The Visual Computer manuscript No.
(will be inserted by the editor)

Template-Based Quadrilateral Mesh Generation from Imaging Data

M. A. S. Lizier · M. F. Siqueira · J. Daniels II · C. T. Silva · L. G. Nonato

Received: date / Accepted: date

Abstract This paper describes a novel template-based meshing approach for generating good quality quadrilateral meshes from 2D digital images. This approach builds upon an existing image-based mesh generation technique called *Imesh*, which enables us to create a segmented triangle mesh from an image without the need for an image segmentation step. Our approach generates a quadrilateral mesh using an indirect scheme, which converts the segmented triangle mesh created by the initial steps of the *Imesh* technique into a quadrilateral one. The triangle-to-quadrilateral conversion makes use of template meshes of triangles. To ensure good element quality, the conversion step is followed by a smoothing step, which is based on a new optimization-based procedure. We show several examples of meshes generated by our approach, and present a thorough experimental evaluation of the quality of the meshes given as examples.

Mario A. S. Liziér
Departamento de Computação
Universidade Federal de São Carlos - Brazil
E-mail: lizier@dc.ufscar.br

Marcelo F. Siqueira
Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte - Brazil
E-mail: mfsiqueira@dimap.ufrn.br

Joel Daniels II
Scientific Computing and Imaging Institute
University of Utah - USA
E-mail: jdaniels@cs.utah.edu

Claudio T. Silva
Scientific Computing and Imaging Institute
University of Utah - USA
E-mail: csilva@sci.utah.edu

L. Gustavo Nonato
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo - Brazil
E-mail: gnonato@icmc.usp.br

Keywords Image-based mesh generation · template-based mesh · quadrilateral mesh · mesh smoothing · mesh segmentation · Bézier patches

1 Introduction

Generating meshes from 2D digital images is an important problem, which has been investigated in several different contexts such as image representation and numerical simulation. Most methods for generating meshes from imaging data rely on schemes based on two-stages: image segmentation and mesh generation itself. The image segmentation stage is responsible for partitioning the image in well defined regions, which are then "meshed" in the mesh generation stage.

An alternative approach was adopted by the *Imesh* algorithm [21, 10] (Figure 1). The *Imesh* approach combines image segmentation and mesh generation into a single processing stage, requiring only a couple of parameters to trigger the meshing process directly from the input image. Moreover, *Imesh* is able to segment the mesh in accordance with image features, making it possible to identify and build a correspondence between regions of the image and partitions of the mesh. The *Imesh* output is a provably good quality triangulation which contains smaller triangles along the boundary of image regions, and larger triangles in their interior.

Triangle meshes have been extensively investigated by the meshing community, and their theoretical properties are now well understood [4]. In addition, algorithms for generating good triangle meshes of polygonal and curved planar domains, such as the one used by *Imesh*, have been proposed and implemented [28, 23]. In contrast, the generation of good-quality quadrilateral meshes is not so well understood [3]. Directly generating quadrilateral meshes from a

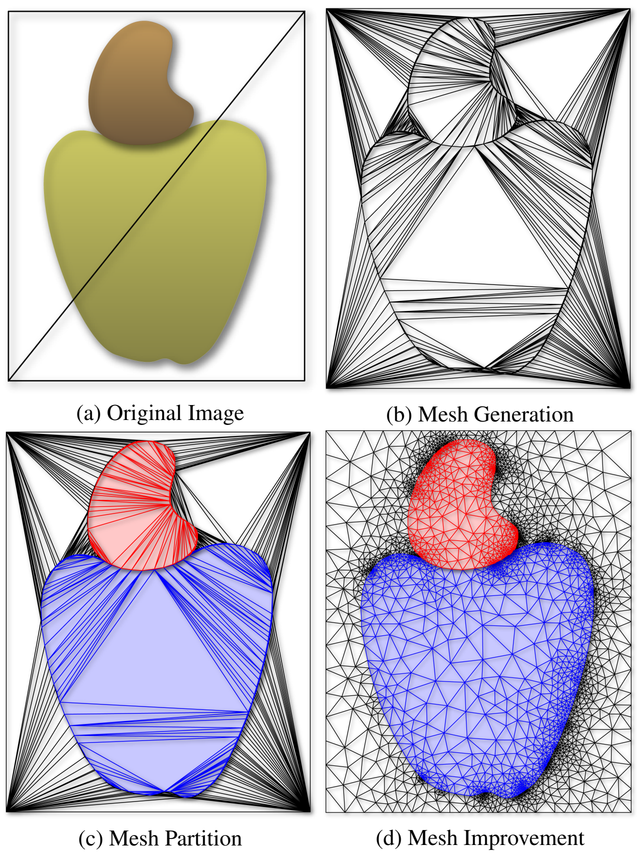

(a) Original Image (b) Mesh Generation (c) Mesh Partition (d) Mesh Improvement

Fig. 1 The original image and the three main steps of *Imesh*.

description of the domain is intrinsically harder than generating triangle meshes. Yet, quadrilateral meshes are more appropriate than triangle meshes for certain applications [2, 19,22].

1.1 Contributions

We describe an extension of *Imesh* which generates quadrilateral meshes directly from imaging data. Our extension combines two ingredients: a template-based triangulation-to-quadrangulation conversion strategy, and an optimization-based smoothing procedure. The former aims at generating quadrilateral meshes that respect the image object boundaries (as defined by the mesh partition step of *Imesh*), while the latter improves the quadrilateral shape quality.

The use of a template-based meshing approach makes it possible for *Imesh* to generate a quadrilateral mesh *indirectly*, i.e., from a triangle mesh rather than directly from a description of a polygonal domain. This indirect strategy makes the quadrilateral mesh generation task easier. Templates also enabled us to devise a novel and simple smoothing procedure to locally improve the quality of the quadrilaterals, while preserving the previously defined image object boundaries. Our experimental results indicate that the combination of our template-based mesh generation approach with the new smoothing procedure is very effective, rendering *Imesh* one of the few techniques to generate quadrilateral mesh *straightly* from images (i.e., without the need for an image segmentation or image object boundary delimitation step).

2 Related Work

In this section we summarize the main techniques devoted to generate meshes from images as well as to convert triangle meshes of planar domains into quadrilateral meshes. A comprehensive overview, mainly in the context of surfaces in 3D, is beyond the scope of this paper and can be found in [1].

2.1 Mesh generation from images

Techniques for generating meshes from digital images can be grouped into two main classes: mesh-based image representation and image modeling for simulation. *Mesh-based image representation techniques* build meshes that minimize the approximation error between the original image and the image represented by the mesh. In this class one finds adaptive methods, which iteratively refine the mesh until a lower-bound error is reached [14,13,9], mixed methods [18], and error diffusion schemes [37]. A main drawback of most mesh-based image representation methods is the use of interpolation error to guide the mesh generation process, which is not effective in textured and color images, impairing the use of such methods in a wide class of problems. *Image modeling for simulation techniques* divide the mesh generation process in two main steps: pre-processing and mesh generation. The pre-processing step aims at filtering and segmenting the image in order to detect regions of interest, which are meshed in the mesh generation step [8]. Binarization combined with implicit function reconstruction [5,38], pre-segmentation with Delaunay meshing [6], and shape deformation [33] figure among the most popular approaches, all of them relying on an image-segmentation stage.

2.2 Quadrilateral mesh generation

Generating a quadrilateral mesh of a polygonal domain is intrinsically harder than producing a triangular mesh. Indeed, if we require the set of vertices of the mesh to be the set of vertices of the input polygon, then a triangular mesh can always be obtained. In contrast, additional vertices may be necessary in order to generate a quadrilateral mesh. In addition, the theoretical properties to generate good quality quadrilateral meshes are not as well understood as the ones for producing good quality triangular meshes [3]. So, several researchers adopted an *indirect approach* to produce quadrilateral meshes [12,7,26,32,24,36,27]: a triangle mesh of the domain is generated, and then converted to a quadrilateral mesh.

The indirect approach relies on the premise that a quadrilateral mesh can be more easily generated from an existing triangle mesh of the target domain. Here, we adopt a two-stage indirect approach. First, we combine adjacent triangles to form quadrilaterals and produce a hybrid, triangle-quadrilateral mesh. Second, we convert the hybrid mesh into an all-quadrilateral mesh using *template subdivisions* of triangles and quadrilaterals. Template subdivisions of mesh elements have been used before for refining quadrilateral meshes [29] and respecting domain boundaries [3]. We use template subdivisions for mesh conversion and optimization purposes. To our best knowledge, this is the first work that exploits the potential of template subdivisions for both purposes.

3 Preliminaries

This section introduces basic concepts from Computational Geometry and Digital Topology, which are used in the description of the *Imesh* algorithm in the following section. We refer the reader to [34,28,16] for detailed discussions of those concepts.

Let S be a finite set of points in $\mathbb{R}^2$. A *triangulation*, $\mathscr{T}(S)$, of S is a set of triangles, along with their edges and vertices, such that (1) the set of vertices of $\mathscr{T}(S)$ is exactly S, and (2) the intersection of any two triangles, σ and τ, of $\mathscr{T}(S)$ is either empty or a common vertex or edge of σ and τ. The *underlying space*, $|\mathscr{T}(S)|$, of $\mathscr{T}(S)$ is the point set consisting of all points of $\mathbb{R}^2$ that belong to the triangles of $\mathscr{T}(S)$. Similarly, we define a *quadrangulation*, $\mathscr{Q}(S)$, of S as a set of quadrilaterals, along with their edges and vertices, such that (1) the set of vertices of $\mathscr{Q}(S)$ is exactly S, and (2) the intersection of any two quadrilaterals, μ and ν, of $\mathscr{Q}(S)$ is either empty or a common vertex or edge of μ and ν. The underlying space, $|\mathscr{Q}(S)|$, of $\mathscr{Q}(S)$ is the point set consisting of all points of $\mathbb{R}^2$ that belong to the quadrilaterals of $\mathscr{Q}(S)$. Hereafter, we will use *quad* as an abbreviation for quadrilateral.

A *Delaunay triangulation*, $\mathscr{DT}(S)$, of S is a triangulation of S such that (1) the underlying space, $|\mathscr{DT}(S)|$, of $\mathscr{DT}(S)$ is the convex hull of S (i.e., the smallest convex set that contains S), and (2) the interior of the circumcircle of every triangle of $\mathscr{DT}(S)$ does not contain any vertex of $\mathscr{DT}(S)$. Given a planar straight-line graph (PSLG), $G=(V,E)$, where V is a set of points of $\mathbb{R}^2$ and E is a set of line segments in $\mathbb{R}^2$ with endpoints in V, we define a *conforming Delaunay triangulation* of G as *any* Delaunay triangulation, $\mathscr{DT}(S)$, for some $S \subset \mathbb{R}^2$, such that (1) $V \subseteq S$ and (2) each edge e of E is an edge of $\mathscr{DT}(S)$ or a union of edges of $\mathscr{DT}(S)$. We say that $\mathscr{DT}(S)$ *conforms* to the vertices in V and edges in E (edges in E are called *constrained edges*).

