at the last iteration

method	gradient-limiting	size-preserving
total faces	64845	111092
correct faces	51506	110847
correct faces (%)	79.42%	99.77%
max $R(e)$	1.268	1.046
min quality	0.333	0.356
max quality	0.999	0.999
mean quality	0.843	0.895
quality deviation	0.121	0.107

element quality of both meshes are similar, although the mesh obtained using the size-preserving approach presents better results. The minimum quality is higher using the size-preserving method and the maximum quality is equal

in both cases. The mean quality is higher and the deviation is smaller when the size-preserving method is used. That is, the smoothing process is able to maintain both the element quality and the element size only when the size-preserving method is used. In the case of the gradient-limiting method, the smoother process obtains a high-quality mesh, but the mesh does not reproduce the size function.

5.2 *Preserving a Complex Size Function in Quadrilateral Mesh Generation*

In this example, we present a quadrilateral mesh generated using a size field derived from a MRI image, courtesy of the Cardiac Atlas website and the Auckland MRI research group [19], see Figure 6(a). The size field is defined in terms of the mean curvature of the MRI field, in order to generate more elements where the variation of the gradient of the MRI field is higher, see Figure 6(b). With this background mesh, we have generated two meshes. In each mesh, we have computed the interpolation error of the initial MRI field on the corresponding mesh, and the ratio $R(e)$ for the elements. The first mesh is obtained with the gradient-limiting technique, $\varepsilon = 0.5$. Figure 7(a) shows the interpolation error of the MRI field on the mesh, and Figure 7(b) presents the ratio $R(e)$ for the elements. Note that there are elements that are more than 90% bigger than the requested size. The second mesh is obtained using the size-preserving approach, $\alpha = 2$. In this case, the results present a reduced interpolation error (Figure 8(a)), and ratio $R(e)$ (Figure 8(b)). The gradient-limiting technique takes 2.11 seconds to compute the new size function, while the size-preserving method takes 6.67 seconds. Although the size-preserving method is slower than the gradient-limiting technique, it is important to point out that the size-preserving method provides a mesh that correctly preserves the element size.

Table 2 summarizes the statistics for the meshes generated with the different size functions. The behavior of the size functions is similar to the ones reported in the previous example. The gradient-limiting technique, although fast is not able to reproduce the initial size function, obtaining a maximum ratio $R(e)$ around 1.90 and only 81% of correct faces. The mesh obtained with the size-preserving approach presents a maximum ratio equal to 1.08. That means that there are elements that are only 8% bigger than the prescribed size. In addition, the percentage of correct faces is around 98% percent. The mesh quality using the gradient-limiting technique is lower than the mesh quality using the size-preserving method. The minimum quality is lower when using the gradient-limiting method, and the maximum quality is the same for both meshes. The mean quality is higher in the mesh generated using the size-preserving method, and deviation is smaller. Only the mesh generated using the size-preserving method contains high-quality elements that correctly reproduce the initial size function.

