A fast algorithm for computing irreducible triangulations of closed surfaces in $\mathbb{E}^d$

Suneeta Ramaswami[a,1], Marcelo Siqueira[b,2,*]

[a] *Rutgers University, Department of Computer Science, 227 Penn Street, BSB Room 321, Camden, NJ 08102-1656, USA*

[b] *Universidade Federal do Rio Grande do Norte, Departamento de Matemática, CCET Sala 70, Natal, RN 59078-970, Brazil*

Abstract

We give a fast algorithm for computing an irreducible triangulation $\mathcal{T}'$ of an oriented, connected, boundaryless, and compact surface $\mathcal{S}$ in $\mathbb{E}^d$ from any given triangulation $\mathcal{T}$ of $\mathcal{S}$. If the genus g of $\mathcal{S}$ is positive, then our algorithm takes $\mathcal{O}(g^2+gn)$ time to obtain $\mathcal{T}'$, where n is the number of triangles of $\mathcal{T}$. Otherwise, $\mathcal{T}'$ is obtained in linear time in n. While the latter upper bound is optimal, the former upper bound improves upon the currently best known upper bound by a $\lg n/g$ factor. In both cases, the memory space required by our algorithm is in $\Theta(n)$.

Keywords: Irreducible triangulations, link condition, edge contractions
2010 MSC: 05C10, 68U05

1. Introduction

Let $\mathcal{S}$ be a compact surface with empty boundary. A triangulation of $\mathcal{S}$ can be viewed as a "polyhedron" on $\mathcal{S}$ such that each face is a triangle with three distinct vertices and the intersection of any two distinct triangles is either empty, a single vertex, or a single edge (including its two vertices). A classical result from the 1920s by Tibor Radó asserts that every compact surface with empty boundary (i.e., usually called a *closed surface*) admits a triangulation [1]. Let e be any edge of a triangulation $\mathcal{T}$ of $\mathcal{S}$. The *contraction* of e in $\mathcal{T}$ consists of contracting e to a single vertex and collapsing each of the two triangles meeting e into a single edge (see Figure 3). If the result of contracting e in $\mathcal{T}$ is still a triangulation of $\mathcal{S}$, then e is said to be *contractible*; else it is *non-contractible*. A triangulation $\mathcal{T}$ of $\mathcal{S}$ is said to be *irreducible* if and only if every edge of $\mathcal{T}$

*Corresponding author
Email addresses: rsuneeta@camden.rutgers.edu (Suneeta Ramaswami), mfsiqueira@mat.ufrn.br (Marcelo Siqueira)

[1] Partially supported by NSF grant CCF-0830589
[2] Partially supported by CNPq Grant 305845/2012-8 and CNPq Grant 486951/2012-0

is non-contractible. Barnette and Edelson [2] showed that all closed surfaces have finitely many irreducible surfaces. More recently, Boulch, de Verdière, and Nakamoto [3] showed the same result for compact surfaces with a nonempty boundary.

Irreducible triangulations have proved to be an important tool for tackling problems in combinatorial topology and discrete and computational geometry. The reasons are two-fold. First, all irreducible triangulations of any given compact surface form a "basis" for all triangulations of the same surface. Indeed, every triangulation of the surface can be obtained from at least one of its irreducible triangulations by a sequence of *vertex splittings* [4, 5], where the vertex splitting operation is the inverse of the edge contraction operation (see Figure 3). Second, some problems on triangulations can be solved by considering irreducible triangulations only. In particular, irreducible triangulations have been used for proving the existence of geometric realizations (in some $\mathbb{E}^d$) of triangulations of certain surfaces, where $\mathbb{E}^d$ is the d-dimensional Euclidean space [6, 7], for studying properties of diagonal flips on surface triangulations [8, 9, 10, 11, 12], for characterizing the structure of flexible triangulations of the projective plane [13], and for finding lower and upper bounds for the maximum number of cliques in an n-vertex graph embeddable in a given surface [14]. Irreducible triangulations are also "small", as their number of vertices is at most linear in the genus of the surface [15, 3]. However, the number of vertices of all irreducible triangulations of the same surface may vary, while any irreducible triangulation of smallest size (known as *minimal*) has $\Theta(\sqrt{g})$ vertices if the genus g of the surface is positive [16].

The sphere has a unique irreducible triangulation, which is the boundary of a tetrahedron [17]. The torus has exactly 21 irreducible triangulations, whose number of vertices varies from 7 to 10 [18]. The projective plane has only two irreducible triangulations, one with 6 vertices and the other with 7 vertices [19]. The Klein bottle has exactly 29 irreducible triangulations with number of vertices ranging from 8 to 11 [20]. Sulanke devised and implemented an algorithm for generating all irreducible triangulations of compact surfaces with empty boundary [5]. Using this algorithm, Sulanke rediscovered the aforementioned irreducible triangulations and generated the complete sets of irreducible triangulations of the double torus, the triple cross surface, and the quadruple cross surface.

The idea behind Sulanke's algorithm is to generate irreducible triangulations of a surface by modifying the irreducible triangulations of other surfaces of smaller Euler genuses (the Euler genus of a surface equals the usual genus for nonorientable surfaces, and equals twice the usual genus for orientable surfaces). The modifications include vertex splittings and the addition of handles, crosscaps, and crosshandles. Unfortunately, the lack of a known upper bound on the number of vertex splittings required in the intermediate stages of the algorithm prevented Sulanke from establishing a termination criterion for all surfaces. Furthermore, his algorithm is impractical for surfaces with Euler genus ≥ 5, as his implementation could take centuries to generate the quintuple cross surface on a cluster of computers with an average CPU speed of 2GHz [5]. To the best of our knowledge, no similar algorithm for compact surfaces with a nonempty boundary exists.

1.1. Our contribution

Here, we give an algorithm for a problem closely related to the one described above: *given any triangulation $\mathcal{T}$ of a compact surface $\mathcal{S}$ with empty boundary, find one irreducible triangulation, $\mathcal{T}'$, of $\mathcal{S}$ from $\mathcal{T}$.* In particular, if the genus g of $\mathcal{S}$ is positive, then we show that $\mathcal{T}'$ can be computed in $\mathcal{O}(gn + g^2)$ time, where n is the number of triangles of $\mathcal{T}$. Otherwise, $\mathcal{T}'$ can be computed in $\mathcal{O}(n)$ time, which is optimal. In either case, the space requirement is in $\Theta(n)$. To the best of our knowledge, the previously best known (time) upper bound is $\mathcal{O}(n \lg n + g \lg n + g^4)$ for the algorithm given by Schipper in [4]. In his complexity analysis, Schipper assumed that g is a constant depending only on $\mathcal{S}$, and thus stated the upper bound as $\mathcal{O}(n \lg n)$. While it is true that g is an intrinsic feature of $\mathcal{S}$, we may have $m \in \Theta(\sqrt{g})$ [16], where m is the number of vertices of $\mathcal{T}$, which implies that $n \in \Theta(g)$ (see Section 2). Thus, we state the time bounds in terms of both g and n. Since our algorithm can more efficiently generate *one* irreducible triangulation from any given triangulation of $\mathcal{S}$, it can potentially be used as a "black-box" by a fast and alternative method (to that of Sulanke's) for generating *all* irreducible triangulations of any given surface.

1.2. Application to the triquad conversion problem

The algorithm for computing irreducible triangulations described here was recently incorporated into an innovative and efficient solution [21] to the problem of converting a triangulation $\mathcal{T}$ of a closed surface into a quadrangulation with the same set of vertices as $\mathcal{T}$ (known as the *triquad conversion* problem [22]). This new solution takes $\mathcal{O}(gn + g^2)$ time, where n is the number of triangles of $\mathcal{T}$, to produce the quadrangulation if the genus g of the surface is positive. Otherwise, it takes linear time in n. The solution improves upon the approach of computing a perfect matching on the dual graph of $\mathcal{T}$, for which the best known upper bound is $\mathcal{O}(n \lg^2 n)$ amortized time [23]. In [21], the new solution is experimentally compared with two simple greedy algorithms [24, 25] and the approach based on the algorithm in [23]. It outperforms the approaches in [24, 25, 23] whenever n is sufficiently large and $g \ll n$, which is typically the case for triangulations used in computer graphics and engineering applications. We hope that the solution in [21] to the triquad conversion problem increases the practical interest for algorithms to compute irreducible triangulations of surfaces.

1.3. Organization

The remainder of this paper is organized as follows: Section 2 introduces the notation, terminology and basic definitions used throughout the paper. Section 3 reviews prior work on algorithms for computing irreducible triangulations and related algorithms (e.g., algorithms for mesh simplification). Section 4 describes our proposed algorithm in detail, and analyzes its time and space complexities. Section 5 presents an experimental comparison of the implementation of three algorithms for computing irreducible triangulations: ours; a randomized, brute-force algorithm; and the one proposed by Schipper in [4]. Finally, section 6 summarizes our main contributions, and discusses future research directions.

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 disk, $\mathbb{B}^2 = \{(x,y) \in \mathbb{E}^2 \mid x^2+y^2 < 1\}$, 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. Here, we restrict our attention to the class consisting of all *oriented, connected, and compact surfaces in* $\mathbb{E}^d$, and we use the term "surface" to designate a member of this class (unless stated otherwise).

The notions of triangle mesh and quadrilateral mesh of a surface are synonyms for the well-known terms triangulation and quadrangulation of a surface, respectively, in topological graph theory and algebraic topology [26, 1]. Informally, 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. To formalize these ideas, we rely on the notions of subdivision of a surface, as nicely stated by Guibas and Stolfi [27], and of a graph embedded on a surface.

Definition 1. *A* subdivision *of a surface $\mathcal{S}$ is a partition, $\mathcal{P}$, of $\mathcal{S}$ into three finite collections of disjoint subsets: the* 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, and satisfy the following conditions:*

(S1) every vertex is a point,

(S2) every edge is an open arc,

(S3) every face is an open disk, and

(S4) the boundary of every face is a closed path of edges and vertices.

Condition (S4) is based on the notion of "closed path" on a surface, which can be formalized as follows: let $\mathbb{S}^1 = \{(x,y) \in \mathbb{E}^2 \mid x^2+y^2 = 1\}$ be the circumference of a circle of unit radius centered at the origin. We define a *simple path* in $\mathbb{S}^1$ as a partition of $\mathbb{S}^1$ into a finite sequence of isolated points and open arcs. Then, condition (S4) is equivalent to the following (refer to Figure 1): for every face τ in $\mathcal{P}$, there exists a simple path, π, in $\mathbb{S}^1$ and a continuous mapping, $g_\tau : \overline{\mathbb{B}}^2 \to \overline{\tau}$, where $\overline{\mathbb{B}}^2$ and $\overline{\tau}$ are the closures of $\mathbb{B}^2$ and

τ, such that g_τ (i) maps $\mathbb{B}^2$ homeomorphically onto τ, (ii) maps each open arc of π homeomorphically onto an edge of $\mathcal{P}$, and (iii) maps each isolated point of π to a vertex of $\mathcal{P}$. So, condition (S4) implies that the images of the isolated points and edges of π under g_τ, taken in the order in which they occur around $\mathbb{S}^1$, constitute a closed, connected path of vertices and edges of $\mathcal{P}$, whose union is the boundary of τ. Note that this path need not be simple, as g_τ may take two or more distinct points or open arcs of π to the same vertex or edge of $\mathcal{P}$, respectively.

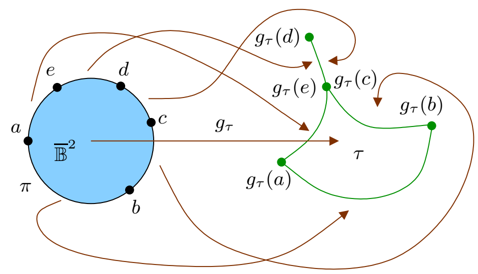

Figure 1: Illustration of condition (S4) of Definition 1.

Since an open disk cannot be entirely covered by a finite number of vertices and edges, every vertex and edge in $\mathcal{P}$ must be incident on some face of $\mathcal{P}$. In fact, using condition (S4), it is possible to show that (i) every edge of $\mathcal{P}$ is entirely contained in the boundary of some face of $\mathcal{P}$, (ii) every vertex is incident on an edge, and (iii) every edge of $\mathcal{P}$ is incident on two (not necessarily distinct) vertices of $\mathcal{P}$. These vertices are called the *endpoints* of the edge. If they are the same, then the edge is a *loop*, and its closure is homeomorphic to the circumference of a circle of unit radius, $\mathcal{S}^1 = \{(x, y) \in \mathbb{E}^2 \mid x^2 + y^2 = 1\}$.

We can define an undirected graph, G, and a one-to-one function, $\imath : G \to \mathcal{S}$ from the collection of all vertices and edges of $\mathcal{P}$. 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 vertices in V incident on edge e in E, then $i(v)$ and $i(u)$ are the two vertices of $V_{\mathcal{P}}(\mathcal{S})$ incident on $\imath(e)$ in $E_{\mathcal{P}}(\mathcal{S})$. Note that $i(G)^c = \mathcal{S} - \imath(G)$ are the faces of $\mathcal{P}$. We say that G is the *graph of* $\mathcal{P}$, and that i is the *embedding* of G on $\mathcal{S}$. Graph G can be viewed as the combinatorial structure of $\mathcal{P}$, while $\imath$ can be viewed as a "geometric realization" of G on the surface.

Two subdivisions of the same surface are *isomorphic* if and only if their graphs are isomorphic. Since $\mathcal{P}$ is fully described by G and $\imath$, it is customary to call the pair, $(G, \imath)$, the subdivision itself. Triangulations of a given surface are specialized subdivisions that adequately capture the practical notion of triangle meshes:

Definition 2. *A* triangulation *of a surface $\mathcal{S}$ is a subdivision $(G, \imath)$ such that each face of $\imath(G)^c$ is bounded by exactly three distinct vertices (resp. edges) from $\imath(G)$. Furthermore, any two edges of a triangulation have at most one common endpoint, and every vertex of a triangulation must be incident on at least three edges.*

The following lemmas state two important properties of surface triangulations:

Lemma 1. *Every edge of a surface triangulation is incident on exactly two faces.*

Proof. See Section Appendix A. □

Lemma 2 ([27])**.** *If G is the graph of a surface triangulation, then G is connected.*

Recall 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 $\mathcal{S}$ 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 [1]. 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 $n_f \in \Theta(n_v)$. As we see in Section 4, our algorithm requires only the graph of a triangulation as its input. Hence, we simply refer to a given triangulation by $\mathcal{T}$.

Let $\mathcal{T}$ be any triangulation of a surface $\mathcal{S}$. Then, every vertex of $\mathcal{T}$ is incident on at least three edges. Since every edge of $\mathcal{T}$ is incident on exactly two faces of $\mathcal{T}$ (see Lemma 1), every vertex of $\mathcal{T}$ must be incident on at least three faces of $\mathcal{T}$ as well. Furthermore, for every vertex v of $\mathcal{T}$, the edges e and faces τ of $\mathcal{T}$ containing v can be arranged as cyclic sequence $e_1, \tau_1, e_2, \ldots, \tau_{k-1}, e_k, \tau_k$, in the sense that e_j is the common edge of τ_{j-1} and τ_j, for all j, with $2 \leq j \leq k$, and e_1 is the common edge of τ_1 and τ_k, with $k \geq 3$ [1]. The set

$$v \cup e_1 \cup \tau_1 \cup e_2 \cup \cdots \cup \tau_{k-1} \cup e_k \cup \tau_k \subset \mathcal{S}\,,$$

is called the *star of v in $\mathcal{T}$* and is denoted by $st(v, \mathcal{T})$ (see Figure 2). It turns out that $st(v, \mathcal{T})$ is homeomorphic to an open disk. Furthermore, the boundary of $st(v, \mathcal{T})$ in $\mathcal{S}$ consists of the boundary edges of $\tau_1, \ldots, \tau_k$ that are not incident on v, as well as the endpoints of $\tau_1, \ldots, \tau_k$, except for v itself. This point set, denoted by $lk(v, \mathcal{T})$, is a simple, closed curve on $\mathcal{S}$ called the *link of v in $\mathcal{T}$* (see Figure 2). If e is an edge of $\mathcal{T}$, then the *star, $st(e, \mathcal{T})$, of e in $\mathcal{T}$* is the set $e \cup \tau \cup \sigma$, where τ and σ are the two faces of $\mathcal{T}$ incident on e. In turn, the *link, $lk(e, \mathcal{T})$, of e in $\mathcal{T}$* is the set consisting of the two vertices, x and y, such that x is incident on τ and y is incident on σ, but none of x and y is incident on e (see Figure 2).

Let τ and e be a face and an edge of $\mathcal{T}$, respectively. Since every vertex of $\mathcal{T}$ is incident on at least three triangulation edges, i.e., since each vertex of $\mathcal{T}$ has *degree* at least three, if u, v, and w are the boundary vertices of τ, then we can uniquely identify τ by enumerating these vertices. In particular, we denote τ by $[u, v, w]$. Similarly, since no two edges of a triangulation share the same two endpoints, if u and v are the two endpoints of edge e, then we can uniquely identify e by enumerating its two endpoints, and therefore we can denote e by $[u, v]$.

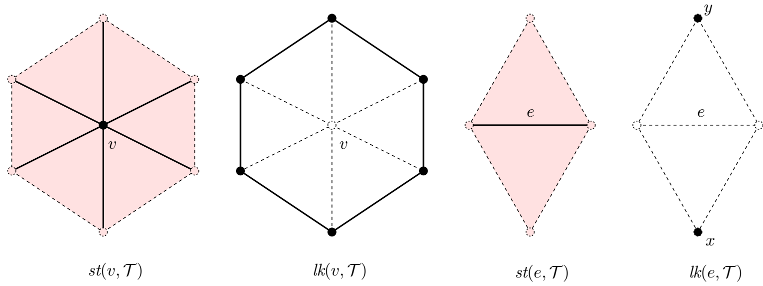

Figure 2: The star and the link of vertex v (left) and the star and the link of edge e (right).

Definition 3. *Let $\mathcal{T}$ be a triangulation of a surface, $\mathcal{S}$, and let $e = [u, v]$ be an edge of $\mathcal{T}$. The* contraction *of e consists of merging u and v into a new vertex w, such that $w \in st(u, \mathcal{T}) \cup st(v, \mathcal{T})$, edges e, $[v, x]$ and $[v, y]$, and faces $[u, v, x]$ and $[u, v, y]$ are removed, edges of the form $[u, p]$ and $[v, q]$ are replaced by $[w, p]$ and $[w, q]$, and faces of the form $[u, r, s]$ and $[v, t, z]$ are replaced by $[w, r, s]$ and $[w, t, z]$, where x and y are the vertices in the link, $lk(e, \mathcal{T})$, of e in $\mathcal{T}$, $p, q \notin \{x, y\}$, $r, s \neq v$, and $t, z \neq u$. If the result is a triangulation of $\mathcal{S}$, then we denote it by $\mathcal{T} - uv$, and call the contraction* topology-preserving *and e a* contractible *edge.*

Figure 3 illustrates the edge contraction operation.

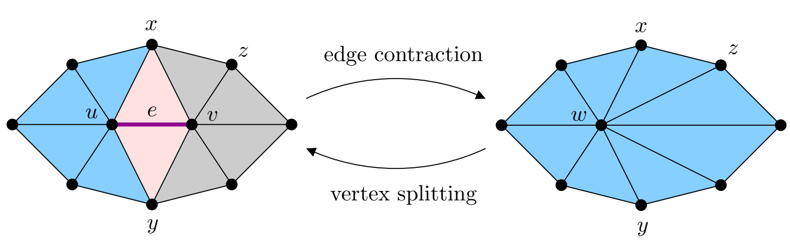

Figure 3: Contraction of edge $[u, v]$ and its inverse: splitting of vertex w.