As customary in Digital Topology, we call each $p \in \mathbb{Z}^2$ a *grid point*, and we regard p as the center of a grid square, denoted by $\square(p)$, with edges of unit length and oriented parallel to the Cartesian coordinate axes. We commonly refer to $\square(p)$ as a *pixel*. A *2D digital (multivalued) image* is a function $\mathscr{I} : \mathbb{G}_{n_1,n_2} \to C$ from a nonempty and finite subset of $\mathbb{Z}^2$, $\mathbb{G}_{n_1,n_2} = \{(g_1,g_2) \in \mathbb{Z}^2 \mid g_i \in [1,n_i], i=1,2\}$, to a nonempty and finite subset, C, of $\mathbb{R}$, where n_1 and n_2 are positive integers. The domain $\mathbb{G}_{n_1,n_2}$ of $\mathscr{I}$ is called a *2D grid* of size $n_1 \times n_2$. The elements of the co-domain C of $\mathscr{I}$ are called *colors*. So, the image $\mathscr{I}$ is a function that assigns a color $\mathscr{I}(p)$ from C to each $p \in \mathbb{G}_{n_1,n_2}$. The union set, $\bigcup_{p \in \mathbb{G}_{n_1,n_2}} \square(p)$, of all pixels whose centers are in $\mathbb{G}_{n_1,n_2}$ is the *continuous analog* of $\mathbb{G}_{n_1,n_2}$, which is denoted by $\square(\mathbb{G}_{n_1,n_2})$. By definition, we have that $\square(\mathbb{G}_{n_1,n_2})$ is a rectangle.

Given a 2D digital image, $\mathscr{I} : \mathbb{G}_{n_1,n_2} \to C$, we define a *triangle mesh* of $\mathscr{I}$ as *any* triangulation, $\mathscr{T}(S)$, for some finite subset S of $\mathbb{R}^2$, such that the underlying space, $|\mathscr{T}(S)|$, of $\mathscr{T}(S)$ is the continuous analog, $\square(\mathbb{G}_{n_1,n_2})$, of the image grid, $\mathbb{G}_{n_1,n_2}$. We can define a *quad mesh* of $\mathscr{I}$ in a similar way. In the following section we describe the *Imesh* algorithm for computing a triangulation $\mathscr{T}(S)$ from the given image, $\mathscr{I}$.

4 Imesh Overview

This section describes *Imesh*, an algorithm for directly generating a triangle mesh from a 2D digital image (see [10] and [21] for a more detailed description of the algorithm). *Imesh* is comprised of three steps: mesh generation, mesh partitioning, and mesh improvement, each of which is described in an individual section below. The input of the algorithm consists of a 2D digital image, $\mathscr{I} : \mathbb{G}_{n_1,n_2} \to C$, a *threshold* $t_e \in [0,1] \subset \mathbb{R}$ for an error measure, and a *classifier*, which is defined as a function, $g : \mathbb{G}_{n_1,n_2} \to L$, where L is a set of "labels". Each label can be viewed as an identifier for a group of interesting features and/or objects represented by the image. For any $p \in \mathbb{G}_{n_1,n_2}$, the value of g at p is determined by $\mathscr{I}(p)$ and the values of $\mathscr{I}$ in a neighborhood of p. The output of the algorithm is a partitioned triangle mesh of $\mathscr{I}$. Each mesh partition (i.e., a subset of triangles of the mesh) corresponds to image regions labeled the same by the classifier, g.

4.1 Mesh Generation

The first step of *Imesh* builds a Delaunay triangulation, $\mathscr{DT}(S)$, from a set S of points in $\square(\mathbb{G}_{n_1,n_2})$. This mesh is built in two stages (refer to Figure 1(a)-(b)). First, an initial Delaunay triangulation is constructed from a set S_0 with four points, namely, the top-left, top-right, bottom-left, and bottom-right vertices of the rectangle $\square(\mathbb{G}_{n_1,n_2})$. Second, the initial triangulation is iteratively refined by inserting more points from $\square(\mathbb{G}_{n_1,n_2})$ into the initial set S_0. The inserted points are approximately located at the boundary of the image regions defined by the classifier. In what follows we briefly describe the details of this iterative point insertion process.

By definition, the Delaunay triangulation $\mathscr{DT}(S_0)$ has only two triangles and four vertices. For every $i \in \mathbb{Z}$ with $i \geq 1$, the i-th iteration of the refinement stage finds exactly one point, say q_i, in $\square(\mathbb{G}_{n_1,n_2})$, inserts q_i into S_{i-1} to produce a set S_i, and then computes a new Delaunay triangulation, $\mathscr{DT}(S_i)$. To choose q_i, the algorithm considers one triangle σ of $\mathscr{DT}(S_{i-1})$ at a time. For each σ, the algorithm computes an *error measure* inside σ. If the error associated with σ is larger than the threshold, t_e, a point from the *interior* of σ is chosen as a candidate to be q_i. Among the candidate points the farther from the existing vertices is added to the triangulation by using an incremental insertion procedure for updating the Delaunay triangulation [15], $\mathscr{DT}(S_i)$ from $\mathscr{DT}(S_{i-1})$. The refinement stage ends when the error associated with every triangle of the current triangulation, $\mathscr{DT}(S_i)$, is no larger than the input threshold t_e.

The *error measure*, $e : \mathscr{T}_t(S) \to \mathbb{R}^*_+$, is a function from the set, $\mathscr{T}_t(S)$, of triangles of a point set triangulation, $\mathscr{T}(S)$, to the set of nonnegative reals. For each $\sigma \in \mathscr{T}_t(S)$, the value of $e(\sigma)$ is given in terms of the classifier g and all image pixels intersected by the three medians of σ. More specifically, let m_1^σ, m_2^σ, and m_3^σ be the three medians of σ. Then, we define the set $P_{m_j^\sigma}$ as made up of pixels $\square(p) \in \mathbb{G}_{n_1,n_2}$ such that (see red square in Figure 2): for each $j = \{1,2,3\}$, $\square(p) \in P_{m_j^\sigma}$ if (1) m_j^σ intersects $\square(p)$, and (2) there exists $r \in \mathbb{G}_{n_1,n_2}$ such that $r \neq p$, $g(p) \neq g(r)$, $m_j^\sigma \cap \square(r) \neq \emptyset$, and $\square(r) \cap \square(p)$ is a common vertex or edge of both squares (i.e., p and r are neighbors in $\mathbb{Z}^2$). Intuitively, the set $P_{m_j^\sigma}$ consists of every pixel $\square(p)$ intersected by m_j^σ and located around the boundary between two image regions labeled distinctly by g. Finally, we can define $e(\sigma)$ as follows: if $P_{m_j^\sigma} = \emptyset$, for all $j \in \{1,2,3\}$, then $e(\sigma) = 0$; otherwise, the error $e(\sigma)$ is equal to the maximum value of the following set:

$$\left\{ \alpha_\sigma(p) \in \mathbb{R}_+ \mid \square(p) \in \bigcup_{j=1}^{3} P_{m_j^\sigma} \right\},$$

where $\alpha_\sigma(p)$ is the smallest barycentric coordinate of point p with respect to the vertices of triangle σ containing p. So,

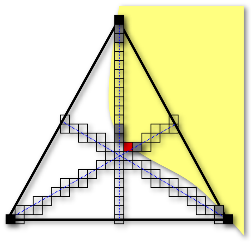


Fig. 2 Image pixels intersected by the three medians. Gray squares are pixels where the classifier g changes its value. The red square is the pixel that will be added to the Delaunay triangulation.

$e(\sigma) \in [0,1] \subset \mathbb{R}$. Note that the value of $e(\sigma)$ is related to how far the pixels in $P_{m_j^\sigma}$ are from the vertices of σ. The farther they are the larger the value of $e(\sigma)$, and the farther the boundary between two distinctly labeled regions of $\mathscr{I}$ is from a vertex of σ. So, if $e(\sigma)$ is "large", the aforementioned boundary is not faithfully approximated by an edge of σ. So, we insert a point q_i in the interior of σ whenever $e(\sigma)$ is larger than the predefined threshold t_e, with $0 \leq t_e \leq 1$. In particular, if $e(\sigma) > t_e$ then q_i is chosen to be any point p, with $\square(p) \in \bigcup_{j=1}^{3} P_{m_j^\sigma}$, such that $e(\sigma) = \alpha_\sigma(p)$ (see red square in Figure). Note that the error $e(\sigma)$ is a "normalized" value, as the error measure e was defined in terms of barycentric coordinates. Finally, note also that point q_i is always a point from $\mathbb{G}_{n_1,n_2}$. Since $\mathbb{G}_{n_1,n_2}$ is a finite set and the same point from $\mathbb{G}_{n_1,n_2}$ is never considered for insertion twice, the termination of the refinement stage is assured. If k is the last iteration, then we let $S = S_k$. From now on, we omit the set S, and denote $\mathscr{DT}(S)$ by simply $\mathscr{DT}$.

4.2 Mesh Partitioning

The mesh partitioning step generates a partition, $\mathscr{P}$, of the set of triangles, $\mathscr{DT}_t$, of the Delaunay triangulation, $\mathscr{DT}$, produced by the mesh generation step. To build $\mathscr{P}$, the *Imesh* algorithm makes use of a function, $h : \mathscr{DT}_t \to L$, which assigns a label from L to each triangle $\sigma \in \mathscr{DT}_t$. Two triangles, σ and τ, of $\mathscr{DT}_t$ belong to the same set $A \in \mathscr{P}$ iff $h(\sigma) = h(\tau)$. To compute $h(\sigma)$, *Imesh* considers the set A_σ of all points $p \in \mathbb{G}_{n_1,n_2}$ such that $\square(p)$ is intersected by a median of σ. Then, label $h(\sigma)$ is defined as the most frequent one among the labels (given by the classifier g) of all points in A_σ. If there is a tie, which may happen when t_e is a large value, *Imesh* picks any of the most frequent labels. Figure 1(c) shows the partitioning of the mesh in Figure 1(b).

Most algorithms for generating meshes from imaging data require an image segmentation preprocessing step [6]. The goal of this preprocessing step is to define a set of boundaries delimiting the distinct objects represented by the image. In contrast, *Imesh* bypasses the image segmentation step, as the distinct objects represented by the image are delimited as a result of the mesh partitioning step. So, in some sense, the image segmentation problem becomes a mesh partitioning problem, which is also a well-known problem [30]. The advantage of delimiting image objects while generating a mesh, as done by *Imesh*, is that there is no need for the mesh to strictly respect predefined boundaries.

4.3 Mesh Improvement

The mesh generated by *Imesh* in the first two steps is in general a mesh with poor-quality triangles, i.e., triangles with very small and/or very large angles. This kind of triangle is extremely unsuitable for many mesh-based applications [31]. As we can see in Figure 1(b), the reason why poor-quality triangles arise is two-fold. First, triangulation vertices are placed along the boundaries and features of the image objects only. Second, the length of triangulation edges along the boundary is in general very small compared to the length of the triangulation edges across the interior of the image objects.

To overcome the aforementioned poor-quality mesh problem, *Imesh* further refines the Delaunay triangulation, $\mathscr{DT}$. To do that, *Imesh* modifies $\mathscr{DT}$ by iteratively inserting points in the triangulation until all current triangles have good quality. Since the mesh has already been partitioned, the insertion of a new point into the triangulation must be carried out carefully, so that the boundaries found by the mesh partitioning step are preserved.

