An Experimental Comparison of Algorithms for Converting Triangulations of Closed Surfaces into Quadrangulations

Thiago Lemos[1], Suneeta Ramaswami[2], and Marcelo Siqueira[3]

[1]Programa de Pós-Graduação em Informática, Universidade Federal do Paraná, Curitiba, Brazil
[2]Department of Computer Science, Rutgers University, Camden, NJ, USA
[3]Departamento de Matemática, Universidade Federal do Rio Grande do Norte, Natal, Brazil

Abstract

We perform an experimental comparison of four algorithms for converting a given triangulation $\mathcal{T}$ of an orientable, connected, boundaryless, and compact surface $\mathcal{S}$ in $\mathbb{E}^d$ into a quadrangulation $\mathcal{Q}$ of $\mathcal{S}$. All algorithms compute $\mathcal{Q}$ by first pairing edge-adjacent triangles of $\mathcal{T}$ (of another triangulation of $\mathcal{S}$ with the same vertex set) so that every triangle belongs to exactly one pair. Such a perfect pairing is guaranteed for triangulations of boundaryless and compact surfaces (also known as closed surfaces). The common edge of each pair of matched triangles is then removed to give rise to a quadrilateral of $\mathcal{Q}$. We implemented two greedy algorithms, a graph matching algorithm, and a new algorithm presented in this paper, which uses edge contractions and vertex splittings to compute a quadrangulation of the surface in two steps by combining pairings with edge flips. If $\mathcal{T}$ has positive genus g, the new algorithm takes $\mathcal{O}(gn_f + g^2)$ amortized time to compute a quadrangulation with the same vertex set, where n_f is the number of triangles of the triangulation; otherwise (i.e., if the genus is zero), it takes $\mathcal{O}(n_f)$ amortized time. We verify that if n_f is large and $g \ll n_f$ (which is typically the case in several applications), then our new algorithm performs better than the other three. However, if n_f is not large enough or g is large (relative to n_f), then a fast and simple greedy algorithm, when combined with an effective heuristic to pair up edge-adjacent triangles, is the best option among the four algorithms.

Keywords: *triangulation, quadrangulation, graph matching, edge contraction, irreducible triangulation*.

1 Introduction

Polygonal meshes have become the *de facto* standard for the representation of surfaces with highly complex geometry and arbitrary genus in computer graphics and geometry processing applications. This is mainly due to (1) their geometric simplicity, (2) the development of efficient algorithms for displaying, editing, smoothing, simplifying, remeshing, parametrizing, and compressing them, (3) the advent, progress, and widespread use of 3D sensing technologies, and (4) the fact that most techniques for manipulating traditional industry standard surface representations, such as NURBS, are currently available for polygonal meshes [1]. The idea of representing a surface by a polygonal mesh can be seen as an application of the divide-and-conquer paradigm: a surface is represented as an assembly of a finite collection (i.e., the *mesh*) of openly-disjoint simpler shapes, which locally approximates the geometry of the surface. The most common shapes used in polygonal meshes of surfaces are triangles and quadrilaterals, along with their vertices and edges. A polygonal mesh made entirely of triangles (resp. quadrilaterals) is called a *triangle* (resp. *quad*) *mesh*.

On the one hand, a triangle mesh is arguably the most popular type of mesh representation for surfaces. Furthermore, provably good algorithms for generating triangle meshes from surfaces given by parametric or implicit functions [2, 3, 4] and point clouds [5] are available in the literature, and have also been implemented

in public and freely-distributed software libraries. On the other hand, quad meshes are better suited than their triangle counterparts for character animation [6], spline-based surface modeling [7], texture mapping [8], mesh compression [9], and some specific finite element analyses [10], to name a few applications. However, the problem of generating a quad mesh from a given surface, which meets the needs of these applications, seems to be intrinsically harder than its triangle mesh counterpart. This difficulty has led several researchers to use an *indirect* approach, which consists of generating a quad mesh from a given triangle mesh of the surface.

There are two broad classes of indirect algorithms [11]: *conversion* algorithms and *remeshing* algorithms. The former class consists of algorithms that convert a triangle mesh into a quad mesh by pairing up adjacent triangles, and possibly flipping edges and/or adding a few vertices to the resulting quad mesh (also known as Steiner points). The latter class consists of algorithms that re-sample the triangle mesh, and connect the new vertices to form a quad mesh. This re-sampling is usually done by parametrizing the given triangle mesh over a base domain, which is then regularly sampled to produce the parameter values of the new vertices.

Remeshing algorithms are usually accompanied by an optimization procedure, which enforces degrees of most vertices to be 4 and edges to be aligned with the principal curvature directions of the surface. These features are highly desirable in most applications requiring a quad mesh of a given surface. However, the optimization procedure is typically very time-consuming and does not offer provable bounds on the values of the optimized mesh features. In contrast, conversion algorithms are generally much faster, but tend to generate unstructured quad meshes, i.e., meshes in which most vertices do not have degree 4 and whose edges are not aligned with the principal curvature directions of the surface. These meshes are not always suitable in applications. Nevertheless, conversion algorithms have been successfully used as a fast preprocessing stage for generating high-quality quad meshes of mechanical parts [12], and as a quick way of pairing up triangles in a triangle mesh for the purposes of fast rendering of subdivision surfaces [13], simplification of quad meshes [14], and stripification [15], traversal [16] and compact representation of triangle meshes [17].

Here, we restrict our attention to conversion algorithms and their application to triangle meshes of *closed* surfaces. (There are several known conversion algorithms for two-dimensional geometric domains, e.g. [18, 19].) In general, conversion algorithms produce a quad mesh from a given triangle mesh in two stages. The first stage, called *pairing*, pairs up adjacent triangles so that each triangle belongs to at most one pair. If every triangle is paired up with another triangle, then the resulting pairing is said to be *perfect*. For the class of triangle meshes we deal with in this paper (i.e., triangle meshes of closed surfaces), a perfect pairing always exists (see Section 2). Pairing is the key stage of conversion algorithms. The second stage obtains a quadrilateral mesh from the perfect pairing by removing the common edge of every pair of matched triangles.

The simplest conversion algorithms compute a perfect pairing in two steps [13, 20, 17]. First, a greedy approach computes a not necessarily perfect pairing, which takes linear time in n_f, where n_f is the number of triangles of the triangle mesh. If the initial pairing is not perfect, then the second step consists of a procedure for completing the pairing. To the best of our knowledge, the procedures available in the literature can only pair up two unpaired triangles at a time, and each execution takes $\mathcal{O}(n_f)$ time [14] or $\mathcal{O}(n_f)$ amortized time [21, 22]. Thus, the (amortized) time for pairing up k unpaired triangles is in $\mathcal{O}(kn_f)$, and the total (amortized) time taken by the pairing stage is in $\mathcal{O}(\max\{1, k\} \cdot n_f)$. An alternative to the two-step, greedy algorithms is to cast the pairing problem as a graph matching one, as a perfect pairing of the triangles in a triangle mesh corresponds to a perfect matching on the *dual graph* of the triangle mesh and vice-versa. The dual graph of a triangle mesh of a closed surface is 3-regular and bridgeless. For such a graph, a perfect matching always exists and can be found in $\mathcal{O}(n \lg^2 n)$ time, where n is the number of graph nodes [23]. If the triangle mesh has genus 0, then a perfect matching can be found in $\mathcal{O}(n)$ amortized time [24].

Our Contributions. We present and analyze the results of an experimental comparison of four conversion algorithms: two greedy algorithms [13, 20], the graph matching algorithm in [23], and a new algorithm presented in this paper, which uses edge contractions and vertex splittings to compute a quadrangulation of the surface in two stages by combining pairings with edge flips. If the input triangle mesh has positive genus g, the new algorithm takes $\mathcal{O}(gn_f + g^2)$ amortized time to compute a quadrilateral mesh with the same vertex set, where n_f is the number of triangles of the mesh; otherwise (i.e., if $g = 0$), it takes $\mathcal{O}(n_f)$ amortized time. All four algorithms have been implemented by the authors. We evaluated their performance against triangles meshes typically found in graphics and geometry processing applications, as well as triangles meshes designed by the authors, which allow us to quantify the influence of parameters n_f and g.

The emphasis of this work is on the run-time efficiency of producing a quad mesh, rather than the evaluation of the resulting quad mesh by specific quality criteria. Indeed, when it comes to pairing algorithms, quadrilateral shape quality is highly dependent on the triangle mesh [11]. Moreover, a quad mesh suitable for applications depend on several quality criteria. As a result, pairing algorithms are better off serving the purpose of rapidly producing a quad mesh that can be used as input to a powerful optimization procedure that improves the mesh with respect to several quality criteria at the same time [14, 12], or by applications in which mesh quality is not a critical factor [25, 13, 15, 17].

2 Notation, terminology, and background

Let $\mathbb{E}^d$ denote the d-dimensional Euclidean (affine) space over $\mathbb{R}$, and let $\mathbb{R}^d$ denote the associated vector space of $\mathbb{E}^d$. A subset of $\mathbb{E}^d$ that is homeomorphic to the open unit interval, $\mathbb{B}^1 = (0, 1) \subset \mathbb{E}$, is called an *open arc*. A subset of $\mathbb{E}^d$ that is homeomorphic to the open circle, $\mathbb{B}^2 = \{(x, y) \in \mathbb{E}^2 \mid x^2 + y^2 < 1\}$, of unit radius is called an *open disk*. Recall that a subset $\mathcal{S} \subset \mathbb{E}^d$ is called a *topological surface*, or *surface* for short, if each point p in $\mathcal{S}$ has an open neighborhood that is an open disk. According to this definition, a surface is a "closed" object in the sense that it has an empty boundary. Throughout this paper, we restrict our attention to the class consisting of all *oriented, connected, and compact surfaces in* $\mathbb{E}^d$, and for short, we assume that the term "surface" designates a member of such a class (unless we explicitly state otherwise).

The terms 'triangle mesh' and 'quadrilateral mesh' are synonyms for triangulation and quadrangulation, respectively. The latter terms are standard in topological graph theory and algebraic topology [26, 27], and we shall use them from now on. A triangulation (resp. quadrangulation) of a surface is a way of cutting up the surface into triangular (resp. quadrilateral) regions such that these regions are images of triangles (resp. quadrilaterals) in the plane, and the vertices and edges of these planar triangles (resp. quadrilaterals) form a graph with certain properties. In what follows, we formalize these ideas. To that end, we rely on the notions of subdivision of a surface, as nicely stated by Guibas and Stolfi [28], and of a graph embedded on a surface.

Definition 2.1. *A* subdivision *of a surface $\mathcal{S}$ is a partition, $\mathcal{P}$, of $\mathcal{S}$ into three finite collections of disjoint subsets:* vertices*,* edges*, and* faces*, which are denoted by $V_{\mathcal{P}}(\mathcal{S})$, $E_{\mathcal{P}}(\mathcal{S})$, and $F_{\mathcal{P}}(\mathcal{S})$, respectively, such that* (i) *every* vertex *is a point,* (ii) *every* edge *is an open arc,* (iii) *every* face *is an open disk, and* (iv) *the boundary of every face is a closed path of edges and vertices.*

One can show that every vertex and edge in $\mathcal{P}$ must be incident to some face of $\mathcal{P}$ [28]. Moreover, every edge in $\mathcal{P}$ is entirely contained in the boundary of some face of $\mathcal{P}$, every vertex is incident on an edge, and every edge of $\mathcal{P}$ is incident on two (not necessarily distinct) vertices of $\mathcal{P}$. These vertices are the *endpoints* of the edge. If they are the same, the edge is a *loop*. From the collection of all vertices and edges of $\mathcal{P}$, we can define an undirected graph G, and a one-to-one function, $\imath : G \to \mathcal{S}$. Specifically, let $G = (V, E)$ be a graph such that $V = \{v_1, \ldots, v_h\}$ and $E = \{e_1, \ldots, e_l\}$, where h and l are the number of vertices and edges of $\mathcal{P}$. Define a one-to-one mapping, $\imath : G \to \mathcal{S}$, such that each $v_j \in V$ and each $e_k \in E$ is

associated with a distinct vertex and a distinct edge of $V_{\mathcal{P}}(\mathcal{S})$ and $E_{\mathcal{P}}(\mathcal{S})$, respectively, for $j = 1, \ldots, h$ and $k = 1, \ldots, l$. Furthermore, if v and u are the two nodes in V incident on edge e in E, then $\imath(v)$ and $\imath(u)$ are the two vertices of $V_{\mathcal{P}}(\mathcal{S})$ incident on $\imath(e)$ in $E_{\mathcal{P}}(\mathcal{S})$. Note that $\imath(G)^c = \mathcal{S} - \imath(G)$ are the faces of $\mathcal{P}$. We say that G is the *graph of* $\mathcal{P}$, and $\imath$ is the *embedding* of G on $\mathcal{S}$. Graph G defines the combinatorial structure of $\mathcal{P}$, while $\imath$ is the "geometric realization" of G on $\mathcal{S}$. Since $\mathcal{P}$ is fully described by G and $\imath$, it is customary to call the pair, $(G, \imath)$, the subdivision itself. Triangulations and quadrangulations are specialized subdivisions.

Definition 2.2. *A* triangulation *(resp.* quadrangulation*) of a surface $\mathcal{S}$ is a subdivision $(G, \imath)$ such that each face of $\imath(G)^c$ is bounded by exactly* 3 *(resp.* 4*) distinct vertices and* 3 *(resp.* 4*) distinct edges from $\imath(G)$. Furthermore, any two edges of a triangulation (resp. quadrangulation) have at most one common endpoint.*

From Definition 2.2, we have that the graph of a triangulation is simple (i.e., it has no loops or parallel edges). The *tri-quad conversion problem* can be formally stated as follows: given a triangulation $(G, \imath)$ of a surface $\mathcal{S}$, find a quadrangulation, $(H, \jmath)$, of $\mathcal{S}$ such that the vertex set of H and G are the same. The following results state a few properties of triangulations and quadrangulations, which are exploited by conversion algorithms:

Lemma 2.1 ([28])**.** *If G is the graph of a surface triangulation, then G is connected.*

Lemma 2.2. *Every edge of a surface triangulation $(G, \imath)$ is incident on exactly two faces of $(G, \imath)$.*

Proof. Let e be any edge of (G, i). Aiming at a contradiction, assume that there are at least three faces, τ_1, τ_2, and τ_3, incident on e. Let p be any point of e. Since every face is an open disk, and since each edge incident on a face is entirely contained in the face boundary, there exists a positive number r_j, for each $j = 1, 2, 3$, such that $\overline{\tau}_j \cap B(p, r_j)$ is homeomorphic to the half-disk, D, where $\overline{\tau}_j$ is the closure of τ_j, $B(p, r_j)$ is the open ball of radius r_j centered at p, and $D = \{(x, y) \in \mathbb{E}^2 \mid x \geq 0, x^2 + y^2 < 1\}$. Furthermore, since e cannot contain a vertex, if each r_j is chosen small enough, then we also have that every point in $B(p, r_j)$ which is also a point on the boundary of τ belongs to e, i.e., $\partial(\tau_j) \cap B(p, r_j) = e \cap B(p, r_j)$, where $\partial(\tau_j)$ is the boundary of τ_j. By definition of subdivision, we know that $\tau_j \cap \tau_k = \emptyset$, for any two $j, k \in \{1, 2, 3\}$, with $j \neq k$. So, if we take $r = \min\{r_1, r_2, r_3\}$, then the intersection of the three half-disks, $\overline{\tau_1} \cap B(p, r)$, $\overline{\tau_2} \cap B(p, r)$, and $\overline{\tau_3} \cap B(p, r)$, is equal to $e \cap B(p, r_j)$, while their union is not homeomorphic to an open disk (no matter how small r is). So, there cannot be any neighborhood of $\mathcal{S}$ around p that is homeomorphic to a disk, which contradicts the fact that $\mathcal{S}$ is a surface. Thus, edge e must be incident on either one or two faces. But, from Definition 2.2, the vertices and edges in the boundary of each face of a triangulation are distinct. Moreover, since the closure $\bar{\tau}$ of a single face τ, which is bounded by three distinct vertices and edges, cannot entirely cover a boundaryless, compact surface in $\mathbb{E}^3$, the complement of $\bar{\tau}$ with respect to $\mathcal{S}$ must contain at least one more face. Consequently, every edge of τ is incident on two faces, and so must be e. □

