International Journal of Computational Geometry & Applications
© World Scientific Publishing Company

CONSTRAINED QUADRILATERAL MESHES OF BOUNDED SIZE

SUNEETA RAMASWAMI*
Department of Computer Science, Rutgers University, 321 Business and Science Building
Camden, NJ 08102, USA
rsuneeta@camden.rutgers.edu

MARCELO SIQUEIRA†
TESSA SUNDARAM‡
JEAN GALLIER†
JAMES GEE‡

†*Department of Computer and Information Science, University of Pennsylvania*
Levine Hall, 3330 Walnut Street
Philadelphia, PA 19104, USA
marcelos@seas.upenn.edu
jean@cis.upenn.edu

‡*Department of Radiology, University of Pennsylvania, 3600 Market Street, Suite 370*
Philadelphia, PA 19104, USA
tessa@mail.med.upenn.edu
gee@rad.upenn.edu

Received (received date)
Revised (revised date)
Communicated by (Name)

We introduce a new algorithm to convert triangular meshes of polygonal regions, with or without holes, into strictly convex quadrilateral meshes of small bounded size. Our algorithm includes all vertices of the triangular mesh in the quadrilateral mesh, but may add extra vertices (called Steiner points). We show that if the input triangular mesh has t triangles, our algorithm produces a mesh with at most $\lfloor \frac{3t}{2} \rfloor + 2$ quadrilaterals by adding at most $t + 2$ Steiner points, one of which may be placed outside the triangular mesh domain. We also describe an extension of our algorithm to convert constrained triangular meshes into constrained quadrilateral ones. We show that if the input constrained triangular mesh has t triangles and its dual graph has h connected components, the resulting constrained quadrilateral mesh has at most $\lfloor \frac{3t}{2} \rfloor + 4h$ quadrilaterals and at most $t + 3h$ Steiner points, one of which may be placed outside the triangular mesh domain. Examples of meshes generated by our algorithm, and an evaluation of the quality of these meshes with respect to a quadrilateral shape quality criterion are presented as well.

Keywords: Quadrilateral mesh; triangulation; the finite-element method.

*The first and second author's research were partially supported by NSF Grant No. CCR-0204293.

1. Introduction

Finite element (FE) analysis is a powerful tool for numerically solving differential equations of variational problems that arise during structural modeling in engineering and the applied sciences. An essential prerequisite for the use of FE analysis is the availability of a mesh over the problem domain. If the problem domain is a subset of the Euclidean plane, triangular or quadrilateral meshes are typically employed. The accuracy of a problem's solution, and the efficiency with which it is obtained using a particular FE implementation are highly dependent on a variety of mesh parameters, such as the number of elements of the mesh and their shape, as well as the regularity, directionality and grading of the mesh [1,2].

Triangular meshes have been extensively investigated by the meshing community, and their theoretical properties are now well understood [3]. Algorithms for generating provably good triangular meshes of polygonal domains have been proposed and implemented [4,5,6,7]. On the other hand, the generation of good quadrilateral meshes is not as well understood. A few algorithms exist to generate quadrilateral meshes of bounded size [8,9,10,11,12], bounded largest angle [10], and controlled density and directionality [13,14,15] for polygonal domains. However, there are no known algorithms to generate quadrilateral meshes of polygonal domains that are provably guaranteed to simultaneously meet several quality criteria. Algorithms to generate such meshes would have great practical value because it has been shown that quadrilateral meshes are more desirable for certain FE-based applications, such as planar stress-strain analysis [16].

The input to two-dimensional meshing algorithms is typically a *planar straight line graph* (PSLG) that defines a polygonal region, possibly with polygonal holes, and an additional set of vertices and edges in its interior [3] (see Fig. 1(a)–(b)). A PSLG is a set of vertices and edges that satisfies two constraints. First, if the PSLG contains an edge then it must also contain the two vertices that are the endpoints of the edge. Second, edges can only intersect each other at their shared endpoints. Meshing algorithms that take PSLGs as input are supposed to generate meshes that *conform to* the input PSLG. In other words, the subset of points of the Euclidean plane covered by the output mesh — the *mesh domain* — is exactly the same as the subset of points of the polygonal region defined by the PSLG, and the set of vertices and edges of the PSLG are respectively included in and covered by the set of vertices and edges of the output mesh. Fig. 1(c) shows a triangular mesh that conforms to the polygonal region and its corresponding PSLG in Fig. 1(a)–(b).

The need for constructing meshes of polygonal regions that also conform to some vertices and edges in their interior often arises in practice. For instance, finite element analysis of two-dimensional biological shapes often requires the construction of meshes from segmented two-dimensional images, whose domain is appropriately represented in the Euclidean plane by polygonal regions with vertices and edges in their interiors [17]. Several meshing algorithms can only handle PSLGs that describe polygonal regions with or without polygonal holes. That is, the vertices and edges of the PSLG are exactly the vertices and edges of the (boundary of the) polygonal region defined by it. However, if the PSLG also defines vertices and edges in the interior of the polygonal region then such meshing algorithms cannot guarantee that these vertices and edges will be included in the output mesh. Some exceptions are the algorithms by Chew [4], Ruppert [5], and Shewchuk [6,7], which are capable of generating triangular meshes for general PSLGs. Hence, their algorithms are particularly useful for building triangular meshes from imaging data of biological shapes.

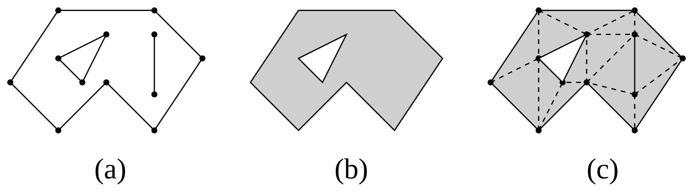

Fig. 1. (a) A PSLG. (b) A polygonal region defined by the PSLG in (a). (c) A triangular mesh that conforms to the polygonal region in (b) and its corresponding PSLG in (a). The edges of the PSLG are shown as solid line segments, and the other edges of the mesh are dashed line segments.

In this paper we are primarily concerned with the generation of strictly convex quadrilateral meshes that are suitable for FE analysis of problems in which directional features are not present or relevant for the analysis accuracy. *Strictly convex quadrilateral meshes* are quadrilateral meshes in which each of the four angles of every quadrilateral is strictly less than 180^o. Strictly convex quadrilateral meshes are the only desirable quadrilateral meshes for FE-based applications [18,2]. The main contributions of our work are two-fold:

(1) We introduce a new algorithm for converting triangular meshes into strictly convex quadrilateral meshes of provably small size. The output quadrilateral mesh contains all vertices in the input triangular mesh, and may contain extra vertices — *Steiner points* — inserted by our algorithm during the conversion process, one of which may be placed outside the triangular mesh domain. In particular, we show that if the input triangular mesh has t triangles then the output quadrilateral mesh generated by our algorithm has at most $\lfloor \frac{3t}{2} \rfloor + 2$ quadrilaterals, obtained by inserting at most $t + 2$ Steiner points. These results improve upon previously known bounds on quadrilateral mesh size [8]. Our algorithm runs in $\mathcal{O}(t)$ time and space.
(2) We also show that given a triangular mesh and a constraining subset of its edges, our algorithm can convert the input triangular mesh into a strictly convex quadrilateral mesh whose set of edges covers the constraining set of edges. Furthermore, we show that if the input triangular mesh has t triangles then the output quadrilateral mesh generated by our algorithm has at most $\lfloor \frac{3t}{2} \rfloor + 4h$

quadrilaterals, obtained by inserting at most $t + 3h$ Steiner points, where h is the number of connected components in the dual graph of the input triangular mesh. One of the Steiner points may be placed outside the triangular mesh domain. Note that this result implies that our algorithm can *indirectly* handle the same class of PSLGs as the algorithms by Chew [4], Ruppert [5], and Shewchuk [6,7]. We just have to use one of these algorithms to generate the triangular mesh, and then define the constraining subset of edges as consisting of the edges of the triangular mesh that cover the edges of the input PSLG. Hereafter, we refer to the input pair consisting of a triangular mesh and a constraining subset of its edges as a *constrained triangular mesh*[a], and we refer to the corresponding output quadrilateral mesh generated by our algorithm as a *constrained quadrilateral mesh*.

While our algorithm may place one Steiner point *outside* the triangulation domain, it is possible, with a small increase in the number of Steiner points, to place all points in the interior of the domain with at most one point *on* the boundary. We discuss this case further in the end of Section 4. The remainder of this paper is organized as follows. In Section 2 we introduce some basic concepts related to our work and review some prior work on quadrilateral meshes. In Section 3 we describe the details of our new algorithm for generating quadrilateral meshes of polygonal domains, and its extension to handle general PSLGs. In section 4 we present some quadrilateral meshes obtained from an implementation of our algorithm, and evaluate their quality according to a quadrilateral shape quality metric. In Section 5 we summarize our results and discuss future work.

2. Background and Prior Work

A *polygonal region* $\mathcal{R}$ is a connected region of the plane whose boundary is one simple and closed polygonal curve or the union of a finite number of disjoint, simple and closed polygonal curves. When the boundary of $\mathcal{R}$ consists of $1 + k$ polygonal curves with $k > 0$, we say that $\mathcal{R}$ has k polygonal holes. Vertices and edges of $\mathcal{R}$ are vertices and edges of its bounding polygonal curves. We denote the set of vertices (resp. edges) of $\mathcal{R}$ by $V_{\mathcal{R}}$ (resp. $E_{\mathcal{R}}$). A *mesh* $\mathcal{M}$ of a polygonal region $\mathcal{R}$ is a decomposition of $\mathcal{R}$ into openly disjoint polygonal regions without holes, called *mesh elements*, that meet each other at a shared vertex or edge only. The set of vertices $V_{\mathcal{M}}$ and the set of edges $E_{\mathcal{M}}$ are the set of vertices and edges of all mesh elements of $\mathcal{M}$, respectively. The set $V_{\mathcal{M}}$ contains the set of vertices $V_{\mathcal{R}}$ of the polygonal region $\mathcal{R}$, and every edge in the set of edges $E_{\mathcal{R}}$ of the polygonal region $\mathcal{R}$ is the union of one or more edges in the set $E_{\mathcal{M}}$. As discussed in the previous section, our interest is in meshing algorithms that take polygonal regions described by PSLGs as input.

[a]We should point out that the term constrained triangular mesh has been used in several other papers with a different meaning.

A mesh $\mathcal{M}$ can be classified according to element type, mesh structure, and mesh geometry [19]. Any two elements of $\mathcal{M}$ are said to be of the *same type* if they have the same number of vertices. When all elements of $\mathcal{M}$ have the same type, we say that $\mathcal{M}$ is *homogeneous.* The most common homogeneous meshes are *triangular meshes* and *quadrilateral meshes*, in which all mesh elements are triangles and quadrilaterals, respectively. The structure of a mesh $\mathcal{M}$ is related to the valence of its vertices. The *valence* of a vertex $v \in V_{\mathcal{M}}$ is the number of edges in $E_{\mathcal{M}}$ incident to v. If this number is the same for all vertices in $V_{\mathcal{M}}$, except for the vertices on the boundary of $\mathcal{M}$, we say that $\mathcal{M}$ is *regular.* Otherwise, it is said to be *irregular.* The geometry of a mesh is related to metric properties. We say that $\mathcal{M}$ is a *uniform mesh* if $\mathcal{M}$ is regular and every edge $e \in E_{\mathcal{M}}$ has the same length. Otherwise, $\mathcal{M}$ is said to be *nonuniform.*

Mesh regularity also affects the geometry of mesh elements, as the valence of a given vertex constrains the shape of all elements incident to it. Regular meshes are often computed by applying a coordinate transformation to a rectangular grid. If such a coordinate transformation can be computed efficiently, a regular mesh can be generated very easily. Besides, there is no need to explicitly store the coordinates of the mesh vertices. However, it may be extremely hard to find a coordinate transformation that fits a given polygonal domain with arbitrarily complex geometry [3]. Furthermore, regular meshes do not offer a flexible way of controlling mesh grading, which is directly related to local variation of element size. If the geometry of the problem domain is complex or a fine control of element size and shape is desirable, irregular meshes are usually preferred. A regular mesh defined as a grid or obtained by applying a coordinate transformation to a grid is often called a *structured mesh* [20].

The problem of generating a quadrilateral mesh of a polygonal region $\mathcal{R}$ is more complex than that of producing a triangular mesh. In fact, if we require the set of vertices of the mesh to be the set of vertices of $\mathcal{R}$, then a triangular mesh can always be obtained but it may not be possible to obtain a quadrilateral one. Hence, additional vertices, called Steiner points, may be necessary in order to quadrangulate polygonal regions. Note that a straightforward parity argument shows that if the number of boundary vertices is odd, it is impossible to quadrangulate $\mathcal{R}$ without adding a Steiner point on or outside the boundary of $\mathcal{R}$. The problem of deciding whether or not a convex quadrilateral mesh of $\mathcal{R}$, with exactly the same set of vertices as $\mathcal{R}$, can be obtained is **NP**-complete for $\mathcal{R}$ with polygonal holes [21]. 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. These facts have led several researchers to adopt an *indirect approach* to produce quadrilateral meshes [22]: The polygonal domain is first triangulated and then the triangulation is converted into a quadrilateral mesh [8,23,11,14,24,15]. This approach relies on the premise that a good quality quadrilateral mesh can be more easily generated from an existing triangular mesh of the problem domain.

Let $\mathcal{R}$ be a polygonal region with n vertices and k polygonal holes, and let $\mathcal{T}$

be any triangular mesh of $\mathcal{R}$. By definition, the set of vertices $V_{\mathcal{R}}$ of $\mathcal{R}$ is contained in the set of vertices $V_{\mathcal{T}}$ of $\mathcal{T}$. From Euler's relation, we know that $\mathcal{T}$ has $t = 2m + 2k - 2 - m_b$ triangles, where m is the number of vertices of $\mathcal{T}$ and m_b is the number of vertices of $\mathcal{T}$ on its boundary, with $n \leq m_b \leq m$. A very simple algorithm for converting such a triangular mesh into a strictly convex quadrilateral mesh was proposed by de Berg [8]: place a Steiner point in the interior of each edge and each triangle of the triangular mesh, and then connect the Steiner point in the interior of each triangle to the three Steiner points on its edges. Fig. 2 illustrates de Berg's algorithm. If the Steiner points are placed carefully, it is always possible to obtain a strictly convex quadrilateral mesh. Despite its simplicity, the size of its output quadrilateral meshes may prevent its practical use on large input triangular meshes. The algorithm by de Berg runs in $\mathcal{O}(t)$ time, produces exactly $3t$ quadrilaterals, and inserts exactly $5m + 5k - 5 - 2m_b$ Steiner points.

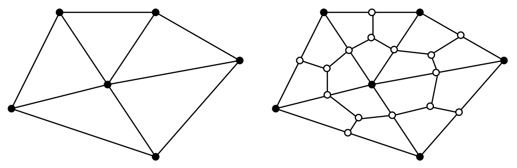
Fig. 2. Example of de Berg's algorithm.

Everett et al. [8] introduced another linear time algorithm to convert triangular meshes into strictly convex quadrilateral ones. Their algorithm also inserts a Steiner point in the interior of each edge of the triangular mesh, but only some of the mesh triangles contain Steiner points in their interiors. This algorithm generates at most $\lfloor \frac{8t}{3} \rfloor$ quadrilaterals and uses the same number of Steiner points as de Berg's. However, the size of the output quadrilateral mesh may still be prohibitive in practice. An interesting feature of this algorithm, which is also present in de Berg's algorithm, is the preservation of the input mesh grading.

Ramaswami et al. [11] presented a linear time and space algorithm to convert triangular meshes into quadrilateral ones that considerably improves upon the bounds on mesh size provided by the algorithms by de Berg and Everett et al. [8]. However, the quadrilateral meshes are not necessarily convex, which rules out the possibility of using their algorithm for generating meshes for FE analysis. Johnston et al. [23] also apply the indirect approach to give an algorithm that uses several heuristics to obtain a strictly convex quadrilateral mesh from a triangular mesh. Their algorithm runs in $\mathcal{O}(t^2)$ time, and selectively combines adjacent triangles to obtain quadrilaterals. However, it is not clear from the description of the algorithm in [23] that the heuristic procedures are always successful in producing a homogeneous quadrilateral mesh.

Shimada et al. [14] proposed an algorithm for generating quadrilateral meshes that takes into account mesh regularity, directionality, and grading, as well as element shape. First, the problem domain is filled with square cells whose size is controlled by a user-defined, scalar density function. Next, the direction of each cell is adjusted by a physically-based relaxation process and a user-defined vector field that specifies directionality over the problem domain. Then, mesh vertices are placed at the center of every cell and connected to generate a triangular mesh of the entire domain. Finally, the triangular mesh is converted into a strictly convex quadrilateral mesh. Later, Viswanath et al. [15] modified this algorithm by using rectangular cells instead of square cells, which enabled them to generate *anisotropic* quadrilateral meshes [2].

