ELSEVIER

Computational Geometry 22 (2002) 75–97

Computational Geometry
Theory and Applications

www.elsevier.com/locate/comgeo

Quadrilateral surface meshes without self-intersecting dual cycles for hexahedral mesh generation ☆

Matthias Müller-Hannemann

Technische Universität Berlin, Department of Mathematics, MA 6-1, Straße des 17. Juni 136, 10623 Berlin, Germany

Communicated by S. Fortune; received 3 July 2000; accepted 27 February 2001

Abstract

Several promising approaches for hexahedral mesh generation work as follows: Given a prescribed quadrilateral surface mesh they first build the combinatorial dual of the hexahedral mesh. This dual mesh is converted into the primal hexahedral mesh, and finally embedded and smoothed into the given domain. Two such approaches, the modified whisker weaving algorithm by Folwell and Mitchell, as well as a method proposed by the author, rely on an iterative elimination of certain dual cycles in the surface mesh. An intuitive interpretation of the latter method is that cycle eliminations correspond to complete sheets of hexahedra in the volume mesh.

Although these methods can be shown to work in principle, the quality of the generated meshes heavily relies on the dual cycle structure of the given surface mesh. In particular, it seems that difficulties in the hexahedral meshing process and poor mesh qualities are often due to self-intersecting dual cycles. Unfortunately, all previous work on quadrilateral surface mesh generation has focused on quality issues of the surface mesh alone but has disregarded its suitability for a high-quality extension to a three-dimensional mesh.

In this paper, we develop a new method to generate quadrilateral surface meshes *without* self-intersecting dual cycles. This method reuses previous b-matching problem formulations of the quadrilateral mesh refinement problem. The key insight is that the b-matching solution can be decomposed into a collection of simple cycles and paths of multiplicity two, and that these cycles and paths can be consistently embedded into the dual surface mesh.

A second tool uses recursive splitting of components into simpler subcomponents by insertion of internal two-manifolds. We show that such a two-manifold can be meshed with quadrilaterals such that the induced dual cycle structure of each subcomponent is free of self-intersections if the original component satisfies this property. Experiments show that we can achieve hexahedral meshes with a good quality.

Keywords: Hexahedral mesh generation; Cycle arrangements; Avoiding self-intersections; b-matching; Balanced cuts

☆ An extended abstract of this paper appeared in Proceedings of the 16th Annual ACM Symposium on Computational Geometry (SCG'00), Hong Kong, 2000, pp. 19–28, under the title "Improving the surface cycle structure for hexahedral mesh generation".

E-mail address: mhannema@math.tu-berlin.de (M. Müller-Hannemann).

URL address: http://www.math.tu-berlin.de/∼mhannema (M. Müller-Hannemann).

0925-7721/01/$ – see front matter
PII: S0925-7721(01)00053-0

1. Introduction

The automatic generation of finite element meshes is essential for the numerical analysis and simulation in a wide variety of applications. Recent years showed many research efforts and brought up several approaches, but up to now, hexahedral mesh generation for an arbitrary 3D solid is still a challenge.

Prescribed surface meshes

In many applications, in particular in structural mechanics simulations, high mesh quality is required near the boundary of the solid and is much more important than "deep inside the domain". Therefore, it is often preferred to start the volume meshing subject to a fixed quadrilateral surface mesh of an excellent quality. Moreover, the complete solid may consist of several components, for example, solid parts of different material or a solid and its complement within a larger box. Many subdomains may also arise for algorithmic reasons: we may want to divide the domain into smaller and simpler regions either to facilitate the meshing process for each part or to allow parallelism which can be crucial for some large-scale applications. In all such cases, it is usually essential to have a compatible mesh at the common boundary of adjacent components, that is the surface meshes must be compatible. The only solution for this problem we can envision is to prescribe the surface mesh for each component.

Thurston [22] and Mitchell [9] independently characterized quadrilateral surface meshes which can be extended to hexahedral meshes. They showed that for a volume which is topologically a ball and which is equipped with an all-quadrilateral surface mesh, there exists a combinatorial hexahedral mesh without further boundary subdivision if and only if the number of quadrilaterals is even. (Similar characterizations can be proved for bodies with handles, see Mitchell [9].) Furthermore, Eppstein [4] used this existence result and proved that a linear number of hexahedra (in the number of quadrilaterals) are sufficient in such cases. Unfortunately, all these results are not completely constructive and it remains unclear whether they can be extended to constructive methods for geometrically well-shaped hexahedral meshes. There are quite simple solids with natural looking quadrilateral surface meshes, for example the quadratic pyramid problem of Schneiders [18], where only rather complicated combinatorial meshes are known, but no mesh with an acceptable quality is available.