If G is the graph of a subdivision $(G, \imath)$ of a surface $\mathcal{S}$, then we can define a dual graph $G^\star$ of G. In particular, we say that $G^\star$ is a *dual graph* of G if and only if there exists a one-to-one correspondence between the vertices of $G^\star$ and the faces of $(G, \imath)$, and two vertices in $G^\star$ are connected by an edge $e^\star$ if and only if their corresponding faces in $(G, \imath)$ are incident on the same edge e in $(G, \imath)$. We say that $e^\star$ is the *dual (edge)* of e in $G^\star$. For any face τ in $(G, \imath)$, we denote its corresponding node in $G^\star$ by $\tau^\star$. All dual graphs of G are unique up to isomorphisms, so we can refer to $G^\star$ as *the* dual graph of G. If $(G, \imath)$ is a triangulation, then every face τ of $(G, \imath)$ has three distinct edges, which implies that $G^\star$ is 3-regular. Furthermore, since the dual of a bridge $e^\star$ in $G^\star$ is a loop e in G, and since G is simple, $G^\star$ must be bridgeless. We can convert the graph G of a triangulation $(G, \imath)$ of a surface $\mathcal{S}$ into the graph of a quadrangulation of $\mathcal{S}$ by *computing a perfect matching M on a dual graph, $G^\star$*. This is the underlying idea of graph-based algorithms for the tri-quad conversion problem. A classic result from graph theory ensures that matching M always exists:

Theorem 2.1 (Petersen's Theorem, [29]). *Every bridgeless, 3-regular graph admits a perfect matching.*

Let M be a perfect matching on $G^\star$. Recall from the definition of perfect matching that M is a subset of the set of edges of $G^\star$ such that (i) every node $\tau^\star$ is incident on an edge of $G^\star$ in M, and (ii) no two edges in M share a common node in $G^\star$. Given such a matching M, we define a graph $H = (W, F)$ from $G = (V, E)$ by letting $W = V$ and $F = \{e \in E \mid e^\star \notin M\}$. By defining an embedding $\jmath : H \to \mathcal{S}$ such that $\jmath(v) = \imath(v)$ and $\jmath(e) = \imath(e)$, for each $v \in W$ and each $e \in F$, we get a quadrangulation, $(H, \jmath)$, of $\mathcal{S}$. Recall also that the genus g of a surface $\mathcal{S}$ is the maximum number of disjoint, closed, and simple curves, $\alpha_1, \ldots, \alpha_g$, on S such that the set $\mathcal{S} - (\alpha_1 \cup \cdots \cup \alpha_g)$ is connected. Up to homeomorphisms, there is only one surface of genus g [27]. For any subdivision, $(G, \imath)$, of $\mathcal{S}$, we have $n_v - n_e + n_f = 2 \cdot (1 - g)$, where n_v, n_e, and n_f are the number of vertices, edges, and faces of $(G, \imath)$ [26]. If $(G, \imath)$ is a triangulation of $\mathcal{S}$, then $3 \cdot n_f = 2 \cdot n_e$, which implies that $2 \cdot n_v - n_f = 4(1 - g)$ and $3 \cdot n_v - n_e = 6(1 - g)$, and thus $n_e \in \Theta(n_v + g)$ and $n_f \in \Theta(n_v + g)$. Here, we assume that $\mathcal{S}$ is such that $g \in \mathcal{O}(n_v)$, which implies that $n_e \in \Theta(n_v)$ and $n_f \in \Theta(n_v)$. Conversion algorithms require the graph of the input triangulation only (see Section 3). So, for the sake of conciseness, we shall ignore the embedding and simply refer to the input triangulation by $\mathcal{T}$.

3 Algorithms

This section briefly describes four conversion algorithms whose performances we compare in Section 4. All four algorithms take the graph G of a triangulation $\mathcal{T}$ of a surface $\mathcal{S}$ as input, and output the graph H of a quadrangulation $\mathcal{Q}$ of $\mathcal{S}$ such that G and H share the same node set (i.e., embeddings of $\mathcal{Q}$ and $\mathcal{T}$ are ignored).

3.1 Greedy algorithms

Greedy algorithms iteratively select one edge at a time from the input triangulation $\mathcal{T}$ and pair up the two triangles incident on it. To ensure that each triangle belongs to at most one pair, every edge of $\mathcal{T}$ is marked as either *free* or *locked*. Initially, all edges are marked *free*. Whenever an edge is selected, the algorithm checks if it is *free*. If so, its two incident triangles are paired up with each other, and their remaining edges are marked as *locked*. If the selected edge is locked, it is discarded and a new edge is selected. This iterative process ends when all edges of $\mathcal{T}$ have been selected and tested. If $\mathcal{T}$ has n_e edges, the algorithm takes $\Theta(n_e)$ time to produce a pairing. Since $n_e \in \Theta(n_v)$ and $n_f \in \Theta(n_v)$, computing the pairing takes $\Theta(n_f)$ time, where n_v and n_f are the number of vertices and triangles of $\mathcal{T}$, respectively.

As we pointed out in Section 1, the above greedy algorithm may not produce a perfect pairing. In fact, in our experiments, it paired up about 87% of the triangles in all test cases. To the best of our knowledge, the algorithm was first mentioned by Puli and Segal [13], and then by Velho [20]. However, in both cases, the selection of edges was combined with a heuristic to increase either the number of paired triangles [13] or the shape quality of the resulting quadrilaterals [20]. The heuristic devised by Puli and Segal was extremely effective in our experiments, pairing up more than 99% of the triangles in all test cases. Moreover, it does not increase the time complexity of the pairing step, which remains $\Theta(n_f)$. In contrast, the heuristic proposed by Velho increases the worst-case time complexity to $\Theta(n_f \lg n_f)$, but does not increase the number of paired triangles. Hence, we only report the results of two greedy algorithms: (1) the greedy algorithm that uses no heuristic to select edges, and (2) the greedy algorithm developed by Puli and Segal [13].

The idea behind the heuristic of Puli and Segal is to assign one of four priorities to each triangle of $\mathcal{T}$. A triangle with priority i is inserted into (priority) set i, for each $i = 0, 1, 2, 3$. Triangles incident on zero or exactly one free edge are given the highest priority, 3, while triangles incident on three free edges are given the lowest priority, 0. Of the remaining triangles (i.e., triangles incident on exactly two free edges),

the ones sharing both free edges with triangles of priority 0 are given priority 1, while the others are given priority 2. Initially, all triangles are assigned priority 0 and inserted into priority set 0. To pair up triangles, the algorithm chooses any triangle from the highest priority set, and then selects a highest priority neighbor incident on a free edge. These two triangles are paired and removed from their sets, while their neighbors update their priorities and relocate to the sets corresponding to their new priorities.

If the pairing produced by the greedy algorithm is not perfect, we augment the implied matching (in the dual graph $G^\star$) along an augmenting path until it becomes perfect. In particular, a procedure to compute one augmenting path is invoked $\frac{k}{2}$ times, where k is the number of triangles left unpaired by the greedy approach. Exactly two unmatched nodes of $G^\star$ are matched every time the procedure is invoked. We implemented the procedure for finding augmenting paths described by Tarjan in [21]. Using a special case of disjoint sets from [22], Tarjan's procedure can find one augmenting path in $\mathcal{O}(n_f)$ amortized time. Thus, the time complexity of the pairing stage of the greedy algorithms that we implemented is in $\mathcal{O}(k\, n_f)$ amortized time.

3.2 A graph-based algorithm

Orrin Frink gave a constructive proof for Petersen's Theorem (see Theorem 2.1) in 1926 [30]. His proof has some minor flaws, but a correct version of it can be found in the first book on graph theory ever written [31]. Using the key idea of "graph reduction" from Frink's proof (see Appendix A) , Biedl, Bose, Demaine, and Lubiw proposed an algorithm for computing a perfect matching on a given 3-regular and bridgeless graph in $\mathcal{O}(n \lg^4 n)$ [24], where n is the number of graph nodes. More recently, Diks and Stanczyk modified the algorithm in [24], and lowered the upper bound to find a perfect matching to $\mathcal{O}(n \lg^2 n)$ [23], which is, to the best of our knowledge, the best known upper bound for computing a perfect matching on a 3-regular and bridgeless graph. It is worth mentioning that a perfect matching on a 3-regular and bridgeless graph can also be computed by the most efficient algorithm for computing maximum cardinality matchings on nonbipartite graphs [32], whose time upper bound is $\mathcal{O}(m\sqrt{n})$, where m is the number of edges of the graph. Since $m = \Theta(n)$ for 3-regular graphs, the upper bound of the algorithm in [32] is $\mathcal{O}(n^{1.5})$ if we restrict the input to 3-regular and bridgeless graphs. We implemented Diks and Stanczyk's algorithm and applied it to the dual graph $G^\star$ of the graph G of the input triangulation $\mathcal{T}$ in order to obtain a perfect pairing of the triangles of $\mathcal{T}$, and since we have $n_f = n$, we obtain a quadrangulation of $\mathcal{S}$ in $\mathcal{O}(n_f \lg^2 n_f)$ amortized time.

3.3 A new algorithm using irreducible triangulations and edge flips

We propose a new algorithm to compute a quadrangulation of surface $\mathcal{S}$ in $\mathcal{O}(g^2 + g\, n)$ time, where g is the genus of $\mathcal{S}$. The quadrangulation is also computed in two stages, but the pairing stage consists of three steps. First, the algorithm computes an *irreducible triangulation* $\mathcal{T}'$ from triangulation $\mathcal{T}$ of $\mathcal{S}$ by applying a sequence of *topology-preserving edge contractions* to the graph G of $\mathcal{T}$. Second, all edge contractions are expanded in reverse order and some *edge flips* may take place in between two expansions. The result is another triangulation, $\mathcal{K}$, of the same surface $\mathcal{S}$ and with the same set of vertices as $\mathcal{T}$. Due to the edge flips, the set of edges and the set of triangles of $\mathcal{K}$ are not the same as the ones of $\mathcal{T}$, but $\mathcal{K}$ has the same number of edges and triangles as $\mathcal{T}$. The second step also builds a matching on the dual graph $J^\star$ of the graph J of $\mathcal{K}$ at the same time $\mathcal{K}$ is built. This matching is not perfect. So, a third step is necessary to augment the matching computed in the second in order to make it perfect. To that end, we also rely on the procedure from [21].

Contracting an edge e of a triangulation consists of removing e and all edges meeting e from the triangulation, and then replacing them by a vertex w which is connected to every remaining vertex that was connected to a vertex of e (see Figure 3.1). The inverse of the edge contraction operation is called *vertex splitting*. If the result of contracting e is a triangulation of the same surface, e is said to be *contractible* and the contraction

is said to be *topology-preserving*; otherwise, e is said to be *non-contractible*. A triangulation is said to be *irreducible* if and only if every edge is non-contractible. There are only finitely many irreducible triangulations for every surface [33]. The sphere has only one irreducible triangulation, which is isomorphic to the boundary of a tetrahedron (and thus it has 4 vertices), while the torus has 21 irreducible triangulations (up to isomorphisms) [34]. The following result regarding the size of irreducible triangulations was proved by Joret and Wood [35]:

Theorem 3.1. *[35] If a surface has positive genus g, the number n'_v of vertices of any irreducible triangulation of the surface is such that $n'_v \leq 26 \cdot g - 4$.*

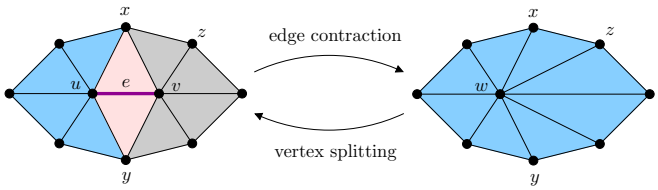

Figure 3.1: The edge contraction operation and its inverse.

Two of the authors of this paper proposed a fast algorithm for computing an irreducible triangulation $\mathcal{T}'$ of a surface $\mathcal{S}$ from any given triangulation $\mathcal{T}$ of the same surface [36]. If the genus g of $\mathcal{S}$ is zero, then the algorithm produces $\mathcal{T}'$ in linear time in n_f. Otherwise, $\mathcal{T}'$ is obtained in $\mathcal{O}(g\, n_f + g^2)$ time. Triangulation $\mathcal{T}'$ is obtained by a sequence of topology-preserving edge contractions, which guarantees that $\mathcal{T}'$ is a triangulation of $\mathcal{S}$. The first step of our proposed algorithm, which we call the *contraction* step, is carried out by the algorithm described in [36]. The second step, which we call the *expansion* step, reverses all edge contractions applied by the first phase using the vertex splitting operation. In between two vertex splitting operations, one or two edges of the current triangulation may be flipped (see Section 3.3.1 for a detailed description of the second step). While reversing the contractions, the expansion step computes a matching on the dual graph of the current triangulation. The output is a triangulation $\mathcal{K}$ of $\mathcal{S}$, with n_f triangles as well, and a matching M on the dual graph $J^\star$ of the graph of $\mathcal{K}$. With respect to M, we have that $J^\star$ has at most 4 unmatched nodes if $\mathcal{S}$ is a genus-0 surface, and at most $56 \cdot g - 12$ unmatched nodes otherwise (which follows from the bound $n'_v \leq 26 \cdot g - 4$, and the relations $3n'_f = 2n'_e$ and $n'_v - n'_e + n'_f = 2(1-g)$, where n'_e and n'_f are the number of edges and triangles of $\mathcal{T}'$). Both $\mathcal{K}$ and M can be obtained in $\mathcal{O}(n_f)$ time.

Matching M is augmented to become a perfect matching on $J^\star$ in the third step. Since M has at most $\ell = \max\{4, 56 \cdot g - 12\}$ unmatched nodes, the augmenting path procedure in [21] is invoked at most $\ell/2$ times. Thus, the third step takes $\mathcal{O}(n_f)$ amortized time if g is zero, and $\mathcal{O}(g\, n_f)$ amortized time otherwise. As we see in Section 4, ℓ is less than 2% of the total number n_f of nodes of $J^\star$ for all test cases in which n_f is large and $g \ll n_f$ (which is typically the case for meshes used in graphics and engineering applications). Finally, a quadragulation $\mathcal{Q}$ of $\mathcal{S}$ is obtained by pairing up the dual triangles corresponding to matched nodes in the perfect matching on $J^\star$. Observe that $\mathcal{Q}$ has the same vertex set as $\mathcal{T}$, but its set of edges is not necessarily a subset of the set of edges of $\mathcal{T}$ (due to the edge flips carried out in the expansion step).

3.3.1 The expansion step of the new algorithm

In this section, we describe the second, i.e., expansion, step of the pairing stage our algorithm to construct a quadrilateral mesh for the input surface $\mathcal{S}$. Before we proceed, we first state a definition and an important lemma that our algorithm relies upon in the preceding step, i.e., the contraction step of the algorithm's pairing stage.