To ensure that boundaries are preserved while new points are inserted, *Imesh* makes use of an adaptation of *Ruppert's algorithm* for generating conforming Delaunay triangulations [28]. The input consists of $\mathscr{DT}$ and a PSLG $G = (V,E)$, where V is the subset of all vertices of $\mathscr{DT}$ incident to a constrained edge of $\mathscr{DT}$, and E is the set of constrained edges of $\mathscr{DT}$. An edge e of $\mathscr{DT}$ is said to be *constrained* iff e belongs to only one triangle of $\mathscr{DT}$ or e is shared by two triangles of $\mathscr{DT}$, each of which belongs to a distinct set of $\mathscr{P}$. In other words, set E contains the edges of $\mathscr{DT}$ that delimit the distinct image objects. The output of Ruppert's algorithm is a conforming Delaunay triangulation, $\mathscr{DT}^*$, of G such that the domains of $\mathscr{DT}^*$ and $\mathscr{DT}$ are the same (i.e., $|\mathscr{DT}^*| = |\mathscr{DT}|$) and $\mathscr{DT}^*$ has no poor-quality triangle.

Triangulation $\mathscr{DT}'$ is also iteratively constructed. In particular, it is the last triangulation of a finite sequence, $(\mathscr{DT}_i)_{i=1}^n$, of conforming Delaunay triangulations of G, where $\mathscr{DT}_1 = \mathscr{DT}$, and $\mathscr{DT}_i$ is obtained from $\mathscr{DT}_{i-1}$ by a point insertion operation. More precisely, for each $i \in \{2,\dots,n\}$, the triangulation $\mathscr{DT}_i$ is obtained from $\mathscr{DT}_{i-1}$ by the insertion of the circumcenter of a "poor-quality" triangle of $\mathscr{DT}_{i-1}$ into the vertex set of $\mathscr{DT}_{i-1}$. However, if the circumcenter of a poor-quality triangle σ lies in a triangle τ such that $h(\sigma) \neq h(\tau)$ (or outside the image domain), then it is not inserted into the vertex set of $\mathscr{DT}_{i-1}$. In this case, the circumcenter belongs to the diametral circle of one or more edges of $\mathscr{DT}_{i-1}$, which in turn belong to the boundary of two distinctly labeled partitions (or to the boundary of the image domain). So, instead of the circumcenter, the midpoint of such edges are computed and inserted in the vertex set of $\mathscr{DT}_{i-1}$.

A triangle of $\mathscr{DT}_i$ is said to have *poor-quality* if it contains an unconstrained angle less than a *quality threshold*, t_q, where t_q must be no larger than 26.4°. An *unconstrained angle* is an internal triangle angle defined by two edges that are not both constrained edges of $\mathscr{DT}_i$. All triangulations $\mathscr{DT}_i$ in $(\mathscr{DT}_i)_{i=1}^n$ have at least one poor-quality triangle, except for $\mathscr{DT}_n$. An upper bound of 26.4° for the quality threshold, t_q, and some further conditions discussed in [10] ensure that $(\mathscr{DT}_i)_{i=1}^n$ is indeed finite and that $\mathscr{DT}^*$ has no poor triangle (see [23] for a detailed discussion on the upper bound of 26.4° for t_q). Figure 1(d) shows the result of the mesh-improvement step of *Imesh* applied to the mesh in Figure 1(c).

5 Quadrilateral Meshing

We now describe our new approach for generating good-quality quadrilateral meshes from imaging data. As we mentioned in Section 2.2, our approach generates a quad mesh *indirectly*, i.e., by converting a given triangle mesh into a quad one. In particular, the quad mesh is generated in two stages, namely: *template-mapping* and *optimization*. The former stage converts a given triangle mesh into a quad one, while the latter stage improves the shape quality of the quads generated by the former. The result is a good-quality quad mesh.

The template-mapping stage consists of three main steps: *boundary and mesh simplification*, *triangle pairing*, and *template-based subdivision*. In turn, the optimization stage consists of two main steps: *boundary adaptation* and *relaxation*. In what follows we give an overview of all steps, and then describe the details of each of them in an individual section.

5.1 Overview

The template-mapping stage starts by pairing up triangles of the given triangle mesh in a greedy manner. This pairing process follows two simple rules: (1) each triangle of the given triangle mesh can belong to at most one pair, and (2) the two triangles of each pair must share an edge, and must belong to the same mesh partition. Next, the common edge of each pair of triangles is removed from the original triangle mesh (see Figure 6(b) and Figure 3). The resulting mesh

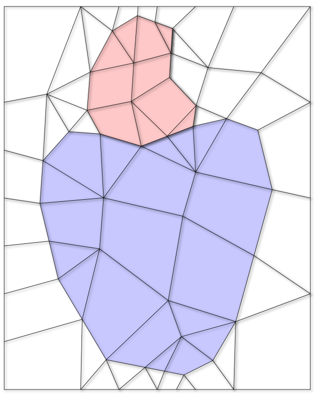


Fig. 3 A mesh obtained by pairing up triangles of the mesh in Figure 6(b).

may be either a quad-only mesh or a *hybrid* mesh consisting of quads and (unpaired) triangles of the given triangle mesh. Finally, each quad and each unpaired triangle (if any) is subdividided into several quads using a subdivision defined template meshes. The resulting mesh is a quad-only mesh of the same domain as the given triangle mesh (see Figure 7(a)).

The pairing and subdivision procedures described above can be applied to any triangle mesh. So, one may find it tempting to consider the output, good-quality triangle mesh of *Imesh* as the input triangle mesh of the template-mapping stage. Unfortunately, the final triangle meshes produced by *Imesh* are not suitable for our purposes. The reason for that is two-fold. First, since each quad and (unpaired) triangle resulting from the pairing procedure are subdivided into several quads, the resulting quad-only mesh could have an unnecessarily large number of quads. Second, *Imesh* tends to generate unnecessarily small triangles along the boundary curves separating adjacent mesh partitions (see Figure 1(d)). As a result of the pairing and subdivision procedures, those triangles would give rise to unnecessarily small quads as well.

The presence of unnecessarily small triangles has to do with the way *Imesh* inserts new vertices into the initial triangulation during its mesh generation step. In particular, these vertices are not guaranteed to be placed along a smooth (and imaginary) curve nor are they guaranteed to be distributed according to the curvature variation of this curve. So, the curves defining the boundary of the mesh partitions may be "jagged" or unnecessarily sampled in some regions. Since Ruppert's algorithm is sensitive to vertex proximity and local curvature variation, the mesh improvement step of *Imesh* tends to create small triangles along the more jagged or overly sampled regions of the curves to ensure mesh quality.

To avoid the two problems described above, we consider the Delaunay triangulation, $\mathscr{DT}$, resulting resulting from the *mesh partition* step of *Imesh* as the input triangle mesh of

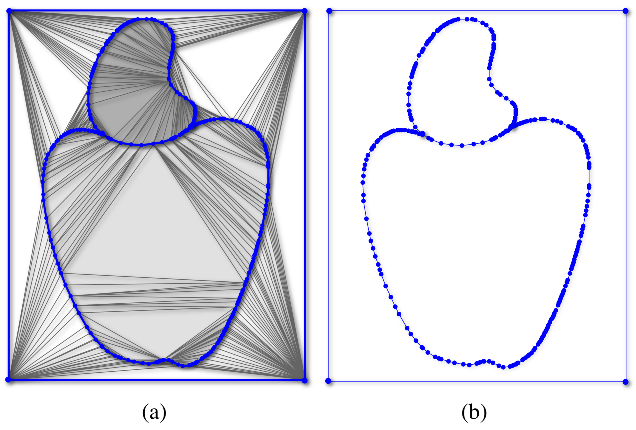


Fig. 4 (a) The Delaunay triangulation resulting from the mesh partition step of *Imesh*. (b) The PSLG defined by its constrained edges and their vertices.

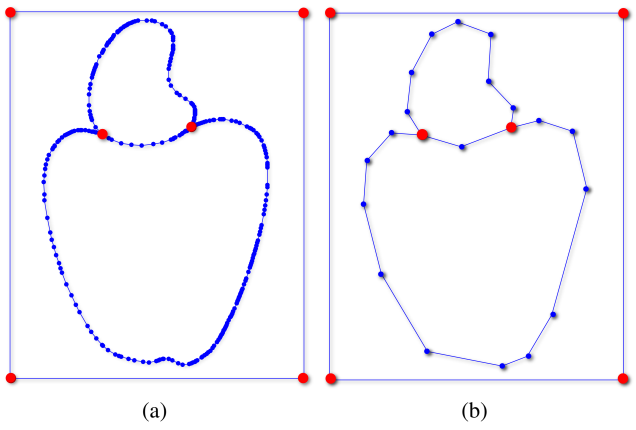


Fig. 5 PLSG in Figure 4(b) before (a) and after (b) simplification (red colored circles represent end points of the polygonal curves generated by our preprocessing).

the template-mapping stage. On the one hand, $\mathscr{DT}$ is usually much coaser than the triangle mesh generated by the mesh improvement step of *Imesh*. On the other hand, the triangles of $\mathscr{DT}$ tend to have poor-quality (i.e., they are very thin and elongated), and the boundary curves are also jagged or unnecessarily sampled in some regions (see Figure 1(c)). However, we can more easily modify $\mathscr{DT}$ to create a triangle mesh that works well with our pairing and subdivision procedures.

To create a new triangle mesh from $\mathscr{DT}$, we first consider the PSLG defined by its constrained edges. An edge of $\mathscr{DT}$ is said to be *constrained* iff it is adjacent to only one triangle (i.e., it belongs to the image boundary) or to two distinctly labeled triangle. Let $G = (V, E)$ be the PSLG such that V consists of all vertices of $\mathscr{DT}$ that are incident to a constrained edge of $\mathscr{DT}$, and E consists of all constrained edges of $\mathscr{DT}$. Figure 4(a) shows a triangulation $\mathscr{DT}$ resulting from the mesh partition step *Imesh*, while Figure 4(b)

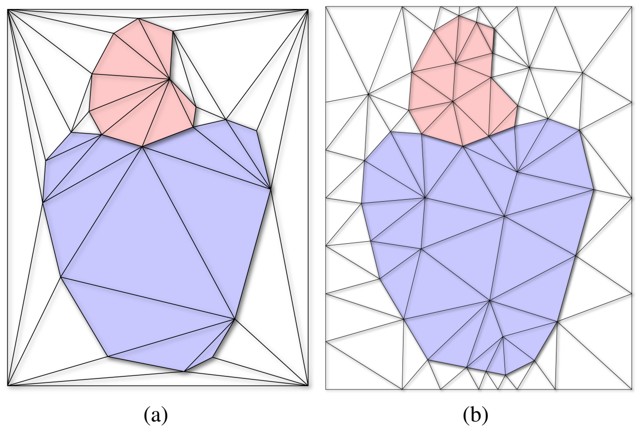


Fig. 6 (a) A Delaunay triangulation that conforms to the PSLG in Figure 5(b). (b) A better quality Delaunay triangulation that also conforms to the PSLG in Figure 5(b).

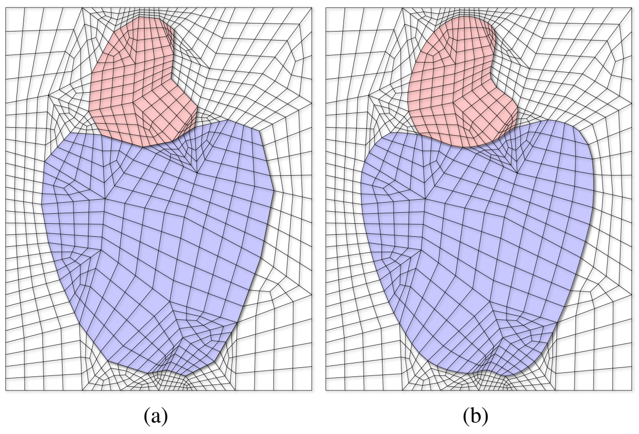


Fig. 7 (a) A template-based quad mesh that conforms to the PSLG in Figure 5(b). (b) A template-based quad mesh that approximates the edges of the PSLG in Figure 5(a).

shows the PSLG G obtained from the triangulation in Figure 4(a).