Related work

We briefly review approaches to hexahedral mesh generation based on prescribed quadrilateral surface meshes. For a more complete survey, online information and data bases on meshing literature see [19] and [15].

Plastering [1,3] is an advancing front based method. It maintains throughout the algorithm the *meshing front*, that is a set of quadrilateral faces which represent the boundary of the region(s) yet to be meshed. The plasterer selects iteratively one or more quadrilaterals from the front, attaches a new hexahedron to them, and updates the front until the volume is completely meshed. However, in practice, the plasterer usually leaves some holes unmeshed. These remaining unplastered regions are filled with tetrahedra.

Whisker weaving [20,21] also meshes from a quadrilateral surface mesh inward. But in contrast to plastering it first builds the combinatorial dual of a mesh. This method is based on the so-called *spatial twist continuum* (*STC*). The STC is an interpretation of the geometric dual of a hexahedral mesh as

an arrangement of surfaces, the *sheets*. More precisely, the mesh dual is the cell complex induced by the intersection of the sheets. A fundamental data structure for an STC is a *sheet diagram* which represents the crossings of one sheet with other sheets. Whisker weaving starts with incomplete sheet diagrams based on the surface mesh. It seeks to complete the sheet diagrams by a set of rules which determine the local connectivity of the mesh. The plastering and whisker weaving algorithms typically create meshes with an invalid mesh connectivity. Heuristic strategies have been developed to resolve such invalidities [20].

Calvo and Idelsohn [2] recently presented rough ideas of a recursive decomposition approach. They select a dual cycle to divide the combinatorial dual of the surface mesh into two subgraphs. This "cut" induces an interior two-manifold which is remeshed simply by mapping or projecting one of the obtained subgraphs onto it. However, fragments from previously used dual cycles are ignored in this mapping. This splitting process is applied recursively until there are no more unused dual cycles.

Hexahedral meshing by cycle elimination

The author developed in [12,13] a purely combinatorial method to decompose a topological ball with a prescribed quadrilateral surface mesh into hexahedra. The step-wise creation of the hexahedral mesh is guided by the cycle structure of the combinatorial dual of the surface mesh. Our method transforms the graph of the surface mesh iteratively by changing the dual cycle structure until we get the surface mesh of a single hexahedron. During the transformation process we keep the invariant that the surface mesh remains simple, planar, and 3-connected (and thus corresponds to a polytope). Our main strategy is a successive elimination of dual cycles. Hence, algorithmically, we try to determine a cycle elimination scheme transforming the given surface mesh to the mesh of a single hexahedron. Independently of us, Folwell and Mitchell [5] changed the original whisker weaving algorithm and incorporated a cycle elimination strategy which is similar to ours. In contrast to us, they have no restrictions on the elimination of a cycle and thereby allow that the surface mesh (as well as the intermediate hex complex) becomes degenerated. Based on the constructions in [9], the new version of whisker weaving applies a number of heuristics to convert a degenerated hex complex into a well-defined one.

Contribution and overview

We first introduce some basic terminology and review the basic ideas for our approach to hexahedral mesh generation in Section 2. Then, in Section 3 we present our new method to generate surface meshes without self-intersecting dual cycles.

Previous work has already successfully used (bidirected) network flow and matching techniques for quadrilateral surface mesh generation [10,11]. However, this work has only focused on high-quality quadrilaterals and a tight mesh density control, whereas the dual cycle structure has not been considered as a side constraint or optimization goal. Although the bidirected flow method typically leads to fewer self-intersecting dual cycles than other methods like advancing front based techniques, the empirical observation is that it still generates a significant number of self-intersections.

As our main result, we introduce a new method which guarantees a quadrilateral surface mesh free of self-intersections. This method builds on a slight modification of the b-matching problem formulation for conformal mesh refinement. In contrast to previous work, the b-matching solution is first decomposed

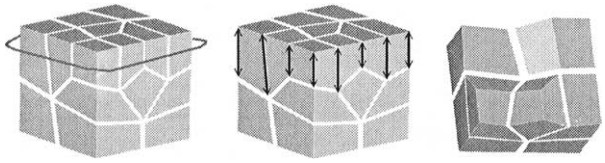

Fig. 1. The elimination of a dual cycle (left) corresponds to the contraction of primal edges (middle) and leads to the removal of a sheet of hexahedra (right).

into carefully chosen cycles and paths which can be realized and embedded as a quadrilateral mesh refinement without self-intersections.