The algorithms in [14,15] allow the user to produce nearly regular quadrilateral meshes with well-shaped elements and precise control over their direction and size distribution. Meshes aligned in specific directions can lead to more accurate FE analysis of problems that have strong directionality and involve anisotropic material properties [2]. However, if precise control of directionality is not critical and the problem domain has complex geometry, neither algorithm may be very attractive due to the cost of the physically-based relaxation process used by the "cell packing" technique. Furthermore, the conversion step is not guaranteed to eliminate all triangles of the triangular mesh. As a result both algorithms may require an extra step to subdivide every triangle and quadrilateral of the mesh resulting from the conversion step into three and four quadrilaterals, respectively, to obtain a homogeneous quadrilateral mesh.

Owen et al. [24] presented another quadrilateral meshing algorithm that takes into account directionality and element shape, and it also preserves mesh grading. It converts a triangular mesh into a strictly convex quadrilateral one using advancing fronts initially defined by the boundary edges of the input mesh. Quadrilaterals are generated by combining and transforming triangles as the fronts move from the boundary to the interior of the input mesh. Local smoothing and topological improvements, commonly performed as post-processing steps, are part of the conversion process. One limitation of this method is that directionality cannot be arbitrarily specified as in [14,15]. Although the algorithms in [24,14,15] do not provide any provable bounds on mesh size or mesh element shape, they have been used in practice to successfully generate good quality quadrilateral meshes.

Our quadrilateral meshing algorithm improves upon the bounds on mesh size provided by the algorithms by de Berg and Everett et al. [8]. This improvement makes it possible to use our algorithm in practical applications, as we shall see in Section 4. Our algorithm is also simpler, and likely faster, than the algorithms by Owen et al. [24], Shimada et al. [14], and Viswanath et al. [15]. Furthermore, our algorithm can deal with constrained triangular meshes, which has not been reported to be possible by the algorithms in [24,14,15]. However, the algorithms in [24,14,15] are more likely to generate quadrilateral meshes in which overall element shape is better for FE analysis than the overall element shape in the quadrilateral meshes generated by our algorithm. Fortunately, we can further improve the overall shape of the quadrilaterals generated by our algorithm, at the expense of runtime, by using standard post-processing techniques, as described in Section 4.

3. The Algorithm

In this section, we first describe a new algorithm for converting triangular meshes of polygonal regions, with or without polygonal holes, into strictly convex quadrilateral meshes. Our algorithm allows interior edges of the triangular mesh to be deleted, but it does not allow deletion of vertices. To construct the quadrilateral mesh, new vertices, referred to as Steiner points, may be inserted along with new edges between Steiner points and/or vertices of the input triangular mesh. We show that the mesh produced by our algorithm has small, bounded size and it consists of strictly convex quadrilaterals only. In particular, we show that if the input triangular mesh has t triangles then our algorithm produces a strictly convex quadrilateral mesh with at most $\lfloor \frac{3t}{2} \rfloor + 2$ quadrilaterals and it adds at most $t + 2$ Steiner points to the mesh.

Next, we present a straightforward extension of our algorithm that makes it possible to convert constrained triangular meshes into strictly convex, constrained quadrilateral meshes. That is, given a triangular mesh and a constraining subset of its edges, our algorithm can be easily modified to prevent the deletion of edges that are in the input constraining subset of edges. We refer to the edges in this subset as *constraining edges*. We show that each constraining edge is represented in the constrained quadrilateral mesh as the union of one or more edges of this mesh. We also show that, for an input constrained triangular mesh, our extended algorithm produces a strictly convex, constrained quadrilateral mesh with at most $\lfloor \frac{3t}{2} \rfloor + 4h$ quadrilaterals, obtained by inserting at most $t + 3h$ Steiner points to the mesh, where h is the number of connected components in the dual graph of the triangular mesh.

Hereafter, we use the terms “triangulation”, “quadrangulation”, “quadrangulate”, and “convex” to mean “triangular mesh”, “quadrilateral mesh”, “decompose into quadrilaterals”, and “strictly convex”, respectively, and we occasionally use “quad” as an abbreviation for “quadrilateral”.

3.1. *Polygonal regions with or without polygonal holes*

The main idea behind our algorithm is to quadrangulate a small triangulated region of the input triangulation at a time until the entire triangulation is converted into a quadrangulation. The domain of each such triangulated region is a small, simple polygon (one with 7 or fewer vertices and no holes). This triangulation is converted into a partial or complete quadrangulation of the polygonal region. By using a spanning tree of the dual graph of the triangulation and a procedure to traverse and prune this spanning tree in a bottom-up fashion, our algorithm systematically groups triangles together to define and quadrangulate the small regions so that no isolated triangles remain in the resulting decomposition.

Let $\mathcal{R}$ be a polygonal region with n vertices and k polygonal holes, and let $\mathcal{T}$ be any triangular mesh of $\mathcal{R}$. Let $m \geq n$ be the number of vertices of $\mathcal{T}$ and t the number of triangles of $\mathcal{T}$. Our algorithm starts by building a rooted spanning tree T of the dual graph G of $\mathcal{T}$. The *dual graph* of $\mathcal{T}$ is the graph that contains a node for every triangle of $\mathcal{T}$ and an edge between two nodes if and only if the corresponding triangles share an edge. Figure 3(b) shows the dual graph of a triangulation of the polygonal region shown in Fig. 3(a). A rooted spanning tree T of G is built as a breadth-first search (BFS) tree. The root of T is any node corresponding to a triangle containing a boundary edge of $\mathcal{T}$. Since every node of G can have degree at most 3, the tree T is a binary tree. Figure 3(c) shows such a spanning tree for the dual graph in Fig. 3(b).

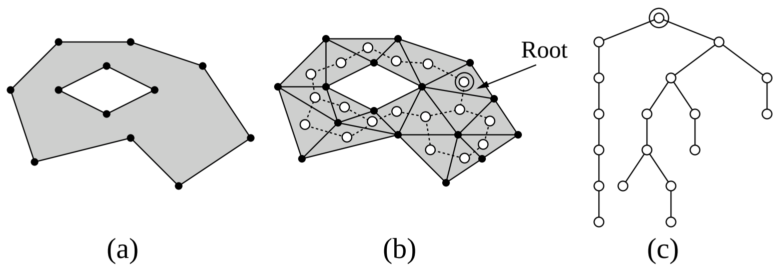

Fig. 3. (a) A polygonal region $\mathcal{R}$. (b) A triangulation of $\mathcal{R}$ and its dual graph. Vertices of the dual graph are shown as white spots, and its edges are shown as dotted edges. (c) A rooted (BFS) spanning tree for the dual graph in (b).

After constructing T, the algorithm builds the set V_l of all nodes of T at level l, for every $l \in \{0, 1, \ldots, d\}$, where d is the depth of T. The root node of T is the singleton node at level 0. Next, the algorithm visits the nodes of T one level at a time and in decreasing order of depth by processing the sets $V_d, V_{d-1}, \ldots, V_0$ in this order. Let $par(v)$ denotes the parent of $v \in V$, $sib(v)$ the sibling of v, and $ele(v)$ the triangle of $\mathcal{T}$ corresponding to v. Note that $ele(v)$ and $ele(par(v))$ necessarily share an edge of $\mathcal{T}$. When visiting a node $v \in V_l$,$1 < l \leq d$, the algorithm considers the subtree rooted at either $par(v)$ or $par(par(v))$ (the nodes of V_0 and V_1 are handled separately at the end of the algorithm). We denote this subtree by T_v and its root by r_v. Let G_v denote the subgraph of G induced by T_v. As we show later, the subgraph G_v corresponds to a triangulated polygonal region $\mathcal{T}_v$ of $\mathcal{T}$ consisting of $4, 5, 6$, or 7 vertices. This triangulation is then converted by the algorithm into a partial or complete quadrangulation of the domain of $\mathcal{T}_v$.

During the conversion of $\mathcal{T}_v$ into a partial or complete quadrangulation of its domain, Steiner points may be added to the interior and boundary of $\mathcal{T}_v$. If the result is a complete quadrangulation of the domain of $\mathcal{T}_v$, the entire subtree T_v is eliminated from T. If the quadrangulation is not complete, there will be only one leftover triangle inside the domain of $\mathcal{T}_v$. The root node r_v of T_v now represents this triangle and the remaining nodes of T_v are eliminated from T. Figure 4 illustrates both cases for a triangulated polygonal region $\mathcal{T}_v$ of $\mathcal{T}$ consisting of 5 vertices on the boundary. After converting $\mathcal{T}_v$ into a partial or complete quadrangulation, the node v is always eliminated from T and V_l by the algorithm. Other nodes of T_v may or may not be eliminated from T and $V_l \cup V_{l-1} \cup V_{l-2}$. In any event, we show that all nodes of V_l are eliminated at the end of this step of the algorithm, and hence the depth of T decreases by at least one. As a result, when the nodes of V_{l-1} are processed during the next step of the algorithm, they are all leaf nodes of (the pruned) T. The sets V_0 and V_1 are handled in a similar way as special cases in the last step, so that after they are processed, the spanning tree T is empty and the underlying triangulation $\mathcal{T}$ has been converted into a strictly convex quadrangulation[b]. We will show that, except at the last step, for every two nodes eliminated from T, at most three quadrilaterals are created by using at most two Steiner points.

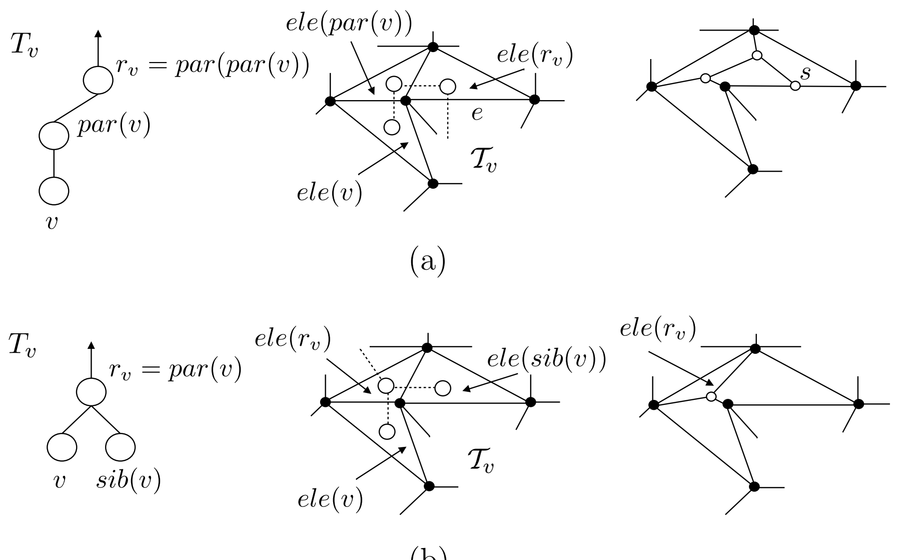

Fig. 4. (a) Complete quadrangulation of the domain of $\mathcal{T}_v$. (b) Partial quadrangulation of the domain of $\mathcal{T}_v$.

Before describing details of the approach used by our algorithm to process the sets $V_d, V_{d-1}, \ldots, V_0$, we discuss two special situations:

[b]This general idea of pruning the dual tree was also used in Ref. [11] to convert triangulations into quadrangulations consisting of quads that are not necessarily convex.

(1) When processing T_v, the algorithm may place a Steiner point s on the edge e between $ele(r_v)$ and $ele(par(r_v))$, as shown in Fig. 4(a). In this situation, the triangle $ele(par(r_v))$ becomes a *degenerate quadrilateral*. Note that $ele(par(r_v))$ can further become a *degenerate pentagon* if the algorithm happens to add another Steiner point to it on the edge shared with $ele(sib(r_v))$. Figure 5 shows a degenerate quadrilateral and a degenerate pentagon.

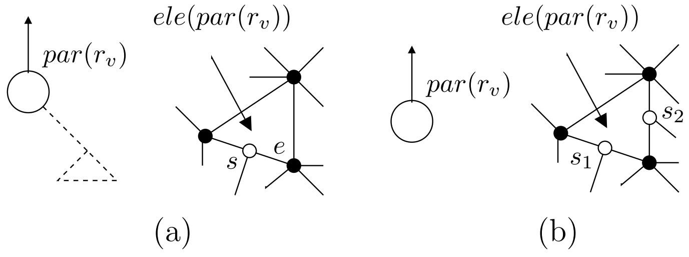

Fig. 5. (a) Degenerate quadrilateral. (b) Degenerate pentagon.

Since T_v gets eliminated from T when a Steiner point is placed on the common edge e of r_v and $par(r_v)$, degenerate pentagons are leaves of T, and degenerate quadrilaterals are either leaves or internal nodes of degree 2. Furthermore, since all quadrilaterals in the quadrangulation constructed by our algorithm are strictly convex, there must be an edge of the quadrangulation incident on s and lying outside $ele(par(r_v))$. We can slightly perturb s along this edge in order to eliminate the degeneracy of $ele(par(r_v))$ without destroying the strict convexity of other quadrilaterals incident to s. Figure 6(a) illustrates the perturbation of the Steiner point of a degenerate quadrilateral to eliminate its degeneracy. We shall show later how our algorithm eliminates the degeneracy of a degenerate pentagon.

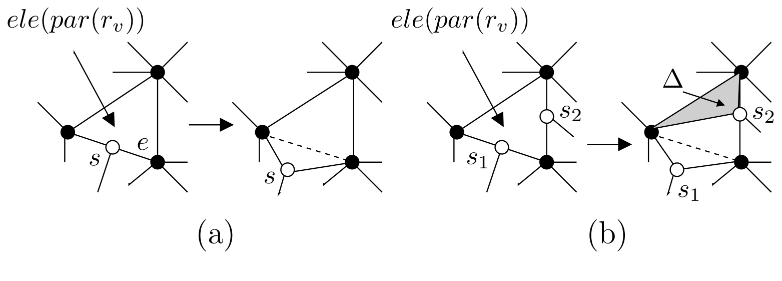

Fig. 6. Elimination of degenerate quadrilaterals and pentagons.

(2) When T_v is a subtree of three nodes, v, $r_v = par(v)$ and $sib(v)$, containing v and there is a cross-edge between v and $sib(v)$ in G_v, the triangulated polygonal region $\mathcal{T}_v$ has three vertices of $\mathcal{T}$ on its boundary and a vertex of $\mathcal{T}$ in its interior, as shown in Fig. 7. In this situation the algorithm eliminates v and $sib(v)$ from T, so that $par(v)$ now represents the *non-empty triangle* Δ with an interior point (see Fig. 7). Note that if v is at level l, then the node corresponding to Δ is a leaf at level $l-1$.

We now describe the steps taken by our algorithm to process the set V_l $(1 < l \leq d)$, where l is the current deepest level of T. During the course of our description, we refer to various lemmas pertaining to quadrangulations of small polygonal regions without holes, which are given and proved in Section 3.3. Let $1 < l \leq d$ be the current deepest level of T. We first eliminate all leaves v of T such that $ele(v)$ is a degenerate quadrilateral, degenerate pentagon, or a non-empty triangle. The first two types of leaves will exist only at levels l, $l-1$, or $l-2$, and the third type will exist only at level l, as each node u of level $l+1$ processed during the previous step had T_u rooted at $par(u)$ or $par(par(u))$.

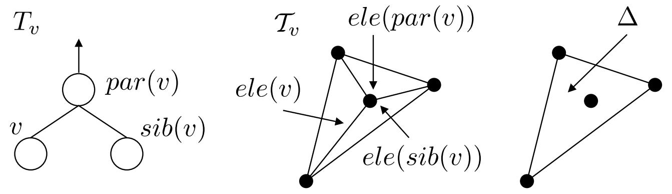

Fig. 7. The non-empty triangle Δ.

Note that, since the spanning tree T is a result of a breadth-first search (BFS) on the dual graph G of $\mathcal{T}$, the subtree T_v of T rooted at either $par(v)$ or $par(par(v))$, where v is a leaf of T, must be one of the five subtrees (up to isomorphism) shown in Fig. 8. Suppose that l is the current deepest level of T. So, the node v of T is at level l. Since a degenerate quadrilateral and a degenerate pentagon can correspond to a node at levels l, $l-1$, or $l-2$ only, and since a non-empty triangle can correspond to a node at level l only, we have that $ele(v)$ can be either a triangle, a degenerate quadrilateral, a degenerate pentagon, or a non-empty triangle, and neither $ele(par(v))$ nor $ele(par(par(v)))$ can be a non-empty triangle. Furthermore, since $par(v)$ has one child in subtrees (1), (3), (4), and (5), and it has two children in subtree (2), $ele(par(v))$ can be either a triangle or a degenerate quadrilateral for $par(v)$ in subtrees (1), (3), (4), and (5), and $ele(par(v))$ has to be a triangle for $par(v)$ in subtree (2). Likewise, $ele(par(par(v)))$ is either a triangle or a degenerate quadrilateral for $par(par(v))$ in subtree (3), but it has to be a triangle for $par(par(v))$ in subtrees (4) and (5).

