Available online at www.sciencedirect.com

SciVerse ScienceDirect

IMR24 00 (2015) 000–000

Procedia Engineering

www.elsevier.com/locate/procedia

24th International Meshing Roundtable (IMR24) (Research Note)

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

Thiago Lemos[a], Suneeta Ramaswami[b,*], Marcelo Siqueira[c]

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

Abstract

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

Peer-review under responsibility of organizing committee of the 24th International Meshing Roundtable (IMR24).

Keywords: triangulation; quadrangulation; graph matching; edge contraction; irreducible triangulation

1. Introduction

Polygonal meshes have become the *de facto* standard for the representation of surfaces with highly complex geometry and arbitrary genus in computer graphics and geometry processing applications. A polygonal mesh made entirely of triangles (resp. quadrilaterals) is called a *triangle* (resp. *quad*) *mesh*. On the one hand, a triangle mesh is arguably the most popular type of mesh representation for surfaces. Furthermore, provably good algorithms for generating triangle meshes from surfaces given by parametric or implicit functions and point clouds are available in the literature, and have also been implemented in public and freely-distributed software libraries. On the other hand, quad meshes are better suited than their triangle counterparts for character animation, spline-based surface modeling, texture mapping, mesh compression, and some specific finite element analyses, to name a few applications. However,

* Corresponding author. Tel.: +1-856-225-6439.
E-mail address: suneeta.ramaswami@rutgers.edu

Peer-review under responsibility of organizing committee of the 24th International Meshing Roundtable (IMR24).

the problem of generating a quad mesh from a given surface, which meets the needs of these applications, seems to be intrinsically harder than its triangle mesh counterpart. This difficulty has led several researchers to use an *indirect* approach, which consists of generating a quad mesh from a given triangle mesh of the surface.

There are two broad classes of indirect algorithms [1]: *conversion* algorithms and *remeshing* algorithms. The former class consists of algorithms that convert a triangle mesh into a quad mesh by pairing up adjacent triangles, and possibly flipping edges and/or adding a few vertices to the resulting quad mesh. The latter class consists of algorithms that re-sample the triangle mesh, and connect the new vertices to form a quad mesh. Here, we restrict our attention to conversion algorithms and their application to triangle meshes of *closed* surfaces (that is, oriented, connected, and compact surfaces with empty boundary).

Our Contributions. We present the results of an experimental comparison of four conversion algorithms: two greedy algorithms [2,3], the graph matching algorithm in [4], and a new algorithm presented in this paper, which uses edge contractions and vertex splittings to compute a quadrangulation of the surface in two stages by combining pairings with edge flips. If the input triangle mesh has positive genus g, the new algorithm takes $O(gn_f + g^2)$ amortized time to compute a quadrilateral mesh with the same vertex set, where n_f is the number of triangles of the mesh; otherwise (i.e., if $g = 0$), it takes $O(n_f)$ amortized time. All four algorithms have been implemented by the authors. We evaluated their performance against triangle meshes typically found in graphics and geometry processing applications, as well as triangle meshes designed by the authors, which allow us to quantify the influence of parameters n_f and g.

The emphasis of this work is on the run-time efficiency of producing a quad mesh, rather than the evaluation of the resulting quad mesh by specific quality criteria. Indeed, when it comes to pairing algorithms, quadrilateral shape quality is highly dependent on the triangle mesh [1]. Moreover, a quad mesh suitable for applications depend on several quality criteria. As a result, pairing algorithms are better off serving the purpose of rapidly producing a quad mesh that can be used as input to a powerful optimization procedure that improves the mesh with respect to several quality criteria at the same time [5,6], or by applications in which mesh quality is not a critical factor [7,2,8,9].

2. Algorithms

This section briefly describes four conversion algorithms whose performances we compare in Section 3. All four algorithms take the graph G of a triangulation $\mathcal{T}$ of a surface $\mathcal{S}$ as input, and output the graph H of a quadrangulation $\mathcal{Q}$ of $\mathcal{S}$ such that G and H share the same node set The existence of a perfect pairing of triangles in $\mathcal{T}$ is guaranteed by the fact that the dual graph of $\mathcal{T}$ is bridgeless and 3-regular and by Petersen's theorem, which states that every bridgeless, 3-regular graph admits a perfect matching [10].