Dey, Edelsbrunner, Guha, and Nekhayev [28] gave a necessary and sufficient condition, called the *link condition*, to determine whether an edge of a surface triangulation is contractible. Let $e = [u, v]$ be any edge of a surface triangulation, $\mathcal{T}$. Then, edge e is contractible if and only if the following condition holds:

$$lk(e, \mathcal{T}) = lk(u, \mathcal{T}) \cap lk(v, \mathcal{T})\,. \tag{1}$$

In other words, edge e is contractible if and only if the links of u and v in $\mathcal{T}$ have no common vertices, except for the two vertices of the link of e in $\mathcal{T}$. The link condition is purely combinatorial. In fact, we can easily test edge e against the link condition by considering the graph, $G_{\mathcal{T}}$, of $\mathcal{T}$ only. In fact, an equivalent characterization of the link condition, which uses the notion of critical cycle on the graph of a triangulation (defined below), had been given before by Barnette in [19].

If edge e passes the test, then the graph of $\mathcal{T} - uv$ can be easily obtained from $G_{\mathcal{T}}$ by merging its vertices $\imath^{-1}(u)$ and $\imath^{-1}(v)$ into a new vertex w, by removing edges $\imath^{-1}(e)$, $\imath^{-1}([v, x])$ and $\imath^{-1}([v, y])$, and by replacing every edge of the form $\imath^{-1}([u, r])$ and $\imath^{-1}([v, s])$ with with $(w, \imath^{-1}(r))$ and $(w, \imath^{-1}(s))$, respectively, where $\imath : G_{\mathcal{T}} \to \mathcal{S}$ is the embedding of $G_{\mathcal{T}}$ in $\mathcal{S}$, x and y are the vertices in the link, $lk(e, \mathcal{T})$, of e in $\mathcal{T}$, and $r, s \notin \{x, y\}$. We can prove that the resulting graph is embeddable in $\mathcal{S}$, and thus the fact that we can define a triangulation (i.e., $\mathcal{T} - uv$) from the resulting graph does not depend on the surface geometry [1].

A ℓ-*cycle* in a triangulation $\mathcal{T}$ consists of a sequence, $e_1, \ldots, e_\ell$, of ℓ edges of $\mathcal{T}$ such that e_j and e_k share an endpoint in $\mathcal{T}$ if and only if $|j - k| = 1$ or $|j - k| = \ell - 1$, for all $j, k = 1, \ldots, \ell$, with $j \neq k$. Since the two endpoints of a triangulation edge cannot be the same, and since no two edges of a triangulation can have two endpoints in common, a triangulation can only have ℓ-cycles, for $\ell \geq 3$. Furthermore, each cycle can be unambiguously represented by enumerating the vertices of its edges rather than the edges themselves. In particular, if $e_1, \ldots, e_\ell$ define a ℓ-cycle in $\mathcal{T}$, then we denote this cycle by $(v_1, \ldots, v_\ell)$, where v_j is the common vertex of edges e_j and e_{j+1}, for each $j = 1, \ldots, \ell - 1$, and v_ℓ is the common vertex of edges e_1 and e_ℓ. A ℓ-cycle of $\mathcal{T}$ is said to be *critical* if and only if $\ell = 3$ and its (three) edges do not belong to the boundary of the same triangulation face. For instance, cycle (u, v, z) is critical in both (partially shown) triangulations in Figure 4. Observe that edge $[u, v]$ fails the link condition (see Eq. 1) in both triangulations, and hence it is non-contractible in both.

Every genus-0 surface (i.e., a surface homeomorphic to a sphere) admits a triangulation with four vertices, six edges, and four faces. We denote this triangulation by $\mathcal{T}_4$. Figure 5 shows a planar drawing of the graph of $\mathcal{T}_4$. Note that every 3-cycle of $\mathcal{T}_4$ consists of (three) edges that bound a face of $\mathcal{T}_4$. So, no 3-cycle of $\mathcal{T}_4$ is critical. Note also that every edge of $\mathcal{T}_4$ fails the link condition, and thus is a non-contractible edge. So, $\mathcal{T}_4$ is a "minimal" triangulation in the sense that no edge of $\mathcal{T}_4$ is contractible. In fact, $\mathcal{T}_4$ is the only triangulation of a genus-0 surface satisfying this property. For a surface of arbitrary genus, we have:

Theorem 3 (Lemma 3 in [4]). *Let $\mathcal{S}$ be a surface, and let $\mathcal{T}$ be any triangulation* *of $\mathcal{S}$. Then, an edge e of $\mathcal{T}$ is a contractible edge if and only if e does not belong to any critical cycle of $\mathcal{T}$ and $\mathcal{T}$ is not (isomorphic to) the triangulation $\mathcal{T}_4$.*

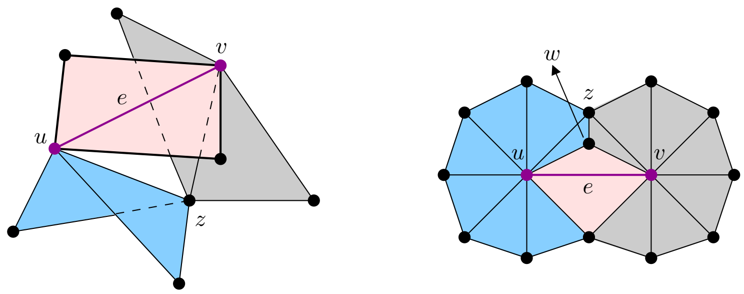

Figure 4: Edges $[u, z]$, $[z, v]$, and $[v, u]$ define critical cycles in both triangulations.

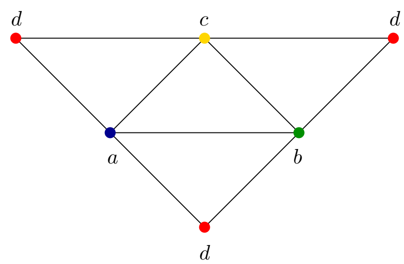

Figure 5: A planar drawing of the graph of $\mathcal{T}_4$. Vertices labeled d are the same vertex.

Definition 4. *Let $\mathcal{T}$ be a triangulation of a surface $\mathcal{S}$, and let v be a vertex of $\mathcal{T}$. If all edges incident on v are non-contractible, then v is called* trapped*; else it is called* loose.

Definition 5. *Let $\mathcal{S}$ be a surface, and let $\mathcal{T}$ be a triangulation of $\mathcal{S}$. If every edge of $\mathcal{T}$ is non-contractible, then $\mathcal{T}$ is called* irreducible*; else it is called* reducible.

A triangulation is irreducible if and only if all of its vertices are trapped. To the best of our knowledge, the best known upper bound on the size of irreducible triangulations was given by Jaret and Wood [29]:

Theorem 4 ([29])**.** *Let $\mathcal{S}$ be a compact surface with empty boundary whose Euler genus, h, is positive, and let $\mathcal{T}$ be any irreducible triangulation of $\mathcal{S}$. Then, the number of vertices, n_v, of $\mathcal{T}$ is such that $n_v \leq 13 \cdot h - 4$. If $\mathcal{S}$ is also orientable, then we have that $g = 2h$, where g is the genus of $\mathcal{S}$, and hence $n_v \leq 26 \cdot g - 4$.*

The largest known irreducible triangulation of an orientable surface of genus g has $\lfloor \frac{17}{2} g \rfloor$ vertices (see [30]).

3. Related work

An irreducible triangulation, $\mathcal{T}'$, of a surface $\mathcal{S}$ can be obtained by applying a sequence of topology-preserving edge contractions to a given triangulation, $\mathcal{T}$, of $\mathcal{S}$. Such a sequence can be found by repeatedly searching for a contractible edge. Whenever a contractible edge is found, it is contracted and the search continues. If no contractible edge is found, then the current triangulation is already an irreducible one, and the search ends. The *link condition test* (defined by Eq. 1 in Section 2) can be used to decide whether an examined edge is contractible. While this approach is quite simple, it can be very time-consuming in the worst-case.

Indeed, if an algorithm to compute $\mathcal{T}'$ relies on the link condition test to compute an irreducible triangulation, then its time complexity is basically determined by two factors: (1) the total number of times the link condition test is carried out by the algorithm, and (2) the time spent with each test. Bounding the number of link condition tests is challenging because *the contraction of an edge can make a previously non-contractible edge contractible and vice-versa.* Moreover, if no special data structure is adopted by the algorithm, then the time to test an edge $e = [u, v]$ against the link condition is in $\Theta(d_u \cdot d_v)$, in the worst case, where d_u and d_v are the degrees of vertices u and v in the current triangulation.

Consider the triangulation of a sphere in Figure 6, which is cut open in two separate pieces. There are exactly $n_v = 3m + 2$ vertices in this triangulation, namely: x, y, $v_0, \ldots, v_{m-1}$, $w_0, \ldots, w_{m-1}$, and $u_0, \ldots, u_{m-1}$. For each $i \in \{0, \ldots, m-1\}$, edges $[v_i, v_{(i+1) \mod m}]$, $[v_i, x]$, or $[v_i, y]$ are all non-contractible, while the remaining ones are all contractible. If all non-contractible edges happen to be tested against the link condition before any contractible edge is tested, then the time for testing all non-contractible edges against the link condition is $\Omega(n_f^2)$, as $n_f \in \Theta(n_v)$ by assumption, and there are as many as $2m$ edges of the forms $[v_i, x]$ and $[v_i, y]$, where each of them is tested in $\Theta(m)$ time because

$$d_x = m = d_y \, .$$

Schipper devised a more efficient algorithm by reducing the time spent on each link condition test [4]. For each vertex u in $\mathcal{T}$, his algorithm maintains a dictionary D_u containing all vertices in $lk(u, K)$, where K is the current triangulation. Determining if an edge $[u, v]$ in K is contractible amounts to verifying if D_v contains a vertex w in $lk(u, K)$, with $w \neq v$ and $w \notin lk([u, v], K)$, which can be done in $\mathcal{O}(d_u \lg d_v)$ time, where d_u and d_v are the degrees of u and v, respectively. He proved that if K is not irreducible then K contains a contractible edge incident on a vertex of degree at most 6. To speed up the search for a contractible edge, the edge chosen to be tested against the link condition is always incident on a vertex of lowest degree. To efficiently find this edge, his algorithm also maintains a global dictionary that stores all vertices of K indexed by their current degree. However, this heuristic does not prevent the same (non-contractible) edge of K from being repeatedly tested against the link condition.

Schipper's algorithm runs in $\mathcal{O}(n_f \lg n_f + g \lg n_f + g^4)$ time and requires $\mathcal{O}(n_f)$ space.

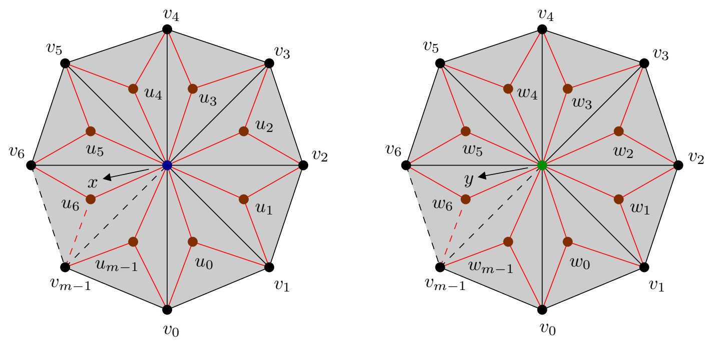

Figure 6: A reducible triangulation of the sphere cut open into two pieces.

Our algorithm allows us to more efficiently compute $\mathcal{T}'$ by testing each edge of K against the link condition at most once, and by reducing the worst-case time complexity for the link condition test even further. By using a time stamp mechanism (see Section 4.3 for details), our algorithm is able to efficiently determine if a previously tested, non-contractible edge becomes contractible as a result of an edge contraction (*without testing the edge against the link condition for a second time*). Our algorithm runs in $\mathcal{O}(g^2 + g\, n_f)$ time if g is positive, and it is linear in n_f otherwise. In either case, the space requirements are linear in n_f.

Edge contraction is a key operation for several *mesh simplification* algorithms [31]. The goal of these algorithms is not to compute an irreducible triangulation, but rather to decrease the level-of-detail (LOD) of a given triangulation by reducing its number of vertices, edges, and triangles. In general, contracted edges are chosen according to some application-dependent criterion, such as preserving geometric similarity between the input and the final triangulation.

Garland and Heckbert [32] show how to efficiently combine the edge contraction operation with a quadric-based error metric for geometric similarity. Furthermore, together with its inverse operation, vertex splitting, the edge contraction operation also allows for the construction of powerful hierarchical representation schemes for storing, transmitting, compressing, and selectively refining very large triangulations [33, 34]. However, topology preservation is not always desirable in the context of mesh simplification applications, and to the best of our knowledge, the greedy algorithm proposed by Cheng, Dey, and Poon in [35] is the only simplification algorithm whose time complexity has been analyzed.

The algorithm in [35] builds a topology-preserving surface triangulation hierarchy of $\mathcal{O}(n_v + g^2)$ size and $\mathcal{O}(\lg n_v + g)$ depth in $\mathcal{O}(n_v + g^2)$ time whenever $n_v \geq 9182g - 222$ and $g > 0$. Each level of the hierarchy is constructed by identifying and contracting a set of independent contractible edges in the triangulation represented by the previous level. A similar result for genus-0 surface triangulations has been known for a long time [36], although the construction of the hierarchy is not based on edge contractions. In general, however, we are not aware of any attempts to bound the number of link condition tests in the mesh simplification literature. If incorporated by simplification algorithms, this distinguishing feature of our algorithm, i.e., carrying out link condition tests faster, can increase their overall simplification speed.

4. Algorithm

Our algorithm takes as input a triangulation $\mathcal{T}$ of a surface $\mathcal{S}$ of genus g, and outputs an irreducible triangulation $\mathcal{T}'$ of the same surface. The key idea behind our algorithm is to iteratively choose a vertex u (rather than an edge) from the current triangulation, K, 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. It was shown in [4] that once vertex u becomes trapped, it cannot become a loose vertex again as the result of a topology-preserving edge contraction.

Lemma 5 ([4])**.** *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 trapped in $\mathcal{T} - e$.*

When the currently processed vertex u becomes trapped (or if u is already trapped when it is chosen by the algorithm), another vertex from the current triangulation is chosen and processed by the algorithm until all vertices are processed, at which point the algorithm ends. Since all vertices in the output triangulation $\mathcal{T}'$ have been processed by the algorithm, and since all edges contracted by our algorithm are contractible, Lemma 5 ensures that all vertices of $\mathcal{T}'$ are trapped. It follows that triangulation $\mathcal{T}'$ is irreducible. It is worth noting that our algorithm requires no knowledge about the embedding of $\mathcal{T}$, as all operations carried out by the algorithm are purely topological, and hence they act on $G_{\mathcal{T}}$ only.

When contracting a contractible edge $e = [u, v]$, our algorithm does not merge vertices u and v into a *new* vertex w. Instead, either u or v is chosen to play the role of w, and the other vertex is merged into the fixed one. If u is the fixed vertex, then we say that v *is identified with* u by the contraction of e (see Figure 7). When v is identified with u during the contraction of e, every edge of the form $[v, z]$ in $\mathcal{T}$ is replaced with an edge of the form $[u, z]$ in $\mathcal{T} - uv$, where $z \in lk(v, \mathcal{T})$ and $z \notin \{u, x, y\}$, and x and y are the vertices in $lk(e, \mathcal{T})$. We denote the set $\{u, x, y\}$ by Λ_{uv}, and the set $\{z \in lk(v, \mathcal{T}) \mid z \notin \Lambda_{uv}\}$ by Π_{uv}.

We assume that $\mathcal{T}$ and all triangulations resulting from the edge contractions executed by our algorithm are stored in an augmented doubly-connected edge list (DCEL) data structure [37], which is briefly discussed in Section 4.5. A detailed description of the algorithm is given in Sections 4.1-4.4. Section 4.6

discusses the particular case of triangulations of genus-0 surfaces. Finally, Section 4.7 analyzes the time and space complexities of the algorithm.

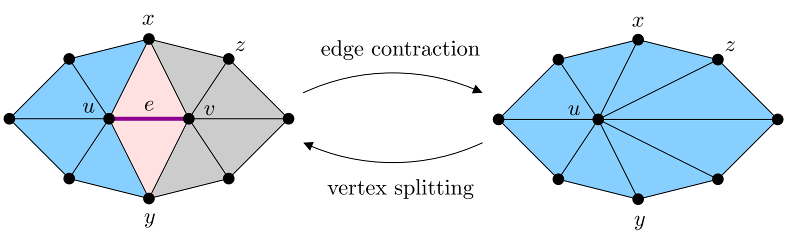

Figure 7: The contraction of $e = [u, v]$ in which v is identified with u.

4.1. Processing vertices

To support the efficient processing of vertices, the vertex record of the DCEL is augmented with six attributes: d, p, n, c, o, and t, where d, c, o, and t store integers, p stores a Boolean value, and n is a pointer to a vertex record (see Table 1). We denote each attribute a of a vertex v of the DCEL by $a(v)$. The value of each vertex attribute is defined with respect to the vertex u being currently processed by the algorithm. When u is chosen to be processed by the algorithm, its attributes and all attributes of its *neighbors*, i.e., the vertices in $lk(u, K)$, where K is the current triangulation, are initialized by the algorithm. As edges are contracted during the processing of u, the attribute values of the neighbors of u may change, while other vertices become neighbors of u and have their attribute values initialized. If a vertex of $\mathcal{T}$ never becomes a neighbor of u during the processing of u, its attribute values do not change while u is processed.

Attribute	**Description**
d	the degree of v
p	indicates whether v has already been processed by the algorithm
n	indicates whether v is a neighbor of u
c	number of critical cycles containing edge $[u, v]$ in K
o	time at which v becomes a neighbor of u
t	time at which edge $[u, v]$ is removed from *lue*

Table 1: Attributes of a vertex v during the processing of a vertex u.

The algorithm starts by creating a queue Q of *unprocessed* vertices, and by initializing the attributes d, p, n, c, o, and t of each vertex u of $\mathcal{T}$ (see Algorithm 4.1). In particular, for each vertex u in $\mathcal{T}$, its degree d_u is computed and stored in $d(u)$, its attribute $p(u)$ is set to *false*, its attribute $n(u)$ is assigned the *null* address, and its attributes $c(u)$, $o(u)$, and $t(u)$ are assigned 0, -1, and -1, respectively. Finally, a pointer to the record of u in the DCEL is inserted into Q.

After the initialization stage, the algorithm starts contracting edges of $\mathcal{T}$ (see Algorithm 4.2). Each edge contraction produces a new triangulation from the one to which the contraction was applied. The algorithm stores the currently modified triangulation in a variable, K. Here, we do not distinguish between K and the triangulation stored in it. Initially, K is set to the input triangulation $\mathcal{T}$ and the vertices in Q are the ones in $\mathcal{T}$. Let u be the vertex at the front of Q. The algorithm uses the value of $p(u)$ to decide whether u should be processed. In particular, $p(u)$ is *false* if and only if u belongs to K *and* u has not been processed yet (i.e., u is in Q). If $p(u)$ is *true* when u is removed from Q, then u is discarded. Otherwise, the algorithm processes u, i.e., it contracts (contractible) edges incident on u until u is trapped (see lines 5-36 of Algorithm 4.2). When vertex u becomes trapped, we say that u has been *processed* by the algorithm.