Getting rid of self-intersections is one fundamental prerequisite for our successive cycle elimination method which is now achieved in a robust way. Another important issue is the order in which cycles are eliminated. Practical experience shows that for achieving an excellent mesh quality a dual cycle should only be eliminated if one of its neighboring primal cycle consists only of sharp edges (see for example the elimination in Fig. 1). Hence, we are often faced with the problem that no dual cycle meets this elimination criterion. In such a situation a split into several subdomains is often very helpful. In contrast to Calvo and Idelsohn [2], we split the domain along a *primal* cycle of the current surface mesh and insert an additional internal two-manifold bounded by this primal cycle. In Section 4, we explain how to find a suitable primal cycle for such a split and show how to mesh such an internal two-manifold subject to the constraint that no self-intersection will be introduced in one of the two induced components. Finally, in Section 5, we report on the mesh quality achieved in experiments and summarize the main features of our approach.

2. Basic definitions and background

Input requirements. We make a few general assumptions on the geometric descriptions of the objects to be meshed. A so-called *macro element* is a smooth, orientable two-manifold surface patch bounded by a polygon. A *surface mesh* is composed of macro elements, which have pairwise disjoint relative interior. The neighborhood relations of the macro elements imply the topology of the surface mesh. *Branchings* are edges which belong to more than two polygons. An edge is a *free boundary edge* if it belongs to only one macro element. A mesh is *closed* if it has no free boundary. For the purpose of volume meshing we will only consider closed meshes. A surface mesh *component* is a closed surface mesh equivalent to a topological ball. If a component is not degenerated (which we will always assume) it is combinatorially a simple, planar and 3-connected graph. In the case of branchings, the volume to be meshed is partitioned in the obvious way into distinct subdomains each of which is bounded by a component.

Next we shortly review the basic notions and ideas of the cycle elimination approach.

Canonical dual cycles. Let G be the graph of a quadrilateral mesh and G^d its combinatorial dual. We say that two adjacent dual edges are *opposite to each other* if and only if they correspond to opposite sides of a quadrilateral, i.e., if they are not neighbored in the cyclic adjacency list of their common node. Hence, the four adjacent edges to each dual node can be partitioned into two pairs of opposite edges.

The dual graph $G^d = (V^d, E^d)$ can be decomposed in a canonical way into a collection of edge-disjoint cycles, say into $C_1, \ldots, C_k$, by putting a pair of adjacent edges e_1, e_2 into the same cycle if they are opposite to each other. In other words, for each quadrilateral the edges which are dual to opposite sides are contained in the same cycle. Observe that by transitivity two edges may belong to the same cycle even if they are neighbored in the cyclic adjacency list of some node. Hence, these dual cycles can be non-simple or *self-intersecting.* Note that the set of dual cycles is well-defined and unique (in the sense that two cycles are equivalent if they have the same set of edges) and can be easily determined in linear time. We call this set the *canonical dual cycles* of G^d, and by a dual cycle we will henceforth always mean a canonical dual cycle (in whisker weaving terminology, these cycles are called *loops* [21]).

Cycle arrangements. Consider a collection of k simple, closed Jordan curves in the plane such that no three curves meet in a point, which we call a *cycle arrangement.* We may associate in the obvious way a planar graph G^d to such a collection of curves (considered as a dual graph): The vertex set is formed by the intersection points between curves, and the edges are induced by the segments between intersection points. If G^d is simple and 3-connected, then so is the primal graph. As the degree of each dual vertex is four, such a primal graph corresponds to a quadrilateral surface mesh. Hence, any cycle arrangement such that the associated planar graph is simple and 3-connected is a *cycle arrangement of a surface mesh.*

Cycle elimination. Next we introduce the concept of a cycle elimination. To get an intuitive idea of the use of cycle eliminations for the meshing see Fig. 1. In terms of the primal graph, a dual cycle elimination means the contraction of each primal edge corresponding to a dual edge contained in the dual cycle, and removing parallel edges afterwards. The elimination corresponds to the removal of a complete layer of hexahedra, that is in STC terminology to the removal of a complete sheet. A *feasible elimination* of a dual cycle C from G^d requires that

(i) C is a simple cycle,
(ii) at least one of two subgraphs of G^d, the first induced by the vertices on the left-hand side, the second by the vertices on the right-hand side of C is connected,
(iii) the resulting graph remains 3-connected.

Note that 3-connectivity can be checked in linear time [6]. So we can test in linear time whether a cycle can be eliminated in a feasible way or not.

A *perfect cycle elimination scheme* of a dual graph G^d of a quadrilateral surface mesh is an order $C_1, C_2, \ldots, C_k$ of its k canonical cycles such that the first $k-3$ cycles with respect to this order can be eliminated one after another in a feasible way, and such that the remaining 3 cycles form the dual surface graph of a single hexahedron. In [12,13] it has been shown that whenever we know a perfect cycle elimination scheme for some graph G^d, we can iteratively build up a hex complex compatible to the prescribed surface mesh by reversing the elimination order and filling in the corresponding layer. Moreover, this explicit construction corresponds to the shelling of the hex complex.