Step 1. *Eliminate all* $v \in V_l \cup V_{l-1} \cup V_{l-2}$ *such that* v *is a leaf and* $ele(v)$ *is a degenerate quadrilateral.* Let s be the Steiner point of $ele(v)$, and let e_s be the edge of the quadrangulation (constructed thus far) incident on s. Convert $ele(v)$ into a strictly convex quadrilateral by perturbing s along the edge e_s, as shown in Fig. 6(a). Remove v from T and V_l.

Step 2. *Eliminate all* $v \in V_l \cup V_{l-1} \cup V_{l-2}$ *such that* $ele(v)$ *is a degenerate pentagon.* Let s_1 and s_2 be the two Steiner points of $ele(v)$ and let e be the shared edge of $ele(v)$ and $ele(par(v))$. It is straightforward to convert $ele(v)$ into a strictly convex quadrilateral and a leftover triangle Δ, as shown in Fig. 6(b). Now, the node v represents the leftover triangle, i.e., $ele(v) = \Delta$.

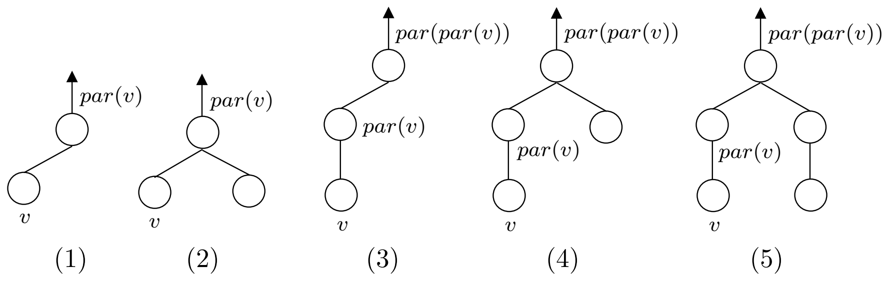

Fig. 8. All possibilities for a subtree T_v of T rooted at $par(v)$ or $par(par(v))$, where v is node at the current deepest level of T (a leaf).

After steps 1 and 2 are carried out, every element v of V_l must correspond to either a triangle or a non-empty triangle, and $par(v) \in V_{l-1}$ must be either a triangle or a degenerate quadrilateral. The next step eliminates all nodes v from V_l such that $ele(v)$ is a non-empty triangle. Assume that $v \in V_l$ corresponds to a non-empty triangle and consider the subtree T_v containing v and rooted at $par(v)$. Up to isomorphism, the subtree T_v must be one of subtrees (1) and (2) in Fig. 8. If T_v is isomorphic to subtree (1), then $par(v)$ is either a triangle or a degenerate quadrilateral. If T_v is isomorphic to subtree (2), then $par(v)$ is a triangle and $sib(v)$, the sibling of v, is either a triangle or a degenerate quadrilateral. So, there are four cases to be considered by the algorithm in order to eliminate the node v from V_l when $ele(v)$ is a non-empty triangle.

Step 3. *Eliminate all* $v \in V_l$ *such that* $ele(v)$ *is a non-empty triangle.* Note that v corresponds to three nodes of the original spanning tree T. Let T_v be the subtree of T rooted at $par(v)$. We consider the following sub-steps (refer to the illustrations in Fig. 9 and Fig. 10):

3a. *Node* $par(v)$ *has degree 2, and* $ele(par(v))$ *is a degenerate quadrilateral.* Let s be the Steiner point of $ele(par(v))$. By connecting s to the node of $ele(par(v))$ that is not incident to the edge of $ele(par(v))$ containing s, the triangulated region $\mathcal{T}_v$ can be decomposed into a triangle Δ and a quadrilateral with a point in its interior (see Fig. 9(a)). By Lemma 3.3.2, the latter can be quadrangulated into five convex quads with three Steiner points in its interior. Carry out the appropriate decomposition and then remove the node v from T and V_l. Node $par(v)$ now corresponds to the triangle Δ.

3b. *Node $par(v)$ has degree 2, and $ele(par(v))$ is a triangle.* In this case, the domain of $\mathcal{T}_v$ is a quadrilateral and this quadrilateral contains a vertex of $\mathcal{T}_v$ in its interior (see Fig. 9(b)). Again, by Lemma 3.3.2, the region can be quadrangulated into five convex quads with three Steiner points in its interior. Carry out this decomposition and then remove v and $par(v)$ from T and from their corresponding node sets.

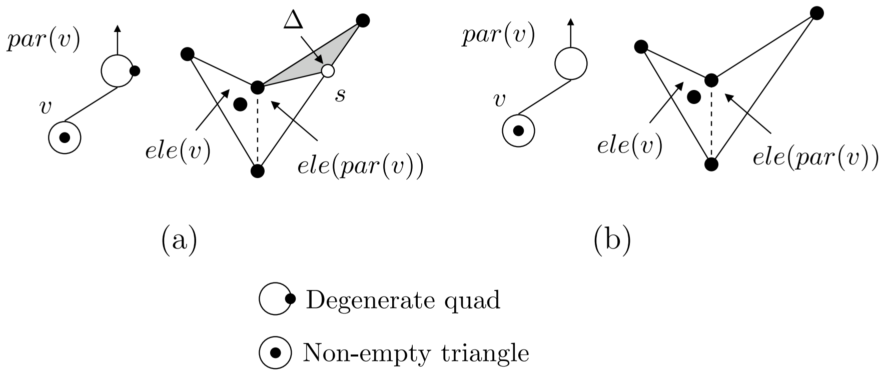

Fig. 9. Cases 3a and 3b of Step 3.

3c. *Node $par(v)$ has degree 3, and $ele(sib(v))$ is a triangle.* If $G_v = T_v$, then the domain of $\mathcal{T}_v$ is a pentagon and this pentagon contains a vertex of $\mathcal{T}_v$ in its interior (see Fig. 10(a)). Then by Lemma 3.3.4, this region can be decomposed into at most six convex quads and one triangle Δ by using at most four Steiner points in the interior. If G_v contains a cross-edge between v and $sib(v)$, then $ele(v)$ and $ele(sib(v))$ form a quadrilateral with a point inside (see Fig. 10(a)), and again Lemma 3.3.2 is applied. In either case, carry out the appropriate decomposition and then remove v and $sib(v)$ from T and from their corresponding node sets. In the former case, the vertex r_v is made correspond to triangle Δ.

3d. *Node $par(v)$ has degree 3, and $ele(sib(v))$ is a non-empty triangle.* If $G_v = T_v$, then the domain of $\mathcal{T}_v$ is a pentagon and this pentagon contains two vertices of $\mathcal{T}_v$ in its interior. If $G_v \neq T_v$, i.e., if G_v contains a cross-edge between v and $sib(v)$, then the domain of $\mathcal{T}_v$ is a triangle and this triangle contains three vertices of $\mathcal{T}_v$ in its interior. In either case, the triangulated region $\mathcal{T}_v$ can be decomposed into two quadrilaterals, each of which has point in its interior, as follows: add a Steiner point on the edge shared by $ele(r_v)$ and $ele(par(r_v))$ and connect it to the vertex of $ele(r_v)$ that is not incident to the edge shared by $ele(r_v)$ and $ele(par(r_v))$ (see Fig. 10(b)). By Lemma 3.3.2, each quadrilateral can be decomposed into five convex quads using three Steiner points. After applying the aforementioned decomposition, the algorithm removes all nodes of T_v from T and from their corresponding node sets. Hence a total of seven nodes were eliminated and ten convex quadrilaterals were created using seven Steiner points. In the next step of the algorithm, the element $ele(par(r_v))$ will be a degenerate quadrilateral or pentagon.

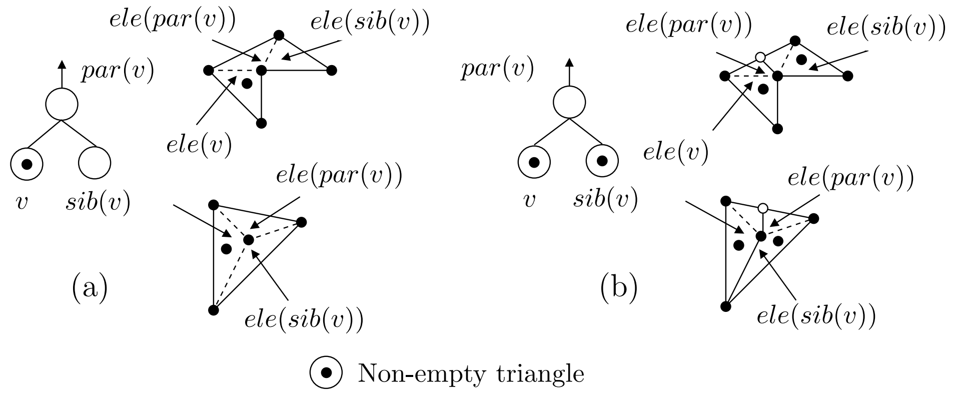

Fig. 10. Cases 3c and 3d of Step 3.

After steps 1, 2 and 3 are carried out, for every node $v \in V_l$, its corresponding element $ele(v)$ in $\mathcal{T}$ is a triangle, and the corresponding element $ele(par(v))$ of its parent $par(v) \in V_{l-1}$ (if any) is either a triangle or a degenerate quadrilateral. The algorithm proceeds by performing two more steps: step 4 and step 5. These steps eliminate all remaining nodes of V_l. Step 4 is carried out if and only if $l \geq 3$, i.e., if and only if the spanning tree T is currently a tree with at least three levels. In the remaining description of our algorithm, assume that $l \geq 3$ after steps 1, 2, and 3 are carried out.

Let v be any node of V_l. If $par(v)$ exists, then consider the subtree T_v containing v and rooted at $par(v)$. Up to isomorphism, the subtree T_v must be one of subtrees (1) and (2) in Fig. 8. If T_v is isomorphic to subtree (1), then $ele(par(v))$ is either a triangle or a degenerate quadrilateral. If T_v is isomorphic to subtree (2), or equivalently, if $par(v)$ has degree 3, then $par(v)$ is a triangle and $sib(v)$, the sibling of v, must be a triangle, as no vertex in V_l can correspond to a non-empty triangle. Our algorithm considers T_v if and only if T_v is isomorphic to subtree (1) and $ele(par(v))$ is a degenerate quadrilateral, or T_v is isomorphic to subtree (2). Whenever T_v is isomorphic to subtree (2) and $ele(par(v))$ is a triangle, the algorithm considers the subtree T_v containing v and rooted at $par(par(v))$. Up to isomorphism, the subtree T_v rooted at $par(par(v))$ is one of subtrees (3), (4) and (5) in Fig. 8. We now give details of step 4.

Step 4. *Eliminate all remaining nodes* $v \in V_l$. From the above observation and from the fact that T is a BFS tree, it follows that the only possible configurations for T_v are those described in the following sub-steps:

4a. *Eliminate all* $v \in V_l$ *such that* $ele(par(v))$ *is a degenerate quadrilateral.* Let T_v be the subtree of T rooted at $r_v = par(v)$. Perturb the Steiner point s of $ele(r_v)$ along the quadrangulation edge incident to it. A Steiner point s' placed in $ele(r_v)$ decomposes the region $\mathcal{T}_v$ into two convex quads and a triangle Δ adjacent to the edge shared by $ele(r_v)$ and $ele(par(r_v))$ (see Fig. 11(a)). Remove v from T and V_l, and let r_v now represent triangle Δ.

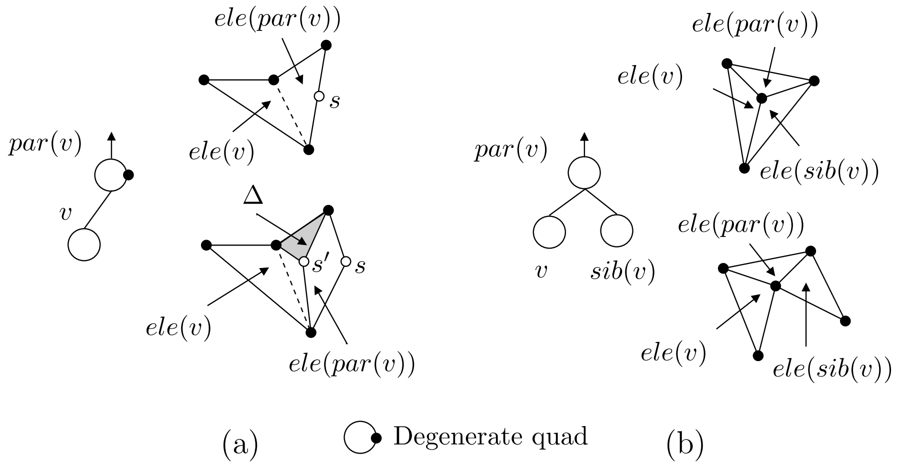

Fig. 11. Cases 4a and 4b of Step 4.

4b. *Eliminate all* $v \in V_l$ *such that* $par(v)$ *is a node of degree 3.* Again, let T_v be the subtree of T rooted at $r_v = par(v)$ and refer to Fig. 11(b). If G_v contains an edge between the nodes v and $sib(v)$, remove v and $sib(v)$ from T_v. The node $par(v)$ is now the non-empty triangle corresponding to the boundary of $\mathcal{T}_v$, with the fourth vertex of $\mathcal{T}_v$ in the interior of its domain (see Fig. 7). If there is no cross-edge between v and $sib(v)$, then by Lemma 3.3.3, the domain of $\mathcal{T}_v$ can be subdivided into two convex quadrilaterals and one triangle Δ (adjacent to the edge shared by $ele(r_v)$ and $ele(par(r_v))$) by adding one Steiner point. Carry out the appropriate decomposition and then remove v and $sib(v)$ from T and the corresponding node sets. The node r_v now represents triangle Δ.

If $ele(par(v))$ is not a degenerate quadrilateral and $par(v)$ is not a node of degree 3, the algorithm considers the subtree T_v containing v and rooted at $par(par(v))$. As we mentioned before, up to isomorphism, the subtree T_v rooted at $par(par(v))$ is one of subtrees (3), (4) and (5) in Fig. 8. If T_v is isomorphic to subtree (3), or equivalently, if node $par(v)$ has degree 2, then $par(par(v))$ can be either a triangle or a degenerate quadrilateral. If T_v is isomorphic to subtree (4), or equivalently, if node $par(par(v))$ has degree 3 and node $sib(par(v))$, the sibling of $par(v)$, is a leaf of T_v, then $sib(par(v))$ is either a triangle or a non-empty triangle previously created by step 4b. If T_v is not isomorphic to subtree (3) nor (4), then it is isomorphic to subtree (5), or equivalently, node $par(par(v))$ has degree 3 and node $sib(par(v))$ has degree 3. So, the algorithm distinguishes these five possible cases in sub-step 4c:

4c. Finally, *eliminate all* $v \in V_l$ *such that* $par(v)$ *is a node of degree 2.* Let T_v be the subtree of T rooted at $r_v = par(v)$. If the domain of $\mathcal{T}_v$ is already a strictly convex quadrilateral, simply eliminate the common edge shared by $ele(v)$ and $ele(par(v))$, and then remove v and $par(v)$ from T and their corresponding node sets. Otherwise, let T_v be the subtree of T rooted at $r_v = par(par(v))$ and consider the following five sub-cases:

i. *Element* $ele(r_v)$ *is a degenerate quadrilateral.* Since T is a BFS tree, we have that $G_v = T_v$. Perturb the Steiner point of $ele(r_v)$ along the quadrangulation edge incident to it so that the domain of $\mathcal{T}_v$ is now a hexagon (see Fig. 12(a)). By Lemma 3.3.1, this region can be subdivided into at most four convex quadrilaterals by using at most three Steiner points in its interior. Carry out this decomposition and then eliminate all the nodes of T_v from T and from their corresponding node sets.