Algorithm 4.1 INITIALIZATION($\mathcal{T}$)

$Q \leftarrow \emptyset$ $\{Q$ is a queue of vertices$\}$
for each vertex u in $\mathcal{T}$ **do**
    $d(u) \leftarrow 0$
    for each v in $lk(u, \mathcal{T})$ **do**
        $d(u) \leftarrow d(u) + 1$
    end for
    $p(u) \leftarrow$ *false*
    $n(u) \leftarrow$ *nil*
    $c(u) \leftarrow 0$
    $o(u) \leftarrow -1$
    $t(u) \leftarrow -1$
    insert a pointer to u into Q.
end for
return Q

Two doubly-connected linked lists, *lue* and *lte*, are used by the algorithm to store edges incident with u during the processing of u. The former is the list of *unprocessed edges*, while the latter is the list of *tested edges*. At any given time, *lue* stores the edges incident on u that have not been tested against the link condition yet, while *lte* stores the edges incident on u that have been tested against the link condition before, during the processing of u, and failed the test. List *lue* is initialized with all edges $[u, v]$ of K such that $p(v)$ is *false* (lines 7-19 of Algorithm 4.2), while list *lte* is initially empty (see line 20 of Algorithm 4.2).

To process u, the algorithm removes one edge, $[u, v]$, from *lue* at a time and determines whether $[u, v]$ is contractible (lines 23-30 of Algorithm 4.2). If so, $[u, v]$ is contracted; else it is inserted into *lte*. Once list *lue* becomes empty, the algorithm considers list *lte* (lines 31-33 of Algorithm 4.2). List *lte* contains all edges incident on u that have been tested against the link condition during the processing of edges in *lue* and failed the test. However, while in *lte*, an edge may have become contractible again as the result of the contraction of another edge in *lue*. If so, Procedure ProcessEdgeList() in Algorithm 4.6 will find and contract this edge.

Algorithm 4.2 Contractions($\mathcal{T}, Q$)

```
S ← ∅ {S is a stack for maintaining edge contraction information}
K ← 𝒯
ts ← 0
while Q ≠ ∅ do
   remove a vertex u from Q {vertex u is chosen to be processed}
   if not p(u) then
      lue ← ∅ {lue is the list of unprocessed edges}
      for each v in lk(u, K) do
         n(v) ← u {mark v as a neighbor of u}
         o(v) ← ts {set the time at which v is found to be a neighbor of u}
         t(v) ← −1 {indicates that [u, v] has not been tested yet}
         if not p(v) then
            if d(v) = 3 then
               insert [u, v] at the front of lue
            else
               insert [u, v] at the rear of lue
            end if
         end if{inserts [u, v] into lue whenever p(v) is false}
      end for{lue stores all vertices in lk(u, K) that have not been processed yet}
      lte ← ∅ {lte is the list of tested edges}
      repeat
         while lue ≠ ∅ do
            remove edge e = [u, v] from lue
            t(v) ← ts
            if d(v) = 3 then
               ProcessVertexOfDegreeEq3(e, K, S, lue, lte, ts )
            else
               ProcessVertexOfDegreeGt3(e, K, S, lue, lte, ts )
            end if
         end while{processes all edges in lue}
         if lte ≠ ∅ then
            ProcessEdgeList(K, S, lue, lte, ts) {process contractible edges in lte}
         end if
      until lue = ∅
      p(u) ← true
   end if{vertex u is now processed}
end while
return (K, S)
```

Recall that if an edge $[u, v]$ in K is contracted, then u becomes incident on edges of the form $[u, z]$ in $K - uv$, where z is a vertex in Π_{uv} (see Figure 7). These *new* edges are always inserted into *lue*, as they have not been processed yet. Hence, the contraction of an edge by Algorithm 4.6 may cause the insertion of new edges into *lue*. If so, list *lue* becomes nonempty and its edges are processed again. Otherwise, list *lue* remains empty, and the processing of u ends with the value of $p(u)$ set to *true*. A key feature of our algorithm is its ability to determine which edges from *lte* become contractible, after the contraction of another edge, without testing those edges against the link condition again. To do so, the algorithm relies on a *time stamp* mechanism described in detail in Section 4.3.

4.2. Testing edges

To decide if an edge $[u, v]$ removed from *lue* is contractible, the link condition test is applied to $[u, v]$, except when the degree d_v of v is 3 (see lines 25-29 of Algorithm 4.2). If $d_v = 3$, then $[u, v]$ is always contractible, unless the degree d_u of u is also 3, which is the case if and only if the current triangulation K is $\mathcal{T}_4$.

Proposition 6. *Let K be a surface triangulation, and let v be any vertex of degree 3 in K. If K is (isomorphic to) $\mathcal{T}_4$, then no edge of K is contractible. Otherwise, every edge of K incident on v is a contractible edge in K.*

Proof. Let v be any vertex of K whose degree, d_v, is equal to 3. Then, $lk(v, K)$ contains exactly three vertices, say u, x, and y (see Figure 8). Since there are exactly two faces incident on $[u, v]$ (see Lemma 1), the other vertices of these two faces are x and y, else v would have degree greater than 3. So, we get $lk([u, v], K) = \{x, y\}$. We claim that $[u, v]$ is contractible if and only if K is not (isomorphic to) $\mathcal{T}_4$. Suppose that K is not isomorphic to $\mathcal{T}_4$. Then, face $[u, x, y]$ is not in K, which means that $lk(u, K) \cap lk(v, K) = \{x, y\}$. Conversely, if K is isomorphic to $\mathcal{T}_4$ then face $[u, x, y]$ is in K, which implies that $lk(u, K) \cap lk(v, K) = \{x, y, [x, y]\}$. By the link condition, $[u, v]$ is contractible if and only if K is not isomorphic to $\mathcal{T}_4$. Since every vertex of $\mathcal{T}_4$ has degree 3, the claim follows. □

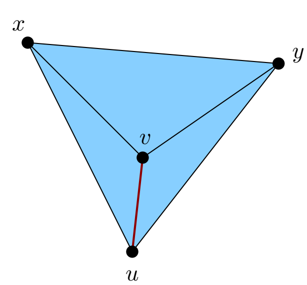

Figure 8: A vertex, v, of degree 3 in K. Edge $[u, v]$ is non-contractible if and only if $[u, x, y] \in K$.

Proposition 6 implies that if $d_v = 3$, we can decide whether $[u, v]$ is contractible by determining if the current triangulation K is isomorphic to $\mathcal{T}_4$. Testing whether K is isomorphic to $\mathcal{T}_4$ amounts to checking if $d_u = d_v = 3$, which can be done in constant time. Procedure ProcessVertexOfDegreeEq3() in Algorithm 4.3 is executed if d_v is equal to 3 (line 26 of Algorithm 4.2). If d_u is also equal to 3, then K is isomorphic to $\mathcal{T}_4$ and nothing is done (line 2). Otherwise, procedure Contract() in Algorithm 4.5 is invoked to contract $[u, v]$. This procedure is discussed in detail in Section 4.3 along with lines 4-24 of Algorithm 4.3, which are related to the time stamp mechanism for counting critical cycles.

Algorithm 4.3 ProcessVertexOfDegreeEq3(e, K, S, lue, lte, ts)

```
 1: get the vertices u and v of e in K
 2: if d(u) ≠ 3 then
 3:    Contract( e, K, S, lue, lte, ts ) {contract edge e = [u, v]}
 4:    let x and y be the vertices in lk(e, K)
 5:    if t(x) ≠ −1 and t(y) ≠ −1 then
 6:       c(x) ← c(x) − 1 {edge [u, x] is in lte; a critical cycle containing it is gone}
 7:       c(y) ← c(y) − 1 {edge [u, y] is in lte; a critical cycle containing it is gone}
 8:       if c(x) = 0 then
 9:          move [u, x] to the front of lte {[u, x] is now contractible}
10:       end if
11:       if c(y) = 0 then
12:          move [u, y] to the front of lte {[u, y] is now contractible}
13:       end if
14:    else if t(x) ≠ −1 and t(x) ≥ o(y) then
15:       c(x) ← c(x) − 1 {edge [u, x] is in lte; a critical cycle containing it is gone}
16:       if c(x) = 0 then
17:          move [u, x] to the front of lte {[u, x] is now contractible}
18:       end if
19:    else if t(y) ≠ −1 and t(y) ≥ o(x) then
20:       c(y) ← c(y) − 1 {edge [u, y] is in lte; a critical cycle containing it is gone}
21:       if c(y) = 0 then
22:          move [u, y] to the front of P {[u, y] is now contractible}
23:       end if
24:    end if{update the value of c(x) and c(y) after contracting [u, v]}
25: end if{if K is not isomorphic to T4}
```

If $d_v > 3$ when line 25 of Algorithm 4.2 is reached, then $[u, v]$ is tested against the link condition. As we pointed out in Section 3, if no special care is taken or no special data structure is adopted, the test $[u, v]$ can take $\Theta(d_u \cdot d_v)$ time. To reduce the worst-case time complexity of the link condition test, our algorithm makes use of the n attribute. During the processing of u, we set $n(w) = u$ for every vertex w in K with $[u, w] \in K$.

Since $d_v > 3$, K cannot be isomorphic to $\mathcal{T}_4$. So, edge $[u, v]$ is non-contractible if and only if $[u, v]$ is part of a critical cycle in K (see Figure 9) , i.e., if and only if u and v have a common neighbor z such that $z \notin lk([u, v], K)$ (i.e., $z \in \Pi_{uv}$). Conversely, if u and v do not have a common neighbor other than the two vertices in $lk([u, v], K)$, then they cannot be part of a 3-cycle in K. By examining the n attribute of the vertices in Π_{uv}, our algorithm can determine if u and v have a common neighbor in Π_{uv}, which can be done in $\mathcal{O}(d_v)$ time.

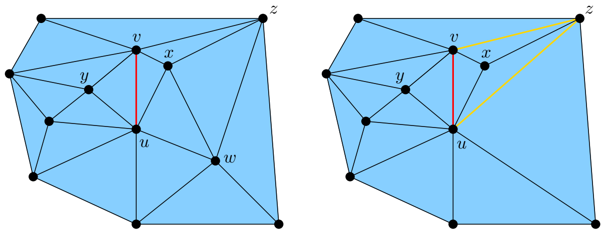

Figure 9: Vertex z is a neighbor of vertex u in the right triangulation, but not in the left one.

Procedure ProcessVertexOfDegreeGt3() in Algorithm 4.4 is the one responsible for testing $[u, v]$ against the link condition when $d_v > 3$ (line 28 of Algorithm 4.2). This procedure tests edge $[u, v]$ against the link condition, which amounts to counting the number of critical cycles in K containing $[u, v]$. Rather than merely checking the value of the n attribute of all vertices in Π_{uv}, Algorithm 4.4 computes the number $c(v)$ of critical cycles in K that contain edge $[u, v]$. To that end, Algorithm 4.4 (lines 2-12) counts the number of vertices z in Π_{uv} such that $n(z) = u$, which is precisely the number of critical cycles in K containing $[u, v]$. If $c(v)$ equals zero, then edge $[u, v]$ is contracted. Otherwise, edge $[u, v]$ is inserted into *lte*, as it has been tested against the link condition and has failed the test (lines 13-17 of Algorithm 4.4).

Algorithm 4.4 ProcessVertexOfDegreeGt3(e, K, S, lue, lte, ts)

```
1: get the vertices u and v of e in K
2: for each z in lk(v, K) do
3:    if z ∈ Π_uv and n(z) = u then
4:       c(v) ← c(v) + 1 {(u, v, z) is a critical cycle in K; increment c(v)}
5:       if t(z) ≠ −1 and t(z) < o(v) then
6:          c(z) ← c(z) + 1 {found a critical cycle in K containing [u, z]}
7:          if c(z) = 1 then
8:             move [u, z] to the rear of lte {c(z) was zero before}
9:          end if
10:      end if
11:   end if{updates the number, c(z), of critical cycles in K containing [u, z]}
12: end for{computes the number, c(v), of critical cycles in K containing [u, v]}
13: if c(v) = 0 then
14:    Contract(e, K, S, lue, lte, ts) {[u, v] in K is contractible}
15: else
16:    insert [u, v] at the rear of lte {edge [u, v] is non-contractible in K, as c(v) > 0}
17: end if
```

Lines 5-10 of Algorithm 4.4 are related to the counting of critical cycles containing edge $[u, z]$, for each $z \in \Pi_{uv}$, in triangulation $K - uv$. See Section 4.3 for further details. Furthermore, while edge $[u, v]$ is being tested by Algorithm 4.4, the degree d_v of v in K may not be the same as the degree d'_v of v in the input triangulation $\mathcal{T}$. In fact, during the processing of any vertex w, the degree of w can only increase or remain the same, while the degree of any other vertex can only decrease or remain the same. Hence, we get $d_v \leq d'_v$, and we can say that the time to test $[u, v]$ against the link condition is in $\mathcal{O}(d'_v)$. In general, the overall time spent with the link condition test during the processing of u is given by

$$\sum_{w \in W_u} \mathcal{O}(d'_w)\,,$$

where W_u is the set of all vertices w of $\mathcal{T}$ such that $[u, w]$ is an edge tested against the link condition during the processing of u, and d'_u and d'_w are the degrees of u and w in $\mathcal{T}$, respectively.

4.3. Counting critical cycles

Let $[u, v]$ be a contractible edge in K during the processing of u, and refer to Figure 10. If $[u, v]$ is contracted, then every ℓ-cycle containing $[u, v]$ in K is shortened and transformed into a $(\ell - 1)$-cycle in $K - uv$ containing u. Thus, every 4-cycle containing $[u, v]$ in K gives rise to a 3-cycle in $K - uv$ containing vertex u.

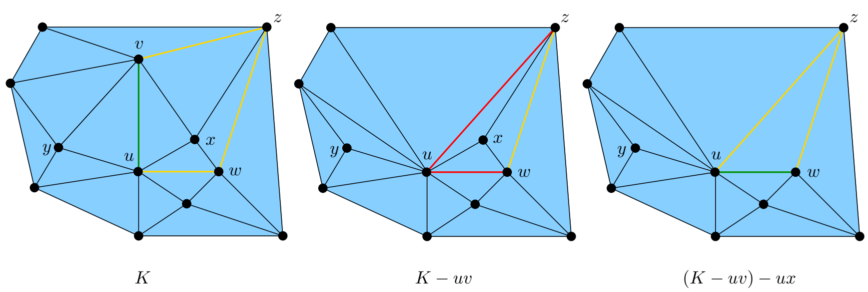

Figure 10: Cycle (u, z, w) is critical in $K - uv$, and non-critical in $(K - uv) - ux$.

Observe that a contractible edge in K may become non-contractible in $K - uv$. In particular, if a newly created 3-cycle, which results from an edge contraction, does not bound a face in $K - uv$, then every edge that belongs to it is non-contractible in $K - uv$. For instance, if edge $[u, v]$ is contracted in triangulation K on the left of Figure 10, then (u, v, z, w) gives rise to (u, z, w) in $K - uv$, which is critical. Observe also that an edge contraction can make a critical cycle non-critical in the resulting triangulation. For instance, if edge $[u, x]$ is contracted in triangulation $(K - uv)$ in Figure 10, then critical cycle (u, z, w) in $K - uv$ becomes non-critical in $(K - uv) - ux$.

In general, if the contraction of an edge $[u, v]$ in a triangulation K identifies a degree-3 vertex v with the currently processed vertex, u, then the cycle defined by the three edges of $lk(v, K)$ bounds a face in $K - uv$, and hence it cannot be critical in $K - uv$. Conversely, if C is a critical cycle in K but not in $K - uv$, then C must bound a face in $K - uv$. But, this is only possible if a vertex z of K is identified with a vertex of C by the edge contraction that produced $K - uv$ from K. Thus, vertex z must be v, vertex u must belong to C, and C must consist of the edges in $lk(v, K)$. Moreover, if a critical cycle C in K becomes non-critical in $K - uv$, a non-contractible edge in K may become contractible in $K - uv$.

Proposition 7. *Let K be a surface triangulation, and let f be a contractible edge of K. If a non-contractible edge e of K becomes contractible in $K - f$, then f must be incident on a degree-3 vertex v of K and e must belong to $lk(v, K)$. Moreover, e belongs to a single critical cycle in K, which consists of the edges in $lk(v, K)$, and this cycle becomes non-critical in $K - f$.*

Proof. By assumption, edge f is contractible in triangulation K. So, we can conclude that K cannot be (isomorphic to) $\mathcal{T}_4$. Thus, if e is a non-contractible edge in K, then e belongs to a critical cycle, say C, in K. Moreover, since e is contractible in $K - f$, we can also conclude that C is non-critical in $K - f$. But, this means that f is incident on a vertex, u, in C and on a degree-3 vertex, v, in K such that C consists of the edges in $lk(v, K)$. Also, the contraction of f identifies v with u. We claim that C is the only critical cycle containing e in K. In fact, if e belonged to another critical cycle, say C', in K, then C' would have to be non-critical in $K - f$; else e would remain non-contractible in $K - f$. But, if C' were non-critical in $K - f$, then C' would have to consist of the edges of $lk(v, K)$ as well. Thus, $C' = C$, i.e., C is the only critical cycle containing e in K. □

Proposition 7 allows us to devise, for each edge e that has been tested against the link condition, a time stamp mechanism to keep track of the number of critical cycles to which e belongs. Recall that all such edges e are stored in the list *lte*. The idea is quite simple. Whenever a contractible edge $[u, v]$, with $d_v = 3$, is contracted, the algorithm checks whether the critical cycle counter of x and y must be *decremented*, where x and y are the two vertices of $lk([u, v], K)$. From Proposition 7, we know that $[u, x]$ and $[u, y]$ are the only edges incident on u that could become contractible in $K - uv$ (if they are non-contractible edges in K). In turn, if $d_v > 3$ then the algorithm checks whether the critical cycle counter of all vertices involved in newly created 3-cycles of $K - uv$ must be *incremented*. This is because contractible edges in K may become non-contractible in $K - uv$, but not the other way around according to Proposition 7. Furthermore, the newly created critical cycles must contain a new neighbor of u in $K - uv$ (i.e, a vertex in Π_{uv}).

The time stamp mechanism relies on a global *time counter*, *ts*, and on the o and t attributes. The value of *ts* is set to zero before any vertex of $\mathcal{T}$ is ever processed (line 3 of Algorithm 4.2). Moreover, the value of *ts* is updated if and only if an edge is contracted. More specifically, the value of *ts* is incremented by one by Algorithm 4.5 immediately before the actual edge contraction occurs.

The o and t attributes of every vertex u of $\mathcal{T}$ are each set to -1 during the initialization stage (Algorithm 4.1). During the processing of a vertex u,

Algorithm 4.5 CONTRACT(e, K, S, lue, lte, ts)

```
get the vertices u and v of e
get the vertices x and y of lk(e, K)
ts ← ts + 1
for each z in lk(v, K) do
   if z ∈ Π_uv then
      n(z) ← u
      c(z) ← 0
      o(z) ← ts
      t(z) ← −1
   end if {vertex z will become a neighbor of u in K − uv}
end for {initializes the n, c, o, and t attributes of the new neighbors of u}
p(v) ← true {prevents v from being selected for processing}
push a record with v, [u, v], [v, x], [v, y], [u, v, x], and [u, v, y] onto S
temp ← ∅ {temp is a temporary list of edges [u, z] such that z ∈ Π_uv}
COLLAPSE(e, K, temp) {updates the DCEL}
d(x) ← d(x) − 1 {updates the degree of x}
d(y) ← d(y) − 1 {updates the degree of y}
d(u) ← d(u) + d(v) − 4 {updates the degree of u}
if d(x) = 3 and not p(x) and t(x) = −1 then
   move [u, x] to the front of lue
end if
if d(y) = 3 and not p(y) and t(y) = −1 then
   move [u, y] to the front of lue
end if
for each [u, z] in temp do
   if not p(z) then
      if d(z) = 3 then
         insert [u, z] at the front of lue
      else
         insert [u, z] at the rear of lue
      end if
   else
      get the vertices x and y of lk([u, z], K)
      for each w in lk(z, K) do
         if w ∉ Λ_uz and n(w) = u and t(w) ≠ −1 then
            c(w) ← c(w) + 1 {increment c(w) to account for (u, w, z)}
            if c(w) = 1 then
               move [u, w] to the rear of lte {[u, w] is now non-contractible}
            end if
         end if
      end for
   end if
end for {updates c(z) if [u, z] ∈ lte and inserts [u, z] in lue otherwise}
```

the value of the o attribute of a vertex v is changed to *ts* if and only if v is or becomes a neighbor of u, i.e., right before $[u, v]$ is inserted into list *lue* because v is already a neighbor of u when the processing of u begins (see line 10 of Algorithm 4.2) or because v becomes a neighbor of u as the result of an edge contraction during the processing of u (in line 8 of Algorithm 4.5). The value of $o(v)$ is changed only once during the processing of u, and after the change is made $o(v)$ can be viewed as the *time* the algorithm discovers that v is in $lk(u, K)$. In turn, the t attribute of a vertex v may be changed at most once during the processing of u. The value of $t(v)$ is set to *ts* immediately before $[u, v]$ is removed from list *lue* (see line 24 of Algorithm 4.2). Hence, after the change is made, $t(v)$ can be viewed as the *time* the algorithm decides whether $[u, v]$ is contractible.