Once we have G, we simplify the polygonal curves corresponding to chains of edges of G, and then generate a Delaunay triangulation that conforms to the graph defined by the vertices and edges of the simplified curves. The simplification yields curves with longer edges (see Figure 5(b)), preventing the triangulation from having thin and elongated triangles (see Figure 6). If the triangulation still contains very poor-quality triangles, these triangles are eliminated by point insertion, just like in the mesh improvement step. For instance, the triangulation in Figure 6(b) was obtained from the one in Figure 6(a) by iteratively inserting points in the latter.

After obtaining a suitable triangle mesh, we effectively start the pairing and subdivision procedures. If the pairing procedure leaves unpaired triangles, these triangles are subdivided into a small, fixed number of quads using a template (see Figures 8(a)-(b)). As a result we obtain an all-quad mesh (see Figure 7(a)). Later, quad vertices are moved toward the original constrained edges (the ones from G) in order to better adapt the quadrilateral mesh to the original object boundaries (see Figure 7(b)). Finally, an optimization-based smoothing is executed to improve mesh quality (see Figure 10).

5.2 Boundary and Mesh Simplification

Consider the PSLG $G = (V, E)$ given as input for the quadrilateral meshing step, that is, edges and vertices in G are shared by triangles from different regions of the partitioned mesh (or they are incident to image boundary edges). Recall that each edge in E is a constrained edge, which we call a *c-edge*. The goal of the boundary simplification step is to simplify the polygonal curves defined by the set of all c-edges and their vertices. To do that we used a well-known line simplification algorithm [11]. This algorithm can only handle simple, open polygonal curves. However, the set of all c-edges defines polygonal "curves" that are not necessarily simple nor open (i.e., they are closed and may form T-junctions). So, we preprocess the set of all c-edges in order to define a set of maximally longer, simple , and open polygonal curves (see Figure 5(a)), and then execute the aforementioned line simplification algorithm on the resulting curves.

Formally, let $C_G = V \cup E$. We wish to partition C_G into a set of *maximal* polygonal curves. A *polygonal curve c from* C_G is a path or a cycle in G such that each vertex of a c-edge of c is incident to at most another c-edge in c, that is, each polygonal curve in C_G is simple. A polygonal curve c is maximal if it is not properly contained by another polygonal curve.

Note that c is an *open* polygonal curve iff c is not a cycle. Note also that every c-edge in E must belong to exactly one maximal polygonal curve. We devised an algorithm for partitioning C_G into maximal polygonal curves. The algorithm starts by picking an arbitrary edge e from E. Initially, all edges in E are said to be *unmarked*. Then, a chain c of edges in E is grown from e. To do so, the algorithm considers one of the two end vertices of c at a time. Let v be such a vertex. If v is not an image boundary corner and v is incident to exactly two edges of E, one in c and the other being an unmarked edge not in c, then the edge not in c, say e', is marked and added to c, and so is its vertex not in c (if c is now a cycle, both vertices of e' are already in c). Otherwise, the algorithm considers the other end vertex of c. If c cannot grow further, all edges of E in c are already marked, and an unmarked edge from E is selected by the algorithm to build a new chain. If there is no unmarked edge in E, the algorithm ends.

Upon termination, every chain c is either an open or a closed, simple polygonal curve from C_G. For each closed curve, we choose an arbitrary edge e in c, and then remove e from c, creating an open curve. Next, the line simplification algorithm is executed on each open polygonal curve, c, producing a simplified curve c' from c. Curve c' approximates c and its vertex set is a subset of the vertex set of c. It is important to point out that the line simplification algorithm provides error bounds that allow us to precisely drive the simplification [11]. As a result we obtain a set of simplified c-edges, which defines a PSLG, $G' = (V', E')$, such that V' and E' are the vertex and edge sets of all simplified curves, respectively. The edges and vertices of G' also delimit the image objects, as shown in Figure 5(b). Finally, a conforming Delaunay triangulation, $\mathscr{DT}'$, is generated from G' (see Figure 6(a)).

It might be the case that $\mathscr{DT}'$ contains some poor-quality triangles (see Figure 6(a)). If so, before executing the triangle pairing step, we run Ruppert's algorithm on $\mathscr{DT}'$ in order to remove poor-quality triangles (see Figure 6(b)). However, to avoid creating a dense triangulation (which would cause the resulting quad-only mesh to be overly dense), we relax the quality measure threshold of the algorithm and limit the number of point insertions. Unfortunately, we noticed that there is no value for this threshold that is suitable for every possible input mesh. Besides, we still have not implemented an automatic way of finding the suitable value for each mesh. In our experiments, we have manually tuned and selected the quality threshold, t_q, for each quad mesh we generated. In Section 6, we offer a discussion on how to automatically set a value to t_q, for any given $\mathscr{DT}'$.

5.3 Triangle Pairing and Template-Based Subdivision

The goal of the template-based subdivision stage is to generate a quad mesh from the previously computed conforming Delaunay triangulation, $\mathscr{DT}'$. To that end, adjacent triangles of $\mathscr{DT}'$ are paired up using a straightforward modification of the *triquad* procedure from [35]. The modified procedure maintains a max-heap H of ordered pairs, (e,k), where e is an *unconstrained* edge from $\mathscr{DT}'$ and k is the length of e. Initially, all edges in H are said to be unmarked. The procedure removes one pair (e,k) at a time from the top of the heap. If e is unmarked, then the two triangles of $\mathscr{DT}'$ sharing e are paired up to form a quad, and all unconstrained edges of this quad are marked. However, if the quad is not strictly convex, the pairing is discarded. The procedure ends when H is empty, and it may leave several unpaired triangles.

Let $\mathscr{M}$ be the collection of triangles and quads resulting from the pairing procedure. Regardless of whether $\mathscr{M}$ consists of quads only, each triangle or quad in $\mathscr{M}$ is subdivided into a small and fixed amount of quads to produce an all-quad mesh. The subdivision of a triangle (or a quad)

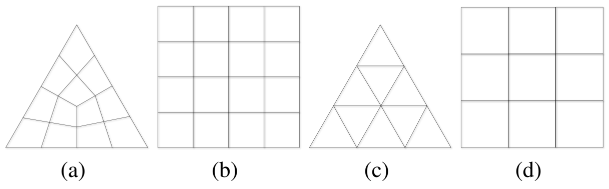


Fig. 8 Templates for the canonical (a) triangle and (b) square subdivisions, and the uniform subdivisions of the canonical (c) triangle and (d) square for defining the control net of triangular and rectangular Bézier patches.

into quads is based on the templates shown in Figures 8(a)-(b). More specifically, each template is a fixed subdivision of a canonical triangle or square, which is then mapped by an affine map or a bilinear map to triangles or quads in $\mathscr{M}$, respectively. After mapping the canonical templates to the triangles and quads in $\mathscr{M}$, we obtain a quadrangulation, $\mathscr{Q}$, that *conforms* to the edges of G' and respect the triangle mesh partition, i.e., every edge of G' is either an edge or a union of edges of $\mathscr{Q}$. Furthermore, we have that $|\mathscr{Q}| = |\mathscr{DT}'|$ and that the partition is preserved (see Figure 7(a)).

5.4 Optimization-Based Smoothing

The final stage of the remeshing step carries out two interrelated tasks. First, the quad mesh, $\mathscr{Q}$, resulting from the previous stage is adapted to the image object boundaries, i.e., to the original polygonal curves from C_G (see Section 5.2). By adapting, we mean to move mesh vertices toward the original polygonal curves of C_G. As a result the quad mesh of each mesh partition element faithfully approximates its corresponding region defined by the PSLG G. Second, the quality of the adapted mesh quads is improved by a new optimization-based relaxation. Both tasks are related with each other by the fact that we adapt and optimize the mesh by moving mesh vertices with the guidance of Bézier surface patches.

5.4.1 Boundary Adaptation

Recall that every quad of $\mathscr{Q}$ belongs to either a triangle or a quad from the mesh $\mathscr{M}$ resulting from the triquad procedure (see Section 5.3). To move the vertices of $\mathscr{Q}$, we assign a triangular Bézier patch b_σ of total degree 3 to each triangle σ in $\mathscr{M}$. Similarly, we assign a rectangular Bézier patch b_μ of bi-degree $(3,3)$ to each quad μ in $\mathscr{M}$. The patch b_σ (resp. b_μ) is responsible for guiding the movement of the mesh vertices of the quads inside triangle σ (resp. quad μ). Since $\mathscr{Q}$ is a planar mesh, and since each b_σ (resp. b_μ) is a mapping from a canonical triangle (resp. square) in $\mathbb{R}^2$ to $\mathbb{R}^3$, we regard all vertices of $\mathscr{Q}$ as points in the xy plane of $\mathbb{R}^3$, and set the z coordinate of all control points of b_σ

(resp. b_μ) to 0. So, vertex movements are constrained to the xy plane.

To compute the control points of each b_σ, we distinguish two cases. If σ does not contain any c-edge of G', then we uniformly subdivide the canonical triangle t associated with b_σ as shown in Figure 8(c), and let the control points of b_σ be the image of the subdivision vertices under the affine map that takes t onto σ. If σ contains a c-edge of G', then we proceed as in the previous case, and then consider each c-edge e of σ. From the simplification stage, c-edge e is an edge of a simplified polygonal curve from $C_{G'}$, which corresponds to a chain of consecutive edges of a polygonal curve, say l, from C_G. So, we redefine the control points of b_σ in e in such a way that the boundary (Bézier) curve of b_σ closely approximates l as shown in Figure 9. The same is done for each b_μ, except for the facts that if μ does not contain a c-edge, then we uniformly subdivide the canonical square q associated with b_μ (see Figure 8(d)), and let the control points of b_μ be the image of the subdivision vertices under the bilinear map that takes q onto μ. In what follows b_τ denotes both b_σ and b_μ.

As we saw in Section 5.2, a c-edge can share a vertex with one or more c-edges. If exactly two c-edges meet at a given vertex, we assume that their corresponding Bézier curves should meet with tangent continuity at that vertex. The only exception is when the vertex is explicitly marked as a "sharp" corner. In this case, we assume that the corresponding Bézier curves should meet with C^0-continuity only. Currently, we manually mark the vertices we consider sharp corners. But, for the time being, this fact is not important. Whenever three or more c-edges meet at a vertex, we assume that the vertex is a sharp corner, and proceed as before. In what follows, we discuss how to redefine the control points of the boundary Bézier curves in order to achieve tangent continuity. The case for C^0-continuity is similar (and easier).