Definition 3.1. *Let $\mathcal{T}$ be a triangulation of a surface $\mathcal{S}$, and let v be a vertex of $\mathcal{T}$. If all edges of $\mathcal{T}$ incident on v are non-contractible, then v is said to be a* trapped *vertex. Otherwise, v is said to be a* loose *vertex.*

Lemma 3.1 (Lemma 7 in [37])**.** *Let $\mathcal{T}$ be a surface triangulation, v a trapped vertex of $\mathcal{T}$, and e a contractible edge of $\mathcal{T}$. If e is contracted in $\mathcal{T}$, then v remains a trapped vertex in the triangulation obtained by contracting e.*

The key idea behind the contraction step of our algorithm is to iteratively choose a vertex u from the current triangulation and then *process* u, which involves contracting (contractible) edges incident on u until no edge incident on u is contractible, i.e., until u becomes a trapped vertex. When u becomes trapped (or if u is already trapped), another vertex from the current triangulation is chosen and processed by the algorithm until all vertices are processed, at which point the algorithm ends. The output of the contraction step is an irreducible triangulation, $\mathcal{T}'$, of $\mathcal{S}$. A vertex u of $\mathcal{T}$ belongs to $\mathcal{T}'$ if and only if u has been processed by the contraction step. Furthermore, the m vertices of $\mathcal{T}'$ define a sequence, $u_1, u_2, \ldots, u_m$, such that u_i was processed before u_{i+1}, for every $i = 1, \ldots, m-1$. For any two vertices $u, v \in \mathcal{T}'$, if u was processed before v, then u was trapped when v was chosen to be processed in the contraction step. As we shall see soon, the following simple lemma plays a useful role in both the contraction and the expansion steps of our algorithm:

Proposition 3.1. *Let K be a surface triangulation and v a vertex of degree 3 in K. If K is (isomorphic to) $\mathcal{T}_4$ (the boundary of a tetrahedron), then no edge of K is contractible; else every edge of K incident on v is a contractible edge.*

To describe the expansion step, we label the vertices of $\mathcal{T}'$ in reverse order (with respect to the order in which they were processed by the contraction step): $u_m, u_{m-1}, \ldots, u_1$. Thus u_m is the first vertex processed by the contraction step, u_{m-1} is the second, and so on. The expansion step of the algorithm computes the graph of a triangulation, $\mathcal{K}$, of $\mathcal{S}$ by expanding edges from each u_i, with $1 \leq i \leq m$, in the reverse order of their contractions (applying the inverse operation — vertex split — of the edge contraction). As new (triangular) faces are created from the expansions, the algorithm pairs faces in a suitable way, possibly by flipping edges, in order to define the initial pairing of triangle. After all expansions are complete, we have a triangulation in which all faces are matched, except possibly for $\mathcal{O}(g)$ of them corresponding to the faces of the irreducible triangulation, $\mathcal{T}'$.

Denote by $n_1, n_2, \ldots, n_m$ the number of vertices of the input triangulation, $\mathcal{T}$, of $\mathcal{S}$ that get collapsed via edge contractions into the vertices $u_1, u_2, \ldots, u_m$, respectively, in the contraction step of our algorithm. Let

$$V_i = \{v_{in_i}, \ \ldots, \ v_{i2}, \ v_{i1}\}$$

be the set of vertices that are collapsed into u_i in the contraction step, with v_{ij} being the $(n_i - j + 1)$-th collapsed vertex, for all $j \in \{1, \ldots, n_i\}$. Note that $V_i \cap V_j = \emptyset$, for all $i, j \in \{1, \ldots, k\}$, with $i \neq j$, as no vertex is inserted into a triangulation in the contraction step. The expansion step first expands from u_1, then from u_2, and so on up to u_m. For each vertex u_i, it first expands v_{i1} from u_i, then v_{i2} from u_i, and so on until v_{in_i}, at which point we say that u_i has been expanded fully.

Let $\mathcal{T}_{ij}$ be the triangulation that results after vertex v_{ij} in V_i is expanded from u_i. In other words, $\mathcal{T}_{ij}$ is the triangulation just before v_{ij} is identified with u_i by an edge contraction in the contraction step. To expand v_{ij} from u_i, the algorithm performs a vertex split operation that reverses the contraction of $[u_i, v_{ij}]$ in the previous step. Recall from Section 3.3 that a vertex split causes two edges, $[u_i, x]$ and $[u_i, y]$, to *split*, resulting in two new faces, $[u_i, x, v_{ij}]$ and $[u_i, y, v_{ij}]$, as shown in Figure 3.2(a). Pairs of triangulation edges that may be split to give rise to new faces in the expansion step of the algorithm are said to be *split edges*. In what follows, we denote the triangulation obtained *after* vertex u_i has been expanded fully by $\mathcal{T}_i$, i.e., $\mathcal{T}_i = \mathcal{T}_{in_i}$.

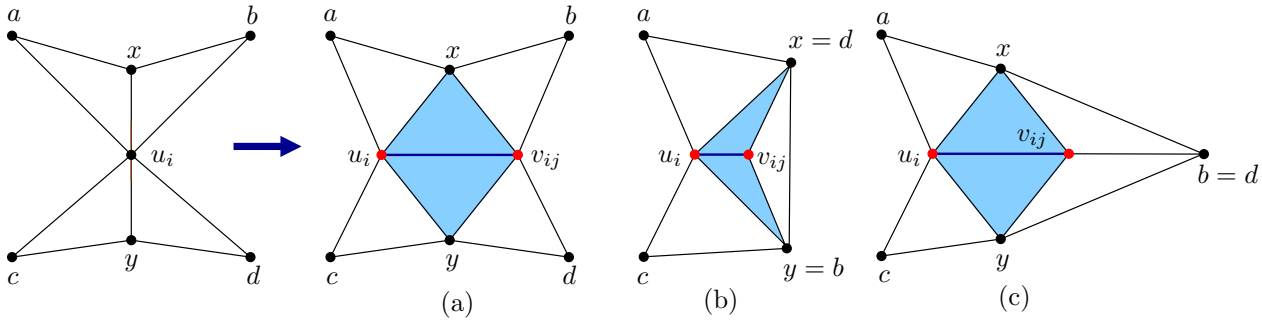

Figure 3.2: (a) Expanding v_{ij} from u_i. (b) Degree of v_{ij} is 3. (c) Degree of v_{ij} is 4.

Definition 3.2. *An edge $e \in \mathcal{T}_{ij}$ is said to be* safe in $\mathcal{T}_{ij}$ *if it is not a split edge for any subsequent expansion.*

The notion of a safe edge is useful because such edges are guaranteed not to cause any conflicts with future edge expansions in subsequent steps of the expansion phase. Observe also that the pair of triangles adjacent on a safe edge will not be separated by future expansions.

Note that every edge in $\mathcal{T}_{ij}$ that is *not* of the form $[u_l, z]$, where $l > i$ and z is a vertex in $\mathcal{T}_{ij}$, is safe in $\mathcal{T}_{ij}$. Indeed, if $l > i$ then no edge can be expanded from u_l after $\mathcal{T}_l$ is obtained. Furthermore, since z is a neighbor of u_l in $\mathcal{T}_{ij}$, either $z = u_k$, for some $k < i$, or $z \in V_k$, for some $k \leq i$. In either case, we get $z \neq u_i$. Note also that an edge safe in $\mathcal{T}_i$ is also safe in $\mathcal{T}_j$, for all $j > i$, and hence belongs to the final triangulation $\mathcal{T}_m$.

Definition 3.3. *Let $\mathcal{T}$ be a triangulation of a surface, $\mathcal{S}$, and let $\tau = [u, v, w]$ and $\sigma = [u, v, z]$ be the two faces of $\mathcal{T}$ incident on edge $e = [u, v]$ in $\mathcal{T}$. Then,* flipping *e replaces τ, σ, and e in $\mathcal{T}$ by $[u, w, z]$ and $[v, w, z]$ and $[w, z]$. Edge e is said to be* flippable *if and only if the flip of e results in a triangulation of $\mathcal{S}$.*

The following lemma states (without proof) a fact relating contractible and flippable edges, which is used by our algorithm:

Lemma 3.2. *Let $\mathcal{T}$ be a triangulation of a surface, $\mathcal{S}$, and let $\tau = [u, v, w]$ and $\sigma = [u, v, z]$ be the two faces of $\mathcal{T}$ incident on edge $e = [u, v]$ in $\mathcal{T}$. If e is a contractible edge and v is a vertex of degree greater than three, then edges $[v, w]$ and $[v, z]$ are both flippable, and thus the flip of either one yields a triangulation of $\mathcal{S}$.*

Definition 3.4. *An edge $e \in \mathcal{T}_{ij}$ is said to be* safely flippable *if it is safe in $\mathcal{T}_{ij}$* and *flippable.*

The following proposition is useful to prove an invariant (see Claim 3.1) of the algorithm's expansion step:

Proposition 3.2. *Suppose $v_{ij} \in V_i$ is incident on a vertex u_k, with $k \neq i$, and $[u_k, v_{ij}]$ is contractible. Then, $k < i$.*

Proof. We know that all vertices u_h, with $h > i$, are trapped in $\mathcal{T}_i$, i.e., every edge incident on u_h in $\mathcal{T}_i$ must be a non-contractible edge. Since, by hypothesis, edge $[u_k, v_{ij}]$ is a contractible edge in $\mathcal{T}_i$, it follows that $k < i$. □

After each expansion in the expansion step, the algorithm obtains a triangulation defined by the underlying vertices such that all *newly* created faces resulting from the expansions are matched. In particular, after expanding v_{ij} from u_i, the algorithm may flip some safely flippable edges in $\mathcal{T}_{ij}$ to construct a triangulation

$\mathcal{K}_{ij}$, finally resulting in a triangulation $\mathcal{K}_i$ after u_i has been expanded fully. All faces in $\mathcal{K}_i$ that result from expansions from u_i, and hence do not belong to $\mathcal{K}_{i-1}$, are said to be *expanded triangles* of $\mathcal{K}_i$. The algorithm matches the expanded triangles of $\mathcal{K}_i$ to incrementally construct an edge set, M, such that the set $M^\star = \{e^\star \mid e \in M\}$ is a matching of the dual graph, $G^\star_{\mathcal{K}_i}$, of $\mathcal{K}_i$. Every edge in M is an edge safe in $\mathcal{T}_i$ that has not been flipped. By using M, rather than $M^\star$, the algorithm does not have to explicitly build and handle $G^\star_{\mathcal{K}_i}$. By an abuse of terminology throughout this section, we refer to set M as a "matching" on $G^\star_{\mathcal{K}_i}$. The expansion step of the algorithm is specified in Algorithm 3.1 with two input parameters, $\mathcal{T}'$ and S, where $\mathcal{T}'$ is the irreducible triangulation and S is the stack that maintains edge contraction information (both are returned by the contraction step). We will show that Algorithm 3.1 maintains the following invariant:

Claim 3.1 (Invariant of Algorithm 3.1)**.** *After vertex u_i has been expanded fully, Algorithm 3.1 obtains triangulation $\mathcal{K}_i$ and a matching M such that (i) for every edge $e \in \mathcal{K}_i$, either $e \in \mathcal{T}_i$ or e is the flip of a safely flippable edge of $\mathcal{T}_i$, and (ii) every edge in the matching M is an edge safe in $\mathcal{T}_i$ that has not been flipped and all nodes of $G^\star_{\mathcal{K}_i}$ that are duals of expanded triangles in $\mathcal{K}_i$ are matched by the duals of edges in M.*

Algorithm 3.1 EXPANSIONS($\mathcal{T}'$, S)

```
1:  M ← ∅
2:  for i ← 1 to m do
3:    for j ← 1 to n_i do
4:      [u_i, x], [u_i, y] ← split edges for v_ij
5:      T_ij ← EXPAND(T_i(j−1), u_i, v_ij, S)
6:      if [u_i, x] ∉ M and [u_i, y] ∉ M then
7:        M ← M ∪ {[u_i, v_ij]}
8:      else if degree(v_ij) = 3 in T_ij then
9:        EXPANDMATCHDEGREE3(T_ij, u_i, v_ij, M)
10:     else if degree(v_ij) = 4 in T_ij then
11:       z ← fourth neighbor of v_ij in T_ij
12:       if not flipped([x, z]) and not flipped([y, z]) then
13:         EXPANDMATCHDEGREE4(T_ij, u_i, v_ij, z, M)
14:       else
15:         EXPANDMATCHDEGREEGREATERTHAN4(T_ij, u_i, v_ij, M)
16:       end if
17:     else
18:       EXPANDMATCHDEGREEGREATERTHAN4(T_ij, u_i, v_ij, M)
19:     end if
20:   end for
21: end for
22: return (T_m, M)
```

Let $[u_i, x, a]$ and $[u_i, x, b]$ be the two faces incident on $[u_i, x]$, and let $[u_i, y, c]$ and $[u_i, y, d]$ be the two faces incident on $[u_i, y]$ just before v_{ij} is expanded from u_i. When incrementally constructing the matching M, Algorithm 3.1 handles two special cases separately: when the degree of v_{ij} in $\mathcal{T}_{ij}$ is 3 (Algorithm 3.2), and when it is 4 (Algorithm 3.3). In the former case, matching M can be updated without any flips. In the latter case, at most one edge, which is safely flippable, may be flipped. The separate handling is necessary to ensure that $\mathcal{K}_{ij}$ has the same topology as $\mathcal{T}_{ij}$. If the degree of v_{ij} in $\mathcal{T}_{ij}$ is greater than 4, then Procedure EXPANDMATCHDEGREEGREATERTHAN4() is invoked (see Algorithm 3.4). In algorithms 3.2-3.4, the input parameter $\mathcal{T}_a$ is one of the $\mathcal{T}_{ij}$. In what follows, we denote the corresponding triangulation K_{ij} by K_a.

Algorithm 3.2 EXPANDMATCHDEGREE3$(\mathcal{T}_a, u, v, M)$

$[u, x], [u, y] \leftarrow$ split edges for v
if flipped$([x, y])$ **then**
    flipped$([x, y]) \leftarrow$ *false*
end if
if $[u, x] \in M$ and $[u, y] \in M$ **then**
    $M \leftarrow M \cup \{[x, y]\}$
else if $[u, x] \in M$ **then**
    $M \leftarrow M \cup \{[v, y]\}$
else
    $M \leftarrow M \cup \{[v, x]\}$
end if

Algorithm 3.3 EXPANDMATCHDEGREE4$(\mathcal{T}_a, u, v, z, M)$

$[u, x], [u, y] \leftarrow$ split edges for v
if $[u, x] \in M$ and $[u, y] \in M$ **then**
    $M \leftarrow M \cup \{[v, z]\}$
else if $[u, x] \in M$ **then**
    flipped$([v, x]) \leftarrow$ *true*
    $M \leftarrow M \cup \{[u, v]\}$
else
    flipped$([v, y]) \leftarrow$ *true*
    $M \leftarrow M \cup \{[u, v]\}$
end if

Algorithm 3.4 EXPANDMATCHDEGREEGREATERTHAN4$(\mathcal{T}_a, u, v, M)$

$[u, x], [u, y] \leftarrow$ split edges for v
if $[u, x] \in M$ **then**
    flipped$([v, x]) \leftarrow$ *true*
end if
if $[u, y] \in M$ **then**
    flipped$([v, y]) \leftarrow$ *true*
end if
$M \leftarrow M \cup \{[u, v]\}$

Algorithm 3.2 is executed when vertex v, after expansion from u, has degree 3 in the resulting triangulation, $\mathcal{T}_a$. Variable M_c contains the set of matching edges for the dual graph of the triangulation just before v's expansion. Algorithm 3.2 updates M_c so that newly expanded nodes are matched. Algorithm 3.3 is executed when vertex v, after expansion from u, has degree 4 in the resulting triangulation, $\mathcal{T}_a$. Like in Algorithm 3.2, M_c contains the set of matching edges for the dual graph of the triangulation just before v's expansion. Algorithm 3.3 also updates M_c so that newly expanded nodes are matched, but possibly by flipping an edge of $\mathcal{T}_a$. Finally, Algorithm 3.4 is executed when v has degree greater than 4 after expansion from u. This is either because v has degree greater than 4 in $\mathcal{T}_a$, or it has degree equal to 4 in $\mathcal{T}_a$ but one (or both) of the edges (x, z) and (y, z) has been flipped, where z is the fourth neighbor of v in $\mathcal{T}_a$. As a result, vertex v has degree 4 in K_a. At last, Algorithm 3.4 updates M_c so that newly expanded nodes are matched, possibly by flipping one or two edges of $\mathcal{T}_a$. The edge record of the DCEL has a flag, *flipped*, to indicate when the flip of an edge of $\mathcal{T}_a$, rather than the edge itself, belongs to K_a. If e is any edge of $\mathcal{T}_a$, then flipped(e) denotes the flag associated with e, and flipped(e) is true if and only if the flip of e belongs to K_a.