Before we describe the time stamp mechanism, we state two invariants regarding list *lue* and *lte*, which will also help us prove the correctness of the algorithm:

Proposition 8. *Let u be any vertex of $\mathcal{T}$ processed by the algorithm. Then, during the processing of vertex u, the conditions regarding lue below are (loop) invariants of the while and repeat-until loops in lines 22-30 and 21-34, respectively, of Algorithm 4.2:*

(1) every edge $[u, w]$ in lue is an edge of the current triangulation, K;

(2) if $[u, w]$ is an edge in lue such that d_w is greater than 3, then edge $[u, w]$ cannot precede an edge $[u, z]$ in lue such that d_z is equal to 3;

(3) the value of $p(z)$ is false, for every vertex z such that $[u, z]$ is in lue;

(4) the value of $o(z)$ is no longer -1, for every vertex z such that $[u, z]$ is in lue;

(5) the value of $t(z)$ is -1, for every vertex z such that $[u, z]$ is in lue;

(6) the value of $c(z)$ is 0, for every vertex z such that $[u, z]$ is in lue; and

(7) no edge in lue has been tested against the link condition before.

Proof. The proof is straightforward (see [38] for the details.) □

Proposition 9. *Let u be any vertex of $\mathcal{T}$ processed by the algorithm. Then, during the processing of u, the conditions regarding lte below are (loop) invariants of the while and repeat-until loop in lines 22-30 and 21-34, respectively, of Algorithm 4.2:*

(1) lists lue and lte have no edge in common;

(2) if $[u, z]$ is an edge in lte then $t(z) \geq o(z) > -1$; and

(3) every edge in lte was tested against the link condition exactly once and failed.

Proof. The proof is straightforward (see [38] for the details.) □

Let $[u, v]$ be an edge removed from *lue* during the processing of vertex u, and let K be the current triangulation at that time. Suppose that $[u, v]$ is contractible. The time stamp mechanism distinguishes two cases: $d_v > 3$ and $d_v = 3$.

Case $\boldsymbol{d_v > 3}$. If d_v is greater than 3 in K, then Algorithm 4.4 is executed on $[u, v]$, and Algorithm 4.5 is invoked in line 14 to contract $[u, v]$ (refer to triangulation K in Figure 10). As we pointed out before, the contraction of $[u, v]$ may give rise to one or more critical cycles in $K - uv$. So, for every neighbor z of v in K that becomes a new neighbor of u in $K - uv$, the algorithm determines if u and z have a common neighbor, w. If so, then (u, v, z, w) is a 4-cycle in K, shortened by the contraction of $[u, v]$, that gave rise to critical cycle (u, z, w) in $K - uv$. If edge $[u, w]$ is in *lte*, then $c(w)$ must be incremented by 1 to account for the newly created critical cycle, (u, z, w), in $K - uv$. Otherwise, nothing needs to be done, as either $[u, w]$ is still in *lue* or vertex w has been processed.

Lines 4-11 of Contract() (see Algorithm 4.5) visit all neighbors z of v in K that become neighbors of u in $K - uv$. Procedure Collapse(), invoked in line 15, contracts $[u, v]$, updates the DCEL, and returns a list, *temp*, with the new neighbors z of u in $K - uv$. Lines 16-18 update the degrees of the vertices x, y, and u, where x and y are the two vertices in $lk([u, v], K)$. Lines 19-24 ensure that Proposition 8(2) holds, and lines 25-43 process the new neighbors z of u that were placed in list *temp*. If $p(z)$ is *true* then the algorithm determines whether the contraction of $[u, v]$ in K produced critical cycles in $K - uv$ involving $[u, z]$. If this is the case, then the critical cycle counter of the third vertex w of the cycle is updated accordingly. If $p(z)$ is *false* then $[u, z]$ is inserted into *lue* in lines 27-31.

Suppose that $p(z)$ is *true*. To determine the occurrence of a new critical cycle in $K - uv$ involving edge $[u, z]$, Contract() compares $n(w)$ with u, for every vertex w in $lk(z, K - uv)$ such that $w \notin \Lambda_{uz}$ (see lines 28-35 of Algorithm 4.5). If $n(w) = u$, then (u, z, w) is a critical cycle in $K - uv$. Otherwise, (u, z, w) is *not* a cycle in $K - uv$. This verification takes $\Theta(d_z)$-time, where d_z is the degree of z in $K - uv$. Since z is a previously processed vertex, it is possible that d_z is greater than the degree of z in the input triangulation, $\mathcal{T}$. The value of $c(w)$ must be incremented by 1 to account for (u, z, w) whenever $[u, w]$ belongs to list *lte*. Line 35 of Algorithm 4.5 checks if $n(w) = u$, $w \notin \Lambda_{uz}$, and $t(w) \neq -1$. If the first two conditions are true, then (u, z, w) is a critical cycle in $K - uv$. If the third is also true, then Propositions 8 and 9 tell us that $[u, w]$ is in *lte*. Accordingly, Contract() increments the critical cycle counter, $c(w)$, of w by 1 in line 36 if and only if the logical expression in line 35 evaluates to *true*.

Suppose now that $p(z)$ is *false*. Then, Contract() simply inserts $[u, z]$ into *lue* (see lines 27-31 of Algorithm 4.5). *Our algorithm need not check whether $[u, z]$ is part of a critical cycle in $K - uv$ at this point.* This verification is postponed to the moment at which $[u, z]$ is removed from *lue*, in line 23 of Algorithm 4.2, with vertex z labeled as v. If $[u, v]$ is part of a critical cycle, then v cannot have degree 3, which means that $[u, v]$ is tested against the link condition in lines 2-12 of Algorithm 4.4. During this test, if $[u, v]$ is found to be part of a critical cycle, then the third vertex involved in the cycle (labeled z in Algorithm 4.4) may have its c attribute value incremented. Indeed, the value of $c(z)$ is incremented by 1 whenever (a) $[u, z]$ is in *lte* (i.e., $t(z) \neq -1$), and (b) $[u, z]$ was inserted in *lte* before v became a neighbor of u (i.e, $t(z) < o(v)$). Condition (b) is necessary to ensure correctness of the counting process. Otherwise, $c(z)$ could be incremented twice for the same cycle, (u, v, z): one time when edge $[u, z]$ is tested against the link condition, and another time when edge $[u, v]$ is tested against the link condition. For an example, let $[u, r]$ be an edge of K such that $[u, r]$ is part of a critical cycle, (u, r, s), of K by the time $[u, r]$ is removed from *lue* in line 23 of Algorithm 4.2 (see Figure 11). Suppose that $d_r > 3$ and $p(s) = \mathit{false}$. Then, when $[u, r]$ is given as input to Algorithm 4.4, we have two possibilities:

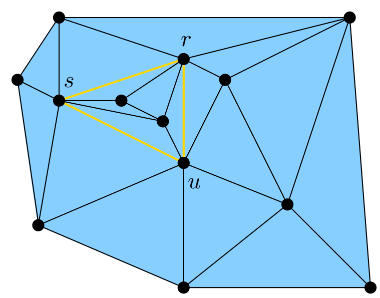

Figure 11: A critical cycle (u, r, s) in K.

(i) $t(s) < o(r)$: edge $[u, s]$ was tested against the link condition *before* r becomes a neighbor of u. Thus, by the time $[u, s]$ was tested against the link condition, edge $[u, r]$ was not an edge of the current triangulation. Consequently, line 6 of Algorithm 4.4 could not be executed to increment $c(r)$ by 1 (with r labeled z) to account for a critical cycle that did not exist at the time. For the same reason, line 4 of Algorithm 4.4 cannot increment $c(s)$ by 1 to account for the same cycle either (with s labeled v). However, when $[u, r]$ is tested against the link condition, $c(r)$ is incremented by 1 in line 4 of Algorithm 4.4 (with r labeled v) to account for (u, r, s). In addition, since $[u, s]$ is in *lte*, we have $t(s) \neq -1$. By hypothesis, we also know that $t(s) < o(r)$. So, line 6 of Algorithm 4.4 is executed to increment $c(s)$ by 1 to account for (u, r, s) for the first time as well (with s labeled z).

(ii) $t(s) \geq o(r)$: vertex r was already a neighbor of u when $[u, s]$ was tested against the link condition. So, we have two cases: (a) $[u, s]$ is tested against the link condition before $[u, r]$, and (b) $[u, r]$ is tested against the link condition before $[u, s]$. If (a) holds, then $c(s)$ is incremented by 1 to account for (u, r, s) when $[u, s]$ is the input edge, e, of Algorithm 4.4 (with s labeled v). However, the value of $c(r)$ is not incremented by 1 to account for the same cycle, as line 6 is not executed. The reason is that $[u, r]$ is still in *lue*. So, $t(r) = -1$, which implies that condition $t(z) \neq -1$ fails (with $z = r$) in line 5 of Algorithm 4.4. When $[u, r]$ is tested against the link condition, the value of $c(r)$ is incremented by 1 to account for (u, r, s) in line 4 of Algorithm 4.4 (with r labeled v). At this point, $c(s)$ is not incremented to account for (u, r, s) for the *second* time, as condition $t(z) < o(v)$ fails for $z = s$ and $v = r$. If (b) holds, then the situation is similar to (a); we just have to interchange the roles of r and s. Therefore, to account for (u, r, s), the values of $c(r)$ and $c(s)$ are incremented by 1 only once.

Case $d_v = 3$. If v is a degree-3 vertex and K is not (isomorphic to) $\mathcal{T}_4$, then procedure ProcessVertexOfDegreeEq3() (Algorithm 4.3) is invoked in line 26 of Algorithm 4.2 to contract $[u, v]$. Let C be the critical cycle of K consisting of the edges in $lk(v, K)$, i.e., $[u, x]$, $[x, y]$, and $[u, y]$, where x and y are the two vertices of $lk([u, v], K)$. Then, Proposition 7 tells us that the contraction of edge $[u, v]$ makes C non-critical in $K - uv$. If edge $[u, x]$ (resp. $[u, y]$) belongs to *lte*, then the value of $c(x)$ (resp. $c(y)$) must be decremented by 1 to account for the fact that one critical cycle in K containing $[u, x]$ (resp. $[u, y]$) is no longer critical in $K - uv$.

The value of $c(x)$ (resp. $c(y)$) should only be decremented if $c(x)$ (resp. $c(y)$) was previously incremented to account for the critical cycle that became non-critical. Algorithm 4.3 uses the values of the o and t attributes of x and y to decide whether $c(x)$ and $c(y)$ should be decremented as follows:

- If, immediately after the contraction of $[u, v]$ in K, the values of $t(x)$ and $t(y)$ are both different from -1, then $[u, x]$ and $[u, y]$ are both in *lte*, and $c(x)$ and $c(y)$ were incremented by 1 to account for the existence of C when either $[u, x]$ or $[u, y]$ was tested against the link condition in line 4 of Algorithm 4.4 (with x or y labeled v). Both x and y are vertices with degree greater than 3 in K, as it was the case when $[u, x]$ and $[u, y]$ were removed from *lue* and then tested against the link condition. From the case $d_v > 3$ (with $v = x$ or $v = y$), we know that $c(x)$ and $c(y)$ were incremented by 1 to account for C exactly once. So, to account for the fact that C is no longer critical in $K - uv$, both $c(x)$ and $c(y)$ are decremented by 1 after the contraction of $[u, v]$ in line 3 of Algorithm 4.3, which is done right after by lines 6 and 7.

- If, immediately after the contraction of $[u, v]$, $t(x) \neq -1$ and $t(y) = -1$, then only $[u, x]$ is in *lte*. Vertex y cannot be trapped, as edge $[v, y]$ is contractible in K (see Proposition 6) and no edge incident on a trapped vertex can be contractible (see Lemma 5). Thus, vertex y has not been processed yet, which means that $[u, y]$ is still in *lue*. Moreover, the value of $c(x)$ is incremented to account for C if and only if $[u, x]$ was inserted into *lte* after y became a neighbor of u (i.e., $n(y) = u$). In fact, if $n(y) = u$ then line 4 of Algorithm 4.4 is executed for $v = x$ and $z = y$, incrementing $c(x)$ by 1 to account for C. Also, since $t(y) = -1$, line 6 of Algorithm 4.4 is *not* executed for $z = y$, and hence $c(y)$ is not incremented by 1 to account for C while $[u, x]$ is tested against the link condition. Conversely, if $[u, x]$ was inserted into *lte* before y became a neighbor of u, then y is not a vertex in Π_{ux}, which means that line 4 of Algorithm 4.4 is not executed for $v = x$ and $z = y$. Thus, the value of $c(x)$ is not incremented by 1 to account for C. This is consistent with the fact that C is not even a cycle in the current triangulation by the time $[u, x]$ is tested against the link condition.

When $[u, x]$ is inserted into *lte* after y becomes a neighbor of u, we must have $o(x) \geq o(y)$, as $[u, y]$ is still in list *lue* (i.e., $t(y) = -1$) and $[u, x]$ was removed from *lue* before $[u, y]$. Since $t(w) \geq o(w)$ for every vertex w such that $[u, w]$ is in *lte*, we must have that $t(x) \geq o(y)$. If $[u, x]$ is inserted into *lte* before y becomes a neighbor of u, then $t(x) < o(y)$, as $t(x)$ is the time at which $[u, x]$ is removed from *lue* and inserted into *lte*, while $o(y)$ is the time at which y becomes a neighbor of u. So, whenever $t(x) \neq -1$, $t(y) = -1$ and $t(x) \geq o(y)$, the value of $c(x)$ (but not the one of $c(y)$) is decremented by 1 in line 15 of Algorithm 4.3, right after the contraction of $[u, v]$ in line 3, to account for the fact that C is no longer critical in $K - uv$.

- If $t(x) = -1$ and $t(y) \neq -1$ immediately after the contraction of $[u, v]$, then we have the same case as before, except that the roles of x and y are interchanged.

- If both $t(x)$ and $t(y)$ are equal to -1, then neither $[u, x]$ nor $[u, y]$ are in *lte*, and thus there is no need for updating $c(x)$ and $c(y)$ (as none of them were incremented by 1 to account for C). Furthermore, since C is no longer critical in $K - uv$, the values of $c(x)$ and $c(y)$ cannot be incremented by 1 to account for C when $[u, x]$ and $[u, y]$ are tested against the link condition, which is also consistent with the fact that C is not critical in $K - uv$. In fact, cycle C may not even be a cycle in K when $[u, x]$ and $[u, y]$ are tested.

To illustrate all cases above, consider the triangulation K_1 in Figure 12. Note that vertex x is a neighbor of u in K_1, but vertex y is not. Suppose that edge $[u, w]$ is contracted, making y a neighbor of u and yielding triangulation K_2 in Figure 12. Next, suppose that edge $[u, z]$ is contracted, yielding triangulation K_3 in Figure 12. Finally, since v is a degree-3 vertex in K_3, edge $[u, v]$ is contracted, which makes (u, x, y) a non-critical cycle in $K_3 - uv$. After $[u, v]$ is contracted, the values of $c(x)$ and $c(y)$ are updated by Algorithm 4.4. To illustrate how the updates are carried out by the algorithm, consider the following scenarios: (i) $[u, x]$ is removed from *lue* before edge $[u, w]$ is, (ii) $[u, x]$ is removed from *lue* after edge $[u, w]$ is, (iii) $[u, y]$ is removed from list *lue* before $[u, z]$ is, and (iv) $[u, y]$ is removed from list *lue* after $[u, z]$ is.

Suppose that (i) and (iii) hold. Then, both $[u, x]$ and $[u, y]$ have been tested against the link condition by the time $[u, v]$ is considered for contraction in K_3.

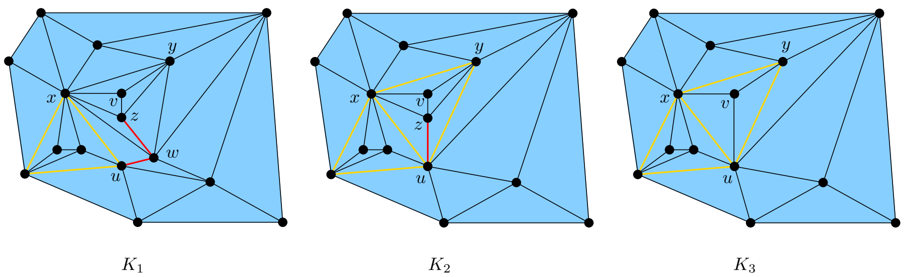

Figure 12: K_2 (resp. K_3) is obtained from K_1 (resp. K_2) by contracting $[u, w]$ (resp. $[u, z]$).

So, $t(x) \neq -1$ and $t(y) \neq -1$ immediately after the contraction of $[u, v]$, as both $[u, x]$ and $[u, y]$ have already been removed from list *lue* (and inserted into list *lte*). Since y is not a neighbor of u when $[u, x]$ is tested against the link condition, the values of $c(x)$ and $c(y)$ are not incremented to account for critical cycle (u, x, y) in K_3. Indeed, both $c(x)$ and $c(y)$ are incremented by 1 to account for (u, x, y) in K_3 while $[u, y]$ is tested against the link condition. This is done by lines 4 and 6 of Algorithm 4.4, with x and y labeled z and v, respectively, as $t(x) \neq -1$ and $t(x) < o(y)$. After the contraction of $[u, v]$ in K_3, both $c(x)$ and $c(y)$ are decremented by 1 in lines 6 and 7 of Algorithm 4.4, which accounts for the fact that (u, x, y) is no longer critical in $K_3 - uv$.

If (iv) holds instead, then only $[u, x]$ has been tested against the link condition by the time $[u, v]$ is considered for contraction in K_3. So, $t(x) \neq -1$ and $t(y) = -1$ immediately after the contraction of $[u, v]$, as $[u, y]$ is still in list *lue*. Since y is not a neighbor of u when $[u, x]$ is removed from *lue* and tested against the link condition, we get $t(x) < o(y)$. This implies that $c(x)$ is not decremented by 1, in line 15 of Algorithm 4.3), to account for the fact that (u, x, y) is not critical in $K_3 - uv$. This is consistent with the fact that $c(x)$ is not incremented by 1 to account for critical cycle (u, x, y) when $[u, x]$ was tested against the link condition.

