

## Size-preserving size functions and smoothing procedures for adaptive quadrilateral mesh generation

E. Ruiz-Gironés · X. Roca · J. Sarrate

**Abstract** The generation of meshes that correctly reproduce a prescribed size function is crucial for quadrilateral meshing due to two reasons. First, quadrilateral meshes are difficult to adapt to a given size field by refining or coarsening the elements without compromising the element quality. Second, after the meshing algorithm is finished, it may be necessary to apply a smoothing algorithm to improve the global quality. This smoothing step may modify the element size and the final mesh will not reproduce the prescribed element size. To solve these issues, we propose to combine the size-preserving method with a smoothing technique that takes into account both the element shape and size. The size-preserving technique allows directly generating a quadrilateral mesh that reproduces the size function, while the proposed smoother allows obtaining a high-quality mesh while maintaining the element size. In adaptive processes, the proposed approach may reduce the number of iterations to achieve convergence, since at each iteration the background mesh is properly reproduced. In addition, we detail new theoretical results that provide more insight to size-preserving size functions. Specifically, we prove that the maximum gradient of a one-dimensional size-preserving size function is bounded. Finally, several applications that show the benefits of applying the proposed techniques are presented.

**Keywords** Mesh size function · background mesh · adaptive process · quadrilateral mesh · gradient-limiting · smoothing

---

E. Ruiz-Gironés · J. Sarrate  
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, jose.sarrate}@upc.edu

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

## 1 Introduction

It is of major importance to generate meshes that correctly preserve the prescribed element size in adaptive processes [1, 2]. The element size is directly related to the accuracy of the numerical solution and the computational time to obtain it. On the one hand, small elements are required to obtain a high-quality numerical solution. On the other hand, large elements are required to obtain the numerical solution in a reasonable amount of time. While several techniques to assign an isotropic size function have been developed, one of the most used 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.

In order to obtain an high-quality mesh, the size function has to satisfy several properties. In addition, each mesh generation algorithm has its own requirements on the size function. For instance, triangular and tetrahedral algorithms [1 – 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 coarsening or refining the mesh is a difficult task which may lead to low-quality elements after repeated modifications. Therefore, special attention is focused on the generation of size functions to ensure that a high-quality mesh is directly obtained [6 – 8]. Current techniques limit the gradient of the size function in order to bound the size ratio of adjacent elements, see [9 – 11]. Thus, it is easier for the mesh generation algorithm to obtain a mesh composed by high-quality elements. However, the final mesh may contain elements that are bigger than the requested size. That is, the mesh does not reproduce the prescribed size function. For instance, if the size function contains small areas with low values of the prescribed size, they can be entirely missed by the final mesh. Thus, the final mesh does not reproduce the prescribed element size.

To solve this issue, we introduced the concept of size-preserving size function in [12, 13]. First, a quantitative criterion to asses when an element reproduces a size function is defined. Then, using this criterion, the size-preserving size function is deduced. The size preserving-size function presents several advantages. First, it enforces that a mesh generation algorithm obtains elements that are smaller than the prescribed size function. Second, the gradient of the new size function is bounded and, for this reason, high-quality elements can be generated. Thus, quadrilateral meshes that reproduce a prescribed element size can be directly generated without needing a refining or coarsening step.

Moreover, quadrilateral mesh generation algorithms require that a smoothing process is applied to the final mesh in order to untangle the inverted quadrilaterals and to improve the element quality of the whole mesh. However, the smoothing process usually does not take into account the size function and, for this reason, when applying the smoothing process, elements that do not follow the size function may appear. In [14], a smoothing process that takes into account the size function is developed. The smoother is based on a minimization approach that optimizes an objective function. The objective function is a combination of two functions. The first is a distortion measure that takes into account the shape of the element. The

second is a measure that takes into account the size of the element with respect to the prescribed size. The result is a smoother process that improves the quality of the whole mesh and, at the same time, obtains elements of the correct size.

The main contribution of this work is to combine the concept of size-preserving size functions with a smoother that takes into account the size function. Thus, we obtain a combined process to generate meshes that correctly reproduce the prescribed size function. Note that the combined process is independent of the meshing algorithm that has been used. It is important to point out that we propose a non-intrusive technique to obtain the final mesh. That is, we do not need to modify the core of the meshing algorithms. In 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 and the call to the smoothing process. In addition, we present several theoretical results to show that the maximum gradient of the size-preserving size function is bounded.

Several applications benefit from the properties of this combined approach. Specifically, we provide two direct applications. First, in quadrilateral mesh generation, the final mesh directly reproduces the size function, without applying coarsening or refining algorithms. Thus, the mesh quality is improved. Second, in adaptive analysis, we can reduce the number of iterations to achieve convergence, since at each iteration the element size prescribed at the background mesh is correctly reproduced.

The outline of this paper is the following. In Section 2, we summarize the concept of size-preserving size function. In Section 3, we deduce some theoretical results of the size-preserving size function. In Section 4, we show how to compute a size-preserving size-function. In Section 5, we present the smoothing procedure. Finally, in Section 6, several examples are presented to illustrate the capabilities of combining the size-preserving size function with a smoothing step that improves the mesh quality while preserves the element size.

## 2 Size-Preserving Size Function

Our goal is to generate a high-quality quadrilateral mesh that correctly reproduces a prescribed element size function. That is, we want to ensure that the size of all the elements of the quadrilateral mesh have the correct size. To this end, we first introduce a quantitative criterion to assess whether an element reproduces a size function. Then, we review the concept of size-preserving size function, previously presented in [12,13].

**Definition 1** A mesh  $M$  reproduces a prescribed size function,  $h(\mathbf{x})$ , if it satisfies

$$\mu(e) \leq \beta \min_{\mathbf{x} \in e} h(\mathbf{x}), \quad \forall e \in M, \quad (1)$$

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

By introducing the ratio

$$R(e) = \frac{\mu(e)}{\beta \min_{\mathbf{x} \in e} h(\mathbf{x})}, \quad (2)$$

a valid mesh has to satisfy

$$R(e) \leq 1 \quad \forall e \in \mathcal{M}. \quad (3)$$

In order to generate a mesh that correctly preserves the prescribed size function, we introduce the concept of *size-preserving size function*. Given the original size function,  $h(\mathbf{x})$ , we will deduce an alternative size function, called size-preserving size function and denoted by  $h^*(\mathbf{x})$ , such that the final mesh reproduces the original size function according to Equation (1).