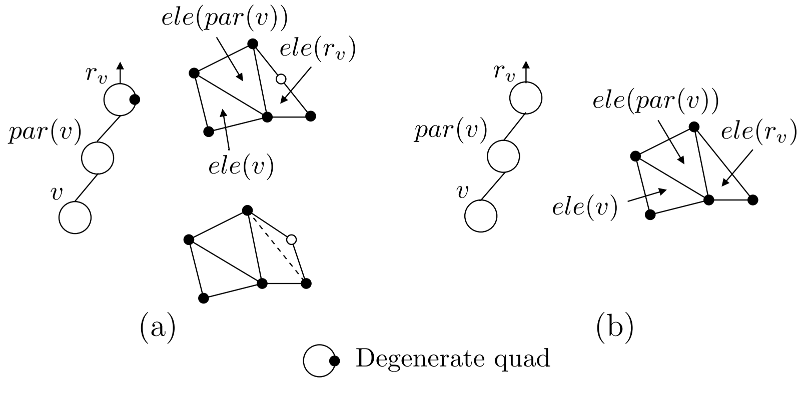

Fig. 12. Cases i and ii of Step 4c.

ii. *Node* r_v *has degree 2 and* $ele(r_v)$ *is a triangle.* Once again $G_v = T_v$ for this case. Therefore, the domain of $\mathcal{T}_v$ is a pentagon (see Fig. 12(b)). Apply Lemma 3.3.3, where the common edge of $ele(r_v)$ and $ele(par(r_v))$ is designated as the "outgoing" edge. There are two possible outcomes from applying Lemma 3.3.3. First, the domain of $\mathcal{T}_v$ is subdivided into three convex quads and one triangle Δ adjacent to the outgoing edge by using two Steiner points. Second, the domain of $\mathcal{T}_v$ is subdivided into four convex quads by using three Steiner points, one of which lies on the outgoing edge. In the first situation, remove v and $par(v)$ from T and from their corresponding node sets, and let r_v now represent triangle Δ. In the second situation, remove all nodes of T_v from T and from their corresponding node sets. Note that triangle $ele(par(r_v))$ becomes a degenerate quadrilateral or a pentagon.

iii. *Node r_v has degree 3, node $sib(par(v))$ is a leaf, and element $ele(sib(par(v)))$ is a triangle.* If $G_v = T_v$, the domain of $\mathcal{T}_v$ is a hexagon. Otherwise, the domain of $\mathcal{T}_v$ is a quadrilateral that contains a vertex of $\mathcal{T}_v$ in its interior (see Fig. 13). In the former case, carry out the decomposition from Lemma 3.3.1. In the latter case, carry out the decomposition from Lemma 3.3.2. In either case, remove all nodes of T_v from T and from their corresponding node sets.

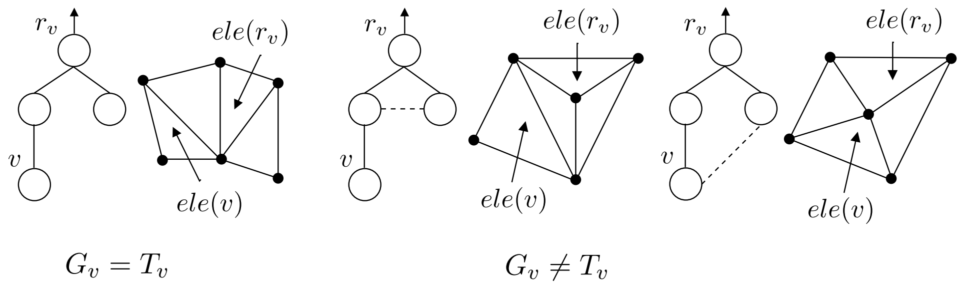

Fig. 13. Case iii of Step 4c (dashed edges are cross-edges).

iv. *Node r_v has degree 3, node $sib(par(v))$ is a leaf, and element $ele(sib(par(v)))$ is a non-empty triangle.* Note that such a non-empty triangle must have been created in Step 4b. Refer to Fig. 14. If $G_v = T_v$, the domain of $\mathcal{T}_v$ is a hexagon with a point in its interior. Otherwise, the domain of $\mathcal{T}_v$ is a quadrilateral that contains two vertices of $\mathcal{T}_v$ in its interior. In both cases, consider the triangulated pentagon P defined by $ele(v)$, $ele(par(v))$, and $ele(r_v)$. Apply Lemma 3.3.3 to P, where the shared edge of $ele(r_v)$ and $ele(sib(par(v)))$ is designated as the "outgoing" edge. If there is a leftover triangle Δ, this triangle and $ele(sib(par(v)))$ form a triangulated quadrilateral with a vertex of $\mathcal{T}_v$ in its interior. Otherwise, a Steiner point has been placed on the edge shared by $ele(r_v)$ and $ele(sib(par(v)))$. In either case, the element $ele(sib(par(v)))$

becomes a triangulated quadrilateral with a vertex of $\mathcal{T}_v$ in its interior. Apply Lemma 3.3.2 to the resulting quadrilateral. Remove all nodes of T_v from T and from their corresponding node sets.

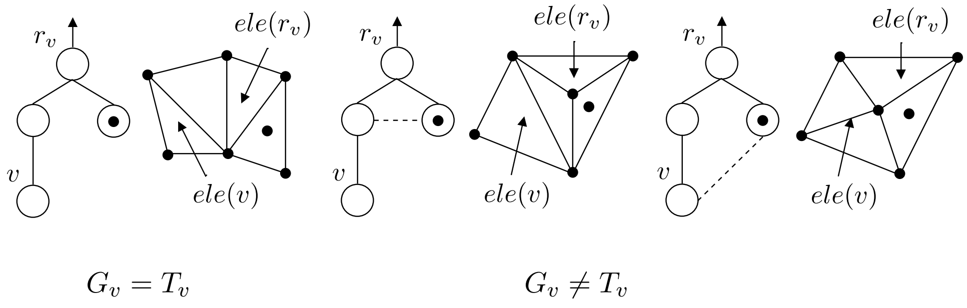

Fig. 14. Case iv of Step 4c (dashed edges are cross-edges).

v. *Node r_v has degree 3 and node $sib(par(v))$ has degree 2.* The different possibilities for the graph G_v are derived from the fact that T is a BFS tree. All cases are illustrated in Fig. 15. Cases (a)–(c) correspond to a pentagon with a point in its interior. For these cases, apply Lemma 3.3.4. In cases (d)–(e), the non-root nodes of T_v (v, v_1, v_2 and v_3 in Fig. 15) correspond to a quadrilateral with a point inside. Apply Lemma 3.3.2 for these cases. Finally, case (f) corresponds to a septagon. For this case, apply Lemma 3.3.5. In all cases, remove all four non-root nodes of T_v from T and from their corresponding node sets.

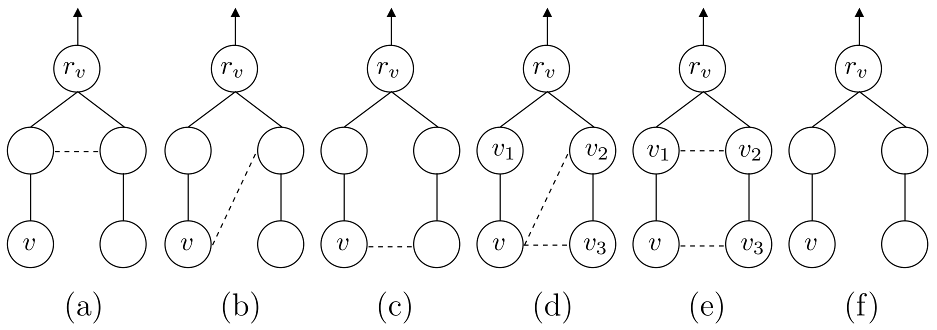

Fig. 15. G_v for Case v of Step 4c (dashed edges are cross-edges).

After the sets $V_d, V_{d-1}, \ldots, V_2$ are processed, they are all empty and the sets V_0 and V_1 are possibly non-empty. If $V_0 \cup V_1$ is empty, we are done. Otherwise, the algorithm carries out the next and last step.

Step 5. *Eliminate all nodes* $v \in V_0 \cup V_1$. Apply steps 1, 2, and 3 to $V_0 \cup V_1$. If $V_0 \cup V_1$ is now empty, we are done. Otherwise, the nodes in V_1, if any, correspond to triangles of $\mathcal{T}$. The singleton node in V_0, if any, corresponds to either a triangle or a degenerate quadrilateral that contains a boundary edge of $\mathcal{T}$ (because of how we choose the root node of T). We now have the following sub-steps:

5a. V_1 *has two elements.* Let v and $sib(v)$ be the two elements of V_1, and let T_v be the subtree of T rooted at $r_v = par(v)$. The tree T_v corresponds to either a triangulated pentagon or a non-empty triangle. Place a single Steiner point on the boundary edge of $ele(r_v)$ and perturb it slightly so that it lies outside the domain of $\mathcal{T}_v$ (see Fig. 16). The domain of $\mathcal{T}_v$ is now either a hexagon, or a quadrilateral with a point in its interior. By Lemmas 3.3.1 and 3.3.2, respectively, each of these regions can be decomposed into strictly convex quadrilaterals by using at most three additional Steiner points in its interior. Eliminate all three nodes from T, V_0, and V_1.

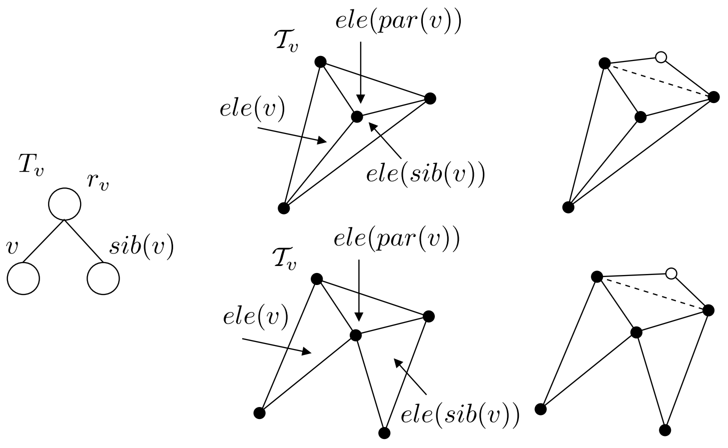

Fig. 16. Decomposition of a non-empty triangle into two quadrilaterals in Step 5a.

5b. V_1 *has only one element and the singleton element of* V_0 *corresponds to a degenerate quadrilateral.* Let v be the singleton element of V_1, and let T_v be the subtree of T rooted at $r_v = par(v)$. Proceed as in case 4a and let r_v correspond to the leftover triangle Δ. Place a Steiner point on the boundary edge of Δ and perturb it slightly as shown in Fig. 17 to convert Δ into a strictly convex quadrilateral. Eliminate v and r_v from T, V_1, and V_0.

5c. V_1 *has only one element and the singleton element of* V_0 *corresponds to a trian-*

gle. Again, let v be the singleton element of V_1, and let $r_v = par(v)$. Let Q be the quadrilateral formed by $ele(v)$ and $ele(r_v)$. If Q is strictly convex, simply eliminate the common edge shared by $ele(v)$ and $ele(r_v)$. If Q is not strictly convex, add a Steiner point in the interior of Q and then apply Lemma 3.3.2 to decompose it into five strictly convex quads by using three additional Steiner points. In either case, remove v and r_v from T and from their corresponding node sets.

5d. *V_1 is empty, V_0 is not empty and its singleton element corresponds to a triangle.* Let v be the singleton element of V_0. Place a Steiner point s on the edge of $ele(v)$ that is also a boundary edge of $\mathcal{T}$. Perturb s, as shown in Fig. 17, so that it lies outside the domain of $\mathcal{T}$ and Q becomes a strictly convex quadrilateral. Remove v from T and V_0.

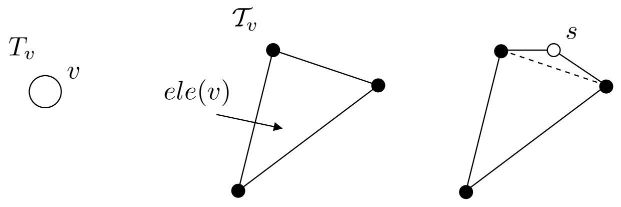

Fig. 17. Converting a triangle into a quadrilateral.

Note that each of step 5a–d can be executed at most once and they are all mutually exclusive. Furthermore, steps 5a, 5b, and 5d are the only cases when our algorithm adds a Steiner point outside the domain of $\mathcal{T}$. When $V_0 \cup V_1$ is empty, the spanning tree T is also empty and the triangulation $\mathcal{T}$ has been converted into a strictly convex quadrangulation. The following theorem states important properties of the above algorithm:

Theorem 3.1.1. *Let $\mathcal{R}$ be a polygonal region with or without polygonal holes. Then, given any triangulation $\mathcal{T}$ of $\mathcal{R}$ with t triangles, the algorithm described above can convert $\mathcal{T}$ into a strictly convex quadrangulation of at most $\lfloor \frac{3t}{2} \rfloor + 2$ quadrilaterals by using at most $t+2$ Steiner points, all except one of which lie within the boundary of $\mathcal{R}$. The algorithm runs in $O(t)$ time and space.*

Proof. The fact that our algorithm generates a strictly convex quadrangulation from $\mathcal{T}$ is an immediate consequence of the lemmas in Section 3.3 and of the fact that the spanning tree T of the dual graph G of $\mathcal{T}$ is empty after cases 1–5 are executed. We now must show that our algorithm produces at most three strictly convex quadrilaterals for every two nodes eliminated from the spanning tree T of the dual graph G of $\mathcal{T}$. In order to see this, consider Table 1. Each row of this table contains, for a given step of the algorithm, the maximum number of nodes of T eliminated, the maximum number of quadrilaterals created, and the maximum number of Steiner points inserted to create those quadrilaterals. Note that the algorithm produces at most three strictly convex quadrilaterals for every two nodes eliminated from T in all steps except steps 2, 3a, 4a, 5a, and 5c. We focus our attention on these cases.

In step 2, $ele(v)$ is a degenerate pentagon. In steps 3a and 4a, $ele(par(v))$ is a degenerate quadrilateral. Degenerate quadrilaterals and pentagons can arise only as the result of the execution of step 4c(ii). Hence, step 2 must be preceded by two executions of step 4c(ii), while steps 3a and 4a must each be preceded by one execution of step 4c(ii). The combination of one execution of step 2 and two executions of step 4c(ii) eliminates 6 nodes from T and produces at most 9 quadrilaterals. The combination of one execution of step 3a and one execution of step 4c(ii) eliminates 7 nodes from T and produces at most 8 quadrilaterals. The combination of one execution of step 4a and one execution of step 4c(ii) eliminates 5 nodes from T and produces at most 5 quadrilaterals. Thus, in all these situations, the algorithm produces at most three strictly convex quadrilaterals for every two nodes eliminated from T. Finally, only one of steps 5a and 5c can be executed and at most once. Step 5a creates 5 quads after eliminating three nodes, thus creating one more quad than the target ratio. Step 5c creates 5 quads after eliminating two nodes, thus creating two more quads than the target ratio. It follows that our algorithm constructs a strictly convex quadrangulation with at most $\lfloor \frac{3t}{2} \rfloor + 2$ quadrilaterals.

From Table 1, we also have that the algorithm uses at most k Steiner points for every k nodes eliminated from T in all steps except for steps 5a and 5c, in which at most $k + 2$ Steiner points are used. Since only one of these steps is executed and at most once, it follows that our algorithm places at most $t + 2$ Steiner points. Furthermore, all Steiner points are placed within the boundary of $\mathcal{R}$, except for one Steiner point inserted in Steps 5a, 5b, or 5d. Since these steps are mutually exclusive and executed at most once, our algorithm places at most one Steiner point outside $\mathcal{R}$.

Finally, after a given node set V_l $(0 \leq l \leq d)$ is processed, where d is the depth of T, the set V_l is empty and hence the depth of T decreases by at least one. Furthermore, while processing V_l, the algorithm visits each node $v \in V_l$ at most twice, and it may also visit a constant number of nodes at levels $l - 1$ and $l - 2$ (the vertices in the subtree T_v at these levels). Hence, each node in T gets visited only a constant number of times, and therefore the algorithm runs in $\mathcal{O}(t)$ time. The space requirements are clearly $\mathcal{O}(t)$ as well. □

3.2. *Constrained quadrilateral meshes*

The algorithm described in Section 3.1 for converting triangulations of polygonal regions, with or without polygonal holes, into quadrangulations can be extended in a straightforward manner to work with constrained triangulations. We are thus able to obtain constrained quadrilateral meshes that conform to an additional set of vertices and edges in the interior of any given polygonal region with or without polygonal holes. Such meshes can be generated in two steps. In the first step, we use an algorithm for generating a constrained triangulation that conforms to the additional set of vertices and edges in the interior of the polygonal region. For instance, any algorithm from [4,5,6] will do. In the second step, we use the aforementioned extension of our algorithm in Section 3.1 to obtain a constrained quadrangulation of the same polygonal region. The input to the algorithm is the constrained triangulation and the subset of constraining edges. Note that any additional vertices in the interior of the polygonal region are also vertices of the quadrangulation, since our algorithm does not allow deletion of vertices. In the rest of this section, we describe the extension of our algorithm to generate constrained quadrilateral meshes.