The following proposition implies the invariant in Claim 3.1. The edge dual to an edge e of $\mathcal{T}_i$ is denoted by $e^\star$:

Proposition 3.3. *After vertex u_i, with $i \in \{1, \ldots, m\}$, has been fully expanded, Algorithm 3.1 produces a triangulation $\mathcal{K}_i$ and a matching M of the dual graph of $\mathcal{K}_i$ such that (i) for every edge $e \in \mathcal{K}_i$ either $e \in \mathcal{T}_i$ or e is the flip of a flippable edge $[a, b] \in \mathcal{T}_i$ such that $a, b \in V_j$, with $j \leq i$, and (ii) an edge $[a, b] \in \mathcal{K}_i$ is inserted into M during u_i's expansion only if $[a, b]$ is not the flip of an edge of $\mathcal{T}_i$, and $a = u_l$, with $l \leq i$ and $b \in V_i$, or $a \in V_l$, with $l \leq i$ and $b \in V_i$. Also, the edges in $\{e^\star \mid e \in M\}$ match all expanded triangles of $\mathcal{K}_i$.*

Proof. We give a proof by induction on i. The base case and the inductive steps will be proved, in turn, by induction on j, for $1 \leq j \leq n_i$. Note that matching M is empty at first.

Base case $(i = 1)$:
We prove by induction on j, for $1 \leq j \leq n_1$, that after v_{1j} is expanded from u_1, Algorithm 3.1 (i) produces a triangulation K_{1j} by flipping at most two safely flippable edges in $\mathcal{T}_{1j}$ of the form $[v_{1j}, v_{1k}]$, with $k < j$, and (ii) inserts an edge $[a, b]$ into M such that $[a, b]$ is not the flip of an edge of $\mathcal{T}_{1j}$ and $[a, b] = [u_1, v_{1j}]$ or $a, b \in \{v_{11}, v_{12}, \ldots, v_{1j}\}$. Furthermore, the edges in the set $\{e^\star \mid e \in M\}$ match all expanded triangles of K_{1j}.

> *Base case* $(j = 1)$: The very first expansion of v_{11} from u_1 results in two expanded triangles, $t_1 = [u_1, x, v_{11}]$ and $t_2 = [u_1, y, v_{11}]$ (see Figure 3.3). Since M is empty, neither $[u_1, x]^\star$ nor $[u_1, y]^\star$ can be matching edges prior to the expansion. Hence, Algorithm 3.1 (line 7) inserts $[u_1, v_{11}]$ into M, whose dual matches expanded triangles t_1 and t_2. In this case, $K_{11} = \mathcal{T}_{11} = \mathcal{T}' + u_1v_{11}$. So, (i) and (ii) hold for $i = j = 1$.
>
> *Inductive step* $(j > 1)$*:* Assume, by the inductive hypothesis, that for all h, with $1 \leq h \leq j - 1$, after expanding v_{1h} from u_1, Algorithm 3.1 (i) produces a triangulation K_{1h} in which only safely flippable edges of the form $[v_{1h}, v_{1k}]$, with $k < h$, are flipped, and (ii) inserts edge $[a, b]$ into M only if $[a, b]$ is not the flip of an edge in $\mathcal{T}_{1h}$ and $[a, b] = [u_1, v_{1h}]$, or $a, b \in \{v_{11}, v_{12}, \ldots, v_{1h}\}$. The edges $\{e^\star \mid e \in M\}$ match all expanded triangles of K_{1h}. Now, consider the expansion of v_{1j} from u_1, with $[u_1, x]$ and $[u_1, y]$ as split edges, and let $t_1 = [u_1, x, v_{1j}]$ and $t_2 = [u_1, y, v_{1j}]$ be the two newly expanded triangles.
>
> We have two cases:
>
> **Case 1:** $(\mathbf{u_1}, \mathbf{x}) \notin \mathbf{M}$ **and** $(\mathbf{u_1}, \mathbf{y}) \notin \mathbf{M}$**.**
> Edge $[u_1, v_{1j}]$ is inserted into M in line 7 of Algorithm 3.1. Let $K_{1j} = K_{1(j-1)} + u_1v_{1j}$ (see

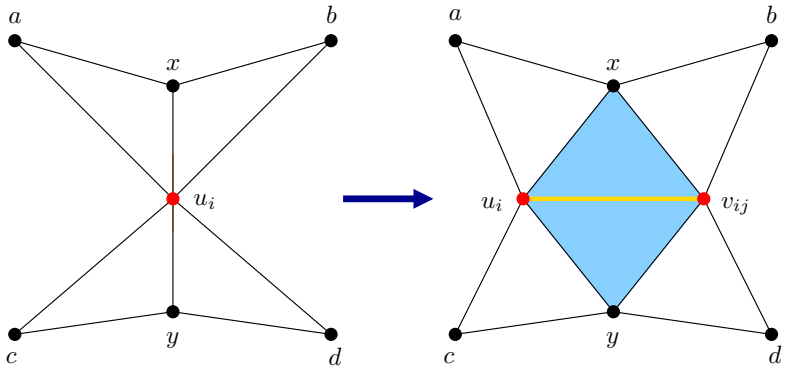

Figure 3.3: Expanding v_{ij} from u_i with $[u_i, x] \notin M$ and $[u_i, y] \notin M$ ($i \geq 1$).

Figure 3.3). Since no edges are flipped, property (i) holds trivially. In addition, the edge $[u_1, v_{1j}]$ inserted into M is obviously not a flipped edge (since it is newly inserted as a result of the expansion of v_{1j}). Since $[u_1, v_{1j}]^\star$ matches the newly expanded triangle nodes t_1 and t_2, it follows from the inductive hypothesis that the edges $\{e^\star \mid e \in M\}$ match all expanded trianges of K_{1j} and hence property (ii) holds.

Case 2: at least one of $[u_1, x]$ and $[u_1, y]$ is in M.

Suppose that $[u_1, x]$ is in M and refer to Figure 3.3 again. Then, it follows by the inductive hypothesis that $x = v_{1k}$, for some $k < j$. Let $t_3 = [u_1, x, a]$ be the face that $[u_1, x]^\star$ matches with $[u_1, x, b]$ in $K_{1(j-1)}$ prior to the expansion of v_{1j}. Similarly, if $[u_1, y] \in M$, $y = v_{1l}$ for some $l < j$. Let $t_4 = [u_1, y, c]$ be the face that $[u_1, y]^\star$ matches with $[u_1, x, d]$ in $K_{1(j-1)}$ prior to the expansion of v_{1j}.

We have three cases depending on the degree of v_{1j} in $\mathcal{T}_{1j}$:

Case 2.1: degree of v_{1j} in $\mathcal{T}_{1j}$ is 3.

Observe that in this case, face $[u_1, x, y]$ must be a face of $\mathcal{T}_{1(j-1)}$. Let $[x, y, z]$ be the other face incident on edge $[x, y]$. In Algorithm 3.1, this case is handled by the procedure in Algorithm 3.2. The edge to be inserted in M depends on whether $K_{1(j-1)}$ contains the flip of edge $[x, y]$ in $\mathcal{T}_{1(j-1)}$.

If $K_{1(j-1)}$ contains the flip of $[x, y]$, then Algorithm 3.2 (lines 2-4) "unflips" $[x, y]$ to make x and y adjacent again in K_{1j} (Figure 3.4). Observe that the "unflip" can be safely carried out without affecting M because it follows by the inductive hypothesis that $[x, y] = [v_{1k}, v_{1l}]$, for some $k, l < j$, and the flip of $[x, y]$ is not in M. In other words, edge $[x, y]$ is safe in $\mathcal{T}_{1j}$, and it is not in M. Let K denote the triangulation resulting from unflipping $[x, y]$ in $K_{1(j-1)}$, i.e., K is the triangulation obtained from $K_{1(j-1)}$ by replacing $\{[u_1, z], [u_1, x, z], [u_1, y, z]\}$ with $\{[x, y], [u_1, x, y], [x, y, z]\}$. Then, we have that $K_{1j} = K + u_1 v_{1j}$. Since a flip was reversed and no new flips were introduced, it follows from the inductive hypothesis that K_{1j} satisfies property (i).

If both $[u_1, x]$ and $[u_1, y]$ are in M, Algorithm 3.2 (line 6) inserts $[x, y] = [v_{1k}, v_{1l}]$ into M, as shown in Figure 3.5(a). Now, $[u_1, x]^\star$ matches faces t_3 and t_1, $[u_1, y]^\star$ matches t_4 and t_2, and $[x, y]^\star$ matches $[v_{ij}, x, y]$ and $[x, y, z]$. So, all newly expanded triangles of K_{1j} are matched. Clearly, property (ii) of the claim is satisfied. If only $[u_1, x]$ is in M, Algorithm 3.2 (line 8) inserts $[v_{1j}, y] = [v_{1j}, v_{1l}]$ into M, as shown in Figure 3.5(b). If $[u_1, y, z]$ is a matched face in $K_{1(j-1)}$, then since $[u_1, y] \notin M$ and the flip of $[x, y]$ is not in M, it must be the case that $[y, z] \in M$. Let t_5 be the face matched with $[u_1, y, z]$ by $[y, z]^\star$ prior to the expansion. In the

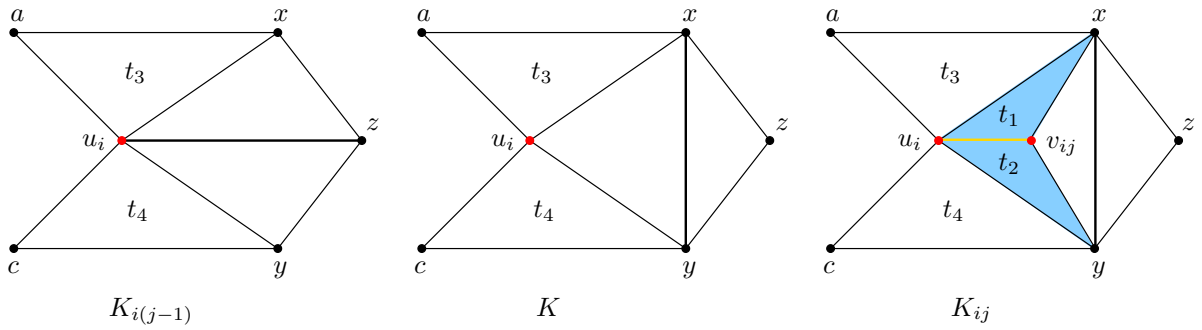

Figure 3.4: Expanding v_{ij} from u_i: degree of v_{ij} in $\mathcal{T}_{ij}$ is 3 and $[x, y]$ is flipped in $K_{i(j-1)}$.

updated matching, $[u_1, x]^\star$ matches t_1 and t_3, $[v_{1j}, y]^\star$ matches t_2 and $[v_{1j}, x, y]$, and $[y, z]^\star$ matches $[x, y, z]$ and t_5. If $[u_1, y, z]$ is not a matched face, $[x, y, z]$ stays unmatched. In either case, all newly expanded triangles are matched and property (ii) of the claim is satisfied. The case of only $[u_1, y]$ in M is symmetric.

Conversely, assume that $[x, y]$ is in $K_{1(j-1)}$. Then, only one of $[u_1, x]$ and $[u_1, y]$ can belong to M. Suppose, without loss of generality, that $[u_1, x] \in M$. In this case, procedure 3.2 (line 8) inserts $[v_{1j}, y]$ into M (see Figure 3.6). Let $K_{1j} = K_{1(j-1)} + u_1 v_{1j}$. It follows immediately from the inductive hypothesis that K_{1j} satisfies property (i). Since v_{1j} is of degree 3 in $\mathcal{T}_{1j}$, it implies that $[v_{1j}, y]$ is contractible (see Proposition 3.1). Since u_1 is the first vertex that is being expanded (and hence the vertex processed at last), it follows from Lemma 3.1 that $y \neq u_k$, for every k. Thus, $y = v_{1l}$, for some $l < j$, and therefore the edge inserted into M is of the form $[v_{1j}, v_{1l}]$. In addition, $[u_1, x]^\star$ and $[v_{1j}, y]^\star$ match the newly expanded triangles of K_{1j}. So, property (ii) holds.

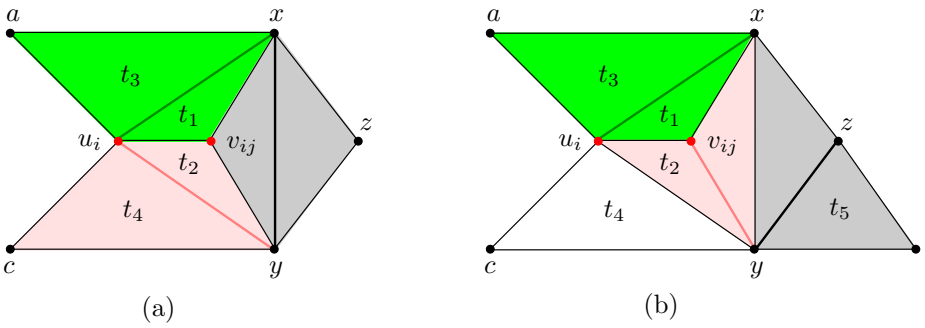

Figure 3.5: (a) $[u_i, x] \in M$ and $[u_i, y] \in M$. (b) $[u_i, x] \in M$ and $[u_i, y] \notin M$.

Case 2.2: degree of v_{1j} in $\mathcal{T}_{1j}$ is 4.

Let z be the fourth vertex of v_{1j} in $\mathcal{T}_{1j}$. As pointed out earlier, faces $[u_1, x, z]$ and $[u_1, y, z]$ are adjacent in $\mathcal{T}_{1(j-1)}$. If both edges $[x, z]$ and $[y, z]$ are in $K_{1(j-1)}$, Algorithm 3.1 (line 13) uses Algorithm 3.3 to update the matching. If the flip of at least one of the edges $[x, z]$ and $[y, z]$ is in $K_{1(j-1)}$, then it implies that v_{1j} has degree greater than 4 in K_{1j} and can be handled in the same manner as a vertex whose degree is greater than 4 in $\mathcal{T}_{1j}$. This is done by Algorithm 3.4 (line 15 in Algorithm 3.1). Here, we prove the inductive step for Algorithm 3.3. The inductive step for Algorithm 3.4 is proved in Case 2.3. First, observe that $[v_{1j}, z]$ must be a contractible edge because the only possible critical cycle with $[v_{1j}, z]$ is (u_1, v_{1j}, z). But, if such a cycle

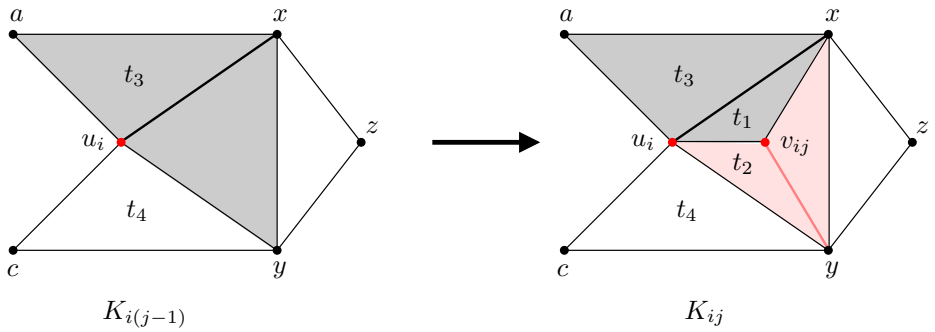

Figure 3.6: Expanding v_{ij} from u_i: degree of v_{ij} in $\mathcal{T}_{ij}$ is 3, $[x, y] \in K_{i(j-1)}$, and $[u_i, x]$ is in M.

existed, then edge $[u_1, v_{1j}]$ would not be contractible, which is a contradiction. Since u_1 is the first expanded vertex, it follows from Lemma 3.1 that $z \neq u_k$, for any k. Thus, $z = v_{1p}$, for some $p < j$.