The new size function,  $h^*(\mathbf{x})$ , can be written in terms of the original one,  $h(\mathbf{x})$ . To obtain a mesh that correctly reproduces the size function, we assume that the new element size around a point  $\mathbf{x} \in \Omega$  has to be  $h^*(\mathbf{x})$ . Then, the new node is created at position  $\mathbf{x} + h^*(\mathbf{x})\mathbf{u}$ , where  $\mathbf{u}$  is a unit vector that denotes the advancing direction of the meshing algorithm. For this reason, the size of the new element,  $e$ , is  $\mu(e) = h^*(\mathbf{x})$ . Taking into account condition (1), we deduce that the following equation has to be satisfied:

$$h^*(\mathbf{x}) = \mu(e) \leq \beta \min_{\mathbf{y} \in B_{h^*(\mathbf{x})}(\mathbf{x})} h(\mathbf{y}),$$

where  $B_r(\mathbf{x})$  is the set of points at distance at most  $r$  from  $\mathbf{x}$ . Note that we have to take the minimum of the original size function over the whole ball, since we do not know a priori the advancing direction of the mesher. If we want  $h^*(\mathbf{x})$  as big as possible to generate the minimum amount of elements, we have that

$$h^*(\mathbf{x}) = \beta \min_{\mathbf{y} \in B_{h^*(\mathbf{x})}(\mathbf{x})} h(\mathbf{y}).$$

To add more flexibility to our method, we include a positive parameter  $\alpha$  that determines the radius of the ball where the minimum of the original size function is extracted:

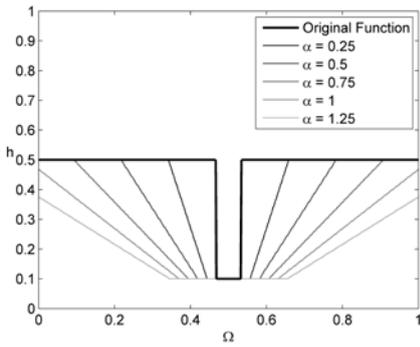
$$h^*(\mathbf{x}) = \beta \min_{\mathbf{y} \in B_{\alpha h^*(\mathbf{x})}(\mathbf{x})} h(\mathbf{y}). \quad (4)$$

Note that Equation (4) is an implicit and non-linear definition of the size-preserving size function. We detail an algorithm to compute it in Section 4. Parameter  $\alpha$  plays an important role in Equation (4). 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. In Section 3, we prove that the maximum gradient of a one-dimensional size-preserving size function is bounded by  $1/\alpha$ .

Although in Equation (4) parameters  $\alpha$  and  $\beta$  are arbitrary, it is recommended to select  $\beta \leq 1$  and  $\alpha \geq 1$  in order to obtain a mesh that satisfies Condition (1). Specifically, we use  $\beta = 1$  through the rest of the paper if it is not explicitly stated.

Figure 1 shows a simple one-dimensional size function with several size-preserving size functions with different values of the  $\alpha$  parameter. They have been computed using the algorithm detailed in Section 4. As  $\alpha$  increases, the maximum gradient of the size-preserving size function decreases, and the area around the minimum size is increased to accommodate more elements of the desired size.



**Fig. 1** Simple size function (thick black line) and several size-preserving size functions with different  $\alpha$  parameters (gray lines).

*Remark 1* By definition of size-preserving size function, an ideal mesh generator should generate a mesh that preserves the initial size field and reproduces all of its features. However, in practice mesh generators are not able to fully preserve a given size function and therefore, we can expect meshes that lead to  $\max R(e) > 1.0$ . For instance, an advancing front mesh generator can close two converging fronts creating an element between them with a larger size than the prescribed size. To facilitate the application of our method in a practical setting, the parameters  $\alpha$  and  $\beta$  can be tuned. As  $\alpha$  increases, the size-preserving size function takes smaller values and conversely, as  $\alpha$  decreases, the size-preserving size function takes larger values. In addition, as  $\beta$  increases, the size-preserving size function increases, and as  $\beta$  decreases, the size-preserving size function decreases. Thus, if larger elements are desired, one can use smaller values of  $\alpha$  and higher values of  $\beta$ . On the contrary, if smaller elements are desired, the user can assign larger values of  $\alpha$  and smaller values of  $\beta$ .

*Remark 2* 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.

### 3 Theoretical analysis

In this section, we prove that the maximum gradient for a one-dimensional size-preserving size function is limited to  $1/\alpha$ . To this end, we first present a definition and several lemmas to introduce preliminary results.

**Definition 2** Let  $h(x)$  be a one-dimensional size function in the interval  $[a, b]$ . Given  $x \in [a, b]$ , we define  $x^*$  as

$$x^* = \operatorname{argmin}_{y \in \mathcal{B}_{ah^*}(x)} \beta h(y). \quad (5)$$

Using Definition 2, it is straightforward to show that  $h^*(x) = \beta h(x^*)$ . In addition, since  $x^*$  is the argument of the minimum of the original size function, we have that

$$x^* \in \mathcal{B}_{ah^*}(x) = [x - ah^*(x), x + ah^*(x)].$$

**Lemma 1** Let  $h(x)$  be a one-dimensional size function in the interval  $[a, b]$ . Assume that  $x^* \in (x - ah^*(x), x + ah^*(x))$ . If  $h^*(x)$  is differentiable at  $x$ , then,

$$(h^*(x))^1 = 0.$$

*Proof* We prove this lemma by contradiction. That is, assume that  $(h^*(x))^1 \neq 0$ . Thus, we consider two cases depending on the sign of the derivative.

(i)  $(h^*(x))^1 > 0$ .

If the derivative is positive, then exists  $\varepsilon > 0$  such that  $h^*(x) < h^*(x + \varepsilon)$ . In addition, since  $x^* \in (x - ah^*(x), x + ah^*(x))$ , if  $\varepsilon$  is small enough,

$$x^* \in \mathcal{B}_{ah^*}(x + \varepsilon).$$

Thus,  $h^*(x + \varepsilon) < h^*(x)$ , which is a contradiction.

(ii)  $(h^*(x))^1 < 0$ .

In this case, there exists  $\varepsilon > 0$  such that  $h^*(x) < h^*(x - \varepsilon)$ . Since  $x^* \in (x - ah^*(x), x + ah^*(x))$ , if  $\varepsilon$  is small enough,

$$x^* \in \mathcal{B}_{ah^*}(x - \varepsilon).$$

Thus,  $h^*(x - \varepsilon) < h^*(x)$ , which is a contradiction. □

**Lemma 2** Let  $h(x)$  be a one-dimensional size function in the interval  $[a, b]$ , and  $x \in [a, b]$  such that  $x^* = x - ah^*(x)$ . If  $h$  is differentiable at  $x^*$ , then  $h^1(x^*) \geq 0$ .

*Proof* We proof this lemma by contradiction. Assume that  $h^1(x - ah^*(x)) < 0$ . Then  $h(x - ah^*(x) + \varepsilon) < h(x - ah^*(x))$ , for some value  $\varepsilon > 0$ . However,  $x - ah^*(x) + \varepsilon$  is located inside the interval  $[x - ah^*(x), x + ah^*(x)]$ . Thus, the hypotheses are invalidated, since  $h(x - ah^*(x) + \varepsilon) < h^*(x)$ . □

**Lemma 3** Let  $h(x)$  be a one-dimensional size function in the interval  $[a, b]$ , and  $x \in [a, b]$  such that  $x^* = x + ah^*(x)$ . If  $h$  is differentiable at  $x^*$ , then  $h^1(x^*) \leq 0$ .

*Proof* The proof of this Lemma follows the same reasoning than the proof of Lemma 2. □

Lemmas 2 and 3 establish the sign of the derivative of the original size function at the end points of the test interval.

**Lemma 4** Let  $h(x)$  be a one-dimensional size function in the interval  $[a, b]$ . Assume that there exists  $x \in [a, b]$  such that  $(h^*(x))^I > 0$ . If  $h^*$  is differentiable at  $x$ , then,  $\exists \varepsilon > 0$  such that  $\forall y \in B_\varepsilon(x)$ ,  $h^*(y) = \beta h(y) - ah^*(y)$ .