The general hexahedral meshing algorithm. For an arbitrary domain which is equipped with a coarse surface mesh but has already been decomposed into components our algorithm consists of the following major steps:

(i) Quadrangulate the surface mesh with the required density such that no component has self-intersecting dual cycles.

(ii) Do for each component
  (a) Search for a perfect cycle elimination scheme.
  (b) Build up a combinatorial hex complex in reversed elimination order.
(iii) Embed the hex complex into the given domain and perform mesh smoothing.

3. Quadrilateral meshes without self-intersections

3.1. Canceling self-intersections

Folwell and Mitchell [5] describe how to modify a quadrilateral surface mesh in order to remove self-intersections for a single domain. The main idea of the latter approach is to collapse a quadrilateral where a self-intersection occurs into two edges by the identification of two opposite vertices. This may result in degenerated vertices with degree two, but such a situation can be resolved by a so-called "pillowing" technique which places an additional ring of quadrilaterals around such a degenerated vertex. In a previous paper we proposed a different kind of local mesh modification [12].

However, without a modification both methods have a serious drawback as they do not work safely in the presence of several components. The problem is that some dual cycles belong to several subdomains and cannot be modified independently. Hence, if a quadrilateral is self-intersecting with respect to one domain but not to the other then the collapsing of a quadrilateral removes the self-intersection in one subdomain but inevitably creates a new self-intersection with respect to the other.

These difficulties can be overcome by an alternative method which can also be applied in the presence of branchings. It works as follows: After decomposing the domain of our solid body into topological balls, we use the quadrilateral surface mesher with only half the required mesh density. Then we subdivide each quadrilateral into four new ones (by halving all edges) to meet the required mesh density. This strategy has its price: In parts of a given model where the desired mesh density is equal to the coarsest possible good-quality quadrilateral mesh, the result will be finer than desired.

This replacement duplicates all dual cycles. In particular, all quadrilaterals where self-intersections occur appear in pairs, see the middle part of Fig. 3. So it is possible to change the surface mesh locally at all such places, by replacing four quadrilaterals with 12 new ones, see the right part of Fig. 3, and Fig. 2 for an example. Obviously, the surface graph remains planar, simple and 3-connected, and we do not change the structure of other cycles than those going through such quadrilaterals. Most importantly, it serves its primary purpose to resolve each existing self-intersection. If a quadrilateral where a self-intersection occurred belongs to exactly one subdomain, the transformation cannot create a new self-intersection, otherwise it might do so with respect to the other subdomain. More precisely, if not only

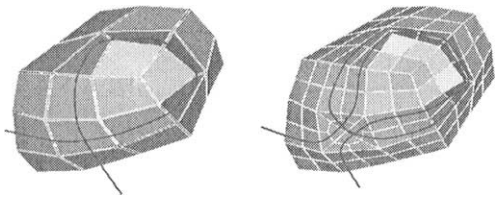

Fig. 2. Application of the transformation to eliminate self-intersections.

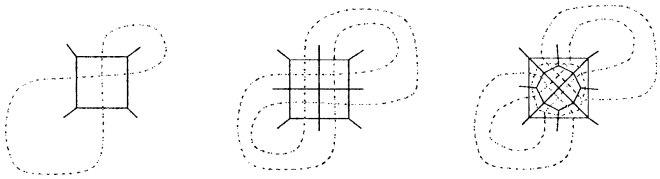

Fig. 3. Getting rid of a self-intersecting dual cycle (left): the dual cycle is first duplicated (middle), and then the surface mesh is locally transformed (right).

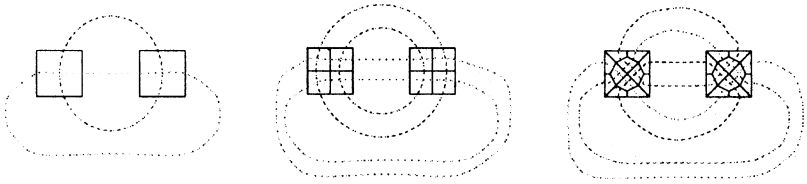

Fig. 4. Simplicity of a pair of simple cycles (before duplication, left) is maintained if we apply the graph transformation at all crossings.

one self-intersecting but two dual cycles are passing through a quadrilateral with respect to some other subdomain, then such a transformation would create a self-intersection right there. Hence, we need some additional step for each pair of simple cycles affected by a transformation for some other subdomain. Fortunately, whenever necessary, we can avoid the complication of creating self-intersecting cycles if we apply the same local transformation at *all* four-tuples of quadrilaterals corresponding to crossings of these cycles, for an example with two crossings see Fig. 4. Note that by transitivity each local transformation may induce further transformations.