First, assume that exactly two c-edges meet at every vertex of G', and that no vertex is marked as a "sharp" corner. Then, to redefine the control points of the boundary Bézier curve of b_τ associated with *each* c-edge e, we solve a curve fitting problem, which can be regarded as a *least squares with equality constraints* (LSE) problem. More specifically, let

$$b_\tau^e : [0,1] \to \mathbb{R}^2$$

be the boundary cubic Bézier curve of b_τ associated with e. Then,

$$b_\tau^e(t) = \sum_{i=0}^{3} B_i^3(t) \cdot b_i^e \,,$$

for every $t \in [0,1]$, where $B_i^3(t)$, for every $i \in \{0,1,2,3\}$, is the i-th Bernstein polynomial of degree 3. Now, given a list $(t_i)_{i=0}^n$ of $n+1$ *parameter* values in $[0,1] \subset \mathbb{R}$, where $n \in \mathbb{N}$, $n \geq 4$, $t_0 = 0$, $t_n = 1$, and $t_j < t_{j+1}$, for every $j \in \{0,\ldots,n-1\}$, a list $(p_i)_{i=0}^n$ of $n+1$ points in $\mathbb{R}^2$, and two unit vectors, $\boldsymbol{n}_0^e$ and $\boldsymbol{n}_3^e$, our fitting problem consists of finding the "best" $b_\tau^e(t)$ (in the least squares sense) satisfying two conditions: (1) the control point b_0^e (resp. b_3^e) of $b_\tau^e(t)$ is the point p_0 (resp. p_n), and (2) the control point b_1^e (resp. b_2^e) must lie in the line by b_0^e (resp. b_3^e) and perpendicular to $\boldsymbol{n}_0^e$ (resp. $\boldsymbol{n}_3^e$). So, we are left with the problem of finding the control points b_1^e and b_2^e.

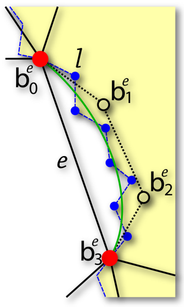


Fig. 9 A Bézier patch that locally approximates a polygonal curve.

To solve the above fitting problem, we define two systems of linear equations, $K\boldsymbol{x} = \boldsymbol{f}$ and $C\boldsymbol{x} = \boldsymbol{d}$. The former system is assembled from $n+1$ equations in four unknowns, namely:

$$\sum_{j=1}^{2} B_j^3(t_i) \cdot b_j^e = p_i - (B_0^3(t_i) \cdot b_0^e + B_3^3(t_i) \cdot b_3^e) \,,$$

for all $i \in \{0,1,\ldots,n\}$, where the four unknowns are $(b_{1,x}^e, b_{1,y}^e)$ and $(b_{2,x}^e, b_{2,y}^e)$, the x and y coordinates of b_1^e and b_2^e, respectively. More specifically, matrix K has $(2n+1)$ rows and 4 columns, and the $(2i+1)$-th and $(2i+2)$-th rows of K are

$$\begin{bmatrix} B_1^3(t_i) & B_2^3(t_i) & 0 & 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 0 & 0 & B_1^3(t_i) & B_2^3(t_i) \end{bmatrix} \,,$$

respectively. In turn, vectors $\boldsymbol{f}$ and $\boldsymbol{x}$ have $2n+1$ and 4 elements, respectively. The $(2i+1)$-th and $(2i+2)$-th elements of $\boldsymbol{f}$ are

$$p_{i,x} - (B_0^3(t_i) \cdot b_{0,x}^e + B_3^3(t_i) \cdot b_{3,x}^e)$$

and

$$p_{i,y} - (B_0^3(t_i) \cdot b_{0,y}^e + B_3^3(t_i) \cdot b_{3,y}^e) \,,$$

where $p_{i,x}$, $b_{0,x}^e$, and $b_{3,x}^e$ (resp. $p_{i,y}$, $b_{0,y}^e$, and $b_{3,y}^e$) are the x (resp. y) coordinates of p_i, b_0^e, and b_3^e. Finally, vector $\boldsymbol{x}$ is given by

$$[\, b_{1,x}^e \quad b_{2,x}^e \quad b_{1,y}^e \quad b_{2,y}^e \,]^{\mathrm{T}} \,.$$

In turn, the linear system $C\boldsymbol{x} = \boldsymbol{d}$ is assembled from two linear equations in two unknowns each, both of which arise from

$$\langle \boldsymbol{n}_0^e, b_1^e - b_0^e \rangle = 0 \quad \text{and} \quad \langle \boldsymbol{n}_3^e, b_3^e - b_0^e \rangle = 0 \,,$$

where $\langle \cdot , \cdot \rangle$ denotes the dot product in $\mathbb{R}^2$. In particular, we have

$$C = \begin{bmatrix} n^e_{0,x} & 0 & n^e_{0,y} & 0 \\ 0 & n^e_{3,x} & 0 & n^e_{3,y} \end{bmatrix}, \quad \boldsymbol{d} = \begin{bmatrix} \langle \boldsymbol{n}^e_0, b^e_0 \rangle \\ \langle \boldsymbol{n}^e_3, b^e_3 \rangle \end{bmatrix},$$

where $(n^e_{0,x}, n^e_{0,y})$ and $(n^e_{3,x}, n^e_{3,y})$ are the coordinates of $\boldsymbol{n}^e_0$ and $\boldsymbol{n}^e_3$, respectively. Note that $C\boldsymbol{x} = \boldsymbol{d}$ represents the *constraints*: b^e_1 (resp. b^e_2) is a point in the line by b^e_0 (resp. b^e_3) and perpendicular to $\boldsymbol{n}^e_0$ (resp. $\boldsymbol{n}^e_3$). Since $K\boldsymbol{x} = \boldsymbol{f}$ is overdetermined, we solve our problem by finding the vector $\boldsymbol{x}$ that *minimizes* $\|K\boldsymbol{x} - \boldsymbol{f}\|_2$ subject to $C\boldsymbol{x} = \boldsymbol{d}$. To that end we used the direct-elimination method as described in Chapter 21 of [20].

For our purposes, the set of points $(p_i)^n_{i=0}$ is simply the set of vertices of the polygonal chain l associated with the c-edge e, while the set of parameter values, $(t_i)^n_{i=0}$, is obtained by a chord length parametrization of the vertices $(p_i)^n_{i=0}$ of l over e:

$$t_0 = 0 \quad \text{and} \quad t_j = t_{j-1} + \frac{\|p_j - p_{j-1}\|}{\|p_n - p_0\|},$$

for $j \in \{1, \ldots, n\}$. In turn, $\boldsymbol{n}^e_0$ and $\boldsymbol{n}^e_3$ are defined in such a way that b^e_τ meets its adjacent curves in a tangent continuous manner. More specifically, each polygonal curve c from C_G is the union of one or more polygonal chains, each of which corresponds to a c-edge e of G'. So, the entire curve c is approximated by a set of Bézier curves. We assumed that the boundary Bézier curves should meet each other with tangent continuity at the endpoints of each polygonal chain l (see Figure 9).

To enforce tangent continuity, we proceed as follows: let e' be a c-edge in G' adjacent to e at $b^e_0 = b^{e'}_3$, and let ν be a triangle (or quad) of $\mathscr{M}$ incident to e'. Then, b^e_τ and $b^{e'}_\nu$ will meet at $b^e_0 = b^{e'}_3$ with tangent continuity iff b^e_1, $b^e_0 = b^{e'}_3$, and $b^{e'}_2$ are points along the same line in $\mathbb{R}^2$. So, we choose $\boldsymbol{n}^e_0 = \boldsymbol{n}^{e'}_3$. Similarly, we let $\boldsymbol{n}^e_3 = \boldsymbol{n}^{e''}_0$, where e'' is the c-edge that meets e at $b^e_3 = b^{e''}_0$. To compute $\boldsymbol{n}^e_0$ (and $\boldsymbol{n}^{e'}_3$), we consider the vectors, $\boldsymbol{m}_e$ and $\boldsymbol{m}_{e'}$, perpendicular to e and e', respectively, and oriented in a consistent manner (i.e., with respect to an orientation given to curve c). Next, we let $\boldsymbol{n}^e_0$ (and $\boldsymbol{n}^{e'}_3$) be the vector

$$\frac{1}{2} \cdot (\boldsymbol{m}_e + \boldsymbol{m}_{e'}) / \left\| \frac{1}{2} \cdot (\boldsymbol{m}_e + \boldsymbol{m}_{e'}) \right\|.$$

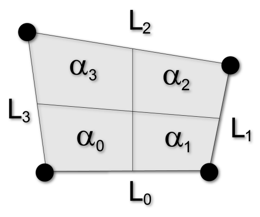
We compute $\boldsymbol{n}^e_3 = \boldsymbol{n}^{e''}_0$ in the same manner. The above strategy is well-defined, except for the case in which an endpoint of e is a meeting point of three or more edges of G'. But, then, the vertex is a sharp corner, and we will deal with this case later.

From the definition of the control points of the Bézier patch b_σ (resp. b_μ) associated with each triangle σ (resp. quad μ) of $\mathscr{M}$, the boundary Bézier curves of adjacent patches are exactly the same (i.e., they have the same control points). This is obviously true for the boundary curves associated with c-edges. For the curves associated with edges that are not constrained, our claim follows from the facts that (1) the curve control points are placed along the edges, and (2) they are images of canonical triangle (resp. quad) subdivision vertices under an affine (resp. bilinear) map. But, both affine and bilinear maps preserve distance ratio along a line.

Now, let us consider the case in which a vertex is a sharp corner. A vertex is a sharp corner if it is the meeting point of three or more c-edges, or if it is the meeting point of exactly two c-edges, but it has been explicitly marked as a sharp corner. In either case, we only require the curves to meet with C^0-continuity. Fortunately, all we have to do is to choose an arbitrary normal to each c-edge at the common vertex, and then solve the same problem as before. In our implementation, we choose one of the two normals of the c-edge itself to be the normal at its endpoints. Since the normals defined by the c-edges at any meeting vertex are not necessarily the same, the solution of the LSE problem ensures C^0-continuity only.

5.4.2 Optimization

The goal of the optimization task is to improve the *shape quality* of the quads of $\mathscr{Q}$. Recall that each quad of $\mathscr{Q}$ is the image of a quad defined in a canonical triangle or quad in $\mathscr{M}$. Triangular and rectangular Bézier patches are also defined on the canonical domain. So, each vertex of $\mathscr{Q}$ can be written in terms of the Bézier patches and by imposing that adjacent Bézier patches share the same cubic Bézier curve, one can modify the quad mesh $\mathscr{Q}$ by moving the control points of the patches.



To improve the shape quality of the quads of $\mathscr{Q}$, we judiciously move the control points of all Bézier patches to improve the shape quality of all quads with respect to the quadrilateral shape quality measure called *Shape and Size* [17]. Let q be a quadrilateral of Q. Then, the Shape and Size metric for q is $A_q \cdot S_q$, where A_q is the area of q and S_q is defined by the formula

$$S_q = 2 \cdot \min\left(\frac{\alpha_0}{\|L_0\|^2 + \|L_3\|^2}, \frac{\alpha_1}{\|L_1\|^2 + \|L_0\|^2}, \frac{\alpha_2}{\|L_2\|^2 + \|L_1\|^2}, \frac{\alpha_3}{\|L_3\|^2 + \|L_2\|^2} \right),$$

where α_i is the area of the i-th quadrilateral in the figure above, and L_j is the length of the j-side of q, for $i, j =$

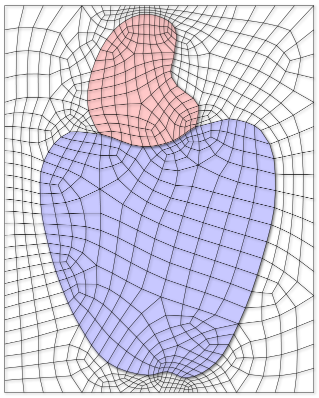


Fig. 10 An optimized template-based quad mesh.