*Proof* We also proof this lemma by contradiction. That is, assume that  $\forall \varepsilon > 0$ ,  $\exists y \in B_\varepsilon(x)$  such that  $h^*(y) \neq \beta h(y) - ah^*(y)$ . Then, we can define a succession of values in the following manner. Let  $\varepsilon = 1/n$ , for  $n \geq 1$ . Thus, there exists  $y_n \in B_{1/n}(x)$  such that  $h^*(y_n) \neq \beta h(y_n) - ah^*(y_n)$ . Since  $h^*(y_n) \neq \beta h(y_n) - ah^*(y_n)$ , we deduce that  $y_n^*$  belongs to the interior of the interval  $[y_n - ah^*(y_n), y_n + ah^*(y_n)]$ . Thus, according to Lemma 1, we obtain

$$(h^*(y_n))^I = 0 \quad \forall n \geq 1.$$

Taking limits,

$$\lim_{n \rightarrow \infty} y_n = x.$$

Then, assuming that  $(h^*(x))^I$  is continuous at  $x$ ,

$$(h^*(x))^I = \lim_{n \rightarrow \infty} (h^*(y_n))^I = 0,$$

which is a contradiction.  $\square$

**Lemma 5** Let  $h(x)$  be a one-dimensional size function in the interval  $[a, b]$ . Assume that there exists  $x \in [a, b]$  such that  $h^I(x^*) < 0$ . If  $h^*$  is differentiable at  $x$ , then,  $\exists \varepsilon > 0$  such that  $\forall y \in B_\varepsilon(x)$ ,  $h^*(y) = \beta h(y) + ah^*(y)$ .

*Proof* The proof of Lemma 5 is performed in a similar manner as Lemma 4.  $\square$

Lemmas 4 and 5 state that the size-preserving size function,  $h^*(x)$ , can be locally expressed as an implicit function using the original size function  $h(x)$ , as long as  $(h^*(x))^I \neq 0$ .

**Theorem 1** Let  $h(x)$  be a one-dimensional size function in the interval  $[a, b]$ . Assuming that  $h^*(x)$  is differentiable, then,  $|(h^*(x))^I| \leq 1/a$ , where  $a$  is the parameter introduced in (4).

*Proof* To prove this proposition, we consider three cases depending on to the position of  $x^*$ .

- (i) Let  $x^* \in (x - ah^*(x), x + ah^*(x))$ .  
According to Lemma 1,  $(h^*(x))^I = 0$ , and the proposition holds.
- (ii) Let  $x^* = x - ah^*(x)$ . According to Lemma 2,

$$(h^*(x))^I \geq 0.$$

If  $(h^*(x))^I = 0$ , then  $|(h^*(x))^I| \leq 1/a$ . Otherwise, using Lemma 4,  $h^*(x)$  can be locally expressed as  $h^*(x) = \beta h(x - ah^*(x))$ . Taking derivatives in both sides of the equality and rearranging the terms, we obtain:

$$(h^*(x))^I = \frac{\beta h^I(x^*)}{1 + a\beta h^I(x^*)}.$$

According to Lemma 2,  $h^I(x^*) \geq 0$ . In addition, the function  $f(t) = \frac{t}{1+at}$  is increasing for  $t \geq 0$ . Thus,

$$(h^*(x))^I \leq \lim_{t \rightarrow \infty} f(t) = \frac{1}{a},$$

and the proposition holds.

(iii) Let  $\mathbf{x}^* = \mathbf{x} + \alpha \mathbf{h}^*(\mathbf{x})$ , and the proposition holds.

This case is proved in a similar manner as case (ii), but using Lemmas 3 and 5.  $\square$

Note that Theorem 1 provides an upper bound of the maximum gradient for a one-dimensional size-preserving size function that is independent of the chosen value of  $\beta$ . The maximum gradient only depends on the value of parameter  $\alpha$  and, for this reason, high-quality meshes can be generated for any value of  $\beta$ . However, as we have already pointed out, in order to obtain a high-quality mesh that correctly preserves the original size function, it is recommended to use  $\beta \leq 1$ .

#### 4 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,  $h(\mathbf{x})$ . Although the algorithm is already presented in [12, 13], we include it for the sake of completeness. The process of computing the value of the size-preserving size function 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 (4) is satisfied.

Given a node,  $\mathbf{n}_0$ , 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 satisfied:

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

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 complete binary tree where each children node is greater than or equal to the parent node. In this case, the nodes in the min-heap are sorted according to the distance to the center of the ball. Since we compute the value of the size-preserving size function node by node, we only need to create and maintain one min-heap at a time. Thus, the memory foot-print is small. Algorithm 1 details the calculation of  $h^*(\mathbf{x})$  for a given node  $\mathbf{n}_0$ . In this algorithm, the distance between an arbitrary node  $\mathbf{n}$  of the background mesh and node  $\mathbf{n}_0$  is computed as the distance along the edges. The distance to node  $\mathbf{n}_0$  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,  $\mathbf{n}_0$ , 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  $r = \alpha \beta h(\mathbf{x}_0)$ . The main loop of the algorithm begins at this point. Each node,  $\mathbf{n}$ , 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  $\mathbf{n}_0$ . Note that in our implementation, the distance between two nodes,  $d$ , is not computed as the Euclidean distance but rather as the length of the shortest path of edges between them. In this way, we can deal with non-convex domains. That is, we can deal with nodes that are close in the Euclidean sense, but separated when the shortest path of edges that connects them is long. In addition, we denote by  $\mathbf{x}_n$  the location of node  $\mathbf{n}$  and we define an auxiliary variable  $r^1 = \alpha \beta h(\mathbf{x}_n)$  that

**Algorithm 1** Computation of  $h^*(x)$ 


---

**Ensure:**  $h^*(x)$

```

1: function sizePreserving(BackgroundMesh  $M$ , Node  $no$ , Real  $\alpha$ )
2:   NodeMinHeap  $N$ 
3:   setDistance( $no, 0$ )
4:   insert( $no, N$ )
5:   Real  $v_0 \leftarrow \beta h(\mathbf{x}_0)$ 
6:   Real  $r \leftarrow \alpha \cdot v_0$ 
7:   while getSize( $N$ ) > 0 do
8:     Node  $n \leftarrow$  firstNode( $N$ )
9:     removeNode( $n, N$ )
10:    Real  $d \leftarrow$  getDistance( $n$ )
11:    Real  $v \leftarrow \beta h(\mathbf{x}_n)$ 
12:    Real  $r' \leftarrow \alpha \cdot v$ 
13:    if  $r' < r$  then
14:      if  $d \leq r'$  then
15:         $r \leftarrow r'$ 
16:        updateAdjacentNodesDistance( $n, r, N$ )
17:      else if  $d \leq r$  then
18:         $r \leftarrow r - r' + \frac{r-d}{r-r'} r'$ 
19:      end if
20:    else if  $r' \geq r$  and  $d \leq r$  then
21:      updateAdjacentNodesDistance( $n, r, N$ )
22:    end if
23:  end while
24:   $h^*(\mathbf{x}_0) \leftarrow r/\alpha$ 
25: end function

```

---

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.

(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.

**Algorithm 2** Update the distance of adjacent nodes

---

```

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

```

---

(v)  $r^l \geq r$  and  $r < d$ .