Greedy algorithms. Greedy algorithms iteratively select one edge at a time from the input triangulation $\mathcal{T}$ and pair up the two triangles incident on it. To ensure that each triangle belongs to at most one pair, every edge of $\mathcal{T}$ is marked as either *free* or *locked*. Initially, all edges are marked *free*. Whenever an edge is selected, the algorithm checks if it is *free*. If so, its two incident triangles are paired up with each other, and their remaining edges are marked as *locked*. If the selected edge is locked, it is discarded and a new edge is selected. This iterative process ends when all edges of $\mathcal{T}$ have been selected and tested. The algorithm takes $\Theta(n_f)$ time, where n_f is the number of triangles of $\mathcal{T}$. Here, we report the results of two greedy algorithms: (1) the greedy algorithm that uses no heuristic to select edges, and (2) the greedy algorithm developed by Puli and Segal [2]. If the pairing produced by the greedy algorithm is not perfect, we augment the implied matching (in the dual graph $G^\star$) to make it perfect. We implemented the procedure for finding augmenting paths described by Tarjan in [11]. Using a special case of disjoint sets from [12], Tarjan's procedure can find one augmenting path in $O(n_f)$ amortized time. Thus, the time complexity of the pairing stage of the greedy algorithms is $O(k\, n_f)$ amortized time, where k is the number of triangles left unpaired by the greedy approach.

Graph-based algorithm. Orrin Frink gave a constructive proof for Petersen's Theorem in 1926 [13]. His proof has some minor flaws, but a correct version of it can be found in the first book on graph theory [14]. Using the key idea of "graph reduction" from Frink's proof , Biedl, Bose, Demaine, and Lubiw proposed an algorithm for computing a perfect matching on a given 3-regular and bridgeless graph in $O(n \lg^4 n)$ [15], where n is the number of graph nodes. More recently, Diks and Stanczyk modified the algorithm in [15], and lowered the upper bound to find a perfect matching to $O(n \lg^2 n)$ [4], which is, to the best of our knowledge, the best known upper bound for computing a perfect matching on a 3-regular and bridgeless graph. We implemented Diks and Stanczyk's algorithm and applied it to the dual graph $G^\star$ of the graph G of the input triangulation $\mathcal{T}$ in order to obtain a perfect pairing of the triangles of $\mathcal{T}$, and since we have $n_f = n$, we obtain a quadrangulation of $\mathcal{S}$ in $O(n_f \lg^2 n_f)$ amortized time.

2.1. A new algorithm using irreducible triangulations and edge flips

We propose a new algorithm to compute a quadrangulation of surface $\mathcal{S}$ in $O(g^2 + g\,n)$ time, where g is the genus of $\mathcal{S}$. The quadrangulation is also computed in two stages, but the pairing stage consists of three steps. First, the algorithm computes an *irreducible triangulation* $\mathcal{T}'$ from triangulation $\mathcal{T}$ of $\mathcal{S}$ by applying a sequence of *topology-preserving edge contractions* to the graph G of $\mathcal{T}$. Second, all edge contractions are expanded in reverse order and some *edge flips* may take place in between two expansions. The result is another triangulation, $\mathcal{K}$, of the same surface $\mathcal{S}$ and with the same set of vertices as $\mathcal{T}$. Due to the edge flips, the set of edges and the set of triangles of $\mathcal{K}$ are not the same as the ones of $\mathcal{T}$, but $\mathcal{K}$ has the same number of edges and triangles as $\mathcal{T}$. The second step also builds a matching on the dual graph $J^\star$ of the graph J of $\mathcal{K}$ at the same time $\mathcal{K}$ is built. This matching is not perfect. So, a third step is necessary to augment the matching computed in the second in order to make it perfect. To that end, we also rely on the procedure from [11].