$1,2,3,4$. The subdivision shown in the figure is obtained by addding line segments connecting the barycenter of q to the midpoints of its sides. In general, we can view the shape measure as a function, $s : \mathscr{Q}_q \to [0,1]$, where $\mathscr{Q}_q$ is the set of quads of $\mathscr{Q}$. Function s is defined in such a way that for each quad $v \in \mathscr{Q}_q$, the larger the value of $s(v)$ the better the quality of v. Therefore, the optimal positioning of the control points can be found by minimizing the following energy function:

$$q_s = \sum_{v \in \mathscr{Q}'_q} (1 - s(v))^2 ,$$

To this end, we used Powell's method [25] defined on the space of the coordinates of the control points. Because the total number of control points is smaller than the number of vertices in $\mathscr{Q}$, the proposed optimization mechanism turns out to be more effective than directly using the coordinates of the quad vertices.

In order to avoid folding (inversion of a quad element), a control point movement must be *feasible*, i.e., every control point must be in the interior of the polygon defined by its adjacent control points in the Bézier control net. This constraint is imposed when applying Powell's method. Our procedure is iterative and runs until q_s is below a predefined threshold or the number of iterations exceeds another predefined threshold. Figure 10 shows the mesh resulting from applying the optimization mechanism to the mesh of Figure 7(b).

6 Experimental Results

This section presents and discusses the results of applying our new template-based quadrilateral meshing approach to five images, which are called Lake, Pyramid, Hub Wheel, Portico and Wrench (see Figure 11). Lake and Pyramid belong to the Berkeley Segmentation Dataset[1]. Hub Wheel is a range image from the Stuttgart Range Image Database[2]. Portico is a photograph from the portico of the city of Gramado, RS, in Brazil. Wrench is an artificial image we generated for this paper.

All experiments were conducted in a PC with one Intel Core i7-975 3.3GHz processor with 3GB RAM. To evaluate the quality of the meshes generated by our new approach, we used an implementation of the *Shape and Size* quadrilateral quality metric available in the VERDICT library[3].

The colored squares in Figures 11(a) and 11(b) correspond to samples used by the built-in texture classifier of *Imesh*. Each color represents a distinct image region of interest (label). Five and four distinct regions of interest are defined in Figures 11(a) and 11(b), respectively. Since the three regions in Figure 11(c) and Figure 11(e) can easily be identified by thresholding, there is no need for using a texture classifier. Since a ground truth segmentation was available for Figure 11(d), we decided to use it so as to show how well boundaries can be preserved by *Imesh*. In order to preserve the sharp corners in Figures 11(d) and 11(e), we only required the Bézier boundary curves meet with C^0-continuity at the corresponding vertices (see Section 5.4.1 for details).

Figure 12 shows the quadrilateral meshes resulting from applying our approach to the images in Figure 11. The quad meshes were generated from the triangle meshes in Figure 13. Observe that the segmentation provided by *Imesh* is naturally preserved by the proposed quadrilateral meshing step. Moreover, as a consequence of the boundary adaptation and smoothing mechanisms, curves between distinctly labeled regions are precisely represented. Furthermore, the "jagged" effect that usually appears in triangle meshes generated by the original version of *Imesh* is not present in the quad meshes.

Figure 15 shows histograms from quality measurements involving the meshes presented in Figure 12. The vertical dashed line represents the lower bound (value equal to 0.2) for quadrilateral shape quality measure. A quad element with quality measure value below this lower bound is considered of poor-quality according to the Shape and Size measure [17]. Reddish and blue vertical lines correspond to element quality measures before and after the smoothing step, respectively. Note that the proposed optimization-based smoothing mechanism was able to improve mesh quality considerably, avoiding bad elements altogether. This fact can also be observed in the fourth column of Table 1, which presents the quality measure of the worst element of the quad meshes in Figure 10 and Figure 12. The fifth and sixth columns confirm the effectiveness of the smoothing mechanism, showing that, on average, the quality measure is above 0.8.

The number of vertices and quadrilaterals of the meshes presented in Figures 10 and 12 are in the first and second

[1] http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/
[2] http://range.informatik.uni-stuttgart.de/htdocs/html/
[3] VERDICT – http://cubit.sandia.gov/verdict.html

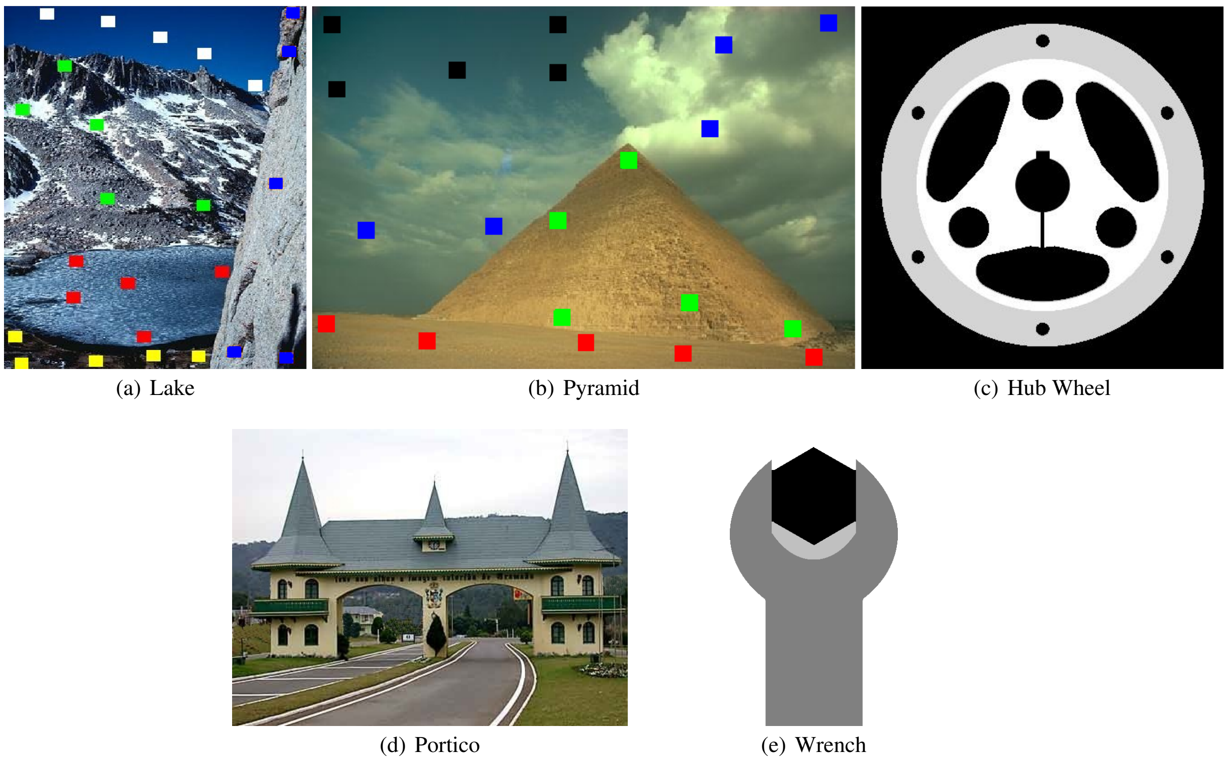


Fig. 11 Images used in our experiments. Colored squares in (a) and (b) correspond to samples used by the built-in texture classifier of *Imesh* (each color corresponds to a label).

columns of Table 1. The computational times in the third column are quite acceptable for an approach that generates good quality quadrilateral meshes. We remark that the optimization-based smoothing is the most time-consuming step of our approach, representing 98% of the total computational time.

The ability of representing image region boundaries and adapting the size of mesh elements locally are important features of our quadrilateral meshing approach. Such features can easily been seen in Figure 16, which shows quad meshes with adaptively sized elements for the Hub Wheel image (Figure 11(c)). Even for the coarser mesh, which contains about 1.2K vertices, our meshing approach was able to satisfactorily represent image region boundaries. It was also able to adapt quad sizes to capture the small chink in the lower part of the Hub. It is worth mentioning that finer quad meshes may be obtained by simply adding new triangles in $\mathscr{DT}'$ before triggering the pairing process and the template mapping.

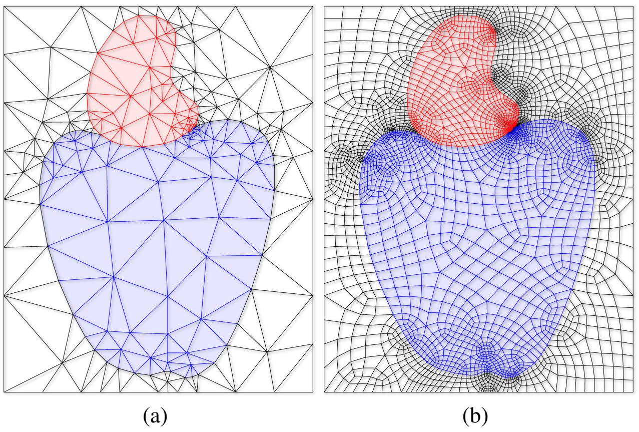


Fig. 14 (a) Delaunay triangulation from a PSLG obtained from Figure 5(a) using one pixel as boundary simplification parameter. (b) The final quad-mesh generated from (a).

6.1 Parameters

We carefully chose the amount of boundary simplification in order to preserve the topology of the image region boundaries (i.e., the PSLG G and G' must remain isomorphic). To enforce topology preservation, we set a lower bound for the number of pixels contained in each image region delimited by the simplified curves. In Figure 14(a), we show the cashew setting one pixel of distance in the boundary simplification step. In Figure 14(b), shows the resulting quad mesh obtained from the triangulation depicted in Figure 14(a).