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

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

Given a node,  $n$ , Algorithm 2 updates the approximation to 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 smallest value contains the correct value of the distance. Recall that this node,  $n$ , is the first node of the min-heap. The new approximation to the distance,  $d_e^l$ , of a node,  $n_e$ , adjacent to  $n$  through edge  $e$  is computed as:

$$d_e^l = \min\{d_e, d + l_e\},$$

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

## 5 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 [14]. 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 [14], the authors propose an element quality that is a combination of the shape quality [15] 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 transforms the ideal element,  $e_I$ , to the physical element,  $e_P$ . Then, the element quality function is:

$$\eta(e_P) = \eta_{sh}(e_P)\eta_{s\tilde{a}}(e_P), \quad (7)$$

where  $\eta_{sh}(e_P)$  corresponds to the shape quality,

$$\eta_{sh}(e_P) = \frac{\|S\|^2}{2|\sigma|}, \quad (8)$$

and  $\eta_{si}(e_P)$  corresponds to the size quality,

$$\eta_{si}(e_P) = \frac{1}{\mu(\sigma)}, \quad (9)$$

In Equations (8) and (9),  $\|S\|$  and  $\sigma$  are the Frobenius norm and the determinant of matrix  $S$ , and

$$\mu(\sigma) = \frac{e}{2} (\sigma e^{-\sigma} + \sigma^{-1} e^{-\sigma^{-1}}),$$

being  $e$  the Euler's number. The  $\eta_{sh}$  function presents asymptotes when  $\sigma = 0$ . For this reason, and according to references [16, 14], we regularize this shape distortion measure in the following way:

$$\eta^*_{sh}(e_P) = \frac{\|S\|^2}{2|\alpha(\sigma)|},$$

$$\alpha(\sigma) = \frac{1}{2} (\sigma + \sqrt{\sigma^2 + 4\delta^2})$$

where

being  $\delta$  is a small parameter.

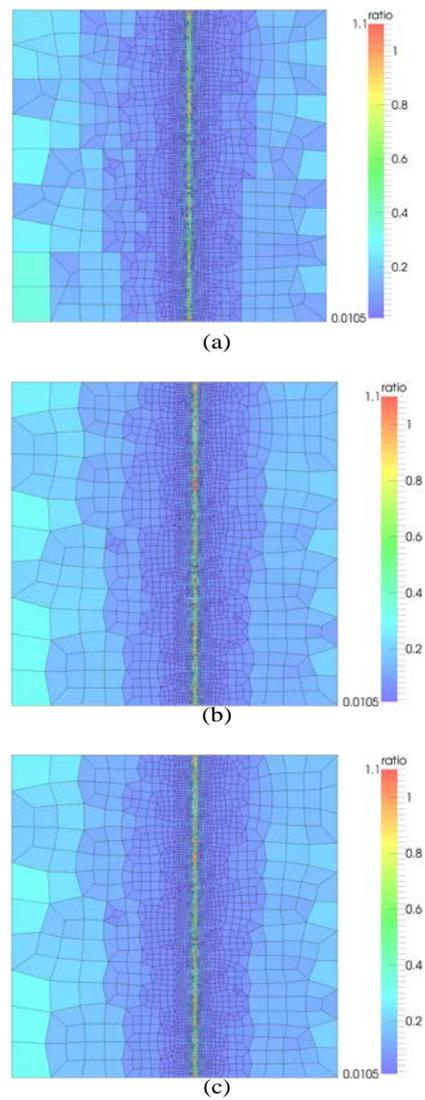
Using the element quality function (7), a continuous minimization problem is stated, and the optimum position of the nodes is computed. For more details on minimizing function (7), see reference [14].

Finally, we present a simple two-dimensional size function to illustrate the behavior of the presented smoother. To this end, we consider the square domain  $[-1, 1] \times [-1, 1]$ , and we define the size function

$$h(x, y) = \min\left\{\frac{1}{3}, 0.005 + |16x - 9|\right\}. \quad (10)$$

We generate three quadrilateral meshes using a background mesh to prescribe the size function in Equation (10). In the first one, we do not smooth the quadrilateral mesh, see Figure 2(a). In this case, we obtain a mesh that correctly reproduces the size function. However, it contains low-quality and inverted elements. Note that quadrilateral elements with triangular shape appear in Figure 2(a). To improve the quality of the elements, in the second case we smooth the mesh by taking into account only the shape of the elements, see Figure 2(b). Now, the mesh is composed by high-quality elements, but it does not reproduce the size function. In the last case, we smooth the mesh with the proposed smoother that takes into account both the element quality and the size function, see Figure 2(c). In this case, we obtain a high-quality mesh that reproduces the original size function.

Table 1 presents the statistical information corresponding to the meshes generated using the size function (10). When no smoothing is applied, the minimum ratio  $R(e)$  is obtained, 1.02. However, inverted elements are present in the mesh. Thus, the minimum quality is zero. When the smoother that only takes into account the element shape is applied, the mesh does not contain inverted elements and the minimum element quality is 0.388. However, the ratio  $R(e)$  is 1.10, the



**Fig. 2** Quadrilateral meshes generated using size function in Equation (10). Final mesh obtained: (a) without any smoothing applied; (b) with a smoothing step that only considers the shape distortion measure; and (c) with a smoothing step that considers the shape distortion measure and the size function.

maximum value of the three meshes. Only when applying the smoother that takes into account both the shape and the size of the element, a high-quality mesh that correctly reproduces the size function is obtained. The ratio  $R(e)$  is 1.06, and the minimum quality is 0.353.

**Table 1** Statistics for the quadrilateral meshes generated using size function (10).

smoothing method	no smoothing	shape smoother	size+shape smoother
max $R(\mathbf{e})$	1.02	1.10	1.06
min quality	0.0	0.388	0.353
max quality	0.999	0.999	0.999
mean quality	0.801	0.897	0.895
quality deviation	0.247	0.121	0.124

## 6 Examples

This section presents three 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 [17 – 19]. The first example presents the advantages of using the size-preserving size function to represent a two-dimensional size field. The second example shows how a size-preserving size function is able to reduce the number of iterations to converge an adaptive process. The third 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 Section 5 has been applied in order to improve the quality of the mesh while preserving the prescribed size of the elements. In all the examples we have used the shape quality to measure the quality of the mesh, see Equation (8). 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  $\varepsilon = 1/\mathbf{a}$ , where  $\varepsilon$  is the maximum gradient allowed in the gradient-limiting technique.