Contracting an edge e of a triangulation consists of removing e and all edges meeting e from the triangulation, and then replacing them by a vertex w which is connected to every remaining vertex that was connected to a vertex of e (see Figure 1). The inverse of the edge contraction operation is called *vertex splitting*. If the result of contracting e is a triangulation of the same surface, e is said to be *contractible* and the contraction is said to be *topology-preserving*; otherwise, e is said to be *non-contractible*. A triangulation is said to be *irreducible* if and only if every edge is non-contractible. There are only finitely many irreducible triangulations for every surface [16]. In general, if the surface has positive genus g, the number n'_v of vertices of any irreducible triangulation of the surface is such that $n'_v \leq 26 \cdot g - 4$ [17].

Two of the authors of this paper proposed a fast algorithm for computing an irreducible triangulation $\mathcal{T}'$ of a surface $\mathcal{S}$ from any given triangulation $\mathcal{T}$ of the same surface [18]. If the genus g of $\mathcal{S}$ is zero, then the algorithm produces $\mathcal{T}'$ in time linear in n_f. Otherwise, $\mathcal{T}'$ is obtained in $O(g\,n_f + g^2)$ time. Triangulation $\mathcal{T}'$ is obtained by a sequence of topology-preserving edge contractions, which guarantees that $\mathcal{T}'$ is a triangulation of $\mathcal{S}$. The first step of our proposed algorithm, which we call the *contraction* step, is carried out by the algorithm described in [18]. The second step, which we call the *expansion* step, reverses all edge contractions applied by the first phase using the vertex splitting operation. In between two vertex splitting operations, one or two edges of the current triangulation may be flipped. While reversing the contractions, the expansion step computes a matching on the dual graph of the current triangulation. The output is a triangulation $\mathcal{K}$ of $\mathcal{S}$, with n_f triangles as well, and a matching M on the dual graph $J^\star$ of the graph of $\mathcal{K}$. With respect to M, we have that $J^\star$ has at most 4 unmatched nodes if $\mathcal{S}$ is a genus-0 surface, and at most $56 \cdot g - 12$ unmatched nodes otherwise (which follows from the bound $n'_v \leq 26 \cdot g - 4$, and the relations $3n'_f = 2n'_e$ and $n'_v - n'_e + n'_f = 2(1 - g)$, where n'_e and n'_f are the number of edges and triangles of $\mathcal{T}'$). Both $\mathcal{K}$ and M can be obtained in $O(n_f)$ time. A detailed description of the expansion phase is available at [19] and is omitted here due to space constraints.

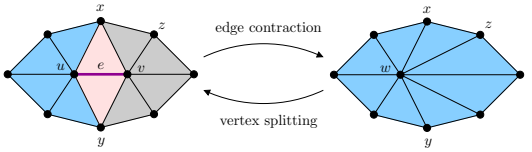

Fig. 1. The edge contraction operation and its inverse.

Matching M is augmented to become a perfect matching on $J^\star$ in the third step. Since M has at most $\ell = \max\{4, 56 \cdot g - 12\}$ unmatched nodes, the augmenting path procedure in [11] is invoked at most $\ell/2$ times. Thus, the third step takes $O(n_f)$ amortized time if g is zero, and $O(g\,n_f)$ amortized time otherwise. As we see in Section 3, ℓ is less than 2% of the total number n_f of nodes of $J^\star$ for all test cases in which n_f is large and $g \ll n_f$ (which is typically the case for meshes used in graphics and engineering applications). Finally, a quadragulation $\mathcal{Q}$ of $\mathcal{S}$ is obtained by pairing up the dual triangles corresponding to matched nodes in the perfect matching on $J^\star$. Observe that $\mathcal{Q}$ has the same vertex set as $\mathcal{T}$, but its set of edges is not necessarily a subset of the set of edges of $\mathcal{T}$ (due to the edge flips carried out in the expansion step). We thus have the following theorem (proof omitted due to lack of space):