Lemma 1. *The described transformation procedure yields a quadrilateral surface mesh without self-intersections.*

Proof. It is obvious that the transformation removes all self-intersections from non-simple cycles and cannot create self-intersections for cycles which are not transformed. Hence, the only interesting case to prove is that the transformation is applied to a quadrilateral belonging to two simple cycles C_1 and C_2 with respect to some component. These two cycles cross each other an even number of times, say exactly $2k$ times with $k \geqslant 1$. We prove this case by induction on k. The base case $k = 1$ is obvious by Fig. 4. For $k > 1$, we compare how the transformation acts on this configuration with a similar configuration with $2k - 2$ crossings. This second configuration is derived from the first by removing two adjacent crossings, see Fig. 5. By the induction hypothesis, the second configuration is free of self-intersections after the transformation. Hence, for the original configuration self-intersections could only occur via the transformation of the two remaining crossings. However, again Fig. 4 shows that this cannot happen, concluding the proof. □

Although this method is robust in the sense that it works without any limitations each application of the transformation degrades the mesh quality.

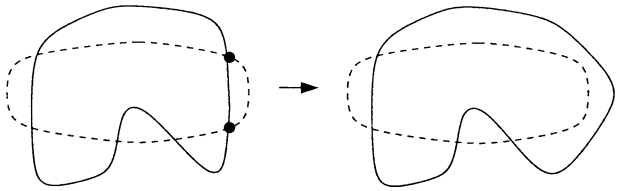

Fig. 5. Two simple curves with $2k = 6$ crossings (left), and a related configuration with only $2k - 2 = 4$ crossings.

3.2. Refinement of conformal meshes

Suppose that we are given a very coarse conformal quadrilateral surface mesh for which all its canonical dual cycles are already simple. In this subsection we want to show how such a mesh can be refined to the desired mesh density without creating self-intersecting cycles.

For ease of exposition, we first restrict ourselves to the case where the mesh is free of branchings (it is not necessarily a topological ball component, the underlying surface might be of higher genus. We only use that the surface mesh is locally planar and its dual is well-defined.). The general case with branchings which induce a partition into several components will be dealt with in Section 3.3.

3.2.1. The weighted perfect b-matching model

Edge capacitated minimum cost perfect b-matching is the following problem: Given a (multi-)graph $G = (V, E)$, a non-negative integral node demand $b(v)$ for each node $v \in V$, and for each edge $e \in E$, non-negative integral lower and upper edge capacities ℓ_e, u_e, and a convex, piecewise linear cost function $C_e(x)$ defined on the interval $[\ell_e, u_e]$, we seek integral matching values x_e for each edge minimizing $\sum_{e \in E} C_e(x)$ subject to the constraints

$$\sum_{e=(v,w),\ v \neq w} x_e + 2x_{(v,v)} = b(v) \quad \text{for all } v \in V, \tag{1}$$

$$\ell_e \leqslant x_e \leqslant u_e \quad \text{for all } e \in E. \tag{2}$$

Note that loop edges (v, v) are allowed in this formulation, and that their matching value $x_{(v,v)}$ counts twice in Eq. (1) (if the loop exists, otherwise we adopt the convention $x_{(v,v)} = 0$).

We define an instance of the perfect b-matching problem defined on the dual graph $G^d = (V^d, E^d)$ of the quadrilateral surface mesh as follows. More precisely, we take the usual dual graph (one vertex for each quadrilateral, and the four dual edges connecting the quadrilateral to its neighbors) but add a loop edge to each vertex. For each dual edge, the matching values x_e will have the interpretation that the corresponding primal edge is subdivided into $x_e + 1$ segments by placing x_e additional points on this edge. Or in other words, $x_e + 1$ dual cycles will go through this primal edge in the refinement. We derive lower and upper bounds ℓ_e, u_e from local density requirements for each primal edge. For reasons which will become important in the later realization of the b-matching solution as a mesh refinement we need the assumption that lower edge capacities satisfy $1 \leqslant \ell_e$. As we have assumed that our input mesh is very coarse, this requirement is not very restrictive. However, if a primal edge is very short and should not be subdivided anymore (for this or some other reason) we allow also a lower bound of $\ell_e = u_e = 0$ for such an edge but demand that all other edges on the corresponding canonical cycle must either be subdivided at least twice, that is, they satisfy $\ell_e \leqslant 2$, or they are also fixed to zero. Loop edges get a lower bound of zero, and a large upper bound (for example, we take the sum of the upper bounds of all other incident

edges). Node demands $b(v)$ are set for each node to an even number which is either the sum of the upper bounds of the incident non-loop edges or the same sum plus one.