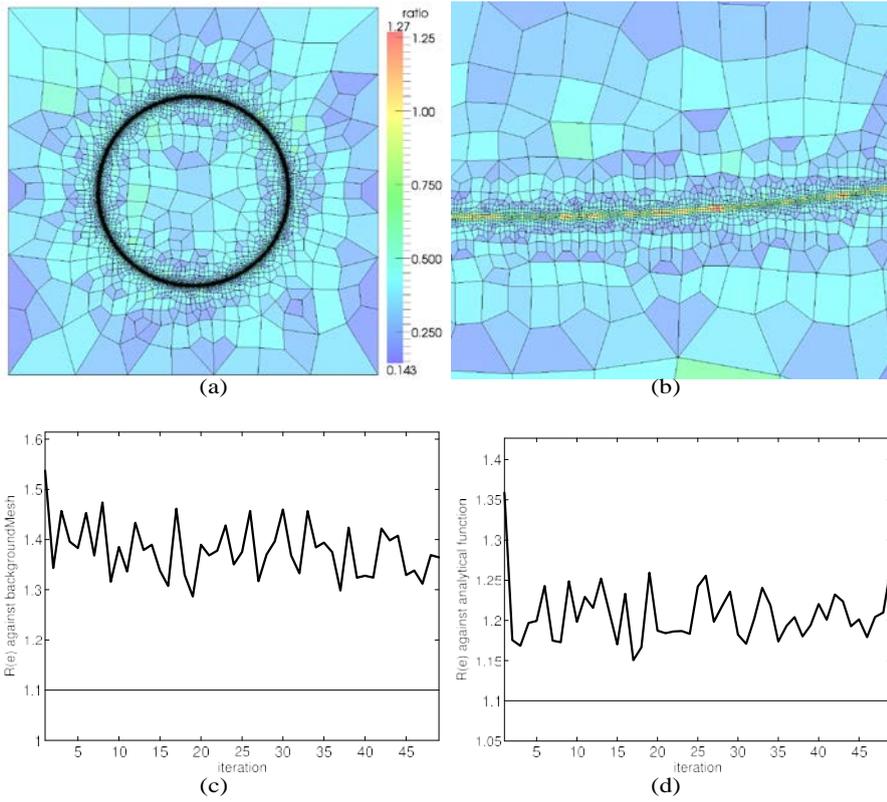
### 6.1 Representing a two-dimensional size field using an adaptive process

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

$$h = \min \left\{ 0.25, |d - 0.2575| + 0.25 \cdot 10^{-3} \right\}, \quad (11)$$

where  $d = \frac{1}{\sqrt{(x - 0.5)^2 + (y - 0.5)^2}}$ , 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, Equation (11), we compute the desired element size at the nodes of this mesh, and we set it as the background mesh of the first iteration of the adaptive process and we generate a new mesh. At each iteration, the new background mesh is constructed from the mesh of the previous iteration. From this background mesh we generate the new mesh. This process is iterated until the mesh reproduces the analytical size function with a relative error below 0.1. That



**Fig. 3** Adaptive process, not converged after 50 iterations, using the gradient-limiting technique: (a) mesh at the last iteration; (b) detail of the mesh; (c) evolution of the ratio  $R(\theta)$  versus the current background mesh, and (d) evolution of the ratio  $R(\theta)$  versus the analytical function.

---

### Algorithm 3 Adaptive process

---

**Ensure:** Mesh  $M$

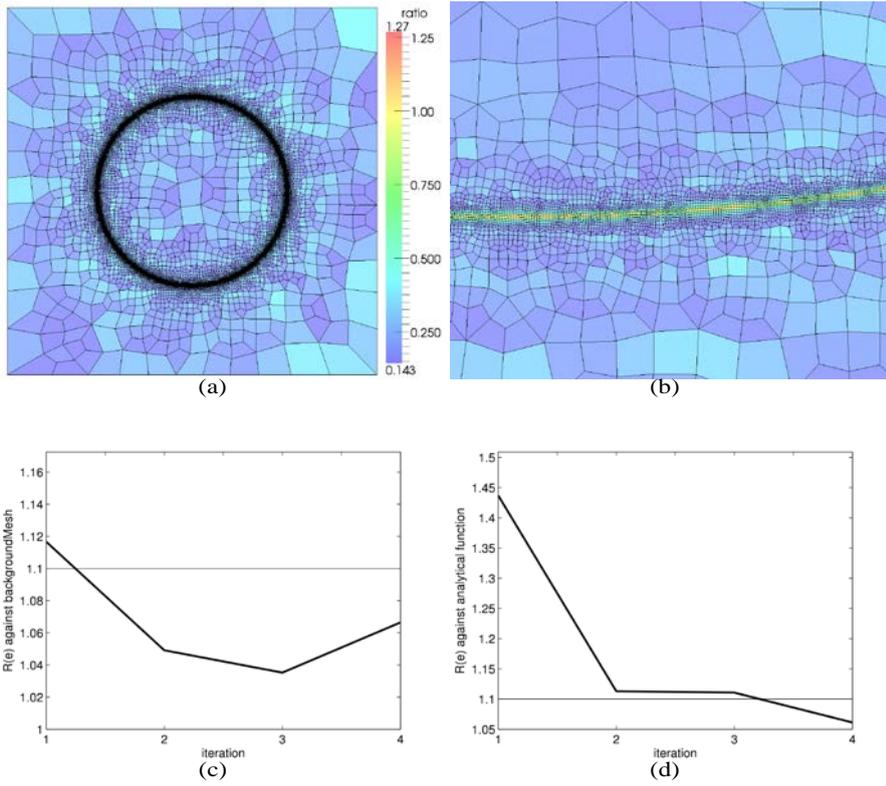
```

1: function AdaptiveProcess
2:   SizeFunction  $h \leftarrow$  getAnalyticalSizeFunction
3:   Mesh  $M \leftarrow$  createUniformMesh
4:   BackgroundMesh  $bm \leftarrow$  getElementSize( $M, h$ )
5:   processBackgroundMesh( $bm$ ) ✦ grad-lim, size-pres.
6:   Boolean converged  $\leftarrow$  checkConvergence( $M, bm$ )
7:   while not converged do
8:      $M \leftarrow$  createNewMesh( $M, bm$ )
9:     BackgroundMesh  $bm \leftarrow$  getElementSize( $M, h$ )
10:    processBackgroundMesh( $bm$ ) ✦ grad-lim, size-pres.
11:    Boolean converged  $\leftarrow$  checkConvergence( $M, bm$ )
12:  end while
13: end function

```

---

is, we accept all the elements whose size is, at most, 10% above the prescribed size of the analytical size function.



**Fig. 4** 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  $R(e)$  versus the current background mesh, and (d) evolution of the ratio  $R(e)$  versus the analytical function.

To check that the mesh generated at each iteration satisfies the prescribed element size we compare the size of the elements against the values stored at the vertices of the current background mesh and against the analytical size function, Equation (11).

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 3(a) presents the generated mesh at the last iteration, while Figure 3(b) shows a detailed view. Figure 3(c) and 3(d) show the evolution of the ratio  $R(e)$  computed versus the background mesh of the current iteration and the analytic function, respectively. Note that the ratio  $R(e)$  computed versus the background mesh is always above 1.28 and, for this reason, the size function prescribed by 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.

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 4(a) and Figure 4(b) show the mesh after the

**Table 2** Statistics for the meshes of the adaptive process after the last iteration.

method	gradient-limiting	size-preserving
total elements	64845	111092
correct elements	51506	110847
correct elements (%)	79.42%	99.77%
max $R(\mathbf{e})$	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

last iteration and a detailed view, respectively. Note that at each iteration, the background mesh is correctly captured, see Figure 4(c) and, for this reason, the process is able to generate a mesh that correctly preserves the analytical size function, see Figure 4(d).

Table 2 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 elements using the gradient-limiting function is around 80%, while the percentage of correct elements using the proposed size-preserving approach is more than 99%. Note that the 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.