Theorem 1. *Given a triangulation $\mathcal{T}$ of a surface $\mathcal{S}$ with genus g, the graph of a quadrangulation $\mathcal{Q}$ of $\mathcal{S}$ with the same vertex set as $\mathcal{T}$ can be generated in $O(gn_f + g^2)$-time if g is positive, and in $O(n_f)$-time otherwise, where n_f is the number of faces of $\mathcal{T}$. In either case, linear space is required.*

3. Experimental results

We implemented the simple greedy algorithm, Puli and Segal's algorithm, Diks and Stanczyk's algorithm, our new algorithm (Section 2.1), and the augmenting path procedure described by Tarjan in [11] in `C++`. All implementations were compiled with the compiler `clang 503.0.40` using the `-O2` option. We ran the implementations and collected data on a Macbook Pro running OS X 10.9.4 at 2.4 GHz (Intel Core 2 Duo), with 3MB of level-two data cache and 4GB of RAM. The implementations of the greedy algorithms and our new algorithm are based on the same data structure for surface triangulations (i.e., an augmented, doubly-connected edge list), while the implementations of the graph-based algorithm and augmenting path procedure are based on specific data structures for graphs and trees [11,20], and for fully-dynamic algorithms for connectivity queries on graphs [21]. Our implementations do not depend on any third-party libraries, and they were made publicly available on the internet[1].

Time measurements refer to the time to compute an initial triangle pairing and the time to make the initial pairing perfect (i.e., the time spent by the augmenting path procedure). If the initial pairing is already perfect, which is always the case for Diks and Stanczyk's algorithm, the second step is ignored. To compare the implementations, we consider three groups of triangulations. The first group consists of low genus triangulations, which are typically found in graphics applications[2] (see table). The second group consists of a hierarchy of 5 triangulations A1-A5 of the same genus-0, box-shaped surface with 3,844 cavities. The third group consists of a hierarchy of 5 triangulations B1-B5 of the same box-shaped surface with 3,500 holes. The second and third groups allow us to compare worst-case scenarios for Puli and Segal's algorithm and for the new algorithm. We denote the simple greedy algorithm, Puli and Segal's algorithm, Diks and Stanczyk's algorithm, and our new algorithm by **SG**, **PS**, **DS**, and **LRS**, respectively. We executed **DS** and **LRS** exactly once on each triangulation of the first group, while **SG** and **PS** were executed ten times on each triangulation. This is because we randomized the edges and the triangles of the input triangulations of **SG** and **PS**, respectively, to avoid selecting edges (resp. triangles) in the fixed order they show up in an input file. The results corresponding to the triangulations of the first group are shown in Table 1.

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

Discussion and final remarks. Our experimental results allow us to analyze the influence of the parameters n_f, g, k and ℓ. More importantly, they allow us to characterize why and when one algorithm can perform better than the others. In particular, we observed that $k = \alpha \cdot n_f$, with $\alpha \in [0.12, 0.13] \subset \mathbb{R}$ for the simple greedy algorithm, and $\alpha \in [10^{-5}, 0.0021] \subset \mathbb{R}$ for the improved greedy algorithm. In turn, ℓ does not depend on n_f, but on g, and $\ell \leq 56 \cdot g - 12$. We verified that if n_f is sufficiently large and $g \ll n_f$, then the new algorithm performs better than the other three, as $|\ell - k|$ is small relative to n_f. However, if g is large (relative to n_f), then the improved greedy algorithm is a far better option than the other three. In addition, in this particular scenario, the new algorithm can do worse than the graph-based algorithm, but still better than the simple greedy algorithm (which performed extremely poorly in all test cases). Despite the facts that (1) the improved greedy algorithm is a far better option than the simple one, and (2) the effectiveness of some heuristics for computing partial matchings on general graphs had already been mentioned in [8], the mesh processing community appears to be unaware of the improved greedy algorithm. In fact, it is not even mentioned among the possible solutions for the tri-quad conversion problem in a recent survey [1], while the simple greedy heuristic is cited and has been implemented in a popular, freely available meshing tool (`http://meshlab.sourceforge.net/`).