Suppose that (ii) holds. Then, vertex y is already a neighbor of u when $[u, x]$ is removed from list *lue*. Furthermore, edge $[u, x]$ is removed from *lue* before $[u, y]$ is, as both edges are inserted at the rear of *lue* and $[u, x]$ is inserted first. Since $t(y) = -1$ and $t(y) < o(x)$ by the time $[u, x]$ is tested against the link condition, both $c(x)$ and $c(y)$ are incremented by 1 in lines 4 and 6 of Algorithm 4.4, respectively, during the test. If (iii) also holds, then we get $t(x) \neq -1$ and $t(y) \neq -1$ by the time $[u, v]$ is tested against the link condition. So, after the contraction of $[u, v]$ in K_3, both $c(x)$ and $c(y)$ are decremented by 1 in lines 6 and 7 of Algorithm 4.3, respectively, to account for the fact that (u, x, y) is not a critical cycle in $K_3 - uv$. If (iv) holds instead, then since v is a degree-3 vertex, edge $[u, v]$ is removed from *lue* before $[u, y]$ is. This means that $[u, y]$ is still in *lue* after the contraction of $[u, v]$. Thus, $t(y) = -1$. But, since y was a neighbor of u when $[u, x]$ was removed from *lue*, we get $t(x) \geq o(y)$. So, both $c(x)$ and $c(y)$ are decremented by 1 in lines 6 and 7 of Algorithm 4.3, respectively, to account for the fact that (u, x, y) is not critical in $K_3 - uv$. Thus, the values of $c(x)$ and $c(y)$ are consistently updated by the algorithm in cases (ii) and (iii).

Finally, suppose that triangulation K_2 in Figure 12 is the initial triangulation in the processing of u, which means that the algorithm finds that both vertices x and y are neighbors of u in lines 8-19 of Algorithm 4.2, and thus $o(x) = o(y) \neq -1$. If edge $[u, z]$ is removed from *lue* before both $[u, x]$ and $[u, y]$, then $[u, x]$ and $[u, y]$ are in list *lue* immediately after the contraction of $[u, z]$. Since v is a degree-3 vertex, edge $[u, v]$ is inserted at the front of *lue*, which implies that $[u, v]$ is removed from *lue* before any of $[u, x]$ and $[u, y]$ is. So, after the contraction of $[u, v]$, we get $t(x) = t(y) = -1$. Thus, the values of $c(x)$ and $c(y)$ are not decremented in Algorithm 4.3 to account for the fact that (u, x, y) is not a critical cycle in $K - uv$. This is consistent with the fact that none of $c(x)$ and $c(y)$ have been incremented yet. This example shows that, to consistently update the values of the c attributes, our algorithm need not increment counters every time a critical cycle arises.

4.4. Processing edges

From Proposition 8, each edge that belongs to list *lue*, during the processing of vertex u, is an edge of the form $[u, z]$ such that $c(z) = 0$, $o(z) \neq -1$, $t(z) = -1$, and $p(z) = \mathit{false}$ (i.e, z is in Q and thus it has not been processed yet). Furthermore, every edge in *lue* is eventually removed from *lue* during the execution of the while loop in lines 22-30 of Algorithm 4.2. Once an edge $[u, z]$ is removed from *lue*, there are 3 possibilities: it is either contracted, inserted into list *lte*, or ignored.

If $[u, z]$ is contracted, then it is removed from triangulation $K - uz$ and $p(z)$ is set to *true*, which prevents the algorithm from trying to process vertex z after it is removed from Q in line 5 of Algorithm 4.2. If $[u, z]$ is inserted into *lte*, then $[u, z]$ has been tested against the link condition and found to be non-contractible by Algorithm 4.4. Moreover, immediately before $[u, z]$ is inserted into *lte* (see line 16 of Algorithm 4.4), the value of $c(z)$ is equal to the number of critical cycles containing $[u, z]$ in K, and the value of $t(z)$ is the time at which $[u, z]$ was removed from *lue*. If $[u, z]$ is ignored, i.e., if the degree, d_u, of u and the degree, d_z, of z are both equal to 3 (see line 25 of Algorithm 4.2 and line 2 of Algorithm 4.3), then K is (isomorphic to) $\mathcal{T}_4$, which means that $[u, z]$ is not contractible.

List *lue* will eventually be empty after finitely many iterations of the while loop in lines 22-30 of Algorithm 4.2. This is because there are finitely many edges in the input triangulation $\mathcal{T}$, each edge contraction yields a triangulation with three fewer edges, no vertex is created by the algorithm, and no edge removed from *lue* is inserted into *lue* again. So, let us consider the moment at which *lue* becomes empty and line 31 of Algorithm 4.2 is reached. For every edge $[u, z]$ in the current triangulation, K, we distinguish two cases: (1) $[u, z] \notin \mathit{lte}$ and (2) $[u, z] \in \mathit{lte}$.

If $[u, z] \notin \mathit{lte}$, then let us consider the value of $p(z)$. If $p(z)$ is *true*, then vertex z was processed before u is removed from Q. So, edge $[u, z]$ is never inserted into *lue*. Since z was processed before, it is trapped, which implies that $[u, z]$ is non-contractible. If $p(z)$ is *false*, then z is still in Q, and edge $[u, z]$ was ignored by the algorithm after being removed from list *lue*. So, triangulation K must be (isomorphic to) $\mathcal{T}_4$, which implies that all edges of K are non-contractible edges.

If $[u, z] \in$ *lte*, then $[u, z]$ has been tested against the link condition after being removed from *lue* and found to be non-contractible at the time. List *lte* is a temporary holder for this kind of edge. Every time list *lue* becomes empty and the while loop in lines 22-30 of Algorithm 4.2 ends, list *lte* is examined by procedure ProcessEdgeList() in Algorithm 4.6, which is invoked by line 32 of Algorithm 4.2 whenever *lte* is nonempty. This procedure checks whether an edge $[u, v]$ in *lte* became contractible (after being inserted into *lte*). If so, at least one contractible edge in *lte* is contracted. The contraction of $[u, v]$ can generate edges in $K - uv$ that are not in K. This is the case whenever $\Pi_{uv} \neq \emptyset$, and the edges are precisely the ones of the form $[u, w]$ in $K - uv$, with $w \in \Pi_{uv}$ (see Figure 7).

Algorithm 4.6 ProcessEdgeList(K, S, lue, lte, ts)

```
while lte ≠ ∅ do
   let e = [u, v] be the edge at the front of lte
   if d(v) = 3 then
      remove edge e = [u, v] from lte
      ProcessVertexOfDegreeEq3(e, K, S, lue, lte, ts)
   else
      break{the edge at the front of lte is not incident on a degree-3 vertex}
   end if
end while{contract edges incident on degree-3 vertices}
if lte ≠ ∅ then
   let e = [u, v] be the edge at the front of lte
   if c(v) = 0 then
      remove edge e from lte{the degree of v is greater than 3}
      Contract( e, K, S, lue, lte, ts ){since c(v) = 0, edge e is contractible}
   end if
end if{the first edge of lte is incident on a vertex with degree greater than 3}
```

To efficiently find a contractible edge in *lte* or find out that one does not exist, our algorithm always moves every edge $[u, z]$ whose value of $c(z)$ is 0 to the front of *lte*. In particular, every time that the value of $c(z)$ is decremented, for any vertex z such that $[u, z]$ is in *lte*, the algorithm verifies if $c(z)$ becomes 0. If so, edge $[u, z]$ is moved to the front of *lte* (see lines 8-13, 16-18, and 21-23 of Algorithm 4.3). In addition, every time that the value of $c(z)$ is incremented, for any vertex z such that $[u, z]$ is in *lte*, the algorithm verifies if $c(z)$ becomes 1. If so, edge $[u, z]$ is moved to the rear of *lte* (see lines 37-39 of Algorithm 4.5 and lines 7-9 of Algorithm 4.4). So, the following invariant regarding *lte* also holds:

Proposition 10. *Let u be the currently processed vertex of the algorithm. Then, the following property regarding list lte is a (loop) invariant of the while and* *repeat-until loops in lines 22-30 and 21-34 of Algorithm 4.2: no edge* $[u, z]$ *in lte such that* $c(z) > 0$ *can precede an edge* $[u, w]$ *in lte such that* $c(w)$ *is equal to* 0.

Proof. The proof is straightforward (see [38] for the details.) □

From Proposition 10, it suffices to check the value of $c(z)$, where $[u, z]$ is the edge at the front of *lte*, to find out whether *lte* contains a contractible edge, which takes constant time. Of course, the correctness of this test relies on the premise that $c(w)$ is indeed equal to the number of critical cycles in K containing edge $[u, w]$, for every edge $[u, w]$ is *lte*. The following states that this premise is valid:

Proposition 11. *Let* u *be any vertex of* $\mathcal{T}$ *processed by the algorithm. Then, whenever line 32 of Algorithm 4.2 is reached, during the processing of* u*, we have that for every edge* $[u, w]$ *in list lte, the value of* $c(w)$ *is the number of critical cycles in* K *containing edge* $[u, w]$*, where* K *is the current triangulation at the time.*

Proof. The proof is straightforward (see [38] for the details.) □

If list *lue* is empty after Algorithm 4.6 is executed, then no edge in *lte* is contractible, which also means that no edge incident on u is contractible. So, vertex u is trapped and the processing of u ends. Otherwise, the while loop in lines 22-30 of Algorithm 4.2 is executed again to process the edges in *lue*. It is worth noting that *no edge is tested against the link condition more than once.* Furthermore, since *lte* is a doubly-connected linked list, moving an element of *lte* from any position to the front or rear of *lte* can be done in constant time if we have a pointer to the element. With that in mind, we included a pointer in the edge record of our augmented DCEL to the edge record of *lte*, which makes it possible to access an edge in *lte* from the DCEL record of the edge in constant time.

4.5. Updating the DCEL

Our algorithm stores the input triangulation, $\mathcal{T}$, in a Doubly-Connected Edge List (DCEL) data structure [37], which is augmented with vertex attributes p, n, c, o, and t and an edge attribute (i.e., a pointer to a node in *lte*). Our DCEL has four records: one for vertices, one for edges, one for triangles, and one for half-edges. For each half-edge, h, that bounds a given triangle of the triangulation represented by the DCEL, there is the *next* half-edge and the *previous* half-edge on the same boundary. The destination vertex of h is the origin vertex of its next half-edge, while the origin vertex of h is the destination vertex of its previous half-edge.

The record of a vertex v stores a pointer, *he*, to an arbitrary half-edge whose origin vertex is v. It also contains fields corresponding to the p, n, c, o, and t attributes. The edge record of an edge e contains two pointers, *h1* and *h2*, one for each half-edge of e. It also contains a pointer to a record of *lte*. The face record of a face τ stores a pointer, *he*, to one of the three half-edges of its boundary. The half-edge record of a half-edge h contains a pointer, *or*, to its origin vertex; a pointer, *pv*, to its previous half-edge; a pointer, *nx*, to its next half-edge; a pointer, *eg*, to its corresponding edge; and a pointer, *fc*, to the face it belongs to.

Our DCEL also has a procedure, called MATE(), that returns the mate of a given half-edge h by comparing a pointer to h with the pointers *h1(eg(h))* and *h2(eg(h))* of the edge $eg(h)$ to which h belongs. If h is equal to *h1(eg(h))*, then MATE() returns a pointer to *h2(eg(h))*. Otherwise, MATE() returns a pointer to *h1(eg(h))*.

Procedure COLLAPSE() in Algorithm 4.7 implements the edge contraction operation in a triangulation represented by our DCEL. If $e = [u, v]$ is the edge to be contracted during the processing of a vertex, u, then COLLAPSE() removes edges $[u, v]$, $[v, x]$, and $[v, y]$, along with faces $[u, v, x]$ and $[u, v, y]$, where x and y are the two vertices in $lk([u, v], K)$, and K is the current triangulation. In addition, COLLAPSE() replaces all edges of the form $[v, z]$, where $z \notin \{u, x, y\}$, by edges of the form $[u, z]$. Each operation in COLLAPSE() takes constant time, but there are $d_v - 3$ edge replacements. So, the time complexity of COLLAPSE() is in $\Theta(d_v)$.

Algorithm 4.7 COLLAPSE($e, K, temp$)

```
get the vertices u and v of e
h ← if or(h1(e)) ≠ u then or(h1(e)) else or(h2(e))
x ← or(pv(h))
y ← or(pv(MATE(h)))
h1 ← MATE(nx(h)) {h1 starts at x and ends at v}
h2 ← pr(MATE(h)) {h2 starts at v and ends at y}
e1 ← eg(h1) {e1 points to edge [v, x]}
e2 ← eg(h2) {e2 points to edge [v, y]}
h3 ← nx(h1)
while h3 ≠ h2 do
    or(h3) ← u
    insert h3 into temp
    h3 ← nx(MATE(h3))
end while{replace v by u}
f ← eg(pv(h)) {f is a pointer to edge [u, x] in K}
g ← eg(nx(MATE(h))) {g is a pointer to edge [u, y] in K}
h1(f) ← MATE(pv(h))
h2(f) ← h1 {the half-edge starting at u and ending at x is now a mate of h1}
h1(g) ← MATE(nx(MATE(h)))
h2(g) ← h2 {the half-edge starting at y and ending at u is now a mate of h2}
he(u) ← h2 {makes sure u points to a half-edge in the final triangulation}
remove edges e, e1, and e2, and triangles fc(h1(e)) and fc(h2(e))
```

4.6. Genus-0 surfaces

In this section, we make some observations about our algorithm with regard to triangulations of surfaces of genus 0, as it takes linear time in n_f to produce an irreducible triangulation from $\mathcal{T}$. Recall that n_f is the number of triangles in $\mathcal{T}$.

We start by noticing that if list *lte* is nonempty when the loop in lines 21-34 of Algorithm 4.2 ends (i.e., when the processing of vertex u ends), then every edge $[u, v]$ in *lte* is a non-contractible edge in the triangulation at the time. Otherwise, the c attribute of v would be zero and $[u, v]$ would have been contracted. From Lemma 5, we know that $[u, v]$ can no longer be contracted, as vertex u is trapped after being processed. Nevertheless, it is still possible that an edge, $[w, v]$, in the link of u is contracted after u is processed, causing $[u, v]$ to be removed from the resulting triangulation, as shown in Figure 13. Of course, the triangulation immediately before the contraction cannot be (isomorphic to) $\mathcal{T}_4$.

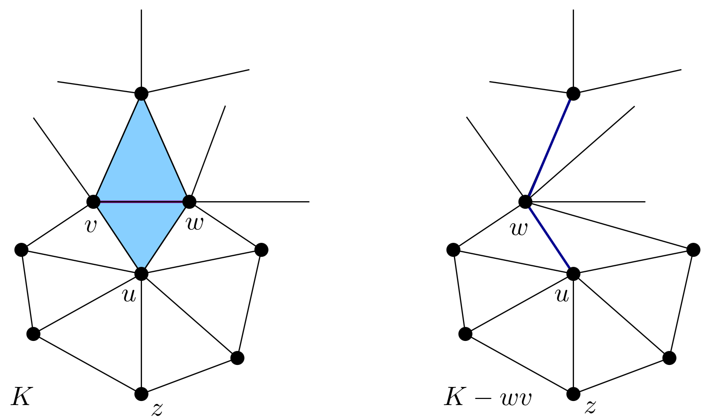

Figure 13: Vertex u is trapped, but its link may be modified by the contraction of a link edge.

Since u is trapped before the contraction of $[v, w]$, edge $[u, v]$ must belong to a critical cycle, say C, in the triangulation K to which the contraction is applied. Similarly, edge $[u, w]$ must also belong to a critical cycle, say C', in K. Both C and C' can have at most one edge in common. If they do have an edge in common, then the contraction of $[v, w]$ identifies C and C' in the resulting triangulation, $K - wv$. Otherwise, C gives rise to another critical cycle containing $[u, w]$ in $K - wv$. In either case, no critical cycle containing $[u, w]$ becomes non-critical in $K - wv$. Thus, if e is any edge incident on u when the contraction stage ends, then every critical cycle containing e immediately after u is processed belongs to or has been merged into a critical cycle in triangulation $\mathcal{T}'$.

When $\mathcal{S}$ is a genus-0 surface, the contraction stage produces a triangulation $\mathcal{T}'$ isomorphic to $\mathcal{T}_4$. No edge of $\mathcal{T}'$ is contractible, of course, but none of them can belong to a critical cycle either. So, from our previous remark, we can conclude that if u is the first vertex removed from Q in line 5 of Algorithm 4.2, then list *lte* must be empty by the time vertex u is processed. Otherwise, every edge in *lte* would be part of a critical cycle in the current triangulation, say L, at the time. But, since exactly three of those edges must be part of $\mathcal{T}'$, the critical cycles containing these three edges in L would also belong to $\mathcal{T}'$. However, this is not possible as $\mathcal{T}_4$ has no critical cycles. Hence, after vertex u is processed, no edge incident on u is part of a critical cycle in L. Since u is trapped, all those edges must be non-contractible. Thus, L must be isomorphic to $\mathcal{T}_4$, and hence *all* contractions occur during the processing of the first vertex u removed from Q.

4.7. Complexity

This section analyzes the time and space complexities of the algorithm described in the previous sections. A key feature of this algorithm is the fact that it tests an edge against the link condition at most once. If an edge is ever tested against the link condition and found to be non-contractible, the edge is stored in an auxiliary list (i.e., *lte*) and a critical cycle counter is assigned to the edge by the algorithm to keep track of the number of critical cycles containing the edge.

It turns out that maintaining the critical cycle counters of all edges in *lte* is cheaper than repeatedly testing them against the link condition. In particular, the time to update critical cycle counters is constant in lines 5-24 of Algorithm 4.3, can be charged to the time spent with the link condition test in lines 2-11 of Algorithm 4.4, and is in $\sum_{z\in\mathcal{J}_u^v}\mathcal{O}(\rho_z)$ in lines 25-41 of Algorithm 4.5, where $\mathcal{J}_u^v$ denotes the set of all vertices z in $\mathcal{T}$ such that $z \in \mathcal{T}'$, z has been processed before u, z becomes a neighbor of u after the contraction of edge $[u, v]$, and ρ_z is the degree of z in the triangulation resulting from the contraction. As we see later in the proof of Theorem 12, if $\mathcal{C}_u$ denotes the set of all vertices v in $\mathcal{T}$ such that edge $[u, v]$ is contracted during the processing of u, then

$$\sum_{u\in\mathcal{T}'}\sum_{v\in\mathcal{C}_u}\left(\mathcal{O}(d_v) + \sum_{z\in\mathcal{J}_u^v}\mathcal{O}(\rho_z)\right)$$

is an upper bound for the total time to test all edges $[u, v]$ against the link condition plus the time to update critical cycle counters in lines 25-41 of Algorithm 4.5, for every $u \in \mathcal{T}'$ and every $v \in \mathcal{C}_u$, where $\mathcal{T}'$ is the irreducible triangulation produced by the algorithm. Furthermore, the above bound can be written as $\mathcal{O}(n_v) + \mathcal{O}(g^2)$ if the genus g of the surface on which $\mathcal{T}$ is defined is positive.