Finally, we specify piecewise linear convex cost functions for each primal edge, which achieve their minimum at the desired subdivision number. Hence, the deviation from the optimal local subdivision number is punished by increasing costs. This allows a tight density control. Loop edges have no cost, i.e., $C(x_{(v,v)}) \equiv 0$.

Note that by the fact that each $b(v)$ is even, any perfect b-matching guarantees that the number of additional subdivision points is even which is necessary and sufficient for an all-quadrilateral refinement of each given quadrilateral. Hence, the following lemma is immediate:

Lemma 2. *Any feasible perfect b-matching of the above specified refinement instance corresponds to an all-quadrilateral conformal mesh refinement respecting the mesh density constraints.*

The class of b-matching problems presented above is efficiently solvable by the primal-dual blossom algorithm. Also in practice, b-matching problems can be solved very fast, for an overview on efficient algorithms and a recent computational study see [14].

3.2.2. Decomposing the b-matching solution

In principle, it is fairly straightforward to decompose any perfect b-matching solution x for our problem into simple cycles. If we delete all loops from G^d and replace each edge $e \in E^d$ by exactly x_e copies (if $x_e = 0$ we delete such an edge), the resulting multigraph is *Eulerian* (each node has an even degree) by the fact that each macro element must have an even number of primal edges for a conformal refinement into quadrilaterals. Hence, we can find in linear time an Eulerian trail in this multigraph which decomposes into simple cycles.

However, we take a slightly different approach. First, we avoid the explicit construction of the multigraph and simulate the decomposition on the original dual graph G^d. Second, we do not want to find an arbitrary Eulerian trail. Instead, our decomposition into cycles should be guided by some quality criteria for the later layout phase. In particular, due to the parallel edges in the multigraph, cycles might consist of only one pair of parallel edges and therefore be shorter than necessary. Third, our decomposition method should be easily adaptable to the general case with branchings. To this end, we decompose the b-matching not only into simple cycles but also into simple paths of even multiplicity (which means that each edge of this path is taken an even number of times).

First we delete all loops from G^d and update the b-values accordingly. To find the cycles and paths one after another, we start with an arbitrary node v_0 with $b(v_0) > 0$ which becomes our current node on a partial path or cycle P. Now we iteratively select an edge incident to our current node, add it to our current partial path, and make its other endpoint the new current node. An edge is *eligible* if it carries a positive matching value $x_e > 0$ and if it is not already on the current path. Clearly, the first node of a path always has an eligible edge $e = (v_0, v_1)$. When we arrive at node v_i in this process, two things may happen. If the node v_i already appears on the current partial path, say as v_k with $k < i$, then the current iteration terminates with the cycle $P = \{v_k, v_{k+1}, \ldots, v_i = v_k\}$. If this is not the case, we proceed with the next eligible edge. Otherwise, if no such edge exist, the last edge of the current path $e = (v_{i-1}, v_i)$ must have an even matching value x_e because $b(v_i)$ is even. We also terminate the current iteration but now take a maximal subpath from some node v_k to v_i of the current path with the property that each edge has matching value at least two.

The *cycle residual capacity* $r(P)$ of a cycle P is defined as $r(P) := \min\{x_{ij}\colon (i,j) \in P\}$, whereas the *path residual capacity* is $r(P) := \min\{\lfloor x_{ij}/2 \rfloor\colon (i,j) \in P\}$. In any case, our construction ensures that the residual capacity is positive. When a path (or cycle) P is completed, we redefine the b-matching $x_{ij} := x_{ij} - r(P)$ for all $(i,j) \in P$ and update the b-values. Clearly, we maintain the invariant that $b(v)$ is an even number for all nodes. Hence, we can delete the path or cycle just determined and continue the next iteration at v_k (if $k = 0$ it might happen that $b(v_0) = 0$ in which case we select a new starting vertex).

In our implementation, we have still some freedom in how to select eligible edges. We use this freedom by the following two selection rules. The first rule says always to select, if possible, an eligible edge which is opposite in the quadrilateral with respect to the last edge on the current path. By choosing only eligible edges subject to this rule we can first identify all cycles which are canonical dual cycles in the original mesh. If the first rule is not applicable, the path has to take a turn to the left or to the right with respect to the quadrilateral which we have just entered. The second rule tries to identify cycles and paths which always "turn in the same direction", i.e., we do not want to find a cycle or path which alternates between left and right turns. Hence, the second rule says that if we have a choice we should take a turn to the same side as before.

3.2.3. Layout of the decomposed b-matching solution

Given the decomposition of the b-matching into a collection of cycles and paths of even multiplicity, we have to show that we can find a corresponding layout as a mesh refinement of the original mesh.