Acknowledgments. Suneeta Ramaswami was partially supported by NSF grant CCF-0830589, and Marcelo Siqueira was partially supported by CNPq grants 305845/2012-8 and 486951/2012-0.

[1] `http://www.mat.ufrn.br/~mfsiqueira/pairing.zip` (for the time being, it is revealed to the reviewers only)

[2] Triangulations `Botijo`, `Eros`, `Filigree`, and `Gargoyle` were downloaded from the Aim@Shape Repository, triangulation `Happy Buddha` was downloaded from the Large Geometric Models Archive, and triangulation `Iphigenia` was downloaded from the website of the book [22]. We would also like to thank Renato Werneck (Microsoft Research, USA) for the source code of his implementation of data structures for dynamic trees (see [23]), which greatly helped us implement Diks and Stanczyk's graph matching algorithm [4].

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

Table 1. Measurements obtained from executing the algorithms on each triangulation of the first group. All time measures are shown in seconds. For **SG** and **PS**, the displayed values are average values over ten executions of each algorithm on each triangulation.

References

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

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

[3] L. Velho, Quadrilateral meshing using 4-8 clustering, in: Proc. Symp. Mesh Generation and Self-Adaptivity (CILANCE), 2000, pp. 61–64.

[4] K. Diks, P. Stanczyk, Perfect matching for biconnected cubic graphs in $O(n \lg^2 n)$ time, in: Proceedings of the 36th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'10), Springer-Verlag, Germany, 2010, pp. 321–333.

[5] M. Tarini, N. Pietroni, P. Cignoni, D. Panozzo, E. Puppo, Practical quad mesh simplification, Comput Graph Forum 29 (2010) 407–418.

[6] C. S. Verma, T. Tautges, Jaal: Engineering a high quality all-quadrilateral mesh generator, in: W. R. Quadros (Ed.), Proc. 20th International Meshing Roundtable, Springer, 2012, pp. 511–530.

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

[8] M. Gopi, D. Eppstein, Single-strip triangulation of manifolds with arbitrary topology, Comput Graph Forum 23 (2004) 371–380.

[9] T. Gurung, D. E. Laney, P. Lindstrom, J. Rossignac, Squad: Compact representation for triangle meshes, CG Forum 30 (2011) 355–364.

[10] J. Petersen, Die theorie der regulären graphs, Acta Mathematica 15 (1891) 193–220.

[11] R. E. Tarjan, Data Structures and Network Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, USA, 1983.

[12] H. N. Gabow, R. E. Tarjan, A linear-time algorithm for a special case of disjoint union, Jour. Comp. and Sys. Sci. 30 (1985) 209–221.

[13] O. Frink, A proof of petersen's theorem, Annals of Mathematics 27 (1926) 491–493.

[14] D. König, The theory of finite and infinite graphs, Birkhäuser, Boston, MA, USA, 1990.

[15] T. C. Biedl, P. Bose, E. D. Demaine, A. Lubiw, Efficient algorithms for petersen's matching theorem, Journal of Algorithms 38 (2001) 110–134.

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

[17] G. Joret, D. R. Wood, Irreducible triangulations are small, Journal of Combinatorial Theory, Series B 100 (2010) 446–455.

[18] S. Ramaswami, M. Siqueira, A fast algorithm for computing irreducible triangulations of closed surfaces in $\mathbb{E}^d$, CoRR (*submitted*) (2014).

[19] T. Lemos, S. Ramaswami, M. Siqueira, An experimental comparison of algorithms for converting triangulations of closed surfaces into quadrangulations, 2015. URL: `http://clam.rutgers.edu/~rsuneeta/hpc/research/imr15_long.pdf`.

[20] D. D. Sleator, R. E. Tarjan, A data structure for dynamic trees, Journal of Computer and System Sciences 26 (1983) 362–391.

[21] J. Holm, K. D. Lichtenberg, M. Thorup, Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge and biconnectivity, Journal of the ACM 48 (2001) 723–760.

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

[23] R. E. Tarjan, R. F. Werneck, Dynamic trees in practice, Journal of Experimental Algorithmics 14 (2010) 5:4.5–5:4.23.