If $[u_1, x] \in M$ and $[u_1, y] \in M$, let $K_{1j} = K_{1(j-1)} + u_1v_{1j}$. Since no edges are flipped, it follows from the inductive hypothesis that property (i) holds. In addtion, Algorithm 3.3 (line 3) inserts $[v_{1j}, z] = [v_{1j}, v_{1p}]$ into M (see Figure 3.7). Now, $[u_1, x]^\star$ matches faces t_3 and t_1, $[u_1, y]^\star$ matches t_4 and t_2, and $[v_{1j}, z]^\star$ matches $[v_{1j}, x, z]$ and $[v_{1j}, y, z]$. So, all newly expanded triangles of K_{1j} are matched. Clearly, property (ii) of the claim holds. Conversely, if $[u_1, x] \in M$ and $[u_1, y] \notin M$, Algorithm 3.3 (lines 5-6) flips $[v_{1j}, x] = [v_{1j}, v_{1k}]$, for $k < j$, and inserts $[u_1, v_{1j}]$ into M, as illustrated by Figure 3.8. In this case, let K_{1j} denote the triangulation resulting from flipping $[v_{1j}, x]$ in $K_{1(j-1)} + u_1v_{1j}$, i.e., K_{1j} is the triangulation obtained from $K_{1(j-1)} + u_1v_{1j}$ by replacing $\{[v_{1j}, x], [u_1, v_{1j}, x], [v_{1j}, x, z]\}$ with $\{[u_1, z], [u_1, x, z], [u_1, v_{1j}, z]\}$. Since Lemma 3.2 implies that $[v_{1j}, x]$ is flippable, and since $[v_{1j}, x]$ is clearly an edge safe in $K_{1(j-1)} + u_1v_{1j}$, it follows from the inductive hypothesis that K_{1j} satisfies property (i). To see that property (ii) also holds, we only need to show that all expanded triangles in K_{1j} are matched. If $[u_1, y, z]$ is a matched face in $K_{1(j-1)}$, then since $[u_1, y] \notin M$ and $[u_1, z]$ cannot be in M, it must be the case that $[y, z] \in M$. Let t_5 be the face matched with $[u_1, y, z]$ by $[y, z]^\star$ prior to the expansion. In K_{1j}, $[y, z]^\star$ matches $[v_{1j}, y, z]$ and t_5, $[u_1, x]^\star$ matches t_3 and $[u_1, x, z]$, and $[u_1, v_{1j}]^\star$ matches $[u_1, v_{1j}, z]$ and t_2. If $[u_1, y, z]$ is not a matched face in $K_{1(j-1)}$, then face $[v_{1j}, y, z]$ stays unmatched in K_{1j}. In either case, all expanded triangles in K_{1j} are matched. The case of $[u_1, y] \in M$ and $[u_1, x] \notin M$ is symmetric.

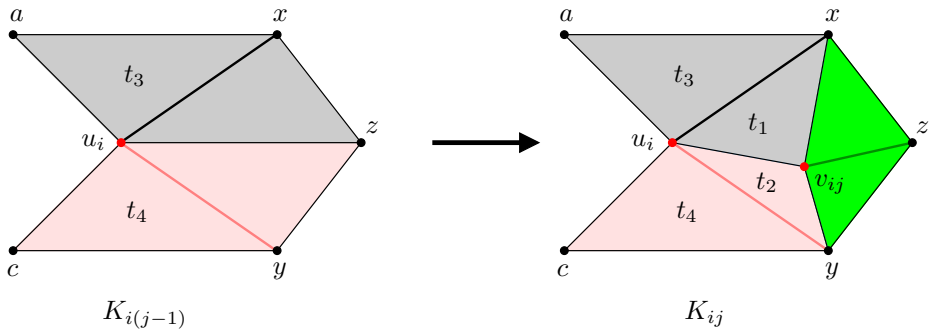

Figure 3.7: Expanding v_{ij} from u_i: degree of v_{ij} in K_{ij} is 4, and $[u_i, x] \in M$ and $[u_i, y] \in M$.

Case 2.3: degree of v_{1j} in $\mathcal{T}_{1j}$ is greater than 4.

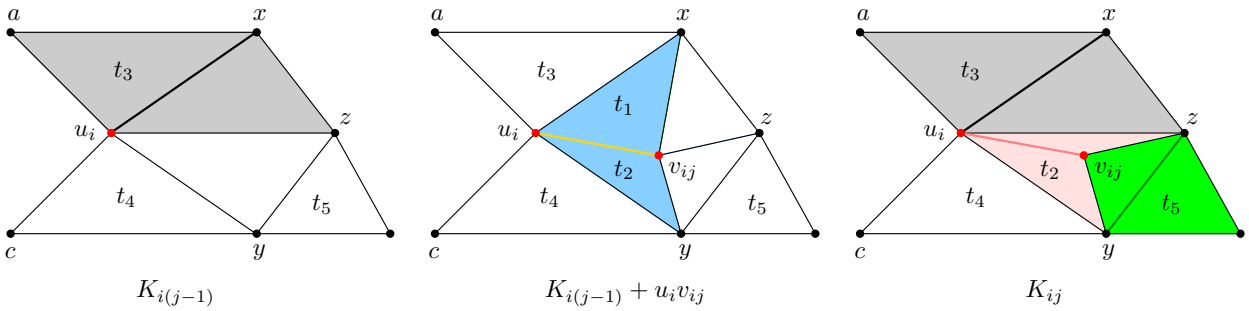

Figure 3.8: Expanding v_{ij} from u_i: degree of v_{ij} in K_{ij} is 4, and $[u_i, x] \in M$ and $[u_i, y] \notin M$.

If $[u_1, x] \in M$, Algorithm 3.4 flips edge $[v_{1j}, x]$ and inserts $[u_1, v_{1j}]$ into M. Let $K = K_{1(j-1)} + u_1 v_{1j}$, and let K_{1j} be the triangulation obtained from K by flipping $[v_{1j}, x]$ (see Figure 3.9). Since $[u_1, x] \in M$, the inductive hypothesis tells us that x is of the form v_{1h}, for some $h < j$. To see that $[v_{1j}, x]$ is flippable, note that $[u_1, b] \in K_{1(j-1)}$ is either an edge of $\mathcal{T}_{1(j-1)}$ or it is the flip of an edge. In the case of the former, it follows that $[v_{1j}, b]$ is an edge of $\mathcal{T}_{1j}$ and hence Lemma 3.2 implies that $[v_{1j}, x]$ is flippable. In the case of the latter, we know by the inductive hypothesis that $[u_1, b]$ is the result of a safe flip at a previous expansion step, and hence it cannot be an edge in $\mathcal{T}_{1j}$. Therefore, edge $[v_{1j}, x]$ is flippable in this case as well. We conclude that K_{1j} satisfies property (i). In addition, $[u_1, x]^\star$ matches faces t_1 and $[u_1, x, b]$, and $[u_1, v_{1j}]^\star$ matches faces $[u_1, v_{1j}, b]$ and $[u_1, v_{1j}, y]$. It is clear that property (ii) is satisfied as well.

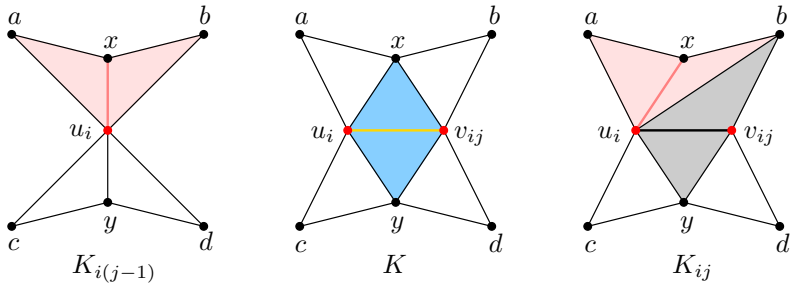

Figure 3.9: Expanding v_{ij} from u_i: degree of v_{ij} in K_{ij} is greater than 4, and $[u_i, x] \in M$ and $[u_i, y] \notin M$.

Similarly, if $[u_1, y] \in M$, then Algorithm 3.4 flips $[v_{1j}, y]$ and inserts $[u_1, v_{1j}]$ into M. An analogous argument shows that the resulting K_{1j} and matching M satisfy properties (i) and (ii), respectively. Finally, if both $[u_1, x]$ and $[u_1, y]$ are in M, Procedure 3.4 flips both $[v_{1j}, x]$ and $[v_{1j}, y]$, and inserts $[u_1, v_{1j}]$ into M. The resulting K_{1j} and M satisfy properties (i) and (ii) (see Figure 3.10).

This proves the proposition for u_1. After u_1 is fully expanded, we have a triangulation $K_1 = K_{1n_1}$ whose edges are the edges of $\mathcal{T}_1$, but with some edges safe in $\mathcal{T}_1$ flipped, as well as a matching M that contains only edges safe in $\mathcal{T}_1$, whose duals match all expanded triangles of K_1. In what follows, we have the inductive step.

Inductive step $(i > 1)$:
Assume, by the inductive hypothesis, that after u_{i-1} is fully expanded, Algorithm 3.1 produces (i) a triangulation K_{i-1} that is the same as $\mathcal{T}_{i-1}$ but with some edges $[a, b]$ safe in $\mathcal{T}_{i-1}$ flipped, where $a, b \in V_h$ and $h \leq (i-1)$, and (ii) a matching M containing only edges of the form $[u_l, b]$, where $b \in V_h$, $l \leq (i-1)$ and

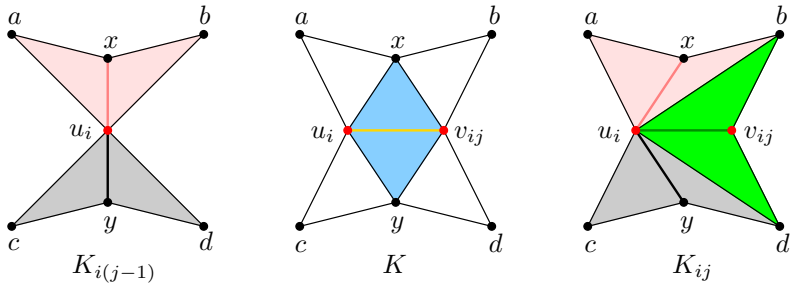

Figure 3.10: Expanding v_{ij} from u_i: degree of v_{ij} in K_{ij} is greater than 4, and $[u_i, x] \in M$ and $[u_i, y] \in M$.

$h \geq l$, or $[a, b]$, with $a \in V_l$ and $l \leq (i-1)$, and $b \in V_h$ and $h \leq (i-1)$. All expanded triangle nodes in the dual of K_{i-1} are matched by duals of edges in M. We prove that (i) and (ii) hold after u_i is expanded fully.

The proof of the inductive step is similar to the base case, and we highlight only the important differences between the two. In particular, we prove by induction on j, with $1 \leq j \leq n_i$, that after v_{ij} is expanded from u_i, Algorithm 3.1 (i) produces a triangulation K_{ij} by flipping at most two safely flippable edges in $\mathcal{T}_{ij}$ of the form $[v_{ij}, v_{ik}]$, with $k < j$, and (ii) inserts an edge $[a, b]$ into M where $[a, b]$ is not the flip of an edge of $\mathcal{T}_{ij}$ and $[a, b] = [u_l, v_{ij}]$, with $l \leq i$, or $a \in V_l$, with $l \leq i$ and $b \in V_i$. The edges in $\{e^\star \mid e \in M\}$ match all expanded triangles of K_{ij}. As in the proof of the base case, let $[u_i, x]$ and $[u_i, y]$ be the split edges when vertex v_{ij} is expanded from u_i, and let $t_1 = [u_i, x, v_{ij}]$ and $t_2 = [u_i, y, v_{ij}]$ be the two newly expanded triangles.

Base case ($j = 1$): When v_{i1} is expanded from u_i, neither $[u_i, x]$ nor $[u_i, y]$ are in M because, by the inductive hypothesis, any edge in M incident on u_l must have $l \leq (i-1)$. Hence, Algorithm 3.1 (line 7) inserts $[u_i, v_{i1}]$ into M, as shown in Figure 3.3. Let $K_{i1} = K_{i-1} + u_i v_{i1}$. By using an argument similar to the base case of u_1, it follows that properties (i) and (ii) are satisfied by K_{i1} and M, respectively.

Inductive step ($j > 1$): Assume, by the inductive hypothesis that for all $1 \leq h \leq j-1$, after expanding v_{ih} from u_i, Algorithm 3.1 (i) produces a triangulation K_{ih} in which only safely flippable edges of the form $[v_{ih}, v_{ik}]$, with $k < h$, are flipped, and (ii) inserts edge $[a, b]$ into M only if $[a, b]$ is not a flipped edge of $\mathcal{T}_{ih}$ and $[a, b] = [u_i, v_{ih}]$ or $a \in \{u_l\} \cup V_l$ and $b = v_{ih}$, with $l < i$, or $a, b \in \{v_{i1}, v_{i2}, \ldots, v_{ih}\}$. In addition, the edges in the set $\{e^\star \mid e \in M\}$ match all expanded triangles of K_{ih}.

Now, consider the expansion of v_{ij} from u_i. Like in the inductive step for u_1, we also have two cases:

Case 1: $(\mathbf{u_i}, \mathbf{x}) \notin \mathbf{M}$ and $(\mathbf{u_i}, \mathbf{y}) \notin \mathbf{M}$.
In this case, Algorithm 3.1 (line 8) inserts $[u_i, v_{ij}]$ into M (see Figure 3.3). Let $K_{ij} = K_{i(j-1)} + u_i v_{ij}$. By using an argument similar to the base case of u_1, it follows that properties (i) and (ii) are satisfied by K_{ij} and M, respectively.

Case 2: at least one of $[\mathbf{u_i}, \mathbf{x}]$ and $[\mathbf{u_i}, \mathbf{y}]$ is in $\mathbf{M}$.
Suppose that $[u_i, x]$ is in M and refer to Figure 3.3 again. Then, it follows by the inductive hypothesis that $x = v_{ik}$, for some $k < j$. Similarly, if $[u_i, y] \in M$, then $y = v_{il}$ for some $l < j$. Let faces t_3 and t_4 be as previously defined for the base case of u_1. We have three cases depending on the degree of v_{ij} in $\mathcal{T}_{ij}$:

Case 2.1: degree of $\mathbf{v_{ij}}$ in $\mathcal{T}_{\mathbf{ij}}$ is 3.
In this case $[u_i, x, y]$ is a face in $\mathcal{T}_{i(j-1)}$. In Algorithm 3.1, this case is handled by Algorithm 3.2.

The edge to be inserted in M depends on whether $K_{i(j-1)}$ contains the flip of $[x, y]$. If so, then the case is identical to the base case (with u_1 replaced by u_i and v_{1j} replaced by v_{ij}) and will not be repeated here (see Figure 3.4 and Figure 3.5). If $K_{i(j-1)}$ contains $[x, y]$, rather than the flip of $[x, y]$, then Algorithm 3.2 inserts $[v_{ij}, y]$ or $[v_{ij}, x]$ into M (see Figure 3.6). Assume, without loss of generality, that $[v_{ij}, y]$ is inserted into M. As explained in Case 2.1 of the base case for u_1, since $[v_{ij}, y]$ is a contractible edge in $\mathcal{T}_{ij}$, it follows from Lemma 3.1 that $y \neq u_k$, for any $k > i$. Hence, either $y = u_k$ or $y \in V_k$ or $y = v_{kj}$, for some $k < i$. With K_{ij} constructed in a similar manner to K_{1j}, it follows that triangulation K_{ij} and matching M satisfy properties (i) and (ii), respectively.

Case 2.2: degree of $\mathbf{v}_{ij}$ in $\mathcal{T}_{ij}$ is 4.
Let z be the fourth vertex of v_{ij} in $\mathcal{T}_{ij}$. So, faces $[u_i, x, z]$ and $[u_i, y, z]$ are adjacent in $\mathcal{T}_{i(j-1)}$. The construction of triangulation K_{ij} and matching M by Algorithm 3.3 is exactly as in the base case for u_1 (with u_1 replaced by u_i and v_{1j} replaced by v_{ij}). Refer to Figures 3.7 and 3.8 and to the discussion of Case 2.2 of the base case for u_1, where we argued that $[v_{ij}, z]$ must be a contractible edge. It follows from Lemma 3.1 that $z \neq u_k$, for every $k > i$. Hence, either $z = u_k$ or $z \in V_k$ or $z = v_{kj}$, for some $k < i$. Since Algorithm 3.3 inserts either $[v_{ij}, z]$ or $[u_i, v_{ij}]$ into M, it follows that triangulation K_{ij} and matching M satisfy properties (i) and (ii), respectively.

Case 2.3: degree of $\mathbf{v}_{ij}$ in $\mathcal{T}_{ij}$ is greater than 4.
Algorithm 3.4 produces triangulation K_{ij} and matching M by flipping at most two edges, namely, $[v_{ij}, x]$ and $[v_{ij}, y]$, and inserting edge $[u_i, v_{ij}]$ into M (see Figures 3.9 and 3.10). Once again, this case is identical to Case 2.3 of the base case for u_1 (with u_1 replaced by u_i and v_{1j} replaced by v_{ij}). Hence, triangulation K_{ij} and matching M satisfy properties (i) and (ii), respectively.