## 6.2 Accelerating a two-dimensional adaptive process

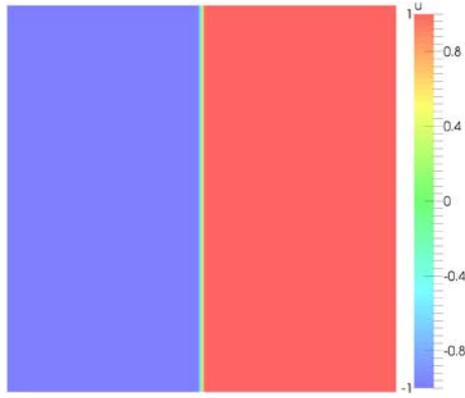
In this example we show that the size-preserving technique is able to reduce the number of iterations needed to converge an adaptive process. To this end, we propose to solve the problem:

$$\begin{aligned}
 \square & \Delta u = -2\alpha^2 \tanh(\alpha x) \left( 1 - (\tanh(\alpha x))^2 \right), \\
 \square & \quad \quad \quad (x, y) \in [-1, 1] \times [-1, 1], \\
 \square & \quad \quad \quad u = \tanh(\alpha x), \quad \quad \quad x = \pm 1, \\
 \square & \quad \quad \quad u_n = 0, \quad \quad \quad y = \pm 1,
 \end{aligned} \tag{12}$$

where  $\alpha$  is a real parameter. The analytical solution of problem (12) is

$$u = \tanh(\alpha x).$$

As  $|\alpha|$  increases, the analytical solution present higher gradients at  $x = 0$ . In this example, we use  $\alpha = 100$ , see Figure 5.



**Fig. 5** Analytical solution of problem (12) for  $a = 100$ .

---

**Algorithm 4** Adaptive process

---

**Ensure:** Mesh  $M$

```

1: function AdaptiveProcess
2:   Mesh  $M \leftarrow$  createUniformMesh
3:   Field  $u^M \leftarrow$  computeSolution( $M$ )
4:   Boolean converged  $\leftarrow$  checkConvergence( $u^M$ )
5:   while not converged do
6:     BackgroundMesh  $bm \leftarrow$  getElementSize( $M, u^M$ )
7:     processBackgroundMesh( $bm$ ) ♦ grad-lim, size-pres.
8:      $M \leftarrow$  createNewMesh( $M, bm$ )
9:     Field  $u^M \leftarrow$  computeSolution( $M$ )
10:    Boolean converged  $\leftarrow$  checkConvergence( $u^M$ )
11:   end while
12: end function

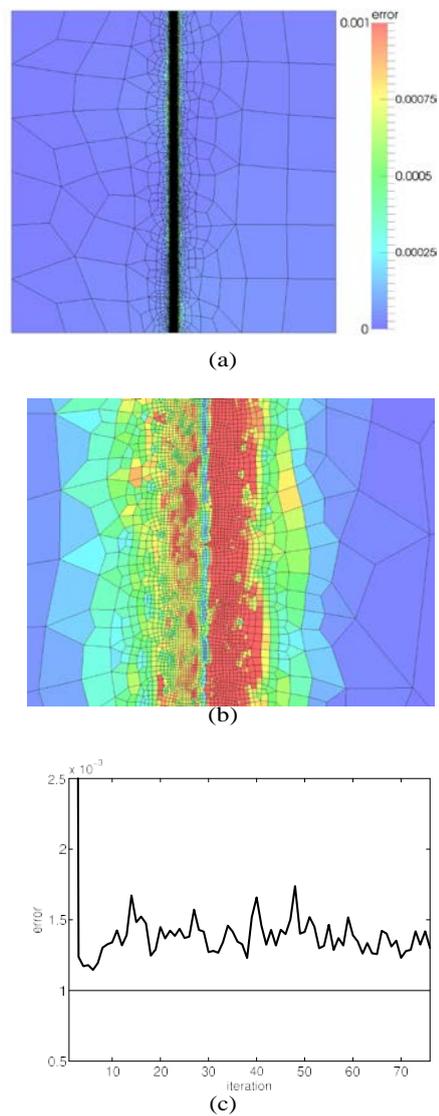
```

---

Since the analytical solution is known *a priori*, we apply the adaptive process defined in Algorithm 4. We first create an initial mesh composed of 20 elements per side. Using this mesh, we solve the problem (12) to obtain a numerical solution. Then, we check if the error of the numerical solution in the  $L^\infty$  norm is less than a threshold prescribed by the user. If it is not the case, we create a background mesh defined using a size function deduced from the error field. Then, a new mesh is generated using a background mesh defined with the size function. The process is iterated until convergence is achieved.

In the first execution, we use the gradient-limiting technique with  $\varepsilon = 0.5$  to process the background mesh. The adaptive process is not able to converge after 76 iterations, since at each iteration, the background mesh is not correctly preserved. Figure 6(a) shows the mesh and the error field in the last iteration, and Figure 6(b) shows a zoom at the sharp feature. Finally, Figure 6(c) show the evolution of the error. Note that the error is oscillating and the adaptive process is not able to converge.

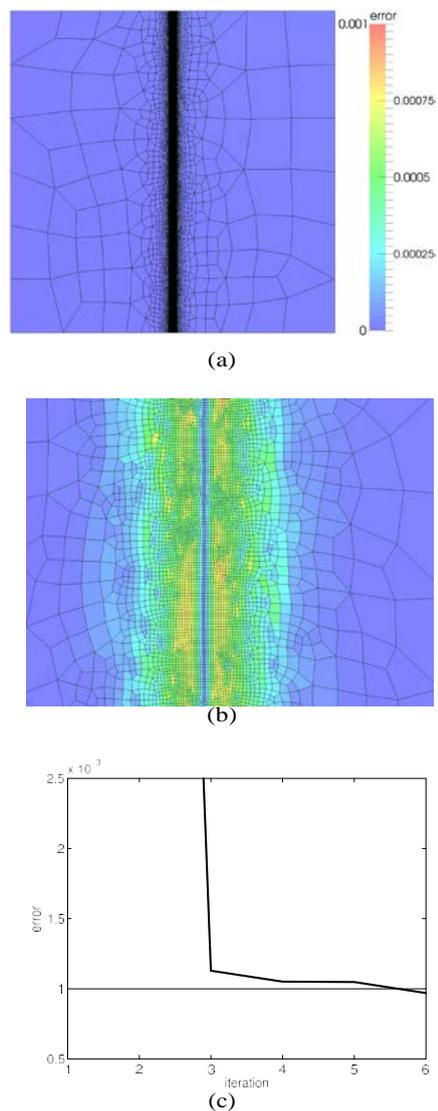
In the second execution, we process the background mesh using the proposed size preserving technique with  $\alpha = 2$ . In this case, the adaptive process is converged taking only 6 iterations. Figure 7(a) shows the mesh and the error field after the last iteration, and Figure 7(b) presents a detailed view. The evolution of the error during the adaptive process is shown in Figure 7(c).



**Fig. 6** Adaptive process, not converged in 76 iterations, using the gradient-limiting technique: (a) mesh and error field after the last iteration; (b) detailed view of the sharp feature; and (c) evolution of the error.