Theorem 12. *Given a triangulation $\mathcal{T}$ of a surface $\mathcal{S}$, our algorithm computes an irreducible triangulation $\mathcal{T}'$ of $\mathcal{S}$ in $\mathcal{O}(g^2 + g \cdot n_f)$ time if the genus g of $\mathcal{S}$ is positive, where n_f is the number of faces of $\mathcal{T}$. If $g = 0$, the time to compute $\mathcal{T}'$ is linear in n_f. In both cases, the space required by the algorithm is linear in n_f.*

Proof. Let n_v and n_e be the number of vertices and edges of the input triangulation $\mathcal{T}$. The initialization of the algorithm (see Algorithm 4.1) takes $\mathcal{O}(n_v)$ time. Indeed, each iteration of the outer for loop in lines 2-13 takes $\Theta(d_u)$ time steps, where d_u is the degree of vertex u in $\mathcal{T}$, as line 3 and lines 7-12 require constant time each, and the inner for loop in lines 4-6 takes $\Theta(d_u)$ time. Since

$$\sum_{u \in \mathcal{T}} d_u = 2 \cdot n_e \, ,$$

the total time taken by the outer for loop in lines 2-13 is given by $\sum_{u \in \mathcal{T}} \Theta(d_u) \in \Theta(n_e)$. So,

$$\Theta(n_e) + \sum_{u \in \mathcal{T}} t_u \tag{2}$$

is the total time complexity of the algorithm, where t_u is the time taken to process vertex u in the outer while loop in lines 4-37 of Algorithm 4.2. If u does not belong to the irreducible triangulation, $\mathcal{T}'$, produced by the algorithm, then t_u is in $\Theta(1)$, as $p(u)$ is *false* immediately after u is removed from Q in line 5 of Algorithm 4.2. Consequently, we can re-write the expression in Eq. (2) as follows:

$$\begin{aligned} \Theta(n_e) + \left(\sum_{u \in \mathcal{T}, u \notin \mathcal{T}'} t_u \right) + \left(\sum_{u \in \mathcal{T}'} t_u \right) &= \Theta(n_e) + \Theta(n_v - n_v') + \left(\sum_{u \in \mathcal{T}'} t_u \right) \quad (3) \\ &= \Theta(n_e) + \mathcal{O}(n_v) + \left(\sum_{u \in \mathcal{T}'} t_u \right) \, , \end{aligned}$$

where n_v' is the number of vertices in $\mathcal{T}'$, and the meaning of '=' is that *every function of the set on the left of '=' is also a function of the set on the right of '='.*

We now restrict our attention to the time, t_u, to process vertex $u \in \mathcal{T}'$, which is the time to execute the lines 7-35 of Algorithm 4.2. Let u be any vertex of $\mathcal{T}'$. After u is removed from Q in line 5 of Algorithm 4.2, the for loop in lines 8-19 is executed. This loop iterates exactly ρ_u times, where ρ_u is the degree of u in the current triangulation, K, i.e., the triangulation at the moment that u is removed from Q. Since u has not been processed yet, we must have that $\rho_u \leq d_u$, where d_u is the degree of u in $\mathcal{T}$. This is because the degree of u can only decrease or remain the same before u is processed. This is also the case after u is processed. So, the total time spent within the for loop in lines 8-19 of Algorithm 4.2 is in $\mathcal{O}(d_u)$. The repeat-until loop in lines 21-34 of Algorithm 4.2 is executed next, and the time taken by this loop is proportional to the time spent to process all edges removed from lists *lue* and *lte*. Thus, time t_u can be bounded from above by

$$\mathcal{O}(d_u) + q_u \, , \tag{4}$$

where q_u denotes the time to process all edges ever removed from lists *lue* and *lte*.

Let $\mathcal{C}_u$ be the subset of vertices of $\mathcal{T}$ such that $v \in \mathcal{C}_u$ if and only if edge $[u, v]$ is contracted during the processing of u. Let $\mathcal{N}_u$ be the subset of vertices of $\mathcal{T}$ such that $v \in \mathcal{N}_u$ if and only if edge $[u, v]$ belongs to $\mathcal{T}_u$, where $\mathcal{T}_u$ is the triangulation resulting from the processing of u. The set $\mathcal{C}_u \cup \mathcal{N}_u$ consists of all vertices of $\mathcal{T}$ that are adjacent to u immediately before u is processed or become adjacent to u during the processing of u. Note that if v is in $\mathcal{N}_u$, then edge $[u, v]$ is non-contractible, as u is trapped in $\mathcal{T}_u$ and thus no edge incident on u can become contractible after u is processed. However, recall from Section 4.6 that $[u, v]$ may still be removed from the final triangulation, $\mathcal{T}'$. This is the case whenever an edge in the link of u and incident on v is contracted, identifying v with another vertex in the link of u (see Figure 13). So, a vertex in $\mathcal{N}_u$ is not necessarily in $\mathcal{T}'$. Note also that $\mathcal{C}_u \cap \mathcal{N}_u = \emptyset$, as each vertex in $\mathcal{C}_u$ is eliminated during the processing of u and hence cannot belong to $\mathcal{T}_u$.

To find an upper bound for q_u, we distinguish two cases: $v \in \mathcal{C}_u$ and $v \in \mathcal{N}_u$. If $v \in \mathcal{C}_u$ then edge $[u, v]$ is contracted during the processing of u. Otherwise, we know that $v \in \mathcal{N}_u$ and edge $[u, v]$ is not contracted during the processing of u (i.e., it is an edge in $\mathcal{T}_u$). Let $\mathcal{A}_u = \{v \in \mathcal{N}_u \mid v \in \mathcal{T}' \text{ and } v \text{ was processed before } u\}$ and $\mathcal{B}_u = \mathcal{N}_u - \mathcal{A}_u$. In what follows we show that

(a) For every vertex $v \in \mathcal{C}_u$, the time required to process edge $[u, v]$ is in

$$\mathcal{O}(d_v) + \sum_{z \in \mathcal{J}_u^v} \mathcal{O}(\rho_z)\,,$$

where $\mathcal{J}_u^v$ denotes the set of all vertices z in $\mathcal{T}$ such that $z \in \mathcal{T}'$, z has been processed before u, z becomes a neighbor of u after the contraction of $[u, v]$, and ρ_z is the degree of z in the triangulation resulting from the contraction.

(b) For every vertex $v \in \mathcal{A}_u$, the time required to process edge $[u, v]$ is constant.

(c) For every vertex $v \in \mathcal{B}_u$, the time required to process edge $[u, v]$ is in $\mathcal{O}(d_v)$.

Let v be any vertex in $\mathcal{C}_u$. From the definition of $\mathcal{C}_u$, we know that $[u, v]$ is contracted during the processing of u. In addition, this contraction occurs during the execution of either (i) line 26, (ii) line 28, or (iii) line 32 of Algorithm 4.2.

If (i) holds, then v is a degree-3 vertex in the triangulation, K, immediately before the contraction. Furthermore, the time required to process $[u, v]$ is proportional to the time spent by the execution of procedure Contract() (Algorithm 4.5) on $[u, v]$ and K. Since the degree ρ_v of v in K is 3, the for loop in lines 25-43 of Algorithm 4.5 is not executed, as the temporary list *temp* returned by Collapse() (Algorithm 4.7) in line 15 is empty. The for loop in lines 4-11 takes $\Theta(\rho_v)$ time, and so does the execution of Algorithm 4.7). The remaining lines of Algorithm 4.5 take constant time each. So, the time required to process edge $[u, v]$ in case (i) is in $\Theta(\rho_v) = \Theta(1)$.

If (ii) holds, then the degree ρ_v of v in the triangulation K immediately before the contraction of $[u, v]$ is greater than 3. Line 28 of Algorithm 4.1 invokes the procedure in Algorithm 4.4. The for loop in lines 2-12 of Algorithm 4.4 iterates $\Theta(\rho_v)$ times to test $[u, v]$ against the link condition. Since v is in *lue*, v is still in Q. Thus, $\rho_v \leq d_v$, where d_v is the degree of v in $\mathcal{T}$. Thus, the time spent by the for loop is in $\mathcal{O}(d_v)$. Since $[u, v]$ was contracted (by assumption), line 14 of Algorithm 4.4 is executed and Algorithm 4.5 is invoked to contract $[u, v]$.

Lines 1-24 of Algorithm 4.5 execute in $\Theta(\rho_v)$ time, including the time for executing Collapse() in line 15. The for loop in lines 25-43 of Contract() iterates $\rho_v - 3$ times, which is the length of list *temp* (i.e., the number of neighbors of v that become adjacent to u after the contraction of $[u, v]$). Lines 26-32 execute in constant time each, while the total time required to execute lines 33-41 is in $\Theta(\rho_z)$, where z is a vertex in $lk(v, K)$ whose degree in K is ρ_z. So, the total time required by Algorithm 4.5 on input $[u, v]$ and K can be bounded above by

$$\mathcal{O}(d_v) + \sum_{z \in \mathcal{J}_u^v} \Theta(\rho_z)\,.$$

Note that $\rho_z \geq d'_z$, where d'_z is the degree of z is $\mathcal{T}'$, as some edges in the link of z may still be contracted before the final triangulation $\mathcal{T}'$ is obtained. In any case, the total time required to process edge $[u, v]$ in case (ii) is bounded above by

$$\mathcal{O}(d_v) + \mathcal{O}(d_v) + \sum_{z \in \mathcal{J}_u^v} \Theta(\rho_z) = \mathcal{O}(d_v) + \sum_{z \in \mathcal{J}_u^v} \Theta(\rho_z)\,.$$

If (iii) holds, then edge $[u, v]$ was tested against the link condition before, and then inserted into *lte* after failing the test. Furthermore, $[u, v]$ is contracted either in line 5 or line 13 of Algorithm 4.6. From our discussion about case (ii), we know that the cost for testing $[u, v]$ against the link condition is in $\mathcal{O}(d_v)$, where d_v is the degree of v in $\mathcal{T}$. In turn, the cost for updating the c attribute of any vertex z such that $[u, z]$ is an edge in *lte* can be charged to the cost of the contraction or link condition test of another edge in *lue* or *lte*, as remarked below:

Remark 1. *If $[u, z]$ is in lte, then the value of $c(z)$ can be updated by either line 6 of Algorithm 4.4, line 30 of Algorithm 4.5, or lines 6-7, 15, and 20 of Algorithm 4.3. However, these lines are always executed to contract or test an edge.*

If $[u, v]$ is contracted in line 5 of Algorithm 4.6, then our discussion about case (i) tells us that the cost for contracting $[u, v]$ is constant, as v has degree 3 at the time. Consequently, the total time required to process $[u, v]$ belongs to $\mathcal{O}(d_v)$, where d_v is the degree of v in $\mathcal{T}$, which is equal to or greater than the degree of v at the time $[u, v]$ was tested against the link condition (immediately before it is inserted into list *lte*). If $[u, v]$ is contracted in line 13 of Algorithm 4.6, then the degree of v immediately before the contraction of $[u, v]$ is greater than 3. So, from our discussion about case (ii), the total time required to process $[u, v]$ is in

$$\mathcal{O}(d_v) + \sum_{z \in \mathcal{J}_u^v} \Theta(\rho_z)\,. \tag{5}$$

From cases (i), (ii), and (iii), we can conclude that Eq. 5 is an upper bound for the time required to process every edge $[u, v]$, with $v \in \mathcal{C}_u$, proving claim (a).

Now, let v be a vertex in $\mathcal{N}_u$. By definition of $\mathcal{N}_u$, we know that $[u, v]$ is in $\mathcal{T}_u$, which means that $[u, v]$ is non-contractible. If $p(v)$ is *true* by the time u is removed from Q, then $v \in \mathcal{A}_u$ and edge $[u, v]$ is not inserted into *lue* (see line 12 of Algorithm 4.2). Thus, the time to process $[u, v]$ is constant, which proves claim (b). If $p(v)$ is *false* by the time u is removed from Q, then $v \in \mathcal{B}_u$ and edge $[u, v]$ is inserted into *lue*. Since $[u, v]$ is not contracted during the processing of u, the algorithm found $[u, v]$ to be non-contractible immediately after removing $[u, v]$ from *lue*. So, either $[u, v]$ failed the link condition test or the current triangulation at the time was (isomorphic to) $\mathcal{T}_4$. If $[u, v]$ is tested against the link condition, then the time required to process $[u, v]$ is in $\mathcal{O}(d_v)$, where d_v is the degree of v in $\mathcal{T}$. While $[u, v]$ is in *lte*, the cost for updating the c attribute of $[u, v]$ is charged to the cost of the contraction or link condition test of another edge in *lue* or *lte* (see Remark 1). If $[u, v]$ is not tested against the link condition, then the time required to process $[u, v]$ is constant. So, claim (c) holds.

From our discussion above, we get

$$\begin{aligned} t_u \in \mathcal{O}(d_u) + q_u = \mathcal{O}(d_u) & \\ + \sum_{v \in \mathcal{C}_u} \left(\mathcal{O}(d_v) + \sum_{z \in \mathcal{J}_u^v} \Theta(\rho_z) \right) + \sum_{v \in \mathcal{A}_u} \Theta(1) + \sum_{v \in \mathcal{B}_u} \mathcal{O}(d_v)\,, & \end{aligned} \tag{6}$$

and thus

$$\begin{aligned} \sum_{u \in \mathcal{T}'} t_u \in \sum_{u \in \mathcal{T}'} \mathcal{O}(d_u) + \sum_{u \in \mathcal{T}'} \sum_{v \in \mathcal{C}_u} \left(\mathcal{O}(d_v) + \sum_{z \in \mathcal{J}_u^v} \Theta(\rho_z) \right) & \\ + \sum_{u \in \mathcal{T}'} \left(\sum_{v \in \mathcal{A}_u} \Theta(1) + \sum_{v \in \mathcal{B}_u} \mathcal{O}(d_v) \right). & \end{aligned} \tag{7}$$

Equation (7) can be rewritten to get rid of ρ_z, $\mathcal{A}_u$, and $\mathcal{B}_u$. Indeed, we know that if z is a vertex in $\mathcal{J}_u^v$, then the degree ρ_z of z in the triangulation $\mathcal{T}_u$ may be greater than d'_z, which is the degree of z in $\mathcal{T}'$. However, $\rho_z - d'_z$ is equal to the number of edges of the link of z that were contracted after $\mathcal{T}_u$ was obtained. So, we can charge the cost of exploring $\rho_z - d'_z$ edges in lines 34-41 of Algorithm 4.5 to the contraction of the $\rho_z - d'_z$ edges of the link of z. More specifically, if

$[w, y]$ is an edge of the link of z that got contracted during the processing of w, which occurs after processing u, then the cost of exploring $[z, y]$ in lines 34-41 of Algorithm 4.5, during the processing of z, can be absorbed by $\mathcal{O}(d_v)$ in the term

$$\sum_{v \in \mathcal{C}_w} \left(\mathcal{O}(d_v) + \sum_{z \in \mathcal{J}_w^v} \Theta(\rho_z) \right)$$

of

$$q_w \in \sum_{v \in \mathcal{C}_w} \left(\mathcal{O}(d_v) + \sum_{z \in \mathcal{J}_w^v} \Theta(\rho_z) \right) + \sum_{v \in \mathcal{A}_w} \Theta(1) + \sum_{v \in \mathcal{B}_w} \mathcal{O}(d_v) \,. \tag{8}$$

Thus,

$$\sum_{u \in \mathcal{T}'} t_u \in \sum_{u \in \mathcal{T}'} \mathcal{O}(d_u) + \sum_{u \in \mathcal{T}'} \sum_{v \in \mathcal{C}_u} \left(\mathcal{O}(d_v) + \sum_{z \in \mathcal{J}_u^v} \Theta(d'_z) \right) \tag{9}$$

$$+ \sum_{u \in \mathcal{T}'} \left(\sum_{v \in \mathcal{A}_u} \Theta(1) + \sum_{v \in \mathcal{B}_u} \mathcal{O}(d_v) \right) .$$

Note that $|\mathcal{A}_u| \leq d'_u$, where d'_u is the degree of vertex u in $\mathcal{T}'$. Then, we get

$$\sum_{u \in \mathcal{T}'} \sum_{v \in \mathcal{A}_u} 1 \leq \sum_{u \in \mathcal{T}'} d'_u = 2n'_e \quad \text{and} \quad \sum_{u \in \mathcal{T}'} \sum_{v \in \mathcal{B}_u} d_v \leq n'_v \cdot \left(\sum_{v \in \mathcal{T}} d_v \right) = 2n'_v n_e \,,$$

where n'_e is the number of edges of $\mathcal{T}'$, and consequently we can write Eq. 9 as follows:

$$\sum_{u \in \mathcal{T}'} t_u \in \sum_{u \in \mathcal{T}'} \mathcal{O}(d_u) + \sum_{u \in \mathcal{T}'} \sum_{v \in \mathcal{C}_u} \left(\mathcal{O}(d_v) + \sum_{z \in \mathcal{J}_u^v} \Theta(d'_z) \right) + \mathcal{O}(n'_e) + \mathcal{O}(n'_v \cdot n_e) \,. \tag{10}$$

Since

$$\sum_{u \in \mathcal{T}'} d_u \leq \sum_{u \in \mathcal{T}} d_u \,,$$

we can conclude that $\sum_{u \in \mathcal{T}'} \mathcal{O}(d_u)$ is in $\mathcal{O}(n_e)$. Moreover, $\mathcal{J}_u^{v_1} \cap \mathcal{J}_u^{v_2} = \emptyset$, for any two vertices v_1 and v_2 of $\mathcal{T}$ such that $[u, v_1]$ and $[u, v_2]$ were contracted during the processing of u. Indeed, a vertex z is in $\mathcal{J}_u^{v_1}$ if and only if it became adjacent to u as a result of the contraction of $[u, v_1]$. So, vertex z cannot become adjacent to vertex u as a result of the contraction of edge $[u, v_2]$. As a result, the union set

$$\bigcup_{v \in \mathcal{C}_u} \mathcal{J}_u^v$$

is a subset of the set V_u of all vertices in $lk(u, \mathcal{T}')$. As a result, we get

$$\sum_{v \in \mathcal{C}_u} \sum_{z \in \mathcal{J}_u^v} d'_z \leq \sum_{z \in V_u} d'_z \Longrightarrow \sum_{u \in \mathcal{T}'} \sum_{z \in V_u} d'_z \leq n'_v \cdot \left(\sum_{u \in \mathcal{T}'} d'_u \right) = 2n'_v n'_e \,,$$

where d'_u is the degree of u in $\mathcal{T}'$.

For any two vertices x and y in $\mathcal{T}'$, we know that $\mathcal{C}_x \cap \mathcal{C}_y = \emptyset$. Also every vertex v in $\mathcal{T}$ that is not in $\mathcal{T}'$ belongs to exactly one set $\mathcal{C}_u$, for some vertex u in $\mathcal{T}'$. So,

$$\sum_{u \in \mathcal{T}'} \sum_{v \in \mathcal{C}_u} d_v \in \sum_{v \in \mathcal{T}, v \notin \mathcal{T}'} d_v \in \mathcal{O}(n_e) \,,$$

which implies that

$$\sum_{u \in \mathcal{T}'} t_u \in \mathcal{O}(n_e) + \mathcal{O}(n_e) + \mathcal{O}(n'_v \cdot n'_e) + \mathcal{O}(n'_e) + \mathcal{O}(n'_v \cdot n_e) \,. \tag{11}$$

So, the total time required for our algorithm to produce $\mathcal{T}'$ from $\mathcal{T}$ can be given by

$$\Theta(n_e) + \mathcal{O}(n_v) + \sum_{u \in \mathcal{T}'} t_u = \Theta(n_v) + \mathcal{O}((n'_v)^2) + \mathcal{O}(n'_v \cdot n_v) \,. \tag{12}$$

If $\mathcal{S}$ is a genus-0 surface, then we know that $n'_v = 4$, which means that Eq. (12) becomes simply $\mathcal{O}(n_v)$. Otherwise, Theorem 4 tells us that $n'_v \leq 26 \cdot g - 4$, which then implies that Eq. (12) can be written in terms of g and n_v as

$$\Theta(n_v) + \mathcal{O}(g^2) + \mathcal{O}(g \cdot n_v) \,. \tag{13}$$

From our assumption that $n_f \in \Theta(n_v)$, the time complexity of our algorithm is in $\mathcal{O}(g^2 + g \cdot n_f)$ if surface $\mathcal{S}$ has a positive genus, g. Otherwise, it is in $\mathcal{O}(n_f)$. As for the space complexity of the algorithm, we note that the space required to store the augmented DCEL is in $\Theta(n_v + n_f + n_e)$. In turn, lists *lue* and *lte* require $\Theta(n_e)$ space each. Since $n_e, n_f \in \Theta(n_v)$, we can conclude that the overall space required by our algorithm on input $\mathcal{T}$ is linear in n_f. $\square$

5. Experimental results

We implemented the algorithm described in Section 4, as well as Schipper's algorithm [4] and a brute-force algorithm. The brute-force algorithm carries out two steps. First, an array with all edges of the input triangulation, $\mathcal{T}$, is shuffled. Second, each edge in the array is visited and tested against the link condition. If an edge passes the test, then it is contracted. Otherwise, it is inserted in an auxiliary array. Once the former array is empty, the edges in the auxiliary array are moved to former one, and the second step is repeated. If no edge is contracted during an execution of the second step, then the algorithm stops, as no remaining edge is contractible, which implies that the output triangulation $\mathcal{T}'$ is irreducible.