This proves the inductive step. Therefore, after u_i is fully expanded, we have a triangulation, $K_i = K_{in_i}$, whose edges are the edges of $\mathcal{T}_i$, but with some edges safe in $\mathcal{T}_i$ flipped, and a matching M that contains only edges safe in $\mathcal{T}_i$, whose duals match all expanded triangle nodes in the dual graph of K_i. So, our claim follows. □

Once the execution of Algorithm 3.1 is finished, we get the input triangulation $\mathcal{T}$ back and a matching $M^\star$ on the dual graph $G^\star_{\mathcal{K}}$ of triangulation $\mathcal{K} = K_{mn}$. There is no need to explicitly construct $\mathcal{K}$ during the execution of Algorithm 3.1, as the flag *flipped* in the edge record of the DCEL allows us to recover $\mathcal{K}$ from $\mathcal{T}$ whenever we want. However, $M^\star$ is not a perfect matching on $G^\star_{\mathcal{K}}$, as no face of $\mathcal{K}$ that is also in the irreducible triangulation $\mathcal{T}'$ is matched by Algorithm 3.1.

3.4 Complexity

In this section, we analyze the time and space complexity of the new conversion algorithm described in this section. The time complexity of the contraction stage was shown in [36] and is stated below:

Lemma 3.3. *[36] Given a triangulation $\mathcal{T}$ of a surface $\mathcal{S}$ of positive genus g, the contraction stage of our algorithm computes an irreducible triangulation $\mathcal{T}'$ of $\mathcal{S}$ in $\mathcal{O}(g^2 + g \cdot n_f)$-time, where n_f is the number of faces of $\mathcal{T}$. If $\mathcal{S}$ is a genus-0 surface, then the contraction stage of our algorithm computes $\mathcal{T}'$ in time linear in n_f.*

The following lemma states the time complexity of the expansion stage:

Lemma 3.4. *Let $\mathcal{T}$ be any triangulation of a surface $\mathcal{S}$. If $\mathcal{T}'$ is the irreducible triangulation obtained from $\mathcal{T}$ by the contraction stage of our algorithm, then the expansion stage of our algorithm computes another triangulation, $\mathcal{K}$, of $\mathcal{S}$, with the same vertex set as $\mathcal{T}$ and the same number of edges and faces as $\mathcal{T}$, as well as a perfect matching M on the dual graph, $G^\star_\mathcal{K}$, of $\mathcal{K}$ in $\mathcal{O}(\max\{1, g\} \cdot n_f)$-time, where n_f is the number of faces of $\mathcal{T}$.*

Proof. Let n_v and n_e be the number of vertices and edges of $\mathcal{T}$. Recall that Algorithm 3.1 reconstructs $\mathcal{T}$ from $\mathcal{T}'$ by carrying out $n_e - n'_e$ vertex split operations, where n'_e is the number of edges of $\mathcal{T}'$, each of which reverses an edge contraction performed in the contraction stage of the algorithm. Each vertex split operation takes constant time to execute. In between every two vertex split operations, Algorithm 3.1 can flip at most two edges of the current triangulation, where each edge flip operation also takes constant time to execute. So, the time complexity of Algorithm 3.1 is in $\mathcal{O}(n_e)$. The flipped edges give rise to another triangulation, $\mathcal{K}$, of $\mathcal{S}$, which has the same vertex set of $\mathcal{T}$, as $\mathcal{K}$ can be derived from $\mathcal{T}$ by carrying out the flips. Since an edge flip operation does not change the number of edges and faces of a triangulation, $\mathcal{K}$ has the same number of edges and faces as $\mathcal{T}$. Proposition 3.3 ensures that the dual nodes in $G^\star_\mathcal{K}$ of all expanded triangles of $\mathcal{K}$ are matched by the edges in $M^\star = \{e^\star \mid e \in M\}$, where M is the "matching" returned by Algorithm 3.1. However, the dual nodes of faces in $\mathcal{T}'$, which are also in $\mathcal{K}$, are not matched by $M^\star$. If $\mathcal{S}$ is a genus-0 surface, then we know that $\mathcal{T}'$ is (isomorphic to) $\mathcal{T}_4$, which means that the number fo faces, n'_f, of $\mathcal{T}'$ is four. Otherwise, Theorem 3.1 and our assumption that $n'_f \in \Theta(n'_v)$ tell us that there are $\mathcal{O}(g)$ faces in $\mathcal{T}'$, where n'_v is the number of vertices of $\mathcal{T}'$. Hence, the path finder procedure [22] is invoked $\mathcal{O}(\max\{1, g\})$ times to make $M^\star$ a perfect matching on $G^\star_\mathcal{K}$. Since each execution of the path finder procedure takes $\mathcal{O}(n_e)$-time, a perfect matching on $G^\star_\mathcal{K}$ can be found in $\mathcal{O}(\max\{1, g\} \cdot n_e)$-time. Finally, since we assumed that $n_e, n_f \in \Theta(n_v)$, where n_f is the number of faces of $\mathcal{T}$, the time complexity of the expansion stage is indeed

$$\mathcal{O}(n_e) + \mathcal{O}(\max\{1, g\} \cdot n_e) = \mathcal{O}(\max\{1, g\} \cdot n_f)\,.$$

□

Finally, the following theorem states the time and space complexity of our new algorithm to compute a quadrangulation of a closed surface $\mathcal{S}$.

Theorem 3.2. *Given a triangulation $\mathcal{T}$ of a surface $\mathcal{S}$ with genus g, the graph of a quadrangulation $\mathcal{Q}$ of $\mathcal{S}$ with the same vertex set as $\mathcal{T}$ can be generated in $\mathcal{O}(g^2 + g \cdot n)$-time if g is positive, and in $\mathcal{O}(n)$-time otherwise, where n is the number of faces of $\mathcal{T}$. In either case, linear space is required.*

Proof. The time complexity of our algorithm follows immediately from Lemma 3.3 and Lemma 3.4. As for the space complexity, we note that the space required to store the augmented DCEL is in $\Theta(n_v + n_f + n_e)$, where n_v, n_e, and n_f are the number of vertices, edges, and faces of $\mathcal{T}$. In turn, the list R of unprocessed vertices, the list S of tested edges, and the edge set M require $\Theta(n_e)$-space each. Since we assumed that $n_e, n_f \in \Theta(n_v)$, we can conclude that the overall space required by our algorithm on input $\mathcal{T}$ is linear in n_f. □

4 Experimental results

We implemented the simple greedy algorithm (Section 3.1), Puli and Segal's algorithm (Section 3.1), Diks and Stanczyk's algorithm (Section 3.2), our new algorithm (Section 3.3), and the augmenting path procedure described by Tarjan in [21] in `C++`. All implementations were compiled with the compiler `clang`

`503.0.40` using the `-O2` option. We ran the implementations and collected data on a Macbook Pro running OS X 10.9.4 at 2.4 GHz (Intel Core 2 Duo), with 3MB of level-two data cache and 4GB of RAM. The implementations of the greedy algorithms and our new algorithm are based on the same data structure for surface triangulations (i.e., an augmented, doubly-connected edge list [38]), while the implementations of the graph-based algorithm and augmenting path procedure are based on specific data structures for graphs and trees [21, 39], and for fully-dynamic algorithms for connectivity queries on graphs [40]. Our implementations do not depend on any third-party libraries, and they were made publicly available on the internet[1].

Time measurements refer to the time to compute an initial triangle pairing and the time to make the initial pairing perfect (i.e., the time spent by the augmenting path procedure). If the initial pairing is already perfect, which is always the case for Diks and Stanczyk's algorithm, the second step is ignored. To compare the implementations, we consider three groups of triangulations. The first group consists of low genus triangulations, which are typically found in graphics applications (Table 4.1). The second group consists of a hierarchy of 5 triangulations of the same genus-0, box-shaped surface with 3,844 cavities (Table 4.2(a)). The third group consists of a hierarchy of 5 triangulations of the same box-shaped surface with 3,500 holes (Table 4.2(b)). The second and third groups allow us to compare worst-case scenarios for Puli and Segal's algorithm and for the new algorithm. Triangulations A2-A5 (resp. B2-B5) in Table 4.2(a) (resp. Table 4.2(b)) were generated from triangulation A1 (resp. B1) by a topology-preserving, mesh simplification algorithm, implemented in the `Graphite` tool[2], which tries to preserve the triangulation geometric shape as well.

Triangulation	# Vertices	# Edges	# Triangles	# Genus
Botijo	20,000	60,024	40,016	5
Eros	197,230	591,684	394,456	0
Filigree	29,129	87,771	58,514	65
Gargoyle	97,130	291,384	194,256	0
Happy Buddha	543,652	1,631,574	1,087,716	104
Iphigenia	351,750	1,055,268	703,512	4

Table 4.1: Euler characteristics of the triangulations in the first group.

From now on, we denote the simple greedy algorithm, Puli and Segal's algorithm, Diks and Stanczyk's algorithm, and our new algorithm by **SG**, **PS**, **DS**, and **LRS**, respectively. We initially executed **DS** and **LRS** exactly once on each triangulation of the first group, while **SG** and **PS** were executed ten times on each triangulation. This is because we randomized the edges and the triangles of the input triangulations of **SG** and **PS**, respectively, in order to avoid selecting edges (resp. triangles) in the fixed order they show up in an input file. For both **SG** and **PS**, we computed and recorded the average execution times over the ten runs on each triangulation. The results corresponding to the triangulations of the first group are shown in Table 4.3.

Triang.	# Vertices	# Edges	# Triangles
A1	2,097,150	6,291,444	4,194,296
A2	991,416	2,974,242	1,982,828
A3	745,697	2,237,085	1,491,390
A4	418,073	1,254,213	836,142
A5	172,354	517,056	344,704

(a)

Triang.	# Vertices	# Edges	# Triangles
B1	2,104,150	6,333,444	4,222,296
B2	994,979	3,005,931	2,003,954
B3	748,496	2,266,482	1,510,988
B4	419,852	1,280,550	853,700
B5	173,370	541,104	360,736

(b)

Table 4.2: Euler characteristics for (a) second group (genus 0), and (b) third group (genus 3,500).

[1] `http://www.mat.ufrn.br/~mfsiqueira/pairing.zip` (for the time being, it is revealed to the reviewers only)
[2] `http://alice.loria.fr/index.php/software/3-platform/22-graphite.html`

Triangulation	Alg.	(a)	(b)	(c)	(d)	(e)
Botijo	**SG**	0.00165190	87.4455%	5,023.8	4.98534900	4.98700090
Botijo	**PS**	0.05538500	99.9735%	10.6	0.01776800	0.07315300
Botijo	**DS**	1.93481000	100.0000%	0.0	0.00000000	1.93481000
Botijo	**LRS**	0.70350300	99.8471%	62.0	0.01820400	0.72170700
Eros	**SG**	0.01006460	87.6632%	48,663.4	813.20450000	813.21456460
Eros	**PS**	0.61721410	99.9988%	4.8	0.06892860	0.68614270
Eros	**DS**	80.59760000	100.0000%	0.0	0.00000000	80.59760000
Eros	**LRS**	1.14758400	99.9990%	4.0	0.01634400	1.16392800
Filigree	**SG**	0.00102340	87.8566%	7,105.6	11.16882000	11.16984340
Filigree	**PS**	0.07795620	99.8756%	72.8	0.08469160	0.16264780
Filigree	**DS**	3.72425000	100.0000%	0.0	0.00000000	3.72425000
Filigree	**LRS**	0.14898500	98.7148%	752.0	0.27949900	0.42848400
Gargoyle	**SG**	0.00395690	87.5094%	24,263.8	213.01330000	213.01725690
Gargoyle	**PS**	0.29626481	99.9877%	23.8	0.20473760	0.50100241
Gargoyle	**DS**	21.62160000	100.0000%	0.0	0.00000000	21.62160000
Gargoyle	**LRS**	0.56195600	99.9979%	4.0	0.00883900	0.57079500
Happy Buddha	**SG**	0.06403060	87.5097%	135,859.0	756.37240000	756.38179630
Happy Buddha	**PS**	1.75697400	99.8856%	1,244.0	54.51173000	56.26870400
Happy Buddha	**DS**	288.18500000	100.0000%	0.0	0.00000000	288.18500000
Happy Buddha	**LRS**	3.76612000	99.8845%	1,474.0	13.88260000	17.64872000
Iphigenia	**SG**	0.00848030	87.4546%	44,294.6	779.35900000	779.36748030
Iphigenia	**PS**	0.52977010	99.9699%	106.4	1.52379900	2.05356910
Iphigenia	**DS**	81.29790000	100.0000%	0.0	0.00000000	81.29790000
Iphigenia	**LRS**	1.00929100	99.9819%	64.0	0.41545500	1.42474600

Table 4.3: Measurements obtained from executing the algorithms on each triangulation of the first group. (a) Time to compute the initial pairing. (b) Percentage of paired triangles. (c) Number of unpaired triangles. (d) Time to make the initial pairing a perfect one — *if the initial pairing is not a perfect pairing already*. (e) Total execution time to produce a perfect pairing. All time measures are shown in seconds. For **SG** and **PS**, the displayed values are average values over ten executions of each algorithm on each triangulation.

Observe that the number k of triangles left unpaired by the initial pairing computed by **SG** is basically a fixed fraction of the number n_f of triangles of the triangulation, i.e., $k = \alpha \cdot n_f$, where $\alpha \in [0.12, 0.13]$ for **SG**. The scenario for **PS** is similar, but α varies slightly more, with $\alpha \in [10^{-5}, 0.0012]$, and α grows as the genus of the surface gets larger. Nevertheless, the pairing ratio associated with **PS** is remarkably larger than the one of **SG**. Since the time to make the initial pairing perfect is proportional to $k \cdot n_f$, the difference between the pairing ratios of **SG** and **PS** causes the average total execution time of **PS** to be over 300 times shorter than that of **SG** for triangulations with a large number of triangles, such as Eros, Gargoyle, and Iphigenia. Observe also that the time spent by **SG** to compute the initial pairing is smaller than that of **PS** by at least one order of magnitude. However, the total execution time is largely dominated by the time to make the initial pairing perfect, and the influence of the latter time over the former increases with n_f. Since **DS** computes a perfect pairing in a single step, the initial pairing is already perfect, and thus the total execution time is the same as the time to compute the initial pairing (see columns (a) and (e) of Table 4.3). It is clear from our time measurements that **DS** is also a much better alternative to the simple greedy algorithm **SG**, but it is inferior to **PS**. This is mainly due to the difference between the ratios of paired triangles of **SG** and **PS**. Finally, the new algorithm, **LRS**, perfomed better than **SG** and **DS** on all triangulations, but it

surpassed **PS** on two triangulations only: Happy Buddha and Iphigenia. To understand why and when **LRS** can perform better than **PS** (and the other two), we notice that (i) the number of triangles left unpaired by the initial pairing of **LRS** does *not* depend on n_f but on g, and (ii) the augmenting path procedure used to make the initial pairing perfect performs better on the initial pairing produced by **LRS**. These two remarks will become evident when we analyze the time measurements obtained from the triangulations in the second and third groups.

From Section 3.3, we know that the number, ℓ, of triangles left unpaired by the initial pairing of **LRS** is always 4 if g is zero; otherwise, we get $\ell \leq 56 \cdot g - 12$. So, if $g \ll n_f$, then the likelihood that $\ell < k$ increases, where k is the number of triangles left unpaired by the initial pairing of **PS** (and **SG**). This is the case for triangulations Eros, Gargoyle, and Iphigenia (see column (c) of Table 4.3). For these three triangulations, the time to make the initial pairing perfect is smaller for the initial pairing produced by **LRS** (see column (d) of Table 4.3). However, the total execution time of **LRS** is smaller than the one of **PS** for triangulation Iphigenia only (see column (e) of Table 4.3). The reason is that the total execution times for Eros and Gargoyle are dominated by the time to compute the initial pairing, and **PS** computes the initial pairing twice as fast as **LRS** for Eros and Gargoyle. The case for Happy Buddha is intriguing, as $\ell > k$ and yet the time to make the initial pairing perfect is smaller for the initial pairing produced by **LRS**. But, this is a mere consequence of the fact that our implemention of the augmenting path procedure relies on a breadth-first search (BFS) to build *augmenting trees* [21]. So, shorter augmenting paths are found first, and thus the closer the unpaired triangles are the faster the augmenting paths are found. Our measuments simply indicate that the triangles left unpaired by the initial pairing produced by **LRS** are "closer" to each other than those of the initial pairing from **PS**. As a result, **LRS** can still perform better than **PS**, with respect to the time for computing the initial pairing, when $\ell > k$ (as long as $(\ell - k)$ is small enough relative to n_f).