We set the lower bounds of 9, 4, 4, 4, 2 and 9 pixels for the images Cashew, Lake, Pyramid, Hub Wheel, Portico and Wrench, respectively. We used the 4-subdivision templates (Figures 8(a)-(b)) in all examples, except for the Hub Wheel

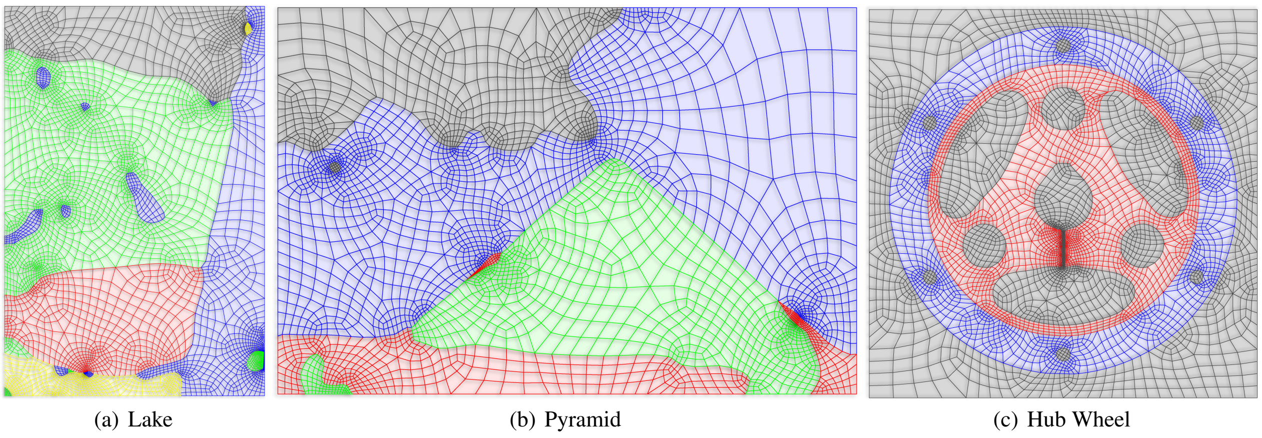
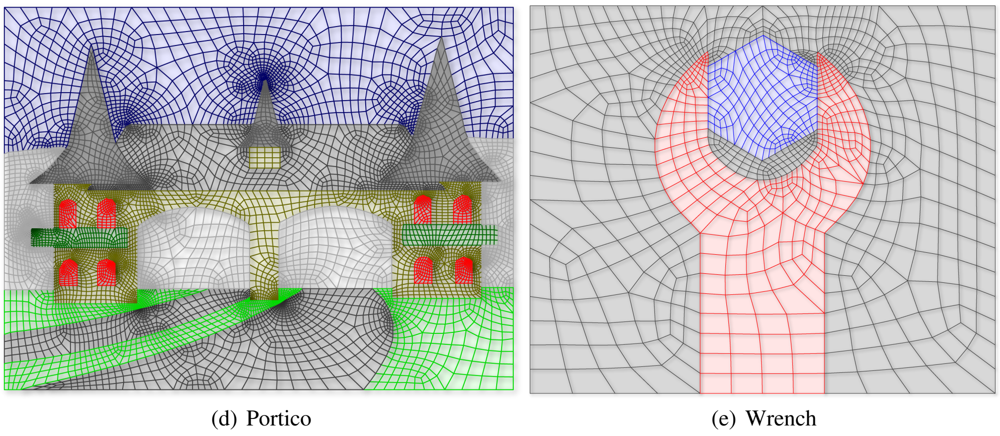



Fig. 12 Quad meshes resulting from the images in Figure 11. The meshes are already segmented, avoiding any post-processing step.

with 1.2K quads, where we used 2-subdivision templates. Finally, to roughly define the density of each mesh, we set Ruppert's refinement quality threshold to 20.7°, 20.7°, 20.7°, 18°, 18°, 26.5°, 30°, 20.7° and 20.7° for the Cashew, Lake, Pyramid, Hub Wheel images (with 1.2K, 5K, 7K and 10K quads), Portico and Wrench, respectively.

Table 1 Number of vertices and quads (first and second columns), computational times (third column), and quality measures (fourth to sixth columns) – minimum (min), mean values and standard deviation (s.d.) – for the meshes in Figures 10 and 12.

	# vert	# quads	meshing (seconds)	min	mean	s. d.
Cashew	1173	1120	1.8	0.227	0.868	0.088
Lake	4735	4588	17.8	0.201	0.873	0.089
Pyramid	2991	2916	7.7	0.215	0.876	0.098
Hub	7307	7220	29.7	0.211	0.900	0.076
Portico	8977	8880	36.6	0.202	0.872	0.094
Wrench	1123	1084	2.1	0.269	0.861	0.085

We conclude our discussion showing a comparison between our quadrilateral meshing approach and the quad meshing algorithm, called `CQMesh`, described in [27]. `CQMesh` generates quad meshes of planar domains defined

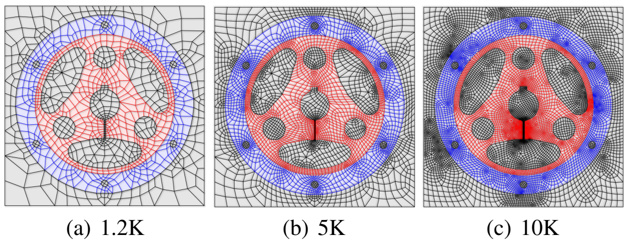
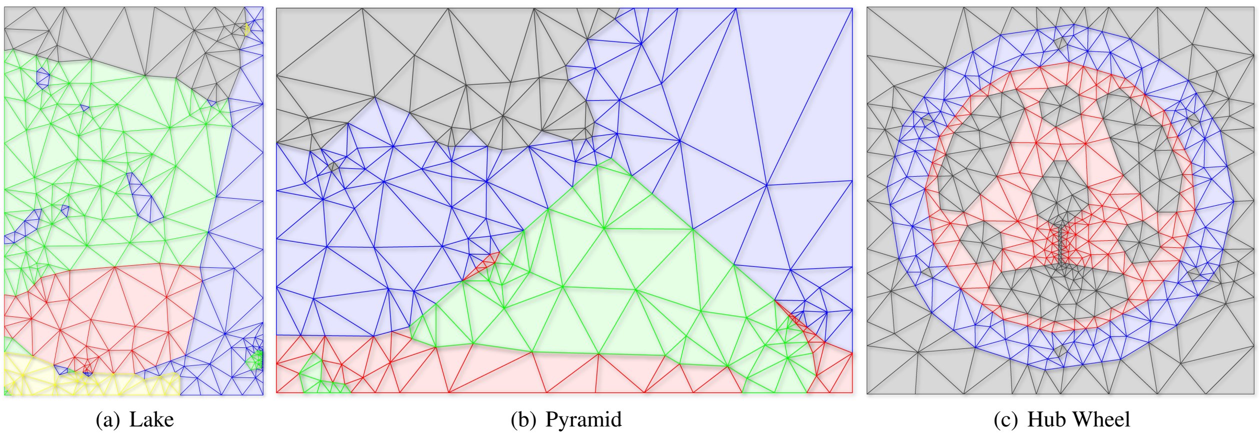


Fig. 16 Quad meshes generated from the Hub Wheel image: a) 1.2K vertices, b) 5K vertices, and c) 10K vertices.

by PLSGs by applying a clustering-based indirect approach coupled with Laplacian smoothing as a post-processing step. Figure 17 shows the 1.2K Hub Wheel quad mesh produced by both approaches. Since most vertices in that coarse mesh are constrained by the boundary curves, the control point-based optimization barely affects the mesh quality. Even so, our proposed approach was able to produce a better quality quad mesh, as one can see in Figure 17(a). The computational times, in seconds, to generate the meshes in Figure 16 are in the second column of Table 2. Computational times for `CQMesh` are shown in the third column. Quality measures are presented in the two last columns. As we can see,



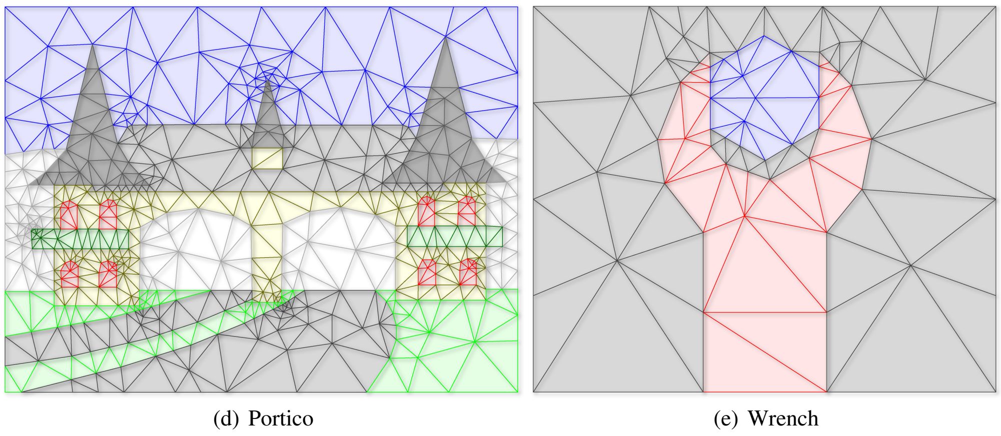


Fig. 13 Triangle meshes from which the quads meshes in Figure 12 were generated.

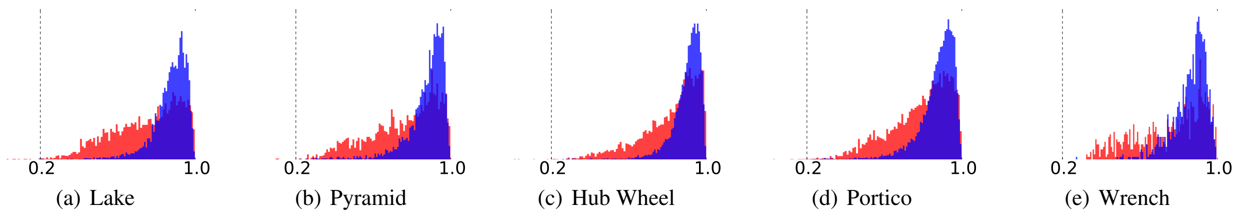


Fig. 15 Quality histograms for the meshes in Figure 12. Vertical dashed lines indicate the lower bound below which a quad element is considered of bad quality. Reddish and blue vertical lines correspond to mesh quality measures before and after the smoothing step, respectively.

Table 2 Computational times (in seconds) and quality measure of the worst quality mesh elements produced by *Imesh* and CQMesh from the Hub Wheel image.

# cells	*Imesh* times	`CQMesh` times	*Imesh* quality	`CQMesh` quality
1.2k	3.4	1.7	0.276	0.08
5k	16.9	13.9	0.211	0.04
10k	39,2	23.0	0.212	0.03

although slower than `CQMesh`, our quadrilateral meshing approach was able to produce meshes with superior element quality.

We finally remark that the template-based meshing strategy used by our approach can be naturally extended to generate triangle meshes from imaging data as well. All we have to do is to use templates made out of triangles instead of quads. Figure 18 shows a triangle mesh of the Hub Wheel image which was produced with templates made out of triangles. It is worth noticing that the boundary simplification and mesh optimization steps of our quad meshing approach were used as they are. The ability of generating triangle meshes with no extra effort certainly adds to the flexibility of the template-based meshing strategy of our approach. However, we need to further investigate the quality of the gen-

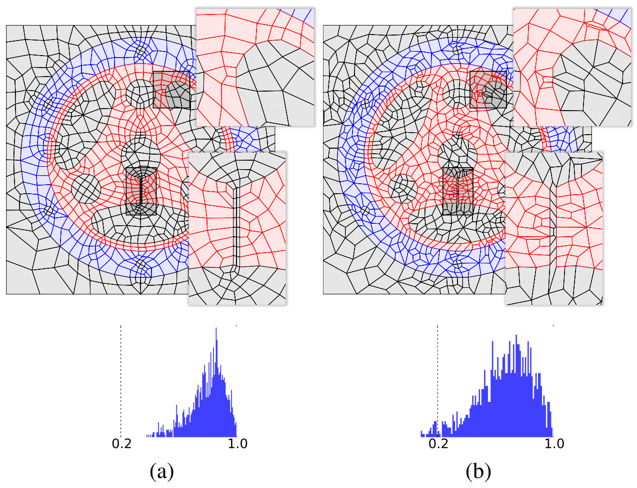


Fig. 17 Comparison between our new approach (a) and the quadrilateral meshing algorithm in [27].