Step	# Nodes Eliminated	# Quads Created	# Steiner Pts. Inserted
1	1	1	0
2	0	1	0
3a	3	5	3
3b	4	5	3
3c	4	6	4
3d	7	10	7
4a	1	2	1
4b	2	2	1
4c,i	3	4	3
4c,ii	3	4	3
4c,iii	4	6	4
4c,iv	6	9	6
4c,v,(a)–(c)	4	5	3
4c,v,(d)–(e)	4	5	3
4c,v,(f)	4	6	4
5a	3	5	4 (1 outside $\mathcal{R}$)
5b	2	3	2 (1 outside $\mathcal{R}$)
5c	2	5	4
5d	1	1	1 (1 outside $\mathcal{R}$)

Table 1. Upper bounds on number of quads created and Steiner points inserted to quadrangulate $\mathcal{R}$.

Let $\mathcal{T}$ be a given constrained triangulation with t triangles, and let S be a subset of edges of $\mathcal{T}$. The edges in S are the constraining edges of $\mathcal{T}$. Let G be the dual graph of $\mathcal{T}$ such that G does not include the dual of the edges of $\mathcal{T}$ that are in S. Let h be the number of connected components of G. In order to construct a convex quadrangulation that also conforms to the constraining edges in S, we obtain all connected components $\{G_1, G_2, \ldots, G_h\}$ of G and their corresponding spanning trees $\{T_1, T_2, \ldots, T_h\}$, and then we run our algorithm in Section 3.1 on each $\mathcal{T}_i$ using T_i as the spanning tree of G_i, for every $(1 \leq i \leq h)$. The root node of each T_i represents a triangle adjacent to a boundary edge e_i of the underlying triangulation $\mathcal{T}_i$. Since the dual graph G of $\mathcal{T}$ does not include the dual of the constraining edges of $\mathcal{T}$ (the edges of S), these constraining edges cannot be interior edges of the small triangulated polygonal regions that are formed by the algorithm in Section 3.1 during the conversion of each $\mathcal{T}_i$ into a quadrangulation. Thus, the constraining edges of $\mathcal{T}$ are kept in the quadrangulation resulting from the conversion of each $\mathcal{T}_i$. However, at the very last step, the algorithm in Section 3.1 may place a Steiner point on at most one of the constraining edges for each $\mathcal{T}_i$, namely the boundary edge belonging to the triangle that corresponds to the root node of T_i.

One difficulty with the above algorithm is that the conversion of one triangulation $\mathcal{T}_i$ into a quadrangulation may "corrupt" the quadrangulation resulting from the conversion of $\mathcal{T}_j$, with $i \neq j$ and $1 \leq i, j \leq h$. This occurs whenever the algorithm places one Steiner point on a boundary edge of $\mathcal{T}_i$ that is also a boundary edge of $\mathcal{T}_j$. This Steiner point will make a quadrilateral of the quadrangulation obtained from $\mathcal{T}_j$ into a biconvex pentagon. Figure 18 illustrates this situation by using a constrained triangulation whose dual graph has three connected components. The constraining edges of $\mathcal{T}$ are heavily drawn. To overcome this difficulty, we carefully choose the root nodes of the spanning trees in $\{T_1, T_2, \ldots, T_h\}$, and run our algorithm on each $\mathcal{T}_i$, for every $1 \leq i \leq h$, in an order that avoids the situation shown in Fig. 18.

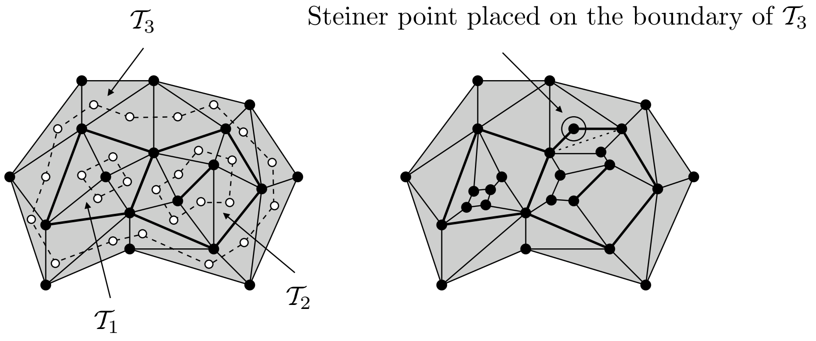

Fig. 18. The conversion of the triangulation $\mathcal{T}_2$ into a quadrangulation corrupted the triangulation $\mathcal{T}_3$.

Let G^c be a graph such that G^c has a node v_i for every connected component G_i of G. G^c has an edge connecting v_i to v_j if and only if there exist a node $u \in G_i$ and a node $w \in G_j$ such that $ele(u)$ and $ele(w)$ in $\mathcal{T}$ share a common edge, with $i \neq j$ and $1 \leq i, j \leq h$. Note that the graph G^c is connected. Let T^c be any spanning tree of G^c such that the root of T^c is a node of G^c corresponding to a connected component of G whose underlying triangulation has a triangle with a boundary edge of $\mathcal{T}$. From the definition of G^c, if $v_i \in T^c$ is not the root of T^c then it has a parent node, say v_j. Then, the underlying triangulations $\mathcal{T}_i$ and $\mathcal{T}_j$ corresponding to v_i and v_j, respectively, must share a common constraining (boundary) edge.

To compute a spanning tree T_i for the connected component G_i of G $(1 \leq i \leq h)$, we consider the following two cases: First, the node v_i is the root of T^c, and hence the triangulation $\mathcal{T}_i$ has at least one triangle $\triangle$ that contains a boundary edge of $\mathcal{T}$. Second, the node v_i has a parent in T^c, say v_j, and hence the triangulation $\mathcal{T}_i$ has at least one triangle $\triangle$ that shares an edge with a triangle in the triangulation $\mathcal{T}_j$, for some $j \neq i$ and $1 \leq j \leq h$. In either case, we choose the root of T_i to be the node of G_i corresponding to $\triangle$. Finally, we obtain T_i by performing a BFS on G_i, starting from the node of G_i corresponding to $\triangle$. Now, suppose we run the algorithm in Section 3.1 on $\mathcal{T}_i$ using T_i as the spanning tree of G_i. If the algorithm places a Steiner point s on a boundary edge of $\mathcal{T}_i$, this boundary edge (call it e) is an edge of the triangle of $\mathcal{T}_i$ corresponding to the root node of T_i. Furthermore, e is also a boundary edge of exactly one other triangulation, namely $\mathcal{T}_j$, where v_j is the parent node of v_i in T^c. Thus, by converting $\mathcal{T}_i$ into a quadrangulation, our algorithm can only affect the triangulation $\mathcal{T}_j$. If $\mathcal{T}_j$ has not yet been converted into a quadrangulation, we can repair $\mathcal{T}_j$ by splitting its "corrupted" triangle, i.e., the triangle adjacent to e that becomes a biconvex quadrilateral by the placement of s, into two triangles. Figure 19 shows this repairing procedure. Note that the graph G, the connected component G_j of G and the spanning tree T_j of G_j must be updated accordingly.

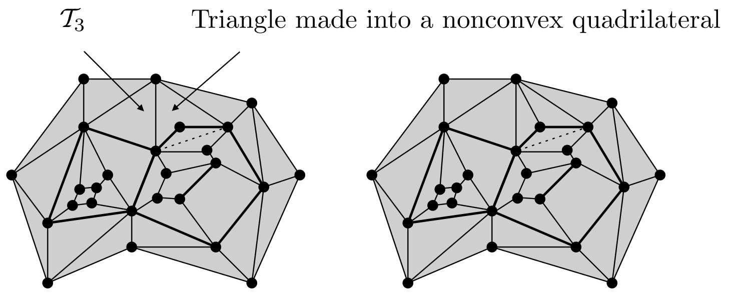

Fig. 19. The "corrupted" triangulation $\mathcal{T}_3$ in Fig. 18 is repaired by splitting a biconvex quadrilateral into two triangles.

Now, in order to avoid the situation shown in Fig. 18, we run the algorithm in Section 3.1 on each $\mathcal{T}_i$ $(1 \leq i \leq h)$ as follows: We traverse the spanning tree T^c by level and in a bottom-up fashion, and apply the algorithm in Section 3.1 to the underlying triangulation $\mathcal{T}_i$ of the spanning tree T_i corresponding to the current node v_i of T^c. As before, let v_j be the parent of v_i. Since we traverse T^c by level and in a bottom-up fashion, the triangulation $\mathcal{T}_j$ has not yet been converted into a quadrangulation at the time that the algorithm visits node v_i of T^c. Hence, the conversion of $\mathcal{T}_i$ into a quadrangulation does not affect any previously computed quadrangulation and can only affect the triangulation $\mathcal{T}_j$. If it does affect $\mathcal{T}_j$, we can repair $\mathcal{T}_j$ as shown in Fig. 19. As a result, when the algorithm visits node v_j, all triangulations whose conversion can affect $\mathcal{T}_j$ have already been converted into quadrangulations, and our repair procedure guarantees that the triangulation $\mathcal{T}_j$ is not corrupted. If v_i is the root node of T^c, the conversion of $\mathcal{T}_i$ into a quadrangulation does not affect any triangulation $\mathcal{T}_j$ or any previously computed quadrangulation. As a result, the extension of our algorithm in Section 3.1 correctly converts constrained triangulations into constrained quadrangulations.

The following lemma provides upper bounds on the size of the constrained quadrangulation produced by the algorithm described above:

Lemma 3.2.1. *Let $\mathcal{T}$ be a constrained triangulation of a polygonal region with or without polygonal holes. The above algorithm converts $\mathcal{T}$ into a constrained and strictly convex quadrangulation with at most $\lfloor \frac{3t}{2} \rfloor + 4h$ quadrilaterals by inserting at most $t + 3h$ Steiner points, where t is the number of triangles of $\mathcal{T}$ and h is the number of connected components in the dual graph of $\mathcal{T}$.*

Proof. The conversion of each triangulation $\mathcal{T}_i$ into a quadrangulation may only affect another triangulation $\mathcal{T}_j$ ($j \neq i$ and $1 \leq j \leq h$) that has not been converted into a quadrangulation yet, except for the last triangulation considered by the algorithm, which does not affect any other triangulation or previously computed quadrangulation. Every time a triangulation $\mathcal{T}_j$ is affected by the conversion of another triangulation into a quadrangulation, the number of triangles in $\mathcal{T}_j$ is increased by 1 due to the repairing procedure shown in Fig. 19. Furthermore, each triangulation $\mathcal{T}_j$ can be affected at most x_j times where x_j is the number of children of the node v_j of T^c corresponding to the spanning tree T_j of G_j. Note that $\sum_{i=1}^{h} x_i = h - 1$, as $h - 1$ is the number of edges in T^c. Let t_i ($1 \leq i \leq h$) be the number of triangles in $\mathcal{T}_i$ before the algorithm converts any triangulation into a quadrangulation, and let t'_i be the number of triangles of $\mathcal{T}_i$ at the time the algorithm converts $\mathcal{T}_i$ into a quadrangulation. Then, we have that $t_i \leq t'_i \leq t_i + x_i$. From Theorem 3.1.1, we know that the algorithm in Section 3.1 produces $\lfloor \frac{3t'_i}{2} \rfloor + 2$ quadrilaterals on input $\mathcal{T}_i$. It follows therefore that the above algorithm produces at most $\sum_{i=1}^{h} \left(\lfloor \frac{3t'_i}{2} \rfloor + 2 \right) \leq \sum_{i=1}^{h} \left(\lfloor \frac{3(t_i + x_i)}{2} \rfloor \right) + 2h \leq \sum_{i=1}^{h} \frac{3t_i}{2} + \sum_{i=1}^{h} \frac{3x_i}{2} + 2h = \frac{3t}{2} + \frac{3h-3}{2} + 2h < \lfloor \frac{3t}{2} \rfloor + 4h$ quadrilaterals to convert $\mathcal{T}$ into a constrained quadrangulation. Furthermore, these quadrilaterals are all strictly convex, as the algorithm in Section 3.1 only generates strictly convex quadrilaterals. From Theorem 3.1.1, we also know that the algorithm in Section 3.1 inserts at most $t'_i + 2$ Steiner points to convert $\mathcal{T}_i$ into a constrained quadrangulation. Thus, the algorithm extension inserts at most $\sum_{i=1}^{h} (t' + 2) \leq \sum_{i=1}^{h} (t_i + x_i + 2) = t + h - 1 + 2h < t + 3h$ Steiner points to convert $\mathcal{T}$ into a constrained quadrangulation. □

3.3. *Small polygonal regions*

As seen in Section 3.1 and Section 3.2, our algorithm for converting a triangulation into a quadrangulation makes use of several facts about the quadrangulation of small and simple polygonal regions, i.e., regions consisting of 4, 5, 6, or 7 boundary edges and no holes. In this section, we list lemmas that formally state such facts and whose proofs can be easily made into algorithmic procedures. We start with two useful facts about convex quadrangulations of 4- and 6-sided polygons, which were given and proved in Bremner et al. [12]:

Lemma 3.3.1. *A hexagon can be decomposed into at most four convex quadrilaterals by using at most three Steiner points in its interior.*

Lemma 3.3.2. *A quadrilateral with a point in its interior can be decomposed into at most five convex quadrilaterals by using at most three Steiner points in its interior.*

For polygonal regions bounded by an odd number of edges, one of the boundary edges is designated as an *outgoing edge*. (In the algorithm described in Section 3.1 the outgoing edge is simply the triangulation edge between the root of subtree T_v and its parent node.) When quadrangulating this region, all Steiner points except one are placed in the interior of the polygon, and one Steiner point may be placed on the outgoing edge. In the following lemmas, these relevant facts are stated and proved formally:

Definition 3.3.1. Given two points p and q, we denote by $L(p, q)$ (resp. $R(p, q)$) the left (resp. right) open half-space defined by the oriented line from p to q. Given a vertex v of a polygon P, we define $wedge(v)$ as follows: If v is reflex then $wedge(v)$ denotes the locus of points inside P that can be connected to v forming strictly convex angles at v. If v is convex then $wedge(v)$ is the interior of the visibility region of v in P. Now, assume that P is a simple polygon. Given a triangulation $\mathcal{T}$ of P such that $V_{\mathcal{T}} = V_P$, a triangle $\Delta = (p, q, r)$ of $\mathcal{T}$ is said to be an *ear* if and only if p, q, and r are consecutive vertices of a counterclockwise (or clockwise) enumeration of the vertices of P.

Definition 3.3.2. Let P be a pentagon and let e be an edge of P. Given a triangulation $\mathcal{T}$ of P such that $V_{\mathcal{T}} = V_P$, the triangulation $\mathcal{T}$ necessarily consists of three triangles, two of which are ears (see Fig. 20). Each of these ears shares two vertices and a distinct diagonal of $\mathcal{T}$ with the third triangle, called the center triangle. Furthermore, the edge e is said to be of *type 1* with respect to $\mathcal{T}$ if it is the edge of P shared with the center triangle of $\mathcal{T}$. If e is not of type 1 and e is adjacent to the *type 1* edge, it is said to be of *type 2* with respect to $\mathcal{T}$. If e is neither of type 1 nor of type 2, then e is incident to the common vertex of all triangles of $\mathcal{T}$ and is said to be of *type 3*.

Lemma 3.3.3. *Let P be a pentagon and let e be the outgoing edge of P. Then, given any triangulation $\mathcal{T}$ of P such that $V_{\mathcal{T}} = V_P$, we have the following:*

(1) If e is of type 1 with respect to $\mathcal{T}$, P can be decomposed into two convex quadrilaterals and one triangle adjacent to e by adding one Steiner point inside P.

(2) If e is of type 2 with respect to $\mathcal{T}$ then, P can be decomposed into three convex quadrilaterals and one triangle adjacent to e by adding two Steiner points inside P.

(3) If e is of type 3 with respect to $\mathcal{T}$ then, P can be decomposed into four convex quadrilaterals by adding two Steiner points inside P and one more on the edge e.

Proof. Let v_1, v_2, v_3, v_4, v_5 be a counterclockwise enumeration of the vertices of P. Let $\mathcal{T}$ be a triangulation of P such that $V_{\mathcal{T}} = V_P$ and refer to Fig. 21. Without loss of generality, assume that the center triangle, T_c, of $\mathcal{T}$ is defined by the vertices v_4, v_1 and v_2, and v_4 is the common vertex of all three triangles of $\mathcal{T}$. If e is of type 1 then e is the edge $\overline{v_1v_2}$ of P. Let $R_1 = wedge(v_3) \cap wedge(v_5) \cap T_c$. Note that the interior of R_1 cannot be empty, and clearly $R_1 \subset P$. Denote the barycenter of R_1 by p_1. By removing edges $\overline{v_4v_1}$ and $\overline{v_4v_2}$ from $\mathcal{T}$, and then adding the point p_1 and the edges $\overline{v_1p_1}$, $\overline{p_1v_4}$, and $\overline{p_1v_2}$, we obtain a decomposition of P into two quadrilaterals, $Q_1 = (v_1, p_1, v_4, v_5)$ and $Q_2 = (v_4, p_1, v_2, v_3)$, and one triangle, $T_1 = (v_1, v_2, p_1)$, which contains $e = \overline{v_1v_2}$. Since p_1 belongs to both $wedge(v_3) \cap T_c$ and $wedge(v_5) \cap T_c$, Q_1 and Q_2 are strictly convex quadrilaterals, and hence Claim 1 is true.