To reinforce our conclusions regarding the performances of both **PS** and **LRS**, we tested these two algorithms against hierarchies of triangulations that favor one algorithm over the other. First, we consider a genus-0 surface triangulation with $4{,}194{,}296$ triangles: triangulation A1 of the second group of triangulations (see Table 4.2(a)). Then, we simplified A1 to generate four more triangulations, A2-A5, of the same surface with a decreasing number of triangles. Since the genus of the surface is zero, the value of ℓ is always equal to 4, while the value of k is much larger, but it gets smaller as n_f decreases. Second, we consider a triangulation of a surface of genus $3{,}500$: triangulation B1 of the third group of triangulations (see Table 4.2(b)). Triangulation B1 has $4{,}222{,}296$ triangles. Then, we simplify B1 to generate four more triangulations, B2-B5, of the same surface with a decreasing number of triangles. Note from Table 4.2(b) that $g \approx 2 \cdot \sqrt{n_f}$ for B1 and $g \approx 10^{-2} \cdot n_f$ for B5. Since g is relatively large compared to n_f, the value of ℓ is greater than k. The results obtained from A1-A5 and B1-B5 are shown in Tables 4.4(i) and 4.4(ii).

As expected, **LRS** ran faster than **PS** on all triangulations A1-A5 of the first group (see column (e) of Table 4.4(i)). However, we notice that the number of triangles left unpaired by the initial pairing produced by **PS** grows as the number of triangles of the triangulation descreases. The reason is that A1 is a triangulation in which almost all vertices have the same degree, but this *connectivity regularity* property is not preserved by the simplification process that generated A2-A5 from A1. Indeed, the more a triangulation is simplified, the more irregular it is likely to get. So, our measurements indicate that the heuristic devised by Puli and Segal is affected by the triangulation connectivity regularity. This is why **LRS** is only 1.08 times faster than **PS** for A1, but 4.69 times faster for A5. We can also notice that the time spent by **LRS** to compute the initial pairing is about twice as long as the one by **LRS** for A1-A5, which is consistent with the fact that the pairing stages of **PS** and **LRS** take $\mathcal{O}(n_f)$ time whenever $g = 0$, but the hidden constant in $\mathcal{O}(n_f)$ for **LRS** is larger.

The scenario for triangulations of the third group is the opposite: **PS** performs far better than **LRS**, as the number ℓ of triangles left unpaired by the initial pairing produced by **LRS** is at least 66 times larger than that of the initial pairing produced by **PS**. Furthermore, ℓ is large enough to cause the total execution time of

(i)

Triang.	Alg.	(a)	(b)	(c)	(d)	(e)
A1	**PS**	5.91165300	99.9968%	136.0	7.19208100	13.10373400
A1	**LRS**	11.98554000	99.9999%	4.0	0.19302800	12.17856800
A2	**PS**	2.78087300	99.9987%	621.2	14.66607000	17.44694300
A2	**LRS**	5.74721000	99.9998%	4.0	0.08340600	5.83061600
A3	**PS**	2.04989300	99.9532%	697.6	12.00453700	14.05443000
A3	**LRS**	4.42929000	99.9997%	4.0	0.06262800	4.49191800
A4	**PS**	1.12364900	99.9124%	732.8	7.17269700	8.29634600
A4	**LRS**	2.38205500	99.9995%	4.0	0.03486000	2.41691500
A5	**PS**	0.46565590	99.7887%	728.4	3.80944600	4.275101900
A5	**LRS**	0.89576500	99.9988%	4.0	0.01407000	0.90983500

(ii)

Triang.	Alg.	(a)	(b)	(c)	(d)	(e)
B1	**PS**	5.85876900	99.9944%	238.0	14.53781100	20.39658000
B1	**LRS**	14.41705000	99.0842%	38,668.0	1,380.94000000	1,395.35705000
B2	**PS**	2.71100000	99.9727%	547.8	9.80479000	12.51579000
B2	**LRS**	7.71557000	98.1895%	36,282.0	690.56300000	698.27857000
B3	**PS**	2.04157000	99.9710%	438.8	7.20557700	9.24714700
B3	**LRS**	5.90831000	97.6365%	35,712.0	541.48100000	547.38931000
B4	**PS**	1.14761700	99.9476%	447.4	4.03661000	5.18422700
B4	**LRS**	3.54605400	95.9082%	34,932.0	350.78800000	354.33405400
B5	**PS**	0.48588230	99.8876%	405.4	2.06146300	2.54734530
B5	**LRS**	1.85803100	90.4856%	34,322.0	208.72100000	210.57903100

Table 4.4: Measurements obtained from executing **PS** and **LRS** on triangulations (i) A1-A5, and (ii) B1-B5. Columns (a)-(e) as in Table 4.3. All time measures shown in seconds. For **PS**, displayed values are averages over ten executions of **PS**.

LRS to be completely dominated by the time to make the initial pairing perfect. We also notice that the ratio $t_{\mathbf{LRS}}/t_{\mathbf{PS}}$ gets larger as the triangulations gets smaller, where $t_{\mathbf{LRS}}$ and $t_{\mathbf{PS}}$ are the times spent by **LRS** and **PS**, respectively, to compute their initial pairings for B1-B5. This is also consistent with the fact that $t_{\mathbf{LRS}}$ is in $\Omega(gn_f)$, as $g > 0$ (see [36]), whilst $t_{\mathbf{PS}}$ is still linear in n_f, and $g \approx 2 \cdot \sqrt{n_f}$ for B1 and $g \approx 10^{-2} \cdot n_f$ for B5. We also obtained time measures for **DS** and **SG** on A1-A5 and B1-B5 (although they have been omitted from Table 4.4), and **DS** performed better than **LRS** on B1-B5 (**DS** was 1.12 and 3.7 times faster on B1 and B5, respectively). But, **LRS** still did better than **SG** on B1-B5, as ℓ was much smaller than the value of k for **SG**.

5 Discussion and Final remarks

We present and analyze the results of an experimental comparison of four algorithms for converting a triangulation of a closed surface into a quadrangulation: a simple greedy algorithm and an improved greedy algorithm [13, 20] (Section 3.1), a graph-based algorithm [23] (Section 3.2), and a new algorithm (Section 3.3). All algorithms are based on the same two-stage process of first pairing triangles and then removing the common edges of each pair. Both greedy algorithms carry out the first (pairing) stage in two steps, while the graph-based algorithm runs in a single step. In turn, the pairing stage of our new algorithm consists of three steps, and the second step (expansion) may flip edges of the input triangulation, $\mathcal{T}$. For the two greedy algorithms, the first step of the two-step pairing stage runs in linear time in n_f, while the second step runs

in $\mathcal{O}(kn_f)$ amortized time, where n_f is the number of triangles of $\mathcal{T}$ and k is the number of triangles left unpaired by the first step. For the new algorithm, the combination of the first two steps of the three-step pairing stage runs in $\mathcal{O}(gn_f + g^2)$ (resp. $\mathcal{O}(n_f)$) amortized time if the genus g of $\mathcal{T}$ is positive (resp. 0), while the third step runs in $\mathcal{O}(\ell n_f)$ amortized time, where ℓ is the number of triangles left unpaired by the second step. Finally, the pairing stage of the graph-based algorithm takes $\mathcal{O}(n_f \lg^2 n_f)$ amortized time.

Our experimental results allow us to analyze the influence of the parameters n_f, g, k and ℓ. More importantly, they allow us to characterize why and when one algorithm can perform better than the others. In particular, we observed that $k = \alpha \cdot n_f$, with $\alpha \in [0.12, 0.13] \subset \mathbb{R}$ for the simple greedy algorithm, and $\alpha \in [10^{-5}, 0.0021] \subset \mathbb{R}$ for the improved greedy algorithm. In turn, ℓ does not depend on n_f, but on g, and $\ell \leq 56 \cdot g - 12$. We verified that if n_f is sufficiently large and $g \ll n_f$, then the new algorithm performs better than the other three, as $|\ell - k|$ is small relative to n_f. However, if g is large (relative to n_f), then the improved greedy algorithm is a far better option than the other three. In addition, in this particular scenario, the new algorithm can do worse than the graph-based algorithm, but still better than the simple greedy algorithm (which performed extremely poorly in all test cases). Despite the facts that (1) the improved greedy algorithm is a far better option than the simple one, and (2) the effectiveness of some heuristics for computing partial matchings on general graphs had already been mentioned in [15], the mesh processing community appears to be unaware of the improved greedy algorithm. In fact, it is not even mentioned among the possible solutions for the tri-quad conversion problem in a recent survey [11], while the simple greedy heuristic is cited and has been implemented in a popular, freely available meshing tool[3].

We noticed that the heuristic devised by Puli and Segal seems to be affected by the connectivity regularity of the input triangulation: the more irregular the triangulation, the less effective the heuristic. Our measurements also indicated that the triangles left unpaired by the initial pairing produced by the new algorithm are closer to each other than the ones of the initial pairing produced by the improved greedy algorithm. As a result, the augmenting path procedure used to make the initial pairing perfect ran faster on the initial pairings produced by the new algorithm. The fact that the new algorithm generates "closer" unpaired triangles is an intrinsic feature of the algorithm given in Section 3.3.1. We intend to further exploit this feature in order to produce a faster algorithm for finding augmenting paths, which we hope to combine with the expansion step in Section 3.3.1, generating a perfect matching in a single step such as the algorithm in [23]. The new algorithm has two main limitations, though. First, it may flip edges, which means that the output quadrangulation is not a pairing of the triangles of the input triangulation, but of a triangulation with the same vertex set and and distinct edge and face sets. Consequently, the new algorithm may not be suitable for applications whose final goal is to pair up triangles only (e.g., see [13, 15, 16, 17]) as opposed to generating a quadrangulation. Second, edge flips may decrease the "shape" quality of the resulting quadrangulation. We did not take shape quality criteria into account in our study, as the goal of conversion algorithms is not to obtain a good quality quadrangulation, but to obtain a quadrangulation as fast as possible. Nevertheless, it may be useful to incorporate an inexpensive quality criterion into the conversion process [20, 14].

Figures 5.11 and 5.12 show the quadrilateral meshes generated from the triangulation `Botijo` by the two greedy algorithms **SG** and **PS**, respectively. Figures 5.13 and 5.14 show the quadrilateral meshes generated by algorithms **DS** and **LRS**, respectively. While the meshes in Figures 5.11, 5.12, and 5.13 look essentially the same, some of the edge flips are visible in Figure 5.14. Our algorithm flips 3392 edges in the input mesh of 60024 edges. For the meshes listed in Table 4.3, our algorithm flips between 5 and 10 percent of the edges. We believe that some additional refinement in the expansion stage to handle vertices of low degree explicitly will lead to further reduction in the number of edge flips without increasing the run-time complexity of the algorithm.

[3] `http://meshlab.sourceforge.net/`

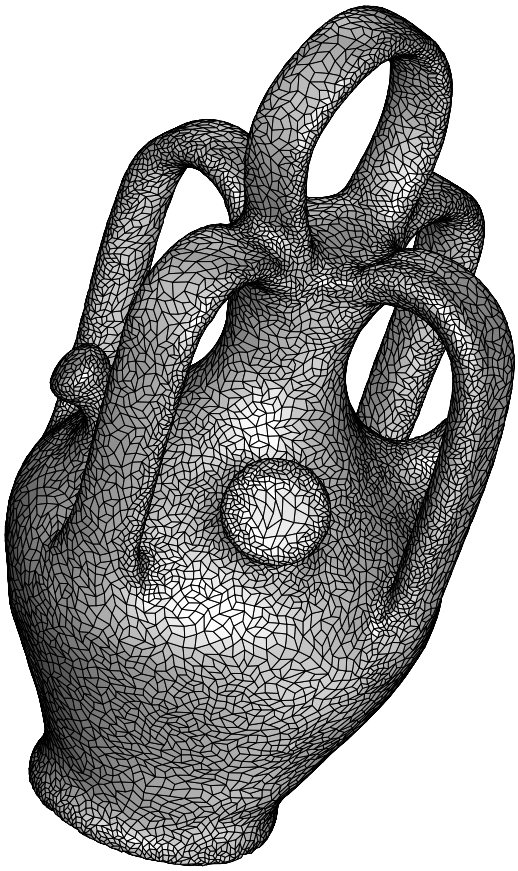

Figure 5.11: Quad mesh generated by **SG**

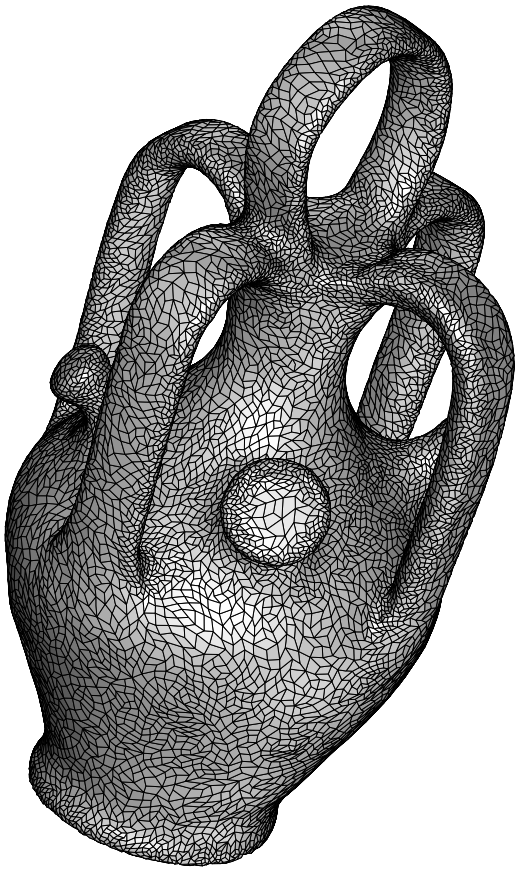

Figure 5.12: Quad mesh generated by **PS**

Acknowledgments. All triangulations in Table 4.1 were obtained from publicly available triangle mesh repositories. Namely, triangulations `Botijo`, `Eros`, `Filigree`, and `Gargoyle` were downloaded from the Aim@Shape Repository, triangulation `Happy Buddha` was downloaded from the Large Geometric Models Archive, and triangulation `Iphigenia` was downloaded from the website of the book [1]. We also would like to thank Renato Werneck (Microsoft Research, USA) for sending us the source code of his implementation of data structures for dynamic trees (see [41]), which greatly helped us implement Diks and Stanczyk's graph matching algorithm [23]. Finally, during the development of this work, Suneeta Ramaswami was partially supported by NSF grant CCF-0830589, and Marcelo Siqueira was partially supported by CNPq grants 305845/2012-8 and 486951/2012-0.