As we pointed out in Section 3, the brute-force algorithm may execute $\Omega(n_f^2)$ link condition tests (see Figure 6), where n_f is the number of triangles in $\mathcal{T}$. In what follows, we describe an experiment in which we compare the implementations of the three aforementioned algorithms against triangulations typically found in graphics applications, as well as triangulations devised to provide us with some insights regarding the behavior of our algorithm and the one by Schipper [4].

5.1. Experimental setup

All algorithms were implemented in `C++` and compiled with `clang 503.0.40` using the `-O3` option. We ran the experiments on an iMac running OSX 10.9.4 at 3.2 GHz (Intel Core i3 — 1 processor and 2 cores), with 256KB of level-one data cache, 4MB of level-two cache, and 8GB of RAM. The implementations are based on the same data structure for surface triangulations (i.e., the augmented DCEL described in Section 4.5), and they do not depend on any third-party libraries[3].

Time measurements refer to the time to compute the irreducible triangulations only (i.e., we did not take into account the time to read in the triangulations from a file and create a DCEL representation in main memory). In particular, each implementation has a function, named `run()`, that computes an irreducible triangulation from a given pointer to the augmented DCEL containing the input triangulation. We only measured the time spent by function `run()`.

To time and compare the implementations, we considered four groups of triangulations. The first group consists of small genus triangulations typically found in graphics papers (see Table 2). The second group consists of 10 triangulations of the same genus-0, brick-shaped surface with 3,844 cavities (see Figure 14). Each triangulation has a distinct number of triangles (see Table 3). The third group consists of 8 triangulations of a brick-shaped surface with 3,500 holes (see Table 4). This surface was obtained from the one in the second group by replacing 3,500 cavities with holes. Finally, the fourth group consists of 10 triangulations of surfaces with varying genus (see Table 5). The triangulations have about the same number of triangles, and the surfaces were also obtained from the ones in the second group by replacing a certain number of cavities with holes.

Triangulations in the first group were chosen to evaluate the performance of the three algorithms on data typically used by mesh simplification algorithms [31].

Recall that the time complexity of both our algorithm and the one given by Schipper [4] is dictated by two parameters: the number of triangles and the genus of the input triangulation. When the genus is zero, the time upper bound we derived for our algorithm depends solely and linearly on the number of triangles (see Section 4.6). Triangulations in the second group were chosen

[3] `http://www.mat.ufrn.br/∼mfsiqueira/Marcelo_Siqueiras_Web_Spot/Software.html`.

to evaluate the performance of the three algorithms on triangulations of genus 0 surfaces.

Triangulation	# Vertices	# Edges	# Triangles	# Genus
Armadillo	171,889	515,661	342,774	0
Botijo	20,000	60,024	40,016	5
Casting	5,096	15,336	10,224	9
Eros	197,230	591,684	394,456	0
Fertility	19,994	60,000	40,000	4
Filigree	29,129	87,771	58,514	65
Hand	195,557	586,665	391,110	0
Happy Buddha	543,652	1,631,574	1,087,716	104
Iphigenia	351,750	1,055,268	703,512	4
Socket	836	2,544	1,696	7

Table 2: Euler characteristics of the triangulations in the first group.

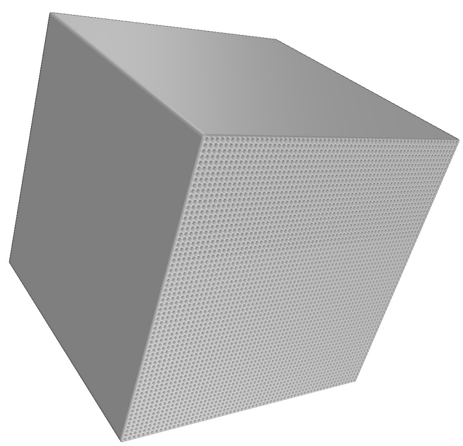

Figure 14: A brick-shaped surface with 3,844 cavities.

Triangulations in the third and fourth groups were chosen to evaluate the influence of both parameters (i.e., genus and number of triangles) separately. Triangulations in the third group have the same genus (i.e, 3,500), but their numbers of triangles vary, which allowed us to evaluate the influence of the number of triangles over the performances of our algorithm and Schipper's algorithm. In turn, triangulations in the fourth group have about the same number of triangles, but their genuses vary, which allowed us to evaluate the influence of the genus over the performances of our algorithm and Schipper's algorithm.

Triangulation	# Vertices	# Edges	# Triangles
B0	2,097,150	6,291,444	4,194,296
B1	1,097,150	3,291,444	2,194,296
B2	597,150	1,791,444	1,194,296
B3	297,150	891,444	594,296
B4	147,150	441,444	294,296
B5	72,150	216,444	144,296
B6	34,650	103,944	69,296
B7	15,900	47,694	31,796
B8	8,400	25,194	16,796
B9	4,400	13,194	8,796

Table 3: Euler characteristics of the triangulations in the second group.

Triangulation	# Vertices	# Edges	# Triangles
C0	2,104,150	6,333,444	4,222,296
C1	1,104,150	3,333,444	2,222,296
C2	604,150	1,833,444	1,222,296
C3	354,150	1,083,444	722,296
C4	179,150	558,444	372,296
C5	91,650	295,444	197,296
C6	41,650	145,944	97,296
C7	16,650	70,994	47,796

Table 4: Euler characteristics of the triangulations in the third group (their genus is 3,500).

Triangulation	# Vertices	# Edges	# Triangles	# Genus
D0	2,104,150	6,333,444	4,222,296	3,500
D1	2,103,150	6,327,444	4,218,296	3,000
D2	2,102,150	6,321,444	4,214,296	2,500
D3	2,101,150	6,315,444	4,210,296	2,000
D4	2,100,150	6,309,444	4,206,296	1,500
D5	2,099,150	6,303,444	4,202,296	1,000
D6	2,098,150	6,297,444	4,198,296	500
D7	2,097,350	6,292,644	4,195,096	100
D8	2,097,250	6,292,044	4,194,696	50
D9	2,097,170	6,291,564	4,194,376	10

Table 5: Euler characteristics of the triangulations in the fourth group.

5.2. Results

From now on, we denote our algorithm, Schipper's algorithm and the brute-force algorithm by **RS**, **S**, and **BF**, respectively. We initially ran **RS** and **S** exactly once on each triangulation of the first group (see Table 2), while **BF** was executed ten times on each triangulation (since we randomized the input by shuffling the edges of the triangulations). So, for **BF**, we computed and recorded the average execution times over the ten runs on each triangulation. A plot of the *time* (in seconds) taken by the three algorithms on every input triangulation versus the *number of triangles* of the triangulations is shown in Figure 15.

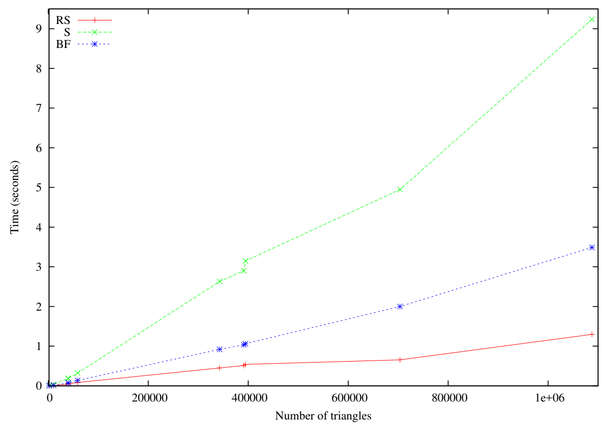

Figure 15: Runtimes for the execution of **RS**, **S**, and **BF** on the first group triangulations.

As we can see in Figure 15, the larger the number of triangles, the larger the ratios t_s/t_{rs} and t_{bf}/t_{rs}, where t_{rs}, t_s, and t_{bf} are the times taken by **RS**, **S**, and **BF**, respectively. In particular, t_s/t_{rs} and t_{bf}/t_{rs} are equal to 2.13 and 1.14 for the triangulation with the smallest number of triangles (i.e., `Socket`), and equal to 7.55 and 3.05 for the triangulation with the largest number of triangles (i.e., `Iphigenia`). Observe that **BF** outperforms **S**. In particular, t_s/t_{bf} is always greater than 1.9 and its largest value is 2.97, which is attained on triangulation `Eros`.

We repeated the experiment for the triangulations in the second group (see Table 3). All triangulations in this group have genus-0. In particular, for every $i = 1, \ldots, 9$, triangulation Bi was obtained from triangulation B$i-1$ by a simplification process that approximately halved the number of triangles of B$i-1$. A plot of the *time* (in seconds) taken by **RS**, **S**, and **BF** on triangulations B0-B9 as a function of the *number of triangles* of the triangulations is shown in Figure 16.

Note that **RS** outperforms **S** and **BF**, and **BF** outperforms **S**. However, this time, ratio t_s/t_{rs} gets smaller as the number of triangles grows. In particular, t_s/t_{rs} is equal to 12.28 for triangulation B9 and equal to 3.56 for triangulation B0. Ratio t_{bf}/t_{rs} presents the same behavior, but it becomes noticeable only for triangulations B0-B3, for which the numbers of triangles exceed 500,000.

Triangulations C0-C7 in Table 4 have a fixed, large genus (i.e, 3,500). In addition, C1-C7 were built as follows: for every $i = 1, \ldots, 7$, triangulation Ci was obtained from triangulation C$i{-}1$ by a simplification process that approximately halved the number of triangles of C$i - 1$. The triangulations in the third group were designed to compare the performances of **RS**, **S**, and **BF** on variable-size triangulations of the same fixed, large genus surface (as opposed to the same genus-0 surface like we did before for the triangulations in the second group). A plot of the *time* (in seconds) taken by **RS**, **S**, and **BF** on triangulations C0-C7 as a function of the *number of triangles* of the triangulations is shown in Figure 17.

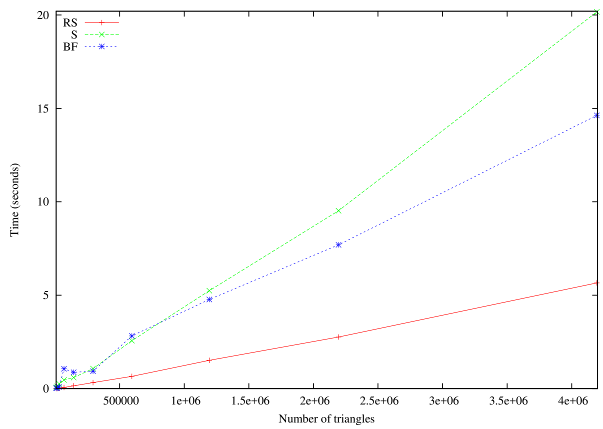

Figure 16: Runtimes for the execution of **RS**, **S**, and **BF** on the second group triangulations.

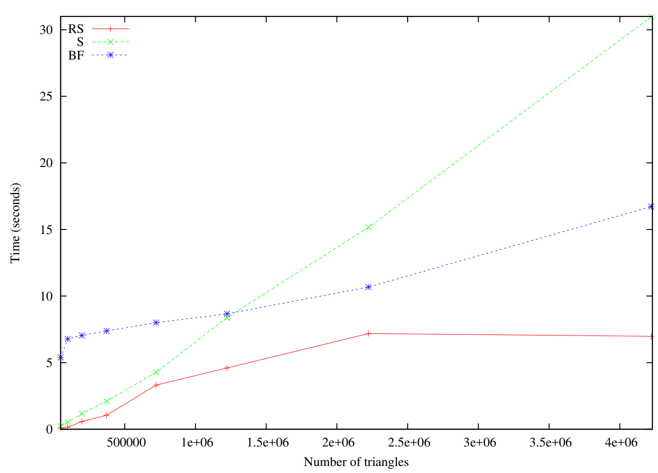

Figure 17: Runtimes for the execution of **RS**, **S**, and **BF** on the third group triangulations.

Once again **RS** outperformed **S** and **BF**, but **BF** outperforms **S** only for C0 and C1, which are the triangulations with the largest number of triangles. Furthermore, contrary to the results obtained from the triangulations in the second group, ratio t_s/t_{rs} gets larger as the number of triangles grows. This is also the case for ratio t_{bf}/t_{rs}, but the behavior can only be noticed for triangulations C0, C1, and C2, which are the ones whose number of triangles is greater than 500,000.

Finally, we ran **RS**, **S**, and **BF** on triangulations D0-D9 of the fourth group. These triangulations have nearly the same number of triangles, but their genus varies from 10 to 3,500 (see Table 5). A plot of the *time* (in seconds) taken by **RS**, **S**, and **BF** on triangulations D0-D9 as a function of the *genus* of the triangulations is shown in Figure 18. Observe that **RS** outperforms **S** and **BF**, and **BF** outperforms **S** for all triangulations. Ratio t_s/t_{rs} gets larger as the genus grows. Its maximum value is 4.7, and is attained for triangulation D0. Unlike, ratio t_{bf}/t_{rs} gets slightly smaller as the genus grows, and it is basically constant and about 2.4 for triangulations D0-D3 whose genuses are greater than 1,999.

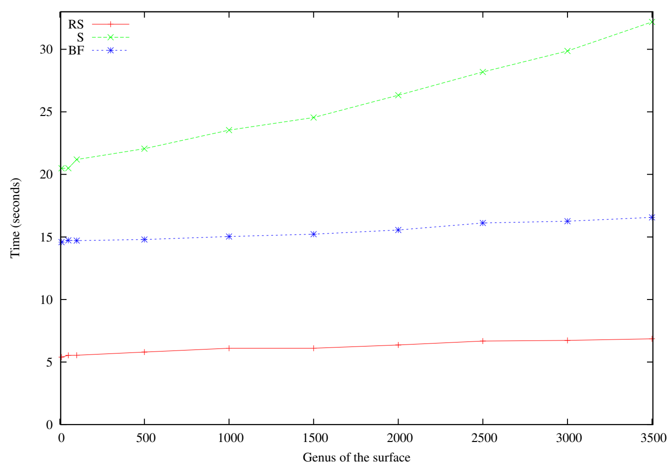

Figure 18: Runtimes for the execution of **RS**, **S**, and **BF** on the fourth group triangulations.

5.3. Discussion

To properly analyze the results in Section 5.2, we take into account the *number of link condition tests* carried out by each algorithm, as well as the *number of edges tested more than once* by **S** and **BF**. We denote the number of link condition tests carried out by **RS** (resp. **S** and **BF**) by ℓ_{rs} (resp. ℓ_s and ℓ_{bf}), the number of edges tested more than once by **S** (resp. **BF**) and by ϵ_s (resp. ϵ_{bf}). In particular, ℓ_{bf} and ϵ_{bf} are the average values over the ten runs of **BF**.

Although **RS** outperforms both **S** and **BF** for the triangulations in the second group, ratios t_s/t_{rs} and t_{bf}/t_{rs} get smaller as the number n_f of triangles of the input triangulations grows. The main reason is that the probability that a randomly chosen edge from a genus-0 surface triangulation is contractible increases as n_f gets larger (and the surface is kept fixed). So, ratios ℓ_s/ℓ_{rs} and ℓ_{bf}/ℓ_{rs} decrease as n_f gets larger, causing t_s/t_{rs} and t_{bf}/t_{rs} to decay, as we can see in Tables 6-7. Note also that $\ell_{bf} > \ell_s$ and $\epsilon_{bf} > \epsilon_s$, for all triangulations B0-B9. Moreover, ϵ_s is very small and does not scale up with n_f. Even so, we get $t_s > t_{bf}$.

Triangulation	$\boldsymbol{\ell}_{rs}$	$\boldsymbol{\ell}_s$	$\boldsymbol{\ell}_{bf}$	$\boldsymbol{\epsilon}_s$	$\boldsymbol{\epsilon}_{bf}$
B0	2,079,539	2,097,158	3,406,091.5	6	19,011.7
B1	1,085,325	1,097,194	1,783,418.4	9	10,034.9
B2	588,335	597,196	968,511.9	7	5,495.3
B3	289,520	297,174	479,536.2	8	2,863.7
B4	145,338	147,193	232,984.4	7	1,482.1
B5	68,454	72,190	111,673.8	7	816.4
B6	29,827	34,675	50,936.6	6	458.9
B7	14,085	15,908	18,838.2	6	292.5
B8	7,979	8,408	9,941.4	6	159.4
B9	4,063	4,408	5,246.4	6	92.1

Table 6: The number of link condition tests and the number of edges tested more than once obtained from the executions of **RS**, **S**, and **BF** on the triangulations B0-B9 in the second group.

Triangulation	$\boldsymbol{\ell}_s/\boldsymbol{\ell}_{rs}$	$\boldsymbol{\ell}_{bf}/\boldsymbol{\ell}_{rs}$	$\boldsymbol{\ell}_s/\boldsymbol{\ell}_{bf}$	$\boldsymbol{t}_s/\boldsymbol{t}_{rs}$	$\boldsymbol{t}_{bf}/\boldsymbol{t}_{rs}$	$\boldsymbol{t}_s/\boldsymbol{t}_{bf}$
B0	1.01	1.64	0.62	3.56	2.58	1.38
B1	1.01	1.65	0.62	3.45	2.79	1.24
B2	1.02	1.65	0.62	3.46	3.15	1.10
B3	1.03	1.66	0.62	3.90	4.30	0.91
B4	1.01	1.60	0.63	3.32	2.88	1.15
B5	1.05	1.63	0.65	3.96	5.86	0.67
B6	1.16	1.71	0.68	7.37	16.90	0.44
B7	1.13	1.34	0.84	11.51	3.76	3.06
B8	1.05	1.25	0.85	10.81	3.35	3.23
B9	1.08	1.29	0.84	12.28	4.15	2.96

Table 7: Ratios between the number of link condition tests performed by **RS**, **S**, and **BF** on the triangulations B0-B9 in the second group, and their corresponding execution time ratios.

The fact that ϵ_s is very small and does not scale up with n_f is due to the strategy used by **S** to find a contractible edge: first, a vertex of lowest degree in the current triangulation K is chosen, and then a contractible edge incident on this vertex is found. Since the genus of the surface is 0, the lowest degree vertex of K is very likely to have degree 3 or 4. If K is not (isomorphic to) $\mathcal{T}_4$ already, then every edge incident on a degree-3 vertex is contractible in K (see Proposition 6), and hence most edges are tested against the link condition by **S** only once. In particular, for the cases in which $\epsilon_s = 6$, only the six edges of the final irreducible triangulation, which is isomorphic to $\mathcal{T}_4$, were tested more than once. **S** chooses a lowest degree vertex from K in $\mathcal{O}(\lg m)$ time, where m is the number of (loose) vertices in K. Our experiments indicate that the $\mathcal{O}(\lg m)$ cost cancels out the gain obtained by reducing the values of ℓ_s and ϵ_s.

For triangulations C0-C7, which have a fixed genus of 3,500 and variable-size, the scenario regarding ℓ_s, ℓ_{bf}, ϵ_s, and ϵ_{bf} is quite the opposite to the one for the genus-0 triangulations, B0-B9 (see Table 8). The reason is that the larger the genus is the smaller the probability that a randomly chosen edge from any of C0-C7 is contractible. Furthermore, this probability decreases even further as n_f gets smaller.