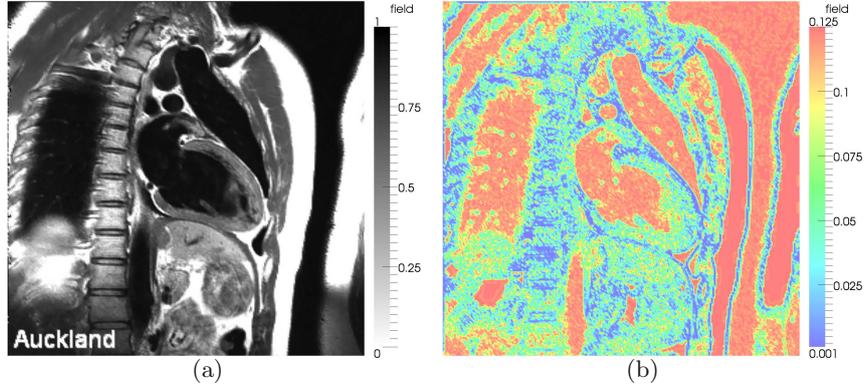


Fig. 6 (a) MRI field defined on a square and (b) its associated size function

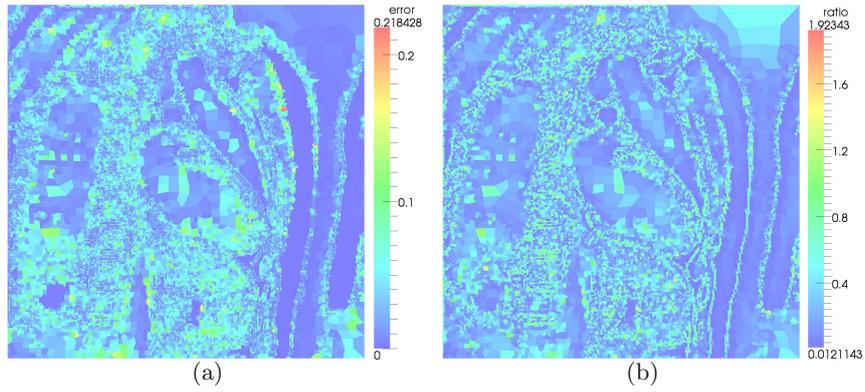


Fig. 7 Mesh generated using the gradient-limiting technique: (a) interpolation error and (b) ratio $R(e)$ of the elements

Table 2 Statistics for the meshes generated for MRI field

method	gradient-limiting	size-preserving
total faces	64918	192121
correct faces	53804	191734
correct faces (%)	81.33%	99.79%
max $R(e)$	1.92	1.11
min quality	0.05	0.35
max quality	0.99	0.99
mean quality	0.78	0.90
quality deviation	0.14	0.09
time (s)	1.05	6.67

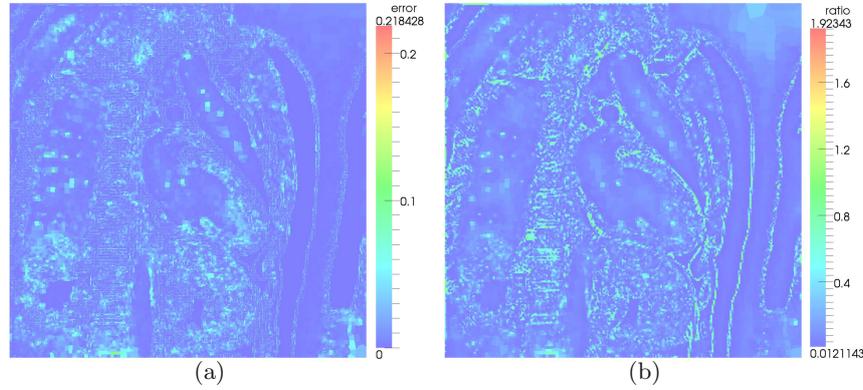


Fig. 8 Mesh generated using the size-preserving technique: (a) interpolation error and (b) ratio $R(e)$ of the elements

6 Conclusions

In this paper, we have presented a novel quantitative criterion to assess when an element correctly reproduces a prescribed size function. Using this criterion, we deduce the novel size-preserving method to compute a new size function by solving an implicit, non-linear equation. The new size function ensures high quality elements that correctly reproduce the prescribed size function.

The size-preserving method computes the new size function by solving a non-linear equation. The proposed algorithm modifies the size function at each node of a background mesh ensuring that an element can be held in that area and, at the same time, limiting the maximum gradient of the new size function. We have shown that to generate a mesh that preserves the original size function, it is recommended to use $\beta \leq 1$ and $\alpha \geq \beta$.

We have proposed to apply a smoothing technique that takes into account both the element quality and the prescribed size in order to obtain a high-quality mesh that correctly reproduces the size function. We have shown that the smoother is able to obtain a high-quality mesh that reproduces the prescribed size function only when the size-preserving method is applied. When a gradient-limiting method is used, the smoother is not able to obtain a mesh that correctly preserves the initial size function.