The easiest way to explain the embedding is to work on the canonical cycle arrangement of the dual graph. That is, our task is to insert cycles to this arrangement one after another. Consider first the case that a cycle C in our decomposition has been determined by always applying the first selection rule, that is, the cycle is identical to some canonical cycle C' of the original mesh. The intuitive idea how to insert a cycle C to the current arrangement is to interpret C as a closed curve embedded on the surface and to route a new closed curve in an ε-distance to C', that is, along the edges of C', but never crossing them, see Fig. 6. Note that we are free on which side of C' to insert the new cycle. If a path P of our decomposition results from always applying the first selection rule, that is, it never turns, its embedding is also easy, see Fig. 7.

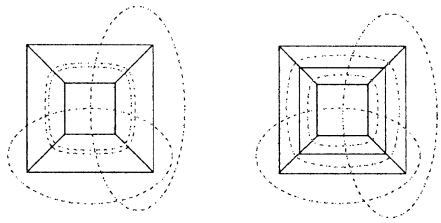

Fig. 6. Insertion of a cycle parallel to an existing canonical cycle.

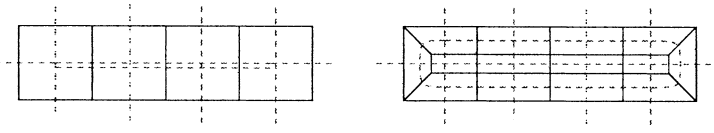

Fig. 7. Insertion of a path of multiplicity two.

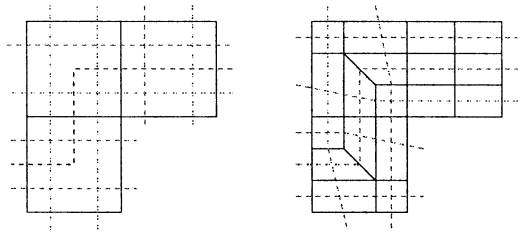

Fig. 8. Insertion of a path with alternating turns.

If a path or cycle C turns (it is composed of dual edges but not a canonical cycle), we have to be more careful in the embedding. Namely, if it turns, say to the right, the new edges must be embedded to the left of the original edges of C. This is no problem, if our cycle turns always in the same direction. Note that if we try to embed the cycle on the wrong side, then the primal mesh which we obtain after dualizing the new cycle arrangement is not necessarily a refinement of the original mesh. That is, although it is certainly also a valid quadrilateral mesh, it might be not embeddable as a subdivision of the original mesh. The latter would be unacceptable as sharp edges of the input geometry would possibly be removed.

For the case that cycle alternates between left and right turns we exploit our first selection rule and the choice of the lower capacities on the edges in the b-matching. The lower capacities guarantee that we can decompose our b-matching such that either at least one cycle runs in parallel to a pre-existing canonical cycle, or for canonical cycles which are blocked by zero capacity edges, all other edges have a lower bound of at least two and therefore appear as paths of even multiplicity in our decomposition if we apply the first selection rule. Hence, we can route any further cycle in such a way that it runs for each edge e of C between e and one of the previously inserted copies. See Fig. 8 for an illustration. Paths with alternating turns could be treated in a similar way. However, even simpler, we split all paths with alternating turns into subpaths without alternating turns and embed them one after another.

Lemma 3. *The insertion of a single dual cycle* (*or path of even multiplicity*) *as described above maintains the* 3*-connectivity of the induced surface mesh.*

Proof. Denote by $G^d = (V^d, E^d)$ the dual graph of the surface mesh before the insertion, and consider the insertion of the dual cycle C. By construction, the resulting dual graph G'^d is simple. Hence, the new primal graph (i.e., the surface mesh) is 3-connected if and only if G'^d is 3-connected. The latter is proved, if we show the existence of three (internally) node disjoint paths between any pair of nodes $u, v \in V^d \cup C$. We do so by a case distinction.

Case 1: $u \notin C$ and $v \notin C$.

By assumption, the original dual graph G^d is 3-connected. Hence there are three disjoint paths between u and v in G^d. These paths translate into disjoint paths in G'^d, as the insertion of C may only subdivide certain edges of them.

Case 2: $u \in C$ and $v \in C$.

Both nodes lying on the same cycle directly yield two disjoint paths formed by the cycle itself. We get a third path which avoids other nodes than u and v on C as follows. Denote by $u' \notin C$ and $v' \notin C$ nodes adjacent to u and v on the same side of the cycle in G'^d. There is a path P from u' to v' avoiding C as the interior of C is connected (by construction). The latter means that $(u - u') \cup P \cup (v' - v)$ is the desired third path.