### 6.3 Preserving a complex size function in quadrilateral mesh generation

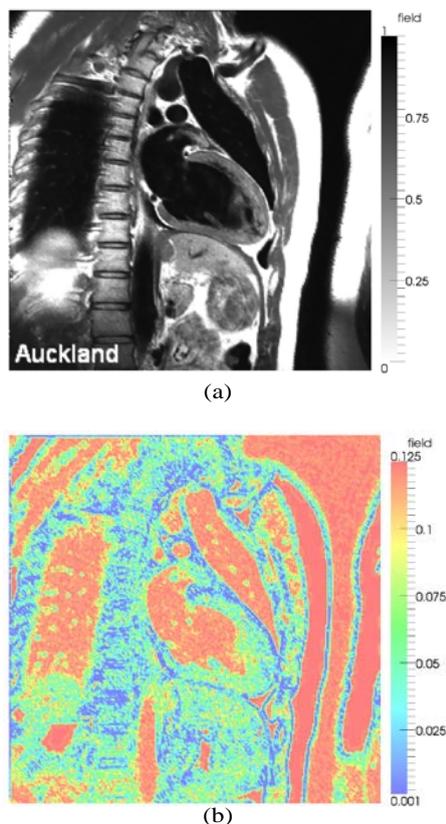
In this example, we present four quadrilateral meshes generated using a size field derived from a MRI image, courtesy of the Cardiac Atlas website and the Auckland MRI research group [20], see Figure 8(a). The size field is defined in terms of the mean curvature of the MRI field. Thus, it determines a higher density of elements where the variation of the gradient of the MRI field is higher, see Figure 8(b). In



**Fig. 7** Adaptive process, converged after 6 iterations, using the size-preserving technique: (a) mesh and error field after the last iteration; (b) detailed view of the sharp feature; and (c) evolution of the error.

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 two meshes are obtained with the gradient-limiting technique,  $\varepsilon = 0.5$ . The first mesh is obtained using a smoothing technique in which only the shape quality is considered. Figures 9(a) and 9(b) show the interpolation error and the ratio  $R(e)$  of the obtained mesh, respectively. Note that in this case, there are elements that are double the prescribed size. The second mesh is generated by

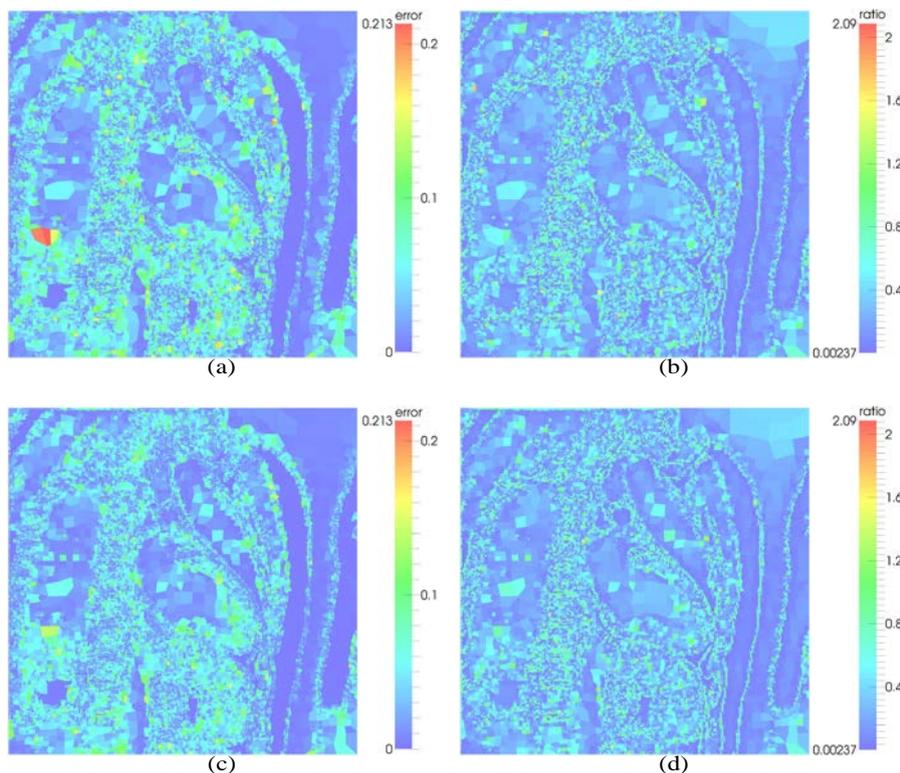


**Fig. 8** (a) MRI field defined on a square and (b) its associated size function.

applying the smoother technique in which the element shape and size is taken into account. Figures 9(c) and 9(d) show the interpolation error and the ratio  $R(e)$  of the obtained mesh, respectively. Although the interpolation error and the ratio  $R(e)$  is lower in this case, there are elements that are more than 90 % bigger than the requested size.

The last two meshes are generated using the size-preserving method. The third one is generated by applying the smoother that only takes into account the shape quality of the elements. Figures 10(a) and 10(b) show the interpolation error and the ratio  $R(e)$ , respectively. Note that the obtained mesh presents lower interpolation error than the previous meshes, and the ratio  $R(e)$  is also lower. Using the size-preserving approach, even if the smoother does not take into account the size function, better results are obtained. To generate the fourth mesh, we have applied the size-preserving technique combined with the smoother that takes into account the element shape and size. The interpolation error and the ratio  $R(e)$  are shown in Figures 10(c) and 10(d). Note that in this case, the lowest interpolation error is obtained as well as the lowest ratio  $R(e)$ .

Table 3 summarizes the statistics for the meshes generated with the different size functions and smoothing techniques. The gradient-limiting technique is not

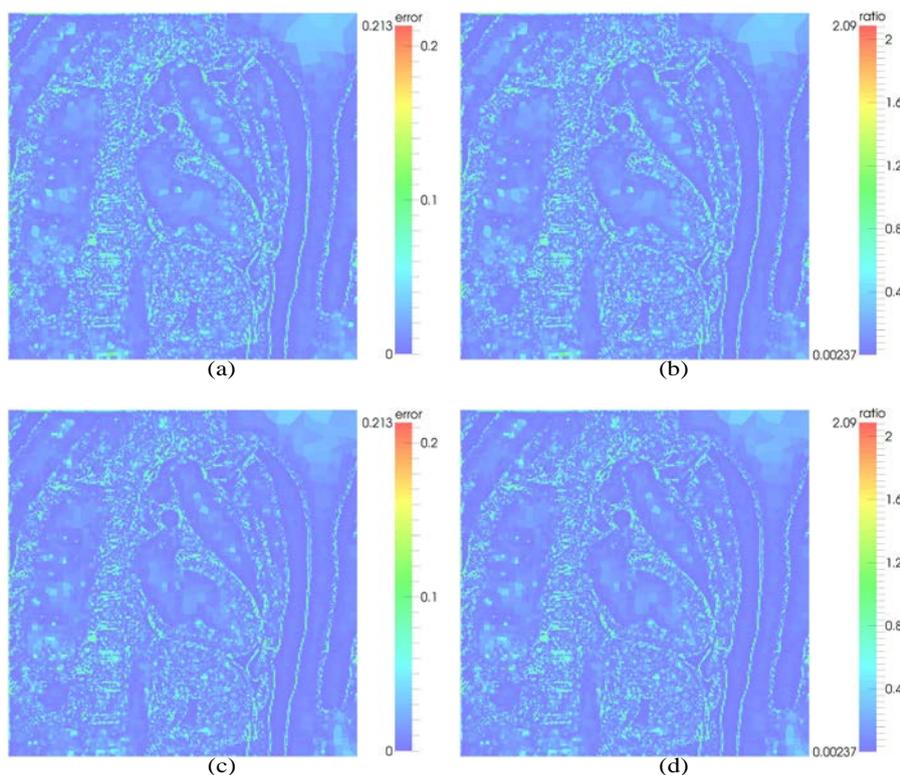


**Fig. 9** Mesh generated using the gradient-limiting technique: (a) and (b) interpolation error and ratio  $R(e)$  when the shape quality is considered; and (c) and (d) interpolation error and ratio  $R(e)$  when the combined shape and size quality is considered.