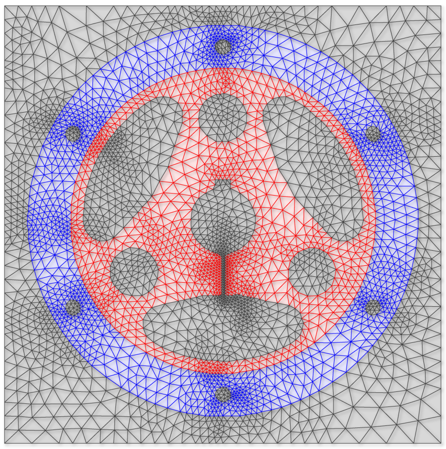


Fig. 18 A triangle mesh produced with templates made out of triangles.

erated triangle meshes in order to incorporate the template-based strategy into *Imesh* to also generate triangle meshes.

6.2 Limitations

Although our proposed approach can produce good-quality quadrilateral meshes, it currently has a few limitations. As we mentioned earlier in Section 5.2, we do not have an automatic procedure for selecting a value for the quality threshold, t_q, of Ruppert's algorithm *in the template-mapping step of our approach*. We experimentally noticed that a value in the range $[18^o, 21^o]$ was able to eliminate poor-quality triangles without producing an excessively fine triangle mesh. In fact, a value from this range was used by all examples presented in this paper, except for the very dense Hub Wheel meshes shown in Figure 16(b) and Figure 16(c). In both cases, we intentionally generated dense quadrilateral meshes. We believe, however, that it is possible to design an automatic procedure to select an appropriate value for t_q. We intend to investigate the possibility of defining a spacing function from the PLSG G' or from a quadtree background mesh [39].

Our proposed approach does not come with any theoretical guarantee on the quality of the produced meshes. In particular, we cannot assure that a better quad mesh will result from a denser and better-quality triangle mesh. However, we believe that a more powerful triangle-pairing procedure could enable us to establish a relation between the quality of the input triangle mesh and the quality of the output quad mesh. Indeed, in a recent work [40], we designed a quality metric for pairing triangles on a triangle *surface* mesh. The proposed metric was combined with a graph-based approach for pairing triangles. This approach allows us to find the pairing that maximizes the total sum of the metric values among all possible pairings. The same approach can be adapted to dealing with the planar case. It remains to be shown that a better input triangle mesh yields a better optimal pairing.

7 Conclusions

This paper described a novel approach for generating quadrilateral meshes from imaging data, which can be regarded as a replacement for the mesh improvement step of *Imesh*. Our approach is able to generate quad-only meshes, while preserving boundaries and regions defined in the first steps of *Imesh*. To produce good quality meshes, our approach is accompanied by a new optimization-based smoothing procedure, which moves mesh vertices around guided by changes of control points of Bézier patches defined on the mesh domain. By changing the location of mesh vertices via control points, our procedure reduces computational effort. We generated quadrilateral meshes from several kinds of digital images. The meshes produced by our approach indicate that our smoothing procedure is also quite effective.

In addition to generating good quality quadrilateral meshes, our approach also produces segmented meshes as a by-product of the mesh generation process. To the best of our knowledge, no other meshing approach generates good quality, segmented quad meshes *directly* from images (i.e., without the need for a preprocessing image segmentation step).

We are currently investigating how to tailor templates so as to reduce the number of extraordinary vertices (i.e., vertices with valence other than four) in the final quad mesh. We intend to incorporate a corner detection algorithm to the boundary adaptation and simplification step, so that we can automatically decide which Bézier curves should meet with C^0 continuity at common vertices. We are also interested in proving lower and upper bounds for the internal angles of the resulting quadrilaterals. Finally, we are looking into ways of extending some of the ideas presented here to approaches for generating hexahedral meshes from 3D imaging data.

Acknowledgments

This work was partially supported by grants from Fapesp-Brazil (#2008/03349-6) and CNPq-NSF (#491034/2008-3), and it was carried out in the framework of the *Instituto Nacional de Ciência e Tecnologia em Medicina Assistida por Computação Científica* (CNPq, Brazil). The second author was partially supported by CNPq (grants #307450/2009-0 and #474710/2010-6) during the development of this work.

References

1. Alliez, P., Ucelli, G., Gotsman, C., Attene, M.: Recent advances in remeshing of surfaces. In: L.D. Floriani, M. Spagnuolo (eds.) Shape Analysis and Structuring, Mathematics and Visualization. Springer Berlin Heidelberg (2008)
2. Allman, D.: A quadrilateral finite element including vertex rotations for plane elasticity analysis. International Journal for Numerical Methods in Engineering **26**, 717–730 (1988)
3. Atalay, F.B., Ramaswami, S., Xu, D.: Quadrilateral meshes with bounded minimum angle. In: Proceedings of the 17th International Meshing Roundtable (IMR), pp. 73–91 (2008)
4. Bern, M., Plassmann, P.: Mesh generation. In: J.R. Sack, J. Urrutia (eds.) Handbook of Comput. Geom. Elsevier (2000)
5. Berti, G.: Image-based unstructured 3d mesh generation for medical applications. In: ECCOMAS - European Congress On Computational Methods in Applied Sciences and Engeneering (2004)
6. Boissonnat, J.D., Pons, J.P., Yvinec, M.: From segmented images to good quality meshes using delaunay refinement. Lecture Notes in Computer Science **5416**, 13–37 (2009)
7. B.P. Johnston, J.S., Kwasnik, A.: Automatic conversion of triangular finite meshes to quadrilateral meshes. International Journal for Numerical Methods in Engineering **31**(1), 67–84 (1991)
8. Cebral, J., Lohner, R.: From medical images to cfd meshes. In: Proc. of the 8th Intern. Meshing Roundtable, pp. 321–331 (1999)
9. Coleman, S., Scotney, B.: Mesh modeling for sparse image data set. In: IEEE ICIP, pp. 1342–1345. IEEE Computer Society (2005)
10. Cuadros-Vargas, A.J., Lizier, M.A.S., Minghim, R., Nonato, L.G.: Generating segmented quality meshes from images. Journal of Mathematical Imaging and Vision **33**, 11–23 (2009)
11. Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a line or its caricature. The Canadian Cartographer **10**(2), 122–122 (1973)
12. Everett, H., Lenhart, W., Overmars, M., Shermer, T., Urrutia, J.: Strictly convex quadrilateralizations of polygons. In: Proceedings of the 4th Canadian Conference on Computational Geometry, pp. 77–82 (1992)
13. García, M., Sappa, A., Vintimilla, B.: Efficient approximation of gray-scale images through bounded error triangular meshes. In: IEEE Intern. Conf. on Image Processing, pp. 168–170 (1999)
14. Gevers, T., Smeulders, A.: Combining region splitting and edge detection through guided delaunay image subdivision. In: IEEE Proceedings of CVPR, pp. 1021–1026 (1997)
15. Green, P., Sibson, R.: Computing dirichlet tesselation in the plane. Computer Journal **21**(2), 168–173 (1977)
16. Herman, G.: Geometry of Digital Spaces. Birkhäuser: Boston, MA, USA (1998)
17. Knupp, P.M.: Algebraic mesh quality metrics. SIAM Journal on Scientific Computing **23**(1), 193–218 (2001)
18. Kocharoen, P., Ahmed, K., Rajatheva, R., Fernando, W.: Adaptive mesh generation for mesh-based image coding using node elimination approach. In: IEEE International Conference on Image Processing, pp. 2052–2056 (2005)
19. Lai, M.J.: Scattered data interpolation and approximation using bivariate c^1 piecewise cubic polynomials. Computer Aided Geometric Design **13**(1), 81–88 (1996)
20. Lawson, C.L., Hanson, R.J.: Solving Least Squares Problem, *Classics in Applied Mathematics*, vol. 15. SIAM (1995)
21. Lizier, M.A.S., Martins-Jr, D.C., Cuadros-Vargas, A.J., Cesar-Jr, R.M., Nonato, L.G.: Generating segmented meshes from textured color images. Journal of Visual Communication and Image Representation **20**, 190–203 (2009)
22. Malanthara, A., Gerstle, W.: Comparative study of unstructured meshes made of triangles and quadrilaterals. In: Proceedings of the 6th International Meshing Roundtable (IMR), pp. 437–447 (1997)
23. Miller, G., Pave, S., Walkington, N.: When and why ruppert's algorithm works. In: Proceedings of the 12th International Meshing Roundtable (IMR), pp. 91–102 (2003)
24. Owen, S., Staten, M., Cannan, S., Saigal, S.: Q-morph: An indirect approach to advancing front quad meshing. International Journal for Numerical Methods in Engineering **9**(44), 1317–1340 (1999)
25. Powell, M.J.D.: An efficient method for finding the minimum of a function of several variables without calculating derivatives. The Computer Journal **7**(2), 155–162 (1964)
26. Ramaswami, S., Ramos, P., Toussaint, G.: Converting triangulations to quadrangulations. Computational Geometry: Theory and Applications **9**(4), 257–276 (1998)
27. Ramaswami, S., Siqueira, M., Sundaram, T., Gallier, J., Gee, J.: Constrained quadrilateral meshes of bounded size. Intern. J. of Comput. Geometry & Applications **15**(1), 55–98 (2005)
28. Ruppert, J.: A delaunay refinement algorithm for quality 2-dimensional mesh generation. J. of Algorithms **18**(3), 548–585 (1995)
29. Schneiders, R.: Refining quadrilateral and hexahedral element meshes. In: Proc. 5th International Conference on Numerical Grid Generation in Computational Field Simulations, pp. 679–688 (1996)
30. Shamir, A.: A survey on mesh segmentation techniques. Computer Graphics Forum **27**(6), 1539–1556 (2007)
31. Shewchuk, J.: What is a good linear element? interpolation, conditioning, and quality measures. In: Proceeding of the 11th International Meshing Roundtable, pp. 115–126 (2002)
32. Shimada, K., Liao, J.H., Itoh, T.: Quadrilateral meshing with directionality control through the packing of square cells. In: Proceedings of the 7th International Meshing Roundtable, pp. 61–76. Dearborn, Michigan, USA (1998)
33. Skrinjar, O., Bistoquet, A.: Generation of myocardial wall surface meshes from segmented mri. International Journal of Biomedical Imaging **2009** (2009)
34. Staples, J., Eades, P., Katoh, N., Moffat, A. (eds.): No Quadrangulation is Extremely Odd, *Lecture Notes in Computer Science*, vol. 1004. Cairns, Australia (1995)
35. Velho, L., Zorin, D.: 4-8 subdivision. Computer Aided Geometric Design **18**(5), 397–427 (2001)
36. Viswanath, N., Shimada, K., Itoh, T.: Quadrilateral meshing with anisotropy and directionality control via close packing of rectangular cells. In: Proceedings of the 9th International Meshing Roundtable, pp. 217–225. New Orleans, Louisiana, USA (2000)
37. Yang, Y., Wernick, M., Brankov, J.: A fast approach for accurate content-adaptive mesh generation. IEEE Transactions on Image Processing **12**(8), 866–881 (2003)

38. Zhang, Y., Bajaj, C., Sohn, B.S.: Adaptive and quality 3d meshing from imaging data. In: SM'03: Proceedings of the eighth ACM symposium on Solid modeling and applications, pp. 286–291 (2003)
39. Teng, S-H., Wong, C.W.: Unstructured mesh generation: theory, practice, and perspectives. International Journal of Computational Geometry and Applications **10**(3), 227–266 (2000)
40. Daniels, J., Nonato, L.G., Siqueira, M., Liziér, M., Silva, C.T. In: SMI'11: Proceedings of the Shape Modeling International (*to appear*)