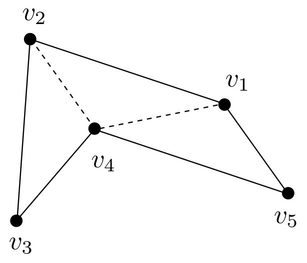

Fig. 20. Triangulation $\mathcal{T}$ of a pentagon P with $V_{\mathcal{T}} = V_P$.

If e is of type 2 then e is either $\overline{v_2v_3}$ or $\overline{v_5v_1}$. Assume that $e = \overline{v_2v_3}$ and refer to Fig. 22. By removing diagonals $\overline{v_4v_1}$ and $\overline{v_4v_2}$ from $\mathcal{T}$, and then adding the point p_1 above and the edges $\overline{v_1p_1}$ and $\overline{p_1v_4}$, we obtain a decomposition of P into the quadrilateral $Q_1 = (v_1, p_1, v_4, v_5)$ and the pentagon $P' = (v_2, v_3, v_4, p_1, v_1)$. Since diagonals $\overline{p_1v_2}$ and $\overline{v_3p_1}$ are inside P', the triangle (p_1, v_2, v_3) is entirely contained in P', and we can decompose P' into triangles $T_c' = (p_1, v_2, v_3)$, (p_1, v_1, v_2), and (v_3, v_4, p_1), where T_c' is the center triangle and (p_1, v_1, v_2), and (v_3, v_4, p_1) are ears of the triangulation (see Fig. 22(a)). Let R_2 be the region $R_2 = wedge(v_1) \cap wedge(v_4) \cap T_c'$. Note that $R_2 \subset P' \subset P$, and the interior of R_2 cannot be empty. Define p_2 to be the barycenter of R_2. We can now decompose P' as in (1) to obtain two strictly convex quads, $Q_3 = (v_2, p_2, p_1, v_1)$ and $Q_4 = (v_3, v_4, p_1, p_2)$, and a triangle $T_2 = (v_3, p_2, v_2)$ adjacent to e. If $e = \overline{v_5 v_1}$ we have the symmetric case, and hence Claim 2 holds.

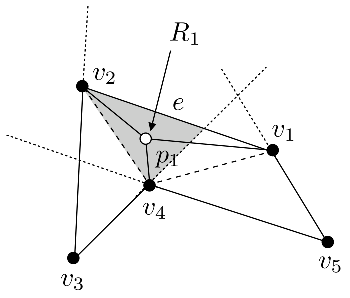

Fig. 21. Illustration of Claim 1 of Lemma 3.3.3.

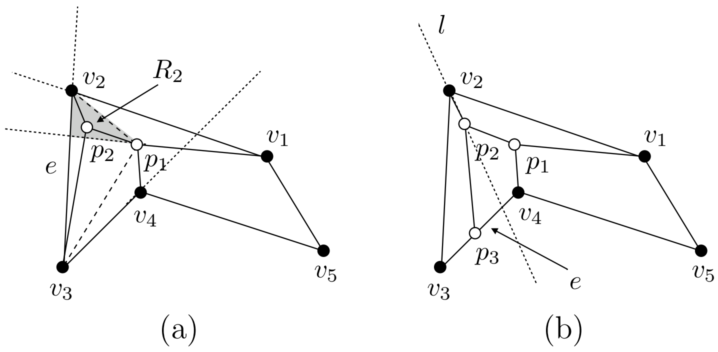

Fig. 22. (a) Illustration of Claim 2, and (b) Claim 3 of Lemma 3.3.3.

If e is of type 3, then e is either $\overline{v_3 v_4}$ or $\overline{v_4 v_5}$. Assume that $e = \overline{v_3 v_4}$. By removing diagonals $\overline{v_4 v_1}$ and $\overline{v_4 v_2}$ from $\mathcal{T}$, and then adding the points p_1 and p_2 above plus the edges $\overline{v_1 p_1}$, $\overline{p_1 v_4}$, $\overline{p_1 p_2}$ and $\overline{p_2 v_2}$, we obtain a decomposition of P into strictly convex quads $Q_1 = (v_1, p_1, v_4, v_5)$ and $Q_3 = (v_2, p_2, p_1, v_1)$, and the pentagon $P'' = (v_2, v_3, v_4, p_1, p_2)$. Let $l = \overrightarrow{v_2 p_2}$ be the oriented line from v_2 to p_2. Note that v_3 is on the right side of l and the line l. If v_4 is also on the right of l, we define p_3 to be the midpoint of the edge $\overline{v_3 v_4}$. Otherwise, the line l intersects the edge $\overline{v_3 v_4}$, and we define p_3 to be the midpoint of the segment defined by v_3 and the intersection point of l and $\overline{v_3 v_4}$ (see Fig. 22b). By adding p_3 and the edge $\overline{p_2 p_3}$ to P'', we obtain a decomposition of P'' into two quadrilaterals, $Q_5 = (v_2, v_3, p_3, p_2)$ and

$Q_6 = (v_4, p_1, p_2, p_3)$. Since p_3 lies on the right side of l, both Q_5 and Q_6 are strictly convex quadrilaterals and therefore the set consisting of the convex quadrilaterals Q_1, Q_3, Q_5 and Q_6 is a decomposition of P that uses three Steiner points, two of which are inside P and the third is on the edge $e = \overline{v_3v_4}$. If $e = \overline{v_4v_5}$ we have the symmetric case, and hence Claim 3 also holds. □

In steps 3c and 4c(v) of our quadrangulation algorithm, the subgraph G_v corresponds to a triangulation $\mathcal{T}_v$ of a pentagon P such that $\mathcal{T}_v$ has exactly one vertex q inside P (see Fig. 14). The following lemma is applied in these cases:

Lemma 3.3.4. *Let P be a pentagon, q a point in its interior, and e the outgoing edge of P. Then, P can be decomposed into at most six convex quadrilaterals and one triangle adjacent to e by inserting at most four Steiner points inside P.*

Proof. Let $\mathcal{T}$ be any triangulation of P such that $V_{\mathcal{T}} = V_P$. Let T_1, T_2, and T_3 be the three triangles of $\mathcal{T}$, where T_2 is the center triangle and T_1 and T_3 are the ears. There are two possibilities: The point q belongs to the triangle containing e, or it does not. In the former case, let Δ be the triangle obtained by connecting q to the two endpoints of e. P is thus decomposed into Δ, which is adjacent to the outgoing edge, and a hexagon H (see Fig. 23(a)). From Lemma 3.3.1, H can be decomposed into at most four convex quadrilaterals using at most three Steiner points in its interior.

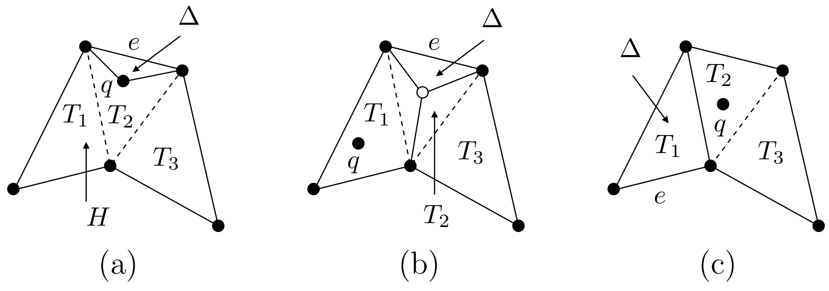

Fig. 23. Illustrations of the possible cases (up to symmetry) of Lemma 3.3.4.

Suppose now that q does not belong to the triangle containing e. If e belongs to T_2, decompose P into a triangle Δ adjacent to e and two convex quadrilaterals, as in case (1) of Lemma 3.3.3 (see Fig. 23(b)). One of these quadrilaterals must contain q, which can be decomposed into five convex quads using three more Steiner points by applying Lemma 3.3.2. Thus, P is decomposed into six convex quads and a triangle adjacent to e by using four Steiner points. If e belongs to T_1, let $\Delta = T_1$. Triangles T_2 and T_3 form a quadrilateral with a point inside (see Fig. 23(c)), to which we apply Lemma 3.3.2. Thus, P is decomposed into five convex quads and a triangle adjacent to e by using three Steiner points. The case when e belongs to T_3 is symmetric. □

In our algorithm, a polygonal region S bounded by seven edges (septagon) is obtained when the subtree T_v is a path of five nodes, with the middle node as the root of T_v. (This is the only case that results in a septagon.) Hence, in this case, the outgoing edge is always the edge of S belonging to the middle node. The triangulation $\mathcal{T}_v$ of S also has a center triangle, the triangle corresponding to the middle node of T_v. This triangle contains only one edge of S, the outgoing one. Figure 24 illustrates a triangulation of a septagon whose dual graph is a path of five nodes.

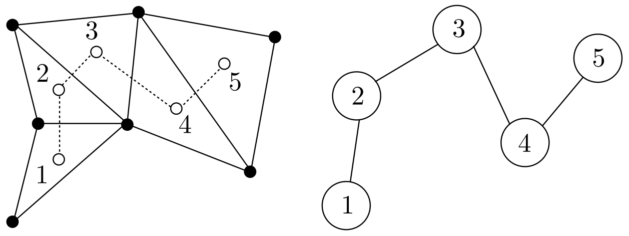

Fig. 24. Triangulation of a septagon and its dual graph.

Lemma 3.3.5. *Let S be a septagon such that S admits a triangulation $\mathcal{T}$, with $V_{\mathcal{T}} = V_S$, whose dual graph is a path. Let the edge of S contained in the center triangle of $\mathcal{T}$ be the outgoing edge e. Then, S can be decomposed into six convex quadrilaterals and one triangle adjacent to e by adding four Steiner points inside S.*

Proof. Let S be a septagon such that S admits a triangulation $\mathcal{T}$, with $V_{\mathcal{T}} = V_S$, whose dual graph is a path. Let $v_1, v_2, \ldots, v_7$ be a counterclockwise enumeration of the vertices of S. Without loss of generality, assume that the center triangle T_c of $\mathcal{T}$ is $T_c = (v_1, v_2, v_5)$. So, there are at most four possibilities for the other four triangles of $\mathcal{T}$, T_1, T_2, T_3 and T_4 (see Fig. 25): (a) $T_1 = (v_5, v_3, v_4)$, $T_2 = (v_5, v_2, v_3)$, $T_3 = (v_5, v_7, v_1)$, and $T_4 = (v_5, v_6, v_7)$; (b) $T_1 = (v_5, v_3, v_4)$, $T_2 = (v_5, v_2, v_3)$, $T_3 = (v_5, v_6, v_1)$, and $T_4 = (v_6, v_7, v_1)$; (c) $T_1 = (v_4, v_2, v_3)$, $T_2 = (v_5, v_2, v_4)$, $T_3 = (v_5, v_7, v_1)$, and $T_4 = (v_5, v_6, v_7)$; and (d) $T_1 = (v_4, v_2, v_3)$, $T_2 = (v_5, v_2, v_4)$, $T_3 = (v_5, v_6, v_1)$, and $T_4 = (v_6, v_7, v_1)$. Assume that $\mathcal{T}$ consists of T_c and the four triangles shown in Fig. 25(a). Below, we describe how to place four Steiner points, p_1, p_2, p_3, and p_4, in order to decompose S into six convex quads and one triangle adjacent to e. The proofs for the other possibilities are very similar and we omit them here.

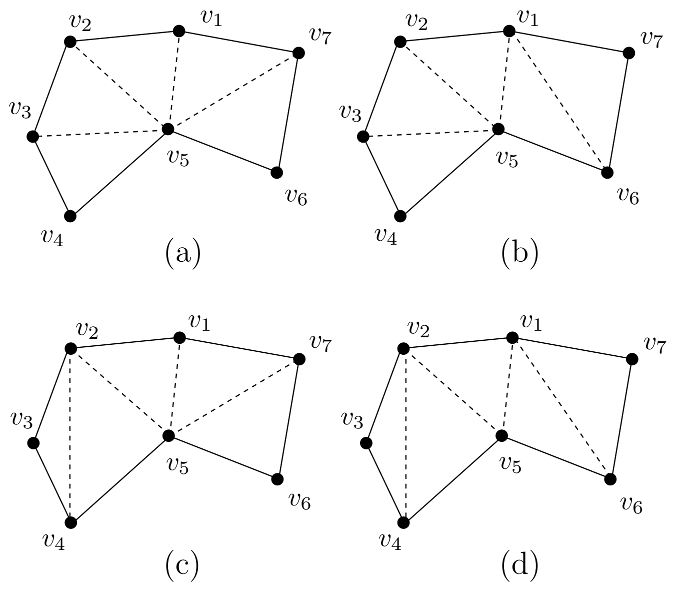

Fig. 25. All triangulations of a septagon whose dual graphs are paths.

Let $R_1 = wedge(v_4) \cap wedge(v_1) \cap T_2$. Clearly, $R_1 \subseteq T_2$, $v_5 \in R_1$, and the interior of R_1 cannot be empty. Similarly, let R_2 be the non-empty region $R_2 = wedge(v_6) \cap wedge(v_2) \cap T_3$. Now, consider the two following cases (see Fig. 26): (i) $v_2 \in R(v_4, v_5)$ and (ii) $v_2 \notin R(v_4, v_5)$. Recall that $R(p, q)$ (resp. $L(p, q)$) is the right (resp. left) open half-space defined by the oriented line from p to q (see Definition 3.3.1). In case (i), because $\overline{v_5v_2}$, $\overline{v_5v_1}$, and $\overline{v_5v_7}$ are edges of $\mathcal{T}$ and $v_2 \notin R(v_4, v_5)$, the intersection region $R_1 \cap R(v_7, v_5)$ cannot be empty. So, we choose p_1 and p_2 to be the barycenters of $R_1 \cap R(v_7, v_5)$ and R_2, respectively. In case (ii), we choose p_1 to be the barycenter of R_1, and we calculate the position of p_2 as follows: If $v_1 \in R(v_6, v_5)$ then p_2 is the barycenter of $R_2 \cap L(p_1, v_5)$. The region $R_2 \cap L(p_1, v_5)$ cannot be empty. Otherwise, the vertex v_6 would belong to $L(p_1, v_5)$ and the vertex v_7 would belong to the complement of $L(p_1, v_5)$, which is an absurd as $v_6 \in R(v_4, v_5)$ and $v_4 \in R(p_1, v_5)$. If $v_1 \notin R(v_6, v_5)$ then p_2 is the barycenter of $R_2 \cap L(v_3, v_5)$. Again, $R_2 \cap L(v_3, v_5)$ cannot be empty as the oriented line $\overrightarrow{v_3v_5}$ from v_3 to v_5 passes through the interior of $wedge(v_2) \cap wedge(v_6)$ and v_1 is on the left of $\overrightarrow{v_3v_5}$.

In both cases (i) and (ii), we have that $p_2 \in L(p_1, v_5)$. Since $p_1 \in T_2$ and $p_2 \in T_3$, we have that the oriented lines $l_1 = \overrightarrow{v_3p_1}$ and $l_2 = \overrightarrow{v_7p_2}$ intersect the edges $\overline{v_5v_2}$ of T_2 and $\overline{v_5v_1}$ of T_3, respectively. Besides, since $p_1 \in wedge(v_1)$ and $p_2 \in wedge(v_2)$, the oriented lines $l_3 = \overrightarrow{p_1v_1}$ and $l_4 = \overrightarrow{p_2v_2}$ intersect the edges $\overline{v_5v_2}$ of T_2 and $\overline{v_5v_1}$ of T_3, respectively. If q and r denote the intersection points of l_1 and l_3 with $\overline{v_5v_2}$, respectively, then v_2, q and r are collinear, and we define p_3 to be the midpoint of $\overline{v_2x}$, where $x = q$ if q is in between v_2 and r, and $x = r$ otherwise. Similarly, If s and t denote the intersection points of l_2 and l_4 with $\overline{v_5v_1}$, respectively, then v_1, s and t are collinear, and we choose p_4 to be the midpoint of $\overline{v_1y}$, where $y = s$ if s is in between v_1 and t, and $y = t$ otherwise. In this way, we have that $p_3 \in L(p_1, v_1)$ and $p_4 \in R(p_1, v_1)$. Now, by removing edges $\overline{v_3v_5}$, $\overline{v_2v_5}$, $\overline{v_1v_5}$, and $\overline{v_7v_5}$ from $\mathcal{T}$, and then adding points p_1, p_2, p_3 and p_4 and edges $\overline{v_5p_1}$, $\overline{p_1v_3}$, $\overline{p_1p_3}$, $\overline{p_1v_3}$, $\overline{v_5p_2}$, $\overline{p_2v_7}$, $\overline{p_2p_4}$, $\overline{p_1p_4}$, and $\overline{p_3v_1}$, we obtain a decomposition of S into six quadrilaterals, $Q_1 = (v_4, v_5, p_1, v_3)$, $Q_2 = (v_3, p_1, p_3, v_2)$, $Q_3 = (p_1, v_5, p_2, p_4)$, $Q_4 = (p_1, p_4, v_1, p_3)$, $Q_5 = (v_5, v_6, v_7, p_2)$, and $Q_6 = (v_7, v_1, p_4, p_2)$, and one triangle, $T = (p_3, v_1, v_2)$, as shown in Fig. 27.