**Table 3** Statistics of the meshes generated for the MRI field.

smoothing method	gradient-limiting		size-preserving	
	shape quality	size+shape quality	shape quality	size+shape quality
total elements	69551	69551	161294	161294
correct elements	55279	56565	158573	160670
correct elements (%)	79.47%	81.32%	98.31%	99.61%
max $R(e)$	2.08	1.92	1.11	1.06
min quality	0.26	0.05	0.39	0.35
max quality	0.99	0.99	0.99	0.99
mean quality	0.80	0.80	0.93	0.88
quality deviation	0.12	0.15	0.09	0.10

able to reproduce the initial size function even when it is combined with a smoother that takes into account the size of the element, obtaining a maximum ratio  $R(e)$  of 1.92 and only 81% of correct elements. In addition, when the smoothing process only takes into account the shape of the element, the percentage of elements with ratio  $R(e)$  lower than one decreases to 79%, approximately. The meshes obtained with the size-preserving method better represent the prescribed size function, in-



**Fig. 10** Mesh generated using the gradient-limiting technique: (a) and (b) interpolation error and ratio  $R(e)$  when the shape quality is considered; and (c) and (d) interpolation error and ratio  $R(e)$  when the combined shape and size quality is considered.

dependently of the smoothing method. When the smoothing method that takes into account the element shape and size is used, the maximum ratio is 1.06, and are more than 99% of elements are correct. When using the smoothing approach that only takes into account the element shape, the maximum ratio is 1.11 and more than 98% percent of elements are correct. As expected, when the smoother takes into account the element size, the maximum ratio  $R(e)$  decreases and the percentage of correct elements increases.

The quality of the meshes generated using the size-preserving approach is better than the meshes generated using the gradient-limiting method. In addition, when the smoother only takes into account the element shape, better qualities are obtained than when the smoother takes into account both the shape and the size of the elements. Nevertheless, the best results are obtained when the mesh is generated by combining the size-preserving method and the proposed smoothing procedure that takes into account the element shape and size.

## 7 Conclusions

In this paper, we have combined the size-preserving technique with a smoother that takes into account both the element quality and the prescribed size function. The size-preserving size function ensures that the mesh generation algorithm obtains a mesh that correctly reproduces the initial size function. However, the mesh may have to be smoothed in order to remove tangled elements and to improve the quality of the whole mesh. Thus, we apply the smoother algorithm that takes into account both the element quality and the size function to obtain a high-quality mesh that reproduces the prescribed size function. The total number of elements of the meshes generated using the combined approach is larger than the number of elements obtained using the gradient-limiting technique. However, it is important to point that using the combined approach, the ratio  $R(\mathbf{e})$  is approximately equal to one around the local minima of the size function. Therefore, the increase in the number of elements remains acceptable.

Two applications 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 technique, the generated mesh already preserves the initial size function and, for this reason, high-quality elements are directly generated. The second application is that the proposed approach can potentially reduce the number of iterations to converge an adaptive process, since at each iteration, the prescribed size function is correctly captured.

In addition, we have shown that the maximum gradient of a one-dimensional size-preserving size function is bounded by  $1/\alpha$ . Note that this bound is independent of the chosen value of  $\beta$  and, for this reason, high-quality meshes can be obtained for any value of the  $\beta$  parameter. However, as we have already pointed out, in order to reproduce the original size function, it is recommended to set  $\beta \leq 1$  and  $\alpha \geq 1$ . Although this result has been proved for the one-dimensional case, further analysis is needed to generalize it for higher dimensions.

The current size-preserving 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 would like to generalize this work to deal with anisotropic size fields.

## References

1. J. Peraire, M. Vahdati, K. Morgan, and O.C Zienkiewicz. Adaptive remeshing for compressible flow computations. *Journal of Computational Physics*, 72:449–466, 1987.
2. R. Löhner. Adaptive remeshing for transient problems. *Computer Methods in Applied Mechanics and Engineering*, 75:195–214, 1989.
3. D. Rypl. *Sequential and Parallel Generation of Unstructured 3D Meshes*. PhD thesis, CTU in Prague, 1998.

4. T.D. Blacker and M.B. Stephenson. Paving: A new approach to automated quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering*, 32(4):811–847, 1991.
5. J. Sarrate and A. Huerta. Efficient unstructured quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering*, 49:1327–1350, 2000.
6. W.R. Quadros, K. Shimada, and S.J. Owen. Skeleton-based computational method for the generation of a 3d finite element mesh sizing function. *Engineering with Computers*, 20(3):249–264, 2004.
7. W.R. Quadros, V. Vyas, M. Brewer, S.J. Owen, and K. Shimada. A computational framework for automating generation of sizing function in assembly meshing via disconnected skeletons. *Engineering with Computers*, 26(3):231–247, 2010.
8. J. Zhu, T.D. Blacker, and R. Smith. Background overlay grid size functions. In *Proceedings of the 11th International Meshing Roundtable*, pages 65–74, 2002.
9. H. Borouchaki, F. Hecht, and P.J. Frey. Mesh gradation control. *International Journal for Numerical Methods in Engineering*, 43(6):1143–1165, 1998.
10. P.J. Frey and L. Marechal. Fast adaptive quadtree mesh generation. In *Proceedings of the 7th International Meshing Roundtable*, 1998.
11. P.O. Persson. Mesh Size Functions for Implicit Geometries and PDE-Based Gradient Limiting. *Engineering with Computers*, 22(2):95–109, 2006.
12. E Ruiz-Gironés, X Roca, and J Sarrate. Preserving isotropic element size functions in adaptivity, quadrilateral and hexahedral mesh generation. *Advances in Engineering Software*, 65:168–181, 2013.
13. E Ruiz-Gironés, X Roca, and J Sarrate. Combining size-preserving and smoothing procedures for adaptive quadrilateral mesh generation. In *Proceedings of the 22nd International Meshing Roundtable*, pages 19–37, 2014.
14. A. Gargallo-Peiró, X. Roca, and J. Sarrate. A surface mesh smoothing and untangling method independent of the CAD parameterization. *Computational Mechanics*, 53(4):587–609, 2014.
15. P. M. Knupp. Algebraic mesh quality metrics. *SIAM Journal on Scientific Computing*, 23(1):193–218, 2001.
16. J.M. Escobar, E. Rodriguez, R. Montenegro, G. Montero, and J.M. González-Yuste. Simultaneous untangling and smoothing of tetrahedral meshes. *Computer Methods in Applied Mechanics and Engineering*, 192(25):2775–2787, 2003.
17. X. Roca, J. Sarrate, and E. Ruiz-Gironés. A graphical modeling and mesh generation environment for simulations based on boundary representation data. In *Congresso de Métodos Numéricos em Engenharia*, 2007.
18. X. Roca. *Paving the path towards automatic hexahedral mesh generation*. PhD thesis, Universitat Politècnica de Catalunya, 2009.
19. X. Roca, E. Ruiz-Gironés, and J. Sarrate. ez4u. mesh generation environment. <http://www-lacan.upc.edu/ez4u.htm>, 2010.
20. Cardiac Atlas website. <http://atlas.scmr.org/>, 2013.