References

[1] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygonal Mesh Processing*. A K Peters, Ltd., 2010. 1, 5

[2] Jean-Daniel Boissonnat and Steve Oudot. Provably good sampling and meshing of surfaces. *Graphical Models*, 67(5):405–451, 2005. 1

[3] Siu-Wing Cheng, Tamal K. Dey, Edgar A. Ramos, and Tathagata Ray. Sampling and meshing a surface with guaranteed topology and geometry. *SIAM Journal on Computing*, 37(4):1199–1227, 2007. 1

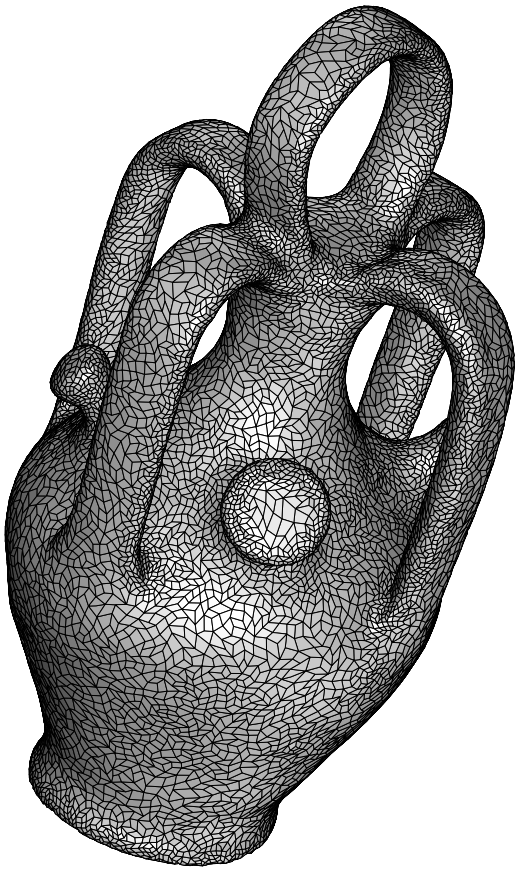

Figure 5.13: Quad mesh generated by **DS**

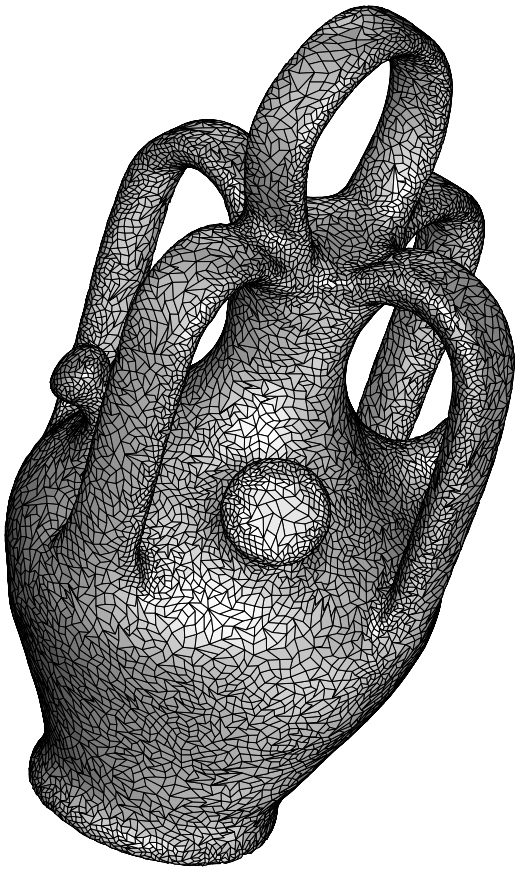

Figure 5.14: Quad mesh generated by **LRS**

[4] Lis Custódio, Tiago Etiene, Sinésio Pesco, and Cláudio T. Silva. Practical considerations on marching cubes 33 topological correctness. *Computers & Graphics*, 37(7):840–850, 2013. 1

[5] Nina Amenta, Sunghee Choi, Tamal K. Dey, and Naveen Leekha. A simple algorithm for homeomorphic surface reconstruction. In *Proceedings of the 16th ACM Symposium on Computational Geometry*, pages 213–222, June 12-14 2000. 1

[6] Giorgio Marcias, Nico Pietroni, Daniele Panozzo, Enrico Puppo, and Olga Sorkine-Hornung. Animation-aware quadrangulation. *Computer Graphics Forum*, 32(5):167–175, 2013. 1

[7] Gerald Farin. *Curves and surfaces for CAGD: a practical guide*. Morgan-Kaufmann, fifth edition, 2002. 1

[8] Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. Texture mapping progressive meshes. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 409–416, New York, NY, USA, 2001. ACM. 1

[9] David Eppstein, Michael T. Goodrich, Ethan Kim, and Rasmus Tamstorf. Motorcycle graphs: Canonical quad mesh partitioning. *Computer Graphics Forum*, 27(5):1477–1486, 2008. 1

[10] E. D'Azevedo. Are bilinear quadrilaterals better than linear triangles? *SIAM Journal on Scientific Computing*, 22(1):198–217, 2000. 1

[11] David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Cláudio Silva, Marco Tarini, and Denis Zorin. Quad-mesh generation and processing: A survey. *Computer Graphics Forum*, 32(6):51–76, 2013. 1, 1, 5

[12] Chaman Singh Verma and Tim Tautges. Jaal: Engineering a high quality all-quadrilateral mesh generator. In William Roshan Quadros, editor, *Proceedings of the 20th International Meshing Roundtable*, pages 511–530. Springer, 2012. 1, 1

[13] Kari Pulli and Mark E. Segal. Fast rendering of subdivision surfaces. In *Proceedings of the Eurographics Workshop on Rendering Techniques*, pages 61–70, August 27-31 1996. 1, 1, 3.1, 5

[14] Marco Tarini, Nico Pietroni, Paolo Cignoni, Daniele Panozzo, and Enrico Puppo. Practical quad mesh simplification. *Computer Graphics Forum*, 29(2):407–418, 2010. 1, 1, 5

[15] Meenakshisundaram Gopi and David Eppstein. Single-strip triangulation of manifolds with arbitrary topology. *Computer Graphics Forum*, 23(3):371–380, 2004. 1, 1, 5

[16] Pedro V. Sander, Diego Nehab, Eden Chlamtac, and Hugues Hoppe. Efficient traversal of mesh edges using adjacency primitives. *ACM Transactions on Graphics*, 27(5):144:1–144:9, December 2008. 1, 5

[17] Topraj Gurung, Daniel E. Laney, Peter Lindstrom, and Jarek Rossignac. Squad: Compact representation for triangle meshes. *Computer Graphics Forum*, 30(2):355–364, 2011. 1, 1, 5

[18] S. Ramaswami, P. Ramos, and G. Toussaint. Converting triangulations to quadrangulations. *Computational Geometry: Theory and Applications*, 9:257–276, 1998. 1

[19] S. Ramaswami, M. Siqueira, T. Sundaram, J. Gallier, and J. Gee. Constrained quadrilateral meshes of bounded size. *International Journal of Computational Geometry and Applications*, 15(1):55–98, 2005. Invited to special issue of selected papers from the 12th IMR. 1

[20] Luiz Velho. Quadrilateral meshing using 4-8 clustering. In *Proceedings of the Symposium on Mesh Generation and Self-Adaptivity (CILANCE 2000)*, pages 61–64, 2000. 1, 1, 3.1, 5

[21] Robert E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, USA, 1983. 1, 3.1, 3.3, 3.3, 4, 4

[22] Harold N. Gabow and Robert E. Tarjan. A linear-time algorithm for a special case of disjoint union. *Journal of Computer and Systems Sciences*, 30(2):209–221, April 1985. 1, 3.1, 3.4

[23] Krzysztof Diks and Piotr Stanczyk. Perfect matching for biconnected cubic graphs in $\mathcal{O}(n \lg^2 n)$ time. In *Proceedings of the 36th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'10)*, pages 321–333, Germany, January 23-29 2010. Springer-Verlag. 1, 1, 3.2, 5, 5, A

[24] Therese C. Biedl, Prosenjit Bose, Erik D. Demaine, and Anna Lubiw. Efficient algorithms for petersen's matching theorem. *Journal of Algorithms*, 38(1):110–134, 2001. 1, 3.2, A

[25] William F. Mitchell. Adaptive refinement for arbitrary finite-element spaces with hierarchical bases. *Journal of Computational and Applied Mathematics*, 36(1):65–78, 1991. Special Issue on Adaptive Methods. 1

[26] Jonathan L. Gross and Thomas W. Tucker. *Topological graph theory*. Dover Publications, Inc., Mineola, NY, USA, 2001. 2, 2

[27] Jean Gallier and Dianna Xu. *A Guide to the Classification Theorem for Compact Surfaces*. Springer-Verlag, 2013. 2, 2

[28] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985. 2, 2, 2.1

[29] Julius Petersen. Die theorie der regulären graphs. *Acta Mathematica*, 15(1):193–220, December 1891. 2.1

[30] Orrin Frink. A proof of petersen's theorem. *Annals of Mathematics*, 27(4):491–493, June 1926. 3.2

[31] Denes König. *The theory of finite and infinite graphs*. Birkhäuser, Boston, MA, USA, 1990. 3.2, A

[32] Silvio Micali and Vijay V. Vazirani. An $\mathcal{O}(|E| \cdot \sqrt{|V|})$ algorithm for finding maximum matchings in general graphs. In *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science*, pages 17–27, Washington, DC, USA, 1980. IEEE Computer Society. 3.2

[33] David W. Barnette and Allan L. Edelson. All 2-manifolds have finitely many minimal triangulations. *Israel Journal of Mathematics*, 67(1):123–128, 1989. 3.3

[34] Serge Lavrenchenko. Irreducible triangulations of the torus. *Journal of Soviet Mathematics*, 51(5):2537–2543, 1990. 3.3

[35] Gwenaël Joret and David R. Wood. Irreducible triangulations are small. *Journal of Combinatorial Theory, Series B*, 100(5):446–455, 1891. 3.3, 3.1

[36] Suneeta Ramaswami and Marcelo Siqueira. A fast algorithm for computing irreducible triangulations of closed surfaces in $\mathbb{E}^d$. *CoRR*, (*submitted*), http://mat.ufrn.br/∼mfsiqueira/Marcelo_Siqueiras_Web_Spot/Geometry_files/ir-arxiv.pdf 2014. 3.3, 3.4, 3.3, 4

[37] Haijo Schipper. Generating triangulations of 2-manifolds. In Hanspeter Bieri and Hartmut Noltemeier, editors, *Computational Geometry: Methods, Algorithms and Applications*, volume 553 of *Lecture Notes in Computer Science*, pages 237–248. Springer, 1991. 3.1

[38] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, third edition, 2008. 4

[39] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983. 4

[40] Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge and biconnectivity. *Journal of the ACM*, 48(4):723–760, 2001. 4

[41] Robert E. Tarjan and Renato F. Werneck. Dynamic trees in practice. *Journal of Experimental Algorithmics*, 14:5:4.5–5:4.23, January 2010. 5

A A sketch of Frink's graph reduction strategy

Let H be a connected, 3-regular and bridgeless graph, and refer to Figure A.15. Suppose that H has a *simple* edge, $e = \{u, v\}$. Let x_1 and x_2 be the other two nodes of H connected to u, and let x_3 and x_4 be the other two nodes of H connected to v. Note that nodes x_i, with $i = 1, 2, 3, 4$, do not have to be pairwise distinct. Frink defined two new graphs, H_1 and H_2, from H by applying two *graph reduction* operations to H. Each (graph) reduction consists of removing two nodes and five edges from H, and then adding two

new edges to the resulting graph; that is, nodes u and v are removed from H along with their incident edges: $\{u, v\}$, $\{u, x_1\}$, $\{u, x_2\}$, $\{v, x_3\}$, and $\{v, x_4\}$. Denote the resulting graph by H'. Then, edges $\{x_1, x_3\}$ and $\{x_2, x_4\}$ (resp. $\{x_1, x_4\}$ and $\{x_2, x_3\}$) are added to H' to define H_1 (resp. H_2). By construction, both H_1 and H_2 are 3-regular graphs. Frink proved that at least one of them is also connected and bridgeless [31], and hence has a perfect matching by Theorem 2.1. Without loss of generality, assume that H_1 is such a graph. Then, from Theorem 2.1, there exists a perfect matching on H_1.

Let M_1 be a perfect matching on H_1. Then, we can distinguish four cases with respect to M_1: (i) $\{x_1, x_3\}$, $\{x_2, x_4\} \notin M_1$, (ii) $\{x_1, x_3\} \in M_1$, $\{x_2, x_4\} \notin M_1$, (iii) $\{x_1, x_3\} \notin M_1$, $\{x_2, x_4\} \in M_1$, and (iv) $\{x_1, x_3\}, \{x_2, x_4\} \in M_1$. If either (i), (ii), or (iii) holds, we can easily build a perfect matching, M, on H from M_1. Indeed, if (i) holds then we let $M = M_1 \cup \{\{u, v\}\}$; else if (ii) holds then we let $M = M_1 \cup \{\{u, x_1\}, \{v, x_3\}\}$; else if (iii) holds then we let $M = M_1 \cup \{\{u, x_2\}, \{v, x_4\}\}$; else we have case (iv). Frink proved that every edge in M_1 belongs to an alternating cycle on H_1 with respect to M_1. By reversing this cycle (i.e., by switching the roles of matching and non-matching edges along the cycle), we obtain another perfect matching, M_1', on H_1 such that at most one of $\{x_1, x_3\}$ and $\{x_2, x_4\}$ is in M_1'. Consequently, with respect to M_1', one of cases (i), (ii), and (iii) must hold, and M can be obtained from M_1' by applying the valid case. Note that $e = \{u, v\}$ was assumed to be simple. If H has no simple edge, then since H is connected and 3-regular, H consists of exactly 2 vertices and 3 parallel edges connecting the 2 vertices. Thus, a perfect matching on H is obtaining by picking one of the 3 edges to be the matching one.

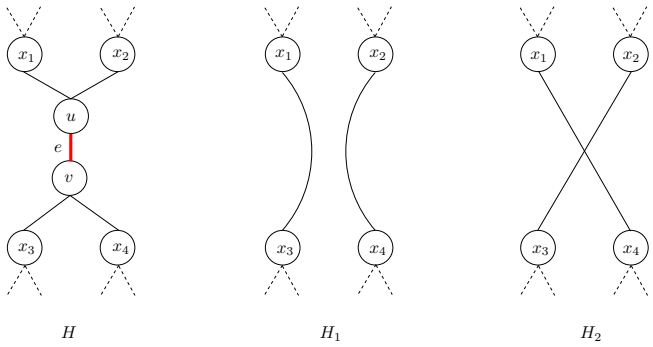

Figure A.15: The two graph reductions used by Frink's proof of Theorem 2.1.

Using the ideas decribed above, Biedl, Bose, Demaine, and Lubiw devised a recursive algorithm for computing M [24]. If the input graph H is the trivial one (i.e., the graph with two vertices and three parallel edges), then one of the three edges, say e, is chosen to be the matching one and the algorithm returns $M = \{e\}$. Otherwise, three main steps take place. The first step determines which of H_1 and H_2 is a connected, 3-regular, and bridgeless graph. This is done in $\mathcal{O}(\lg^4 n)$ amortized time using the procedure suggested by the authors, where n is the number of nodes of H. The second step *recursively* computes a perfect matching M' on the graph found by the first step (i.e., either H_1 or H_2). The third step computes M from the matching M' returned by the second step. The third step takes *constant* time if case (iv) of Frink's proof never occurs. Indeed, a perfect matching on H can be found in $\mathcal{O}(n \lg^4 n)$ amortized time if case (iv) never occurs.

One of the main contributions from [24] is a constructive proof for a slight variation of Petersen's theorem that states that *every 3-regular, bridgeless graph H admits a perfect matching M such that M does not contain a given edge f of H*. Based on this proof, the algorithm fixes an edge f of H to be a non-matching edge, and then chooses the reduction edge $e = \{u, v\}$ (in the second step) to be an edge that shares a vertex with f. After the reduction, edge f becomes one of two edges added to H' to obtain either H_1 or H_2 (i.e., f becomes one of $\{x_1, x_3\}$ and $\{x_2, x_4\}$ if H_1 is found by the first step, and one of $\{x_1, x_4\}$ and $\{x_2, x_3\}$ otherwise). As a result, the algorithm can always avoid case (iv) and find M in $\mathcal{O}(n \lg^4 n)$ amortized time.

More recently, Diks and Stanczyk proposed a modification in the first step of the algorithm that allows us to determine which of H_1 or H_2 is connected and bridgeless in $\mathcal{O}(\lg^2 n)$ amortized time [23]. Thus, the resulting algorithm can find M in $\mathcal{O}(n \lg^2 n)$ amortized time, which is, *to the best of our knowledge*, the best known upper bound for computing a perfect matching on a 3-regular and bridgeless graph. We implemented the algorithm by Diks and Stanczyk and applied it to the dual graph $G^\star$ of the graph G of the input triangulation $\mathcal{T}$ in order to obtain a perfect pairing of the triangles of $\mathcal{T}$ (and thus obtain a quadrangulation of $\mathcal{S}$) in $\mathcal{O}(n_f \lg^2 n_f)$ amortized time.