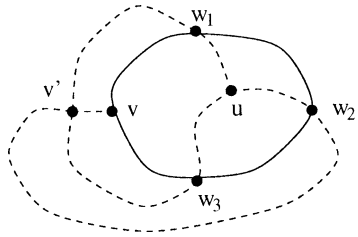

Fig. 9. Case 3 in the proof of Lemma 3.

Case 3: $v \in C$ and $u \notin C$.

Assume without loss of generality that u lies in the interior of C and let v' be a vertex in the exterior of C adjacent to v. As v' and u are vertices in G^d, there are three node disjoint paths. With respect to G'^d each of these paths has to cross C at least once, denote the corresponding vertices on C by w_1, w_2, w_3 in clockwise order. We get three internally node disjoint paths by taking the path from u to w_1 plus a portion of C from w_1 to v, taking the path containing w_2 unchanged, and finally the path from u to w_3 plus a portion of C from w_3 to v. See Fig. 9. □

Hence, an iterative insertion of dual cycles clearly yields a 3-connected surface mesh. In summary, we have derived the following main result of this section.

Lemma 4. *Given a conformal quadrilateral mesh without self-intersecting dual cycles, we can construct a mesh refinement which also has no self-intersecting dual cycles.*

As a by-product, we also obtain an existence result on perfect elimination schemes for such meshes.

Lemma 5. *Given a coarse quadrilateral mesh with a perfect cycle elimination scheme, any mesh refinement of it which is constructed by a b-matching decomposition and layout* (*as described above*) *also has a perfect cycle elimination scheme.*

Proof. We claim that the following cycle order is a perfect elimination order: As an initial sequence we take the reversed order of cycle and path insertions of the layout phase after the b-matching decomposition. This order can be completed by taking the elimination order of the coarse mesh which exists by assumption.

Hence, all we have to check is that each cycle from the initial sequence can be feasibly eliminated at the time when it is inserted. However, this is fairly obvious: by construction, the new cycle is simple, at least one of its enclosed subgraphs is connected, and using Lemma 3 by induction, we know that the surface mesh is 3-connected before the insertion. □

Note that the previous lemma asserts the existence of a perfect elimination order, and its proof also yields such an order which comes for free after the construction of the refinement. In practice, however, our implementation tries to find a reordering of this elimination scheme subject to additional geometric criteria to improve the mesh quality.

Fig. 10 shows an example where the existence of a perfect elimination scheme is almost obvious for the coarse input mesh but much more tricky for the refinement.

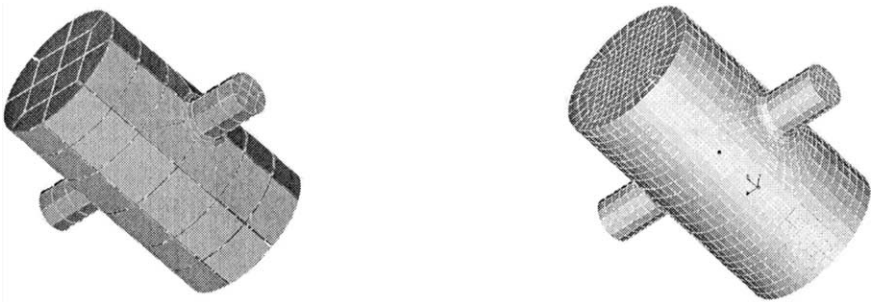

Fig. 10. An example of a fairly coarse conformal input mesh for which the decomposition into hexahedra and the existence of a perfect elimination scheme are easy to find (left), and our refinement into a quadrilateral surface mesh without self-intersections (right).

3.3. Meshes with many components

We now sketch the modifications which are necessary to deal with meshes consisting of many components. We still assume that each component is conforming and free of self-intersections.

Finding the subdivision numbers for all edges. In the case of branchings, we also first need to find a refinement which subdivides each edge such that the number of subdivision point for each macro element is even. This can be done by a two-phase approach as described in [11]. In the first phase a system of linear equations is solved over $GF(2)$ to guarantee a conforming subdivision on branching edges. Afterwards these subdivision numbers become fixed, and the b-matching formulation can be applied as described above. The only difference is that we now solve the problem with only half the required density first and double all subdivision numbers afterwards. This means that we require for the final subdivision number of an edge a lower bound of at least two (instead of one), or the subdivision numbers of the corresponding edges are fixed to zero. The advantage of this duplication is that all paths and cycles in the b-matching decomposition will come with even multiplicity.

Decomposition of the b-matching. We iterate over the components, and start with an arbitrary one. We proceed as in the case of a single component to build up a simple cycle with respect to this component. For some other components, this may induce partial paths (but of even multiplicity!). For each corresponding component