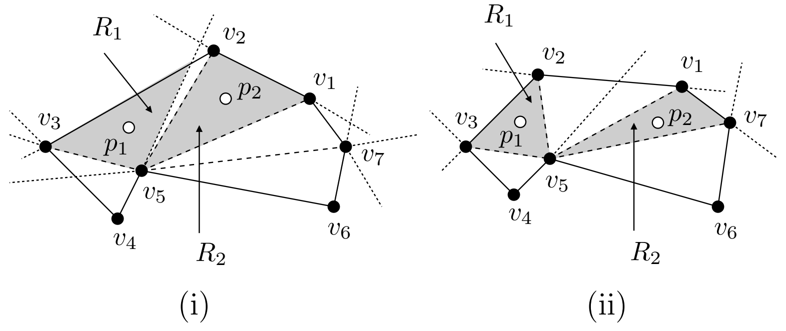

Fig. 26. Choosing points p_1 and p_2 when (i) $v_2 \in R(v_4, v_5)$ and (ii) $v_2 \notin R(v_4, v_5)$.

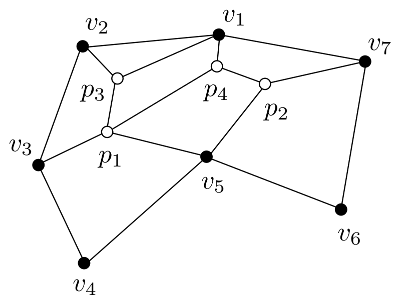

Fig. 27. Resulting decomposition from the triangulation in Fig. 25(a).

Since p_1 and p_2 belong to the interior of $wedge(v_4) \cap T_2)$ and $wedge(v_6) \cap T_3$, respectively, we have that Q_1 and Q_5 are strictly convex quadrilaterals. Since

$p_3 \in L(v_3, p_1)$ and $p_4 \in R(v_7, p_2)$, we get that Q_2 and Q_6 are also strictly convex quadrilaterals. Since $p_2 \in L(p_1, v_5)$, $p_2 \in R(v_5, v_1)$, $p_1 \in L(v_5, v_1)$, $p_4 \in L(p_1, p_2)$, and p_4 is a point on $\overline{v_5 v_1}$, we get that Q_3 is a strictly convex quadrilateral. Finally, because $p_3 \in L(p_1, v_1)$, $p_4 \in R(p_1, v_1)$, p_3 is a point on $\overline{v_5 v_2}$, and p_4 is a point on $\overline{v_5 v_1}$, we get that Q_4 is also a strictly convex quadrilateral. Thus, our claim holds when T_1, T_2, T_3 and T_4 are the triangles corresponding to possibility (a). □

4. Implementation, Results and Quality Measurements

We implemented the algorithm described in Section 3.1, and its extension in Section 3.2, using C++ and the open source and freely available software *Computational Geometry and Algorithms Library* (CGAL)[c] [25]. Figure 28(a) and Figure 28(b) show the quadrilateral meshes obtained from the triangular meshes in Fig. 29(a) and Fig. 29(b), respectively, using the aforementioned implementation of our algorithm. The triangular meshes in Figures 29(a)–(b) were produced by the software `Triangle` [6] from the PSLGs shown on top of the meshes. `Triangle` allows us to set a lower bound for the smallest angle of every triangle in the mesh, except for the angles defined by two triangle edges covering an edge of the input PSLG. Note that a lower bound on the smallest angle of a triangle implies the following upper bound on its largest angle: if θ is the smallest angle then the largest angle is at most $180^{\circ} - 2\theta$. For several FE-based applications, triangles with very small angles are extremely undesirable [1]. Both meshes in Figures 29(a)–(b) were generated by choosing a lower bound of 30° for the smallest angle.

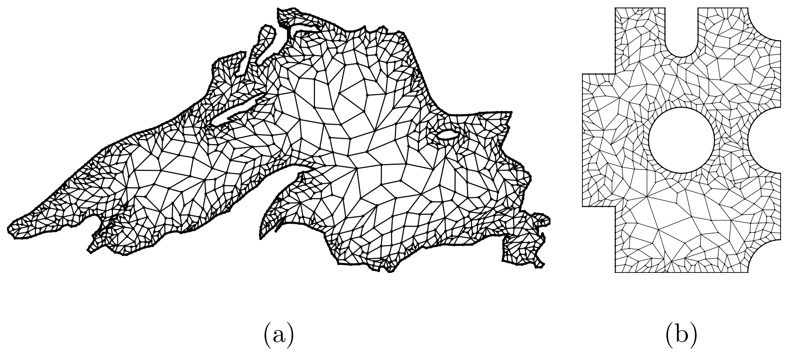

Fig. 28. (a) Quadrilateral mesh of a polygonal region of Lake Superior's shape. (b) Quadrilateral mesh of a CAD model.

The quadrilateral meshes in Fig. 28(a) and Fig. 28(b) have 1249 and 511 quadrilaterals, while their triangular counterparts in Fig. 29(a) and Fig. 29(b) have 2160

[c]`http://www.cgal.org`

and 912 triangles, respectively. Our quadrilateral meshing algorithm also used 169 and 55 Steiner points to generate the quadrilateral meshes in Fig. 28(a) and Fig. 28(b), respectively. So, even though our algorithm may, in the worst case, produce up to $\lfloor \frac{3t}{2} \rfloor$ quadrilaterals and use up to $t+2$ Steiner points on an input triangular mesh with t triangles, in these examples the number of quadrilaterals and the number of Steiner points are about 60% and 8%, respectively, of the number of triangles of the input triangular meshes. It turns out that this reduction in mesh size and the use of only a few Steiner points were observed in almost all the test cases on which we ran our algorithm.

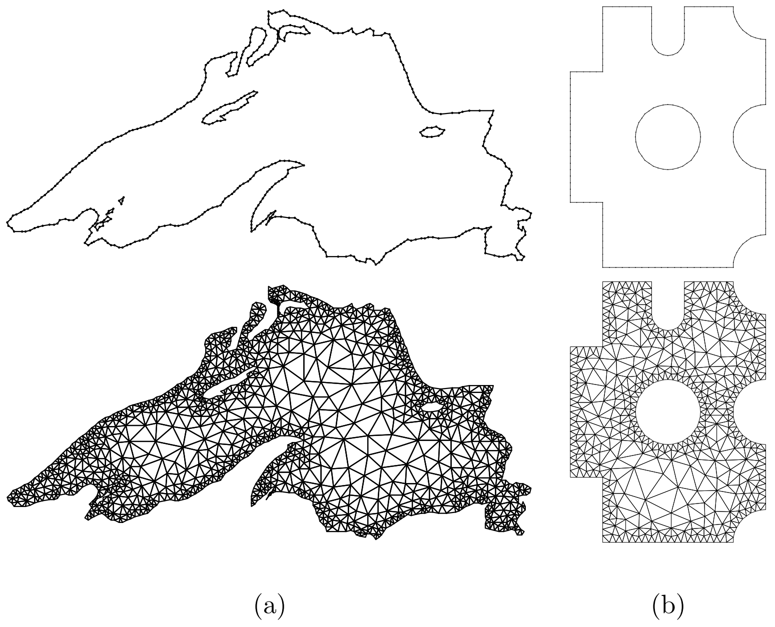

Fig. 29. (a) Triangular mesh of an outline of Lake Superior given by the PSLG on the top. (b) Triangular mesh of a CAD model given by the PSLG on the top. Both meshes were generated by Triangle.

We also noticed that, for a given polygonal region, the greater the number of triangles in the triangular mesh, and the greater the lower bound on the smallest angle of its triangles, the smaller the percentage of quadrilaterals generated and Steiner points inserted by our algorithm (with respect to the number of triangles of the input triangular mesh). Table 2 illustrates this observation. Each row of Table 2 contains the number of triangles of an input triangular mesh generated by Triangle from the PSLG of Lake Superior in Fig. 29(a) with a distinct lower bound for the smallest angle of the triangles. The table also shows the number of quadrilaterals and Steiner points of the corresponding quadrilateral meshes generated by our algorithm, and their percentages with respect to the number of triangles in the input triangular meshes.

Smallest Angle	# Tris.	# Quads.	# Steiner Points	% Size Reduction	% Steiner Points
25°	1422	908	197	36%	14%
30°	2160	1249	169	42%	8%
33°	3346	1846	173	45%	5%

Table 2. Percentage of output quadrilaterals and Steiner points of meshes generated by our algorithm from distinct triangular meshes of the outline of Lake Superior in Fig. 29(a).

We support the observation illustrated in Table 2 with the following conjecture: *If we increase the number of triangles of a triangular mesh for a particular polygonal region and we also increase the lower bound of the smallest angle of its triangles, then the likelihood of two triangles sharing a common edge to form a strictly convex quadrilateral also increases.* As a result, Case 4c of our algorithm will be applied more often (see Section 3.1), which causes the number of Steiner points and the ratio between the number of output quadrilaterals and input triangles to be smaller than they would be if, for the same polygonal region, we had used a smaller input triangular mesh with a larger bound on the smallest triangle angle.

Both Fig. 28(a) and Fig. 28(b) highlight another feature of our algorithm: Preservation of mesh grading, which is also present in the algorithms by Owen et al.[24], de Berg[8], and Everett et al[8]. Since triangles of a small triangulated region of the triangulation have about the same area and our algorithm is based on the conversion of small polygon triangulations into quadrangulations with similarly sized quadrilaterals, the average area of the resulting quadrilaterals tend to be proportional to the average area of the triangles in the same region. Hence, the grading of the input triangular mesh tends to be preserved.

While our algorithm constructs quadrilateral meshes of bounded size and tends to preserve mesh grading, it does not provide any guarantee on the shape quality of the output quadrilaterals, no matter what criteria are used to measure shape quality. Since the shape of the elements of a mesh has a strong influence on the results of numerical simulations involving the mesh [1,2] this is a considerable weakness of our algorithm. Fortunately, we can improve the shape quality of the quadrilaterals generated by our algorithm by using post-processing techniques. These techniques optimize the shape quality of a mesh according to some quality criterion and at the expense of runtime. Figure 30(a) and Figure 30(b) show quadrilateral meshes obtained from the ones in Fig. 28(a) and Fig. 28(b), respectively, by using a smoothing technique called *angle-based smoothing* [26]. The idea behind this technique is to iteratively change the position of the vertices of the quadrilateral mesh, so that the two adjacent angles defined at the same vertex by each diagonal of a quadrilateral become equal or distinct within a certain ratio. The resulting quadrilaterals are supposed to be less distorted with respect to a reference square. Topological operations, such as the *cleanup* procedures in Canann et al. [27] can also be used to improve mesh regularity. We can quantitatively evaluate the effect of angle-based smoothing on the quadrilateral meshes generated by our algorithm by analyzing a measure of shape quality defined by Joe[18].

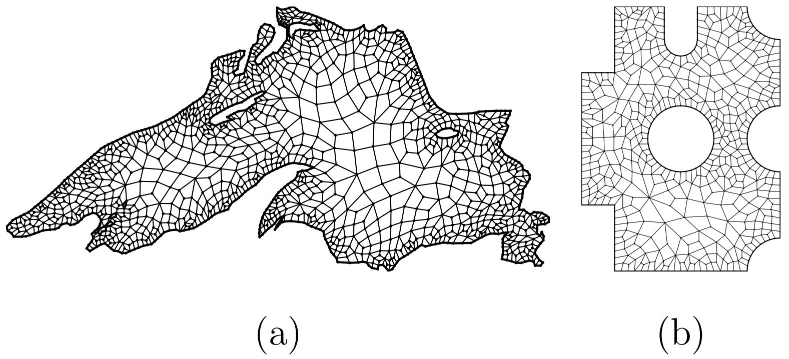

Fig. 30. (a) Quadrilateral mesh resulting from post-processing of the mesh in Fig. 28(a). Quadrilateral mesh resulting from post-processing of the mesh in Fig. 28(b).

Let $Q = [a, b, c, d]$ be a strictly convex quadrilateral with vertices a, b, c, and d. Then, we define a quality measure μ as follows. The quadrilateral Q can be decomposed into two triangles by either adding the diagonal $\overline{bd}$ or the diagonal $\overline{ac}$ to it. In the former case, we have the triangles $T_1 = [a, b, d]$ and $T_2 = [c, d, b]$. In the latter case, we have the triangles $T_3 = [a, b, c]$ and $T_4 = [d, a, c]$. We define μ to be the smallest angle among the angles of the triangles T_1, T_2, T_3, and T_4 (see Fig. 31). Note that the value of μ is 45^o if Q is a square, and that the value of μ approaches 0^o if one or two edges of Q is much shorter than the other edges or Q has an angle near 0^o or 180^o. In fact, it can be shown that the maximum value of μ is 45^o, and that this value is only attained by the square [18]. We can interpret the value of μ as a measure of how distorted the triangles T_1, T_2, T_3, and T_4 are with respect to a reference right, isosceles triangle. This interpretation was recently formalized by Pébay[28] in a study of the asymptotic behavior of another angle-based quality measure of quadrilaterals, which is very similar to μ.

Table 3 shows the frequency distribution of the μ values of the quadrilaterals of six quadrilateral meshes generated by our algorithm. Each quadrilateral mesh was obtained from a triangular mesh of either the outline of Lake Superior in Fig. 29(a) or the CAD model in Fig. 29(b) using a distinct lower bound for the smallest angle of the triangles. Table 4 shows the same information as Table 3 for the quadrilateral meshes obtained by post-processing the quadrilateral meshes in Table 3 using five iterations of the angle-based smoothing. By examining both tables, we can see that the angle-based smoothing increases the frequency distribution of higher values of μ and reduces the frequency distribution of lower values of μ.

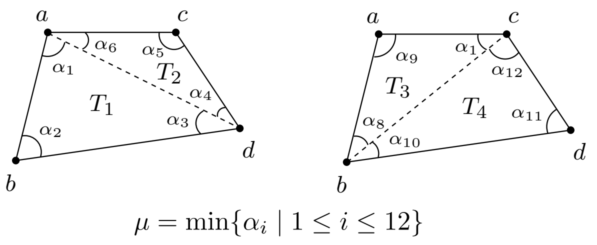

Fig. 31. Illustration of the definition of the quality measure μ.

Mesh	# Quads.	$[0^o, 5^o)$	$[5^o, 10^o)$	$[10^o, 25^o)$	$[25^o, 35^o)$	$[35^o, 45^o]$	Runtime
1	910	30	409	362	102	5	1021 ms
2	1249	11	430	566	228	14	1401 ms
3	1846	16	478	878	427	47	2992 ms
4	404	10	149	190	52	3	369 ms
5	511	2	170	225	110	4	429 ms
6	674	1	170	317	174	12	644 ms

Table 3. Meshes 1 (resp. 4), 2 (resp. 5) , and 3 (resp. 6) were obtained by our algorithm from a triangular mesh of the outline of Lake Superior (resp. CAD model) in Fig. 29(a) (resp. Fig. 29(b)) using a lower bound of 25^o, 30^o, and 33^o for the smallest triangle angle, respectively. The second column from left to right shows the number of quadrilaterals of the meshes. The rightmost column shows the time our meshing algorithm took to generate the mesh.

Mesh	# Quads.	$[0^o, 5^o)$	$[5^o, 10^o)$	$[10^o, 25^o)$	$[25^o, 35^o)$	$[35^o, 45^o]$	Runtime
7	910	9	239	433	205	24	7555 ms
8	1249	2	166	587	458	36	10956 ms
9	1846	1	124	776	830	115	17788 ms
10	404	1	75	221	93	14	3380 ms
11	511	2	38	266	191	14	4791 ms
12	674	0	40	268	325	41	6723 ms

Table 4. Meshes 7, 8, 9, 10, 11, and 12 are the resulting meshes from the application of five iterations of the angle-based smoothing to the meshes 1, 2, 3, 4, 5, and 6 in Table 3. The rightmost column shows the time our implementation of the angle-based smoothing took to produce the improved mesh in five iterations.