Two applications of the size-preserving size function have been proposed. The first one is the generation of quadrilateral meshes. When using classical gradient-limiting techniques, the generated mesh does not fully reproduce the initial size function. For this reason, refining algorithms have to be applied, which can potentially reduce the quality of the mesh. Using the proposed size-preserving technique, the generated mesh already preserves the initial size function and, for this reason, high-quality elements are generated. The

second application is that the size-preserving size function can potentially reduce the number of iterations to converge an adaptive process, since at each iteration, the prescribed size function is correctly captured.

The current algorithm can be improved in several aspects. For instance, we are using an edge-based solver to compute the size-preserving function. However, we can use a Hamilton-Jacobi solver in order to obtain more accurate solutions. In addition, we can improve the speed of computing the size-preserving size function. Since the value of each node is computed independently, we can parallelize the code. Note that the size-preserving size function has been derived for any dimension. In the near future, we would consider to analyze its application to unstructured hexahedral meshing. Finally, we can generalize the ideas of this paper to work with anisotropic size fields.

References

1. Peraire, J., Vahdati, M., Morgan, K., Zienkiewicz, O.C.: Adaptive remeshing for compressible flow computations. *Journal of Computational Physics* 72, 449–466 (1987)
2. Löhner, R.: Adaptive remeshing for transient problems. *Computer Methods in Applied Mechanics and Engineering* 75, 195–214 (1989)
3. Ryppl, D.: Sequential and Parallel Generation of Unstructured 3D Meshes. PhD thesis, CTU in Prague (1998)
4. Blacker, T.D., Stephenson, M.B.: Paving: A new approach to automated quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering* 32(4), 811–847 (1991)
5. Sarrate, J., Huerta, A.: Efficient unstructured quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering* 49, 1327–1350 (2000)
6. Quadros, W.R., Shimada, K., Owen, S.J.: Skeleton-based computational method for the generation of a 3d finite element mesh sizing function. *Engineering with Computers* 20(3), 249–264 (2004)
7. Quadros, W.R., Vyas, V., Brewer, M., Owen, S.J., Shimada, K.: A computational framework for automating generation of sizing function in assembly meshing via disconnected skeletons. *Engineering with Computers* 26(3), 231–247 (2010)
8. Zhu, J., Blacker, T.D., Smith, R.: Background overlay grid size functions. In: *Proceedings of the 11th International Meshing Roundtable*, pp. 65–74 (2002)
9. Yerry, M.: Modified quad-tree approach to finite element mesh generation. *IEEE Computer Graphics & Applications* 3(1), 39–46 (1983)
10. Borouchaki, H., Hecht, F., Frey, P.J.: Mesh gradation control. *International Journal for Numerical Methods in Engineering* 43(6), 1143–1165 (1998)
11. Frey, P.J., Marechal, L.: Fast adaptive quadtree mesh generation. In: *Proceedings of the 7th International Meshing Roundtable* (1998)
12. Persson, P.O.: Mesh Size Functions for Implicit Geometries and PDE-Based Gradient Limiting. *Engineering with Computers* 22(2), 95–109 (2006)

13. Gargallo-Peiró, A., Roca, X., Sarrate, J.: A surface mesh smoothing and untangling method independent of the cad parameterization. Submitted to *Computer Methods in Applied Mechanics and Engineering*
14. Patrick, M.K.: Algebraic mesh quality metrics. *SIAM Journal on Scientific Computing* 23(1), 193–218 (2001)
15. Escobar, J.M., Rodriguez, E., Montenegro, R., Montero, G., González-Yuste, J.M.: Simultaneous untangling and smoothing of tetrahedral meshes. *Computer Methods in Applied Mechanics and Engineering* 192(25), 2775–2787 (2003)
16. Roca, X., Sarrate, J., Ruiz-Gironés, E.: A graphical modeling and mesh generation environment for simulations based on boundary representation data. In: *Congresso de Métodos Numéricos em Engenharia* (2007)
17. Roca, X.: Paving the path towards automatic hexahedral mesh generation. PhD thesis, *Universitat Politècnica de Catalunya* (2009)
18. Roca, X., Ruiz-Gironés, E., Sarrate, J.: ez4u. mesh generation environment (2010), <http://www-lacan.upc.edu/ez4u.htm>
19. Cardiac Atlas website (2013), <http://atlas.scmr.org/>