Triangulation	$\boldsymbol{\ell_{rs}}$	$\boldsymbol{\ell_s}$	$\boldsymbol{\ell_{bf}}$	$\boldsymbol{\epsilon_s}$	$\boldsymbol{\epsilon_{bf}}$
C0	3,499,517	3,336,889	3,595,222.5	346,735	52,815.0
C1	1,846,458	1,780,868	1,967,691.3	200,636	49,757.9
C2	1,018,801	1,006,051	1,131,182.2	128,339	48,063.1
C3	605,195	615,112	742,667.4	91,493	47,180.6
C4	318,114	347,941	459,221.8	66,975	46,630.6
C5	171,599	209,767	316,699.6	54,353	46,300.9
C6	90,867	135,245	232,041.0	47,094	46,105.7
C7	54,594	100,181	188,700.6	45,856	46,000.5

Table 8: The number of link condition tests and the number of edges tested more than once obtained from the executions of **RS**, **S**, and **BF** on the triangulations C0-C7 in the third group.

The fact that $\epsilon_s > \epsilon_{bf}$, for all triangulations in the third group, but C7, tells us that a few low degree vertices are chosen over and over again by **S** before they become trapped, and several edges incident on these vertices are tested more than once against the link condition and failed the test. So, the strategy adopted by **S** is not so effective when the genus of the surface is large. Moreover, as we can see in Tables 8 and 9, we have $\ell_{bf} > \ell_s$, for all triangulations C0-C7, and ℓ_{bf}/ℓ_s gets smaller as n_f gets larger, varying from 1.88 (for C7) to 1.08 (for C0). The larger values of ℓ_{bf}/ℓ_s for C2-C7 explain why **S** outperforms **BF** on C2-C7 (see Figure 17).

It is worth mentioning that the number of link condition tests carried out by **RS** (i.e., ℓ_{rs}) is larger than the number of link condition tests carried out by **S** (i.e., ℓ_s) for triangulations C0, C1, and C2, which are the ones with a larger number, n_f, of triangles. Even so, ratio t_s/t_{rs} gets larger as n_f gets larger for C0, C1, and C2. This fact indicates that the $\mathcal{O}(\lg m)$ cost for choosing a lowest degree vertex from the set of m loose vertices of the current triangulation cancels out the gain obtained by **S** by executing a smaller number of link condition tests.

Since ϵ_s is large (compared to the genus-0 scenario), the value of m decreases more slowly, which increases the overall cost of picking lowest degree vertices. Also, the overall cost of testing a set of edges in **RS** is smaller than the overall cost of testing the same set of edges in **S** (see discussions in Section 4.2 and Section 3).

Triangulation	$\boldsymbol{\ell}_s/\boldsymbol{\ell}_{rs}$	$\boldsymbol{\ell}_{bf}/\boldsymbol{\ell}_{rs}$	$\boldsymbol{\ell}_s/\boldsymbol{\ell}_{bf}$	$\boldsymbol{t}_s/\boldsymbol{t}_{rs}$	$\boldsymbol{t}_{bf}/\boldsymbol{t}_{rs}$	$\boldsymbol{t}_s/\boldsymbol{t}_{bf}$
C0	0.95	1.03	0.93	4.44	2.39	1.85
C1	0.96	1.07	0.91	2.11	1.49	1.42
C2	0.99	1.11	0.89	1.82	1.89	0.97
C3	1.02	1.23	0.83	1.29	2.41	0.53
C4	1.09	1.44	0.76	2.00	7.02	0.29
C5	1.22	1.85	0.66	2.05	12.26	0.17
C6	1.49	2.55	0.58	3.41	43.97	0.08
C7	1.84	3.46	0.53	2.61	58.88	0.04

Table 9: Ratios between the number of link condition tests performed by **RS**, **S**, and **BF** on the triangulations C0-C7 in the third group, and their corresponding execution time ratios.

The experiment with triangulations D0-D9 indicates that the performance of **S** decreases as the genus of the triangulation gets larger and the number of triangles is kept about the same. As we can see in Table 10, the values of ϵ_s scales up very quickly with the genus, which is not the case for the values of ϵ_{bf}. This observation tells us that an edge chosen by **S** to be tested against the link condition is much more likely to be non-contractible than an edge chosen at random (as it is the case in **BF**). As we pointed out before, those "bad" choices increase the time **S** takes to choose a lowest degree vertex and a contractible edge incident on it. This is why t_s/t_{rs} and t_s/t_{bf} get larger as the triangulation genus grows, despite the fact that ratios ℓ_s/ℓ_{rs} and ℓ_s/ℓ_{bf} get smaller (see Table 11). Moreover, since n_t is large and about the same for triangulations D0-D9, the values of ϵ_{bf}/n_e for D4-D9, where n_e is the number of edges of the triangulation, are much smaller than the ones for triangulations C2-C7. Thus, contrary to what we observe for C2-C7, **BF** outperforms **S** on the smallest genus triangulations, D4-D9, as well (see Figure 18).

Finally, observe that the experiments with triangulations B0-B9 corroborates the fact that **RS** runs in linear time in n_f for triangulations of genus-0 surfaces (see Figure 16). Likewise, the experiments with triangulations D0-D9 indicates that the runtime of **RS** is proportional to the term $g \cdot n_f$. To see that, we computed $\delta f_i = (n_{f_i} - n_{f_9})/n_{f_9}$, $\delta g_i = (g_i - g_9)/g_9$, and $\delta t_i = (t_i - t_9)/t_9$, for every $i = 8, \ldots, 0$, where n_{f_j} and g_j are the number of triangles and the genus of triangulation Dj, respectively, and t_j is the time taken by **RS** on triangulation Dj, for every $j = 0, \ldots, 9$. Then, we verified that $\delta t_i/(\delta f_i \cdot \delta g_i)$ is approximately constant for triangulations D0-D4, which have genus greater than or equal to 1,500.

Triangulation	$\boldsymbol{\ell}_{rs}$	$\boldsymbol{\ell}_{s}$	$\boldsymbol{\ell}_{bf}$	$\boldsymbol{\epsilon}_{s}$	$\boldsymbol{\epsilon}_{bf}$
D0	3,499,517	3,336,889	3,585,933.5	346,735	52,866.6
D1	3,298,209	3,161,516	3,576,091.5	297,628	47,936.2
D2	3,095,251	2,981,813	3,554,938.0	247,822	43,124.9
D3	2,895,440	2,804,562	3,522,016.4	197,884	38,400.2
D4	2,693,425	2,628,433	3,495,457.5	148,587	33,570.1
D5	2,493,458	2,451,622	3,466,912.2	99,075	28,706.5
D6	2,293,492	2,274,066	3,436,328.4	49,415	23,811.3
D7	2,129,696	2,132,148	3,412,062.2	9,718	19,991.1
D8	2,110,987	2,115,194	3,409,289.6	4,948	19,357.0
D9	2,087,627	2,100,756	3,406,560.1	1,019	19,072.6

Table 10: The number of link condition tests and the number of edges tested more than once obtained from the executions of **RS**, **S**, and **BF** on the triangulations D0-D9 in the fourth group.

Triangulation	$\boldsymbol{\ell}_s/\boldsymbol{\ell}_{rs}$	$\boldsymbol{\ell}_{bf}/\boldsymbol{\ell}_{rs}$	$\boldsymbol{\ell}_s/\boldsymbol{\ell}_{bf}$	$\boldsymbol{t}_s/\boldsymbol{t}_{rs}$	$\boldsymbol{t}_{bf}/\boldsymbol{t}_{rs}$	$\boldsymbol{t}_s/\boldsymbol{t}_{bf}$
D0	0.95	1.02	0.93	4.69	2.42	1.94
D1	0.96	1.08	0.88	4.44	2.42	1.84
D2	0.96	1.15	0.84	4.22	2.41	1.75
D3	0.97	1.22	0.80	4.14	2.45	1.69
D4	0.98	1.30	0.75	4.02	2.49	1.61
D5	0.98	1.39	0.71	3.86	2.46	1.57
D6	0.99	1.50	0.66	3.80	2.55	1.49
D7	1.00	1.60	0.62	3.82	2.65	1.44
D8	1.00	1.62	0.62	3.70	2.66	1.39
D9	1.01	1.63	0.62	3.80	2.70	1.40

Table 11: Ratios between the number of link condition tests performed by **RS**, **S**, and **BF** on the triangulations D0-D9 in the fourth group, and their corresponding execution time ratios.

6. Conclusions

We presented a new algorithm for computing an irreducible triangulation $\mathcal{T}'$ from a given triangulation $\mathcal{T}$ of a connected, oriented, and compact surface $\mathcal{S}$ in $\mathbb{E}^d$ with empty boundary. If the genus g of $\mathcal{S}$ is positive, then $\mathcal{T}'$ can be computed in $\mathcal{O}(g^2 + g\,n_f)$ time, where n_f is the number of triangles in $\mathcal{T}$. Otherwise, $\mathcal{T}'$ is computed in linear time in n_f. In both cases, the space required by the algorithm is in $\Theta(n_f)$. The time upper bound derived in this paper improves upon the previously best known upper bound in [4] by a $\lg n_f/g$ factor.

We also implemented our algorithm, the algorithm given by Schipper in [4], and a randomized, brute-force algorithm, and then experimentally compared these implementations on triangulations typically found in graphics applications, as well as triangulations specially devised to study the runtime of the algorithms in extreme scenarios. Our algorithm outperformed the other two in all case studies, indicating that the key ideas we use to reduce the worst-case time complexity of our algorithm are also effective in the average case and for triangulations typically encountered in practice. Our experiments also indicated that the key ideas behind Schipper's algorithm are not very effective for the same type of data, as his algorithm was outperformed by the brute-force one.

In the description of our algorithm, we required $\mathcal{S}$ be orientable, as the augmented DCEL we used in the implementation of the algorithm does not support nonorientable surfaces. However, our algorithm also works for nonorientable surfaces within the same time bounds, as Theorem 4 is stated in terms of the Euler genus of $\mathcal{S}$, which is half the value of the (usual) genus g for orientable surfaces. We have not yet extended our algorithm to deal with compact surfaces with a nonempty boundary either. A starting point towards this extension is the very recent work by Boulch, de Verdière and Nakamoto [3], which gives an analogous result to that of Theorem 4 for (non-orientable) surfaces with a nonempty boundary.

We are interested in investigating the possibility of lowering the $\mathcal{O}(g^2 + g\,n_f)$ upper bound, so that the g^2 is replaced with g and the bound becomes $\mathcal{O}(g\,n_f)$. If this is possible, is the resulting bound tight? Another important research venue is the development of a fast algorithm for generating the complete set of *all* irreducible triangulations of a surface from a given triangulation of the surface. We are interested in developing such an algorithm by using ours as a building block, providing an alternative method to that of Sulanke [5].

It would be interesting to find out whether some ideas behind our algorithm could speed up some topology-preserving, mesh simplification algorithms [31]. In particular, one can devise a parallel version of our algorithm to take advantage of the increasingly popular and powerful graphics processing units (GPU). The idea is to process a few vertices of the input triangulation at a time, rather than only one vertex. Theorem 4 can be used to give us an idea of the size of the initial set of vertices. The algorithm must handle the case in which the same vertex becomes a neighbor of two currently processed vertices. At this point, an edge contraction could make two currently processed vertices neighbors of each other. If their common edge is contractible, then one of two vertices can be removed from the current triangulation by contracting the edge. This parallel algorithm can efficiently build a hierarchy of triangulations such as the one in [35].

7. Acknowledgments

All triangulations in Table 2 were obtained from publicly available triangle mesh repositories. Namely, triangulations `Armadillo`, `Botijo`, `Casting`, `Eros`, `Fertility`, `Filigree`, `Hand`, and `Socket` were taken from the Aim@Shape Repository, triangulation `Happy Buddha` was taken from the Large Geometric Models Archive, and triangulation `Iphigenia` was taken from the website of the book in [39].

References

[1] J. Gallier, D. Xu, A Guide to the Classification Theorem for Compact Surfaces, Springer-Verlag, 2013. 1, 2, 2, 2

[2] D. W. Barnette, A. L. Edelson, All 2-manifolds have finitely many minimal triangulations, Israel Journal of Mathematics 67 (1) (1989) 123–128. doi:10.1007/BF02764905. 1

[3] A. Boulch, Éric Colin de Verdière, A. Nakamoto, Irreducible triangulations of surfaces with boundary, Graphs and Combinatorics 29 (6) (2013) 1675–1688. doi:10.1007/s00373-012-1244-1. 1, 6

[4] H. Schipper, Generating triangulations of 2-manifolds, in: H. Bieri, H. Noltemeier (Eds.), Computational Geometry: Methods, Algorithms and Applications, Vol. 553 of Lecture Notes in Computer Science, Springer, 1991, pp. 237–248. doi:10.1007/3-540-54891-2_18. 1, 1.1, 1.3, 3, 3, 4, 5, 5, 5.1, 6

[5] T. Sulanke, Generating irreducible triangulations of surfaces, CoRR arXiv:math/0606687. 1, 6

[6] D. Archdeacon, C. P. Bonnington, J. A. Ellis-Monaghan, How to exhibit toroidal maps in space, Discrete & Computational Geometry 38 (3) (2007) 573–594. doi:10.1007/s00454-007-1354-3. 1

[7] C. P. Bonnington, A. Nakamoto, Geometric realization of a triangulation on the projective plane with one face removed, Discrete & Computational Geometry 40 (1) (2008) 141–157. doi:10.1007/s00454-007-9035-9. 1

[8] S. Negami, Diagonal flips in triangulations of surfaces, Discrete Mathematics 135 (1-3) (1994) 225–232. doi:http://dx.doi.org/10.1016/0012-365X(93)E0101-9. 1

[9] S. Negami, Diagonal flips of triangulations on surfaces, a survey, The Yokohama Mathematical Journal 47 (1999) 1–40. doi:http://dx.doi.org/10.1016/j.jctb.2008.06.006. 1

[10] C. Cortés, C. Grima, A. Marquez, A. Nakamoto, Diagonal flips in outer-triangulations on closed surfaces, Discrete Mathematics 254 (1-3) (2002) 63–74. doi:http://dx.doi.org/10.1016/S0012-365X(01)00353-3. 1

[11] K. Kawarabayashi, A. Nakamoto, Y. Suzuki, N-flips in even triangulations on surfaces, Journal of Combinatorial Theory, Series B 99 (1) (2009) 229–246. doi:http://dx.doi.org/10.1016/j.jctb.2008.06.006. 1

[12] Y. Higuchi, A. Nakamoto, K. Ota, T. Sakuma, N-flips in even triangulations on the torus and dehn twists preserving monodromies, Discrete Mathematics 311 (13) (2011) 1128–1135, selected Papers from the 22nd British Combinatorial Conference. doi:http://dx.doi.org/10.1016/j.disc.2010.08.003. 1

[13] B. Chen, S. Lawrencenko, Structural characterization of projective flexibility, Discrete Mathematics 188 (1-3) (1998) 233–238. doi:http://dx.doi.org/10.1016/S0012-365X(98)00052-1. 1

[14] V. Dujmović, G. Fijavž, G. Joret, T. Sulanke, D. R. Wood, On the maximum number of cliques in a graph embedded in a surface, European Journal of Combinatorics 32 (8) (2011) 1244–1252. doi:http://dx.doi.org/10.1016/j.ejc.2011.04.001. 1

[15] A. Nakamoto, K. Ota, Note on irreducible triangulations of surfaces, Journal of Graph Theory 20 (2) (1995) 227–233. doi:10.1002/jgt.3190200211. 1

[16] M. Jungerman, G. Ringel, Minimal triangulations on orientable surfaces, Acta Mathematica 145 (1) (1980) 121–154. doi:10.1007/BF02414187. 1, 1.1

[17] E. Steinitz, H. Radamacher, Vorlesungen über die Theorie der Polyder, Springer, 1934. 1

[18] S. Lavrenchenko, Irreducible triangulations of the torus, Journal of Soviet Mathematics 51 (5) (1990) 2537–2543. doi:10.1007/BF01104169. 1

[19] D. Barnette, Generating the triangulations of the projective plane, Journal of Combinatorial Theory, Series B 33 (3) (1982) 222–230. doi:http://dx.doi.org/10.1016/0095-8956(82)90041-7. 1, 2

[20] T. Sulanke, Note on the irreducible triangulations of the klein bottle, Journal of Combinatorial Theory, Series B 96 (6) (2006) 964–972. doi:http://dx.doi.org/10.1016/j.jctb.2006.05.001. 1

[21] T. Lemos, S. Ramaswami, M. Siqueira, An experimental comparison of algorithms for converting triangulations of closed surfaces into quadrangulations, *in preparation*. 1.2

[22] D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. Silva, M. Tarini, D. Zorin, Quad-mesh generation and processing: A survey, Computer Graphics Forum 32 (6) (2013) 51–76. 1.2

[23] K. Diks, P. Stanczyk, Perfect matching for biconnected cubic graphs in $\mathcal{O}(n \lg^2 n)$ time, in: J. van Leeuwen, A. Muscholl, D. Peleg, J. PokornÃ¡, B. Rumpe (Eds.), SOFSEM 2010: Theory and Practice of Computer Science, Vol. 5901 of Lecture Notes in Computer Science, Springer-Verlag, Germany, 2010, pp. 321–333. doi:10.1007/978-3-642-11266-9_27. 1.2

[24] K. Pulli, M. E. Segal, Fast rendering of subdivision surfaces, in: Proceedings of the Eurographics Workshop on Rendering Techniques, 1996, pp. 61–70. 1.2

[25] L. Velho, Quadrilateral meshing using 4-8 clustering, in: Proceedings of the Symposium on Mesh Generation and Self-Adaptivity (CILANCE 2000), 2000, pp. 61–64. 1.2

[26] J. L. Gross, T. W. Tucker, Topological graph theory, Dover Publications, Inc., Mineola, NY, USA, 2001. 2, 2

[27] L. Guibas, J. Stolfi, Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams, ACM Transactions on Graphics 4 (2) (1985) 74–123. doi:10.1145/282918.282923. 2, 2

[28] T. K. Dey, H. Edelsbrunner, S. Guha, D. V. Nekhayev, Topology preserving edge contraction, Publications de l'Institut Mathematique (Beograd) 60 (80) (1999) 26–45. 2

[29] G. Joret, D. R. Wood, Irreducible triangulations are small, Journal of Combinatorial Theory, Series B 100 (5) (2010) 446–455. doi:http://dx.doi.org/10.1016/j.jctb.2010.01.004. 2, 4

[30] T. Sulanke, Irreducible triangulations of low genus surfaces, CoRR arXiv:math/0606690. 2

[31] D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson, R. Huebner, Level of Detail for 3D Graphics, Morgan Kaufmann Publishers, 2003. 3, 5.1, 6

[32] M. Garland, P. S. Heckbert, Surface simplification using quadric error metrics, in: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1997, pp. 209–216. doi:10.1145/258734.258849. 3

[33] H. Hoppe, Progressive meshes, in: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '96, ACM, New York, NY, USA, 1996, pp. 99–108. doi:10.1145/237170.237216. 3

[34] L. Velho, Mesh simplification using four-face clusters, in: Proceedings of the International Conference on Shape Modeling & Applications, SMI '01, IEEE Computer Society, Washington, DC, USA, 2001, pp. 200–208. doi:10.1109/SMA.2001.923391. 3

[35] S.-W. Cheng, T. K. Dey, S.-H. Poon, Hierarchy of surface models and irreducible triangulations, Computational Geometry: Theory and Applications 27 (2) (2004) 135–150. doi:http://dx.doi.org/10.1016/j.comgeo.2003.07.001. 3, 6

[36] D. Kirkpatrick, Optimal search in planar subdivisions, SIAM Journal on Computing 12 (1) (1983) 28–35. doi:10.1137/0212002. 3

[37] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational Geometry: Algorithms and Applications, 3rd Edition, Springer-Verlag, 2008. 4, 4.5

[38] S. Ramaswami, M. Siqueira, A fast algorithm for computing irreducible triangulations of closed surfaces in $\mathbb{E}^d$, CoRR arXiv:1409.6015. 4.3, 4.3, 4.4, 4.4

[39] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, B. Lévy, Polygonal Mesh Processing, A K Peters, Ltd., 2010. 7

Appendix A. Proof of Lemma 1

A proof for Lemma 1 is given below:

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, 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. $\square$