The angle-based smoothing also increases the uniformity of the area of the mesh elements. Table 5 illustrates this effect by showing the standard deviation of the average area of the elements of meshes 1–12 in Table 3 and Table 4. Although our implementation of the angle-based smoothing considerably improved the quality of the meshes in Table 3 generated by our meshing algorithm, the time to produce each improved mesh in Table 4 was at least 7 times longer than the time our meshing algorithm took to generate the corresponding mesh in Table 3. However, this post-processing step may be necessary to improve the accuracy of FE analysis using the quadrilateral meshes generated by our algorithm.

Mesh	Average Area	Standard Deviation	Mesh	Standard Deviation
1	1.7×10^{-4}	3.1×10^{-4}	7	2.1×10^{-4}
2	1.2×10^{-4}	1.6×10^{-4}	8	1.2×10^{-4}
3	8.2×10^{-5}	6.9×10^{-5}	9	5.7×10^{-5}
4	8.9	10.9	10	6.7
5	7.1	6.5	11	4.1
6	5.4	3.2	12	2.1

Table 5. Average area of the quadrilaterals in the meshes in Table 3 (before post-processing) and Table 4 (after post-processing). Note that the average area of the meshes in the same row are equal, as the angle-based smoothing does not change the mesh domain.

In a recent work [17], we used our implementation of the algorithm in Section 3.2 to create constrained quadrilateral meshes from two-dimensional images of the human brain. The quality of the meshes generated by our algorithm were compared to the quality of their triangular counterparts and uniform quadrilateral grids with respect to the performance of a FE-based image registration method. The results in [17] show that the target image registration application has a better performance if it uses the meshes generated by our algorithm instead of approximately same-sized uniform quadrilateral grids. The results also show that the target image registration application performs about the same if either the meshes generated by our algorithm, or their denser triangular counterparts are used. Figure 32(c) shows a constrained quadrilateral mesh generated by our algorithm from the constrained triangular mesh of a human brain in Fig. 32(b). Figure 32(d) shows the quadrilateral mesh resulting from post-processing the mesh in Fig. 32(c) using 5 iterations of the angle-based smoothing. The constrained triangular mesh was generated by `Triangle` from the PSLG in Fig. 32(a).

The fact that our algorithm places a Steiner point outside the triangulation domain may be highly undesirable in practice. However, our algorithm can be slightly modified so that no Steiner point is placed outside the triangulation domain. Note that a Steiner point is placed outside the triangulation domain only if one of sub-steps 5a, 5b, and 5d is carried out. We can modify these sub-steps so that, in sub-steps 5b and 5d, instead of adding a Steiner point to the triangulation boundary in order to convert the leftover triangle into a quadrilateral, we subdivide the leftover triangle into five strictly convex quadrilateral by adding five Steiner point in the interior of the triangle and one on the boundary (see Ref. 18 for a proof), as shown by Fig. 33. By doing so, sub-cases 5b and 5d insert 6 and 5 Steiner points into the resulting quadrangulation, respectively.

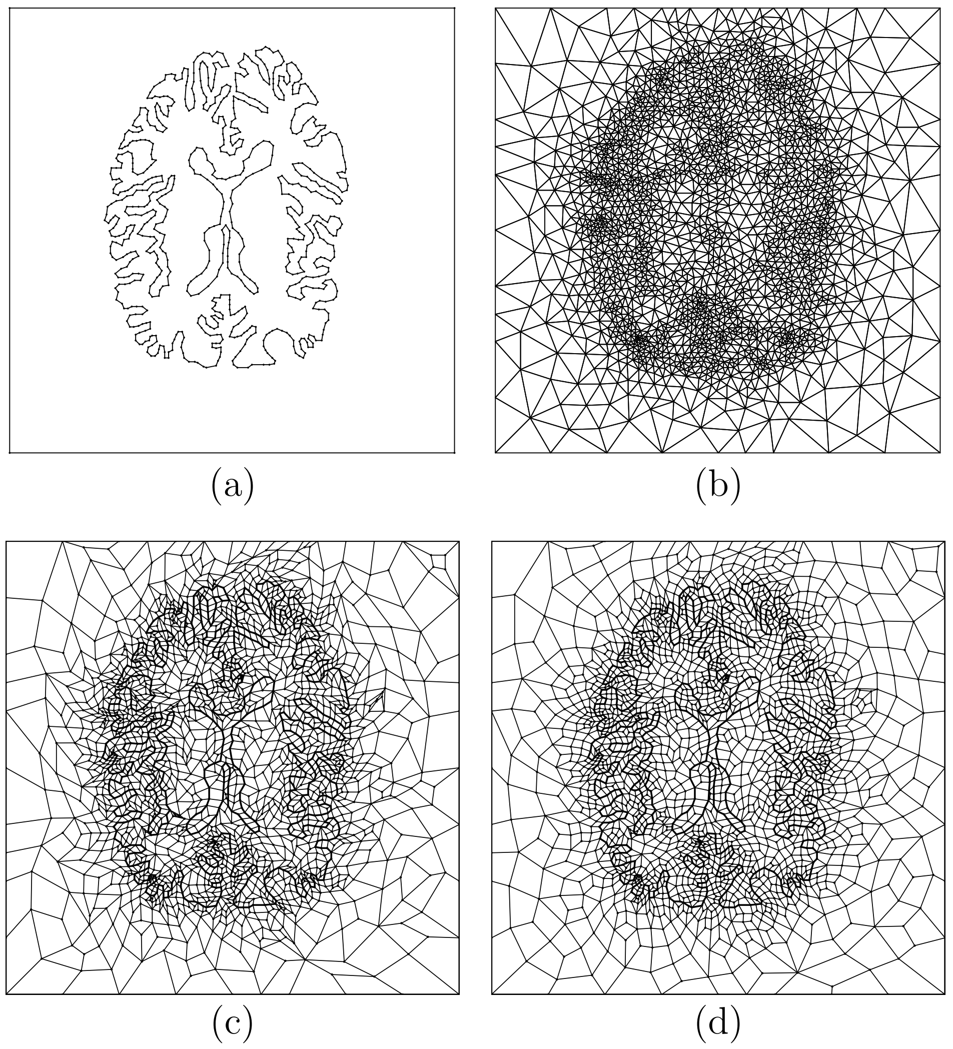

Fig. 32. (a) PSLG describing a polygonal approximation for the domain of an image of a human brain. (b) Constrained triangular mesh generated by Triangle from the PSLG in (a). (c) Constrained quadrilateral mesh generated by the algorithm in Section 3.2. (d) Quadrilateral mesh resulting from the post-processing of the mesh in (c) using 5 iterations of the angle-based smoothing.

In sub-step 5a, recall that we have two possible situations for the subtree T_v of T rooted at $par(v)$: T_v corresponds to either a triangulated pentagon or a non-empty triangle. If T_v corresponds to a triangulated pentagon, then we can apply Lemma 3.3.3 to convert the triangulation $\mathcal{T}_v$ into a quadrangulation and a leftover triangle that contains a boundary edge. Next, we subdivide the leftover triangle into five strictly convex quadrilaterals by using five Steiner points inside the triangle. In this case, we insert exactly six Steiner points into the resulting quadrangulation. If T_v corresponds to a non-empty triangle, then we can insert a Steiner point in the interior of $ele(v)$, so that the elements $ele(v)$ and $ele(sib(v))$ form a quadrilateral with a vertex in its interior. Hence, we can apply Lemma 3.3.2 to subdivide the quadrilateral formed by $ele(v)$ and $ele(sib(v))$ into at most five strictly convex quadrilateral using at most three Steiner points. All Steiner points are placed in the interior of the quadrilateral formed by $ele(v)$ and $ele(sib(v))$. Now, we are left with only one triangle, $ele(par(v))$. We can subdivide $ele(par(v))$ into five strictly convex quadrilaterals by adding five Steiner points to the interior of $ele(par(v))$. The total number of Steiner points inserted into the resulting quadrangulation by sub-step 5d is at most nine. This implies that our algorithm can convert a triangulation with t triangles into a strictly convex quadrangulation by using at most $t+7$ Steiner points, all of which are placed in the interior or on the boundary of the triangulation domain.

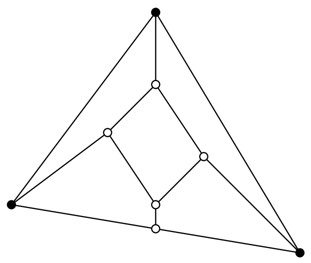

Fig. 33. Subdivision of a triangle into five strictly convex quadrilaterals using five Steiner points, one of which is placed on the boundary of the triangle.

5. Conclusions

We presented an algorithm for converting triangulations of polygonal regions, with or without polygonal holes, into strictly convex quadrangulations. Our algorithm has a runtime linear in the number of triangles of the input triangulation, offers better bounds than similar algorithms that also produce strictly convex quadrilateral meshes of bounded size [8], tends to preserve local mesh grading, and is simpler and likely faster than algorithms that produce better quality meshes — in terms of element shape, regularity and directionality control — at the expense of runtime [14,24,15]. We also introduced an extension of our algorithm for converting constrained triangulations of polygonal regions, with or without polygonal holes, into constrained, strictly convex quadrangulations, and we showed that these constrained quadrilateral meshes also have small bounded size.

We also provided examples of quadrilateral meshes generated by our algorithm and its extension to produce constrained quadrilateral meshes, and we evaluated the quality of these meshes with respect to a quadrilateral shape quality criterion defined in [18]. We used a post-processing technique, called angle-based smoothing [26], to improve the shape quality of the quadrilaterals of the meshes used in the examples. Our empirical results show that this technique effectively led to improvements in the shape quality of the quadrilaterals. We also compared the size of the quadrilateral meshes of our examples with the size of their triangular counterparts, and we observed that the sizes of the quadrilateral meshes is about 60% of their triangular counterparts. A comparison between the quality of the quadrilateral meshes generated by our algorithm and the quality of their triangular counterparts, with respect to a particular FE-based application, is given in [17].

Although the combination of our algorithm with the angle-based smoothing can produce quadrilateral meshes with "well-shaped" quadrilaterals, it may be a very time-consuming combination in practice, as the angle-based smoothing is very slow and so are most post-processing techniques to effectively improve mesh element shape quality. Thus, it is natural to try to improve the shape quality of the quadrilaterals produced by our algorithm by modifying the algorithm itself. However, the problem of generating strictly convex quadrilateral meshes with small bounded size and high-quality quadrilaterals is a very difficult one [10].

Some algorithms for generating quadrilateral meshes with high-quality quadrilaterals and no rigorous bounds on mesh size, such as paving [13], may produce many more than $\mathcal{O}(n)$ quadrilaterals, where n is the number of vertices of the input polygonal region, but they are likely to produce better shaped quadrilaterals than algorithms that reconcile provably good bounds on element shape quality and mesh size [10]. Nevertheless, similar ideas to the ones in [10], such as bounding the largest angle of the quadrilaterals, can be used to improve the shape quality of the quadrilaterals of the small quadrangulations produced by our algorithm (see Section 3.3) without increasing its linear bound on mesh size. We intend to focus on this problem as future work. We also intend to investigate an extension of our algorithm to produce hexahedral meshes from a set of quadrilateral meshes of planar contours. This investigation has been motivated by applications in medical imaging, namely, the generation of three-dimensional meshes of volumetric images from planar image sections [29].

Acknowledgments

We thank the anonymous reviewers for their corrections and helpful suggestions. The second author also thank CNPq and Universidade Federal de Mato Grosso do Sul, Brazil, for partially supporting his research during the development of this work.

References

1. J. Shewchuk. What is a good linear element? interpolation, conditioning, and quality measures. In *Proceedings of the 11th International Meshing Roundtable*, pages 115–126, Ithaca, New York, USA, September 2002.
2. P. Frey and P.-L. George. *Mesh Generation*. Hermes Science Publishing, 2000.
3. M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In F.K. Hwang and D.-Z. Du, editors, *Computing in Euclidean Geometry*. World Scientific, 1992.
4. P. Chew. Constrained delaunay triangulations. *Algorithmica*, 4:97–108, 1989.
5. J. Ruppert. A delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, 1989.
6. J. Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *First Workshop on Applied Computational Geometry*, pages 124–133, Philadelphia, PA, USA, May 1996.
7. J. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1–3):21–74, 2002.
8. H. Everett, W. Lenhart, M. Overmars, T. Shermer, and J. Urrutia. Strictly convex quadrilateralizations of polygons. In *Proceedings of the 4th Canadian Conference on Computational Geometry*, pages 77–82, 1992.
9. Matthias Müller-Hannemann and Karsten Weihe. Minimum strictly convex quadrangulations of convex polygons. In *Proceedings of the 13th ACM Symposium on Computational Geometry*, pages 193–202, 1997.
10. M. Bern and D. Eppstein. Quadrilateral meshing by circle packing. In *Proceedings of the 6th International Meshing Roundtable*, pages 7–19, Park City, Utah, USA, October 1997.
11. S. Ramaswami, P. Ramos, and G. Toussaint. Converting triangulations to quadrangulations. *Computational Geometry: Theory and Applications*, 9:257–276, 1998.
12. D. Bremner, F. Hurtado, S. Ramaswami, and V. Sacristán. Small convex quadrangulations of point sets. In *Proceedings of the International Symposium on Algorithms and Computation (ISAAC)*, volume 2223 of *Lecture Notes in Computer Science*, pages 623–635. Spriger-Verlag, 2001.
13. T. Blacker. Paving: A new approach to automated quadrilateral mesh generation. *International Journal For Numerical Methods in Engineering*, 32(4):811–847, 1991.
14. K. Shimada, J-H. Liao, and T. Itoh. Quadrilateral meshing with directionality control through the packing of square cells. In *Proceedings of the 7th International Meshing Roundtable*, pages 61–76, Dearborn, Michigan, USA, October 1998.
15. N. Viswanath, K. Shimada, and T. Itoh. Quadrilateral meshing with anisotropy and directionality control via close packing of rectangular cells. In *Proceedings of the 9th International Meshing Roundtable*, pages 217–225, New Orleans, Louisiana, USA, October 2000.
16. A. Malanthara and W. Gerstle. Comparative study of unstructured meshes made of triangles and quadrilaterals. In *Proceedings of the 6th International Meshing Roundtable*, pages 437–447, Park City, Utah, USA, October 1997.
17. S. Ramaswami, M. Siqueira, T. Sundaram, J. Gallier, and J. Gee. A new algorithm for generating quadrilateral meshes and its application to fe-based image registration. In *Proceedings of the 12th International Meshing Roundtable*, pages 159–170, Santa Fe, New Mexico, USA, September 2003.

18. B. Joe. Quadrilateral mesh generation in polygonal regions. *Computer-Aided Design*, 27(3):209–222, 1995.
19. L. Velho and J. Gomes. Variable resolution 4-k meshes: Concepts and applications. *Computer Graphics Forum*, 19(4):195–214, 2000.
20. G. Carey. *Computational grids: Generation, Adaptation, and Solution Strategies.* Taylor & Francis, 1997.
21. A. Lubiw. Decomposing polygonal regions into convex quadrilaterals. In *Proceedings of the 1st ACM Symposium on Computational Geometry*, pages 97–106, 1985.
22. S.J. Owen. A survey of unstructured mesh generation technology. In *Proceedings of the 7th International Meshing Roundtable*, pages 239–267, Dearborn, MI, USA, October 1998.
23. B.P. Johnston, J.M. Sullivan, and A. Kwasnik. Automatic conversion of triangular finite meshes to quadrilateral meshes. *International Journal of Numerical Methods in Engineering*, 31(1):67–84, 1991.
24. S. Owen, M. Staten, S. Cannan, and S. Saigal. Q-morph: An indirect approach to advancing front quad meshing. *International Journal for Numerical Methods in Engineering*, 9(44):1317–1340, 1999.
25. A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the Design of CGAL, the Computational Geometry Algorithms Library. *Software - Practice and Experience*, 30:1167–1202, 2000.
26. T. Zhou and K. Shimada. An Angle-Based Approach to Two-Dimensional Mesh Smoothing. In *Proceedings of the 9th International Meshing Roundtable*, pages 373–384, New Orleans, Louisiana, USA, October 2000.
27. S. Canann, S. Muthukrishnan, and R. Phillips. Topological Improvement Procedures for Quadrilateral Finite Element Meshes. *Engineering with Computers*, 14:168–177, 1998.
28. P. Pébay. Planar Quadrangle Quality Measures: Is There Really a Choice? In *Proceedings of the 11th International Meshing Roundtable*, pages 53–62, Ithaca, New York, USA, September 2002.
29. G. Nonato, R. Minghim, M. Oliveira, and G.Tavares. A Novel Approach for Delauney 3D Reconstruction with a Comparative Analysis in the Light of Applications. *Computer Graphics Forum*, 20(2):1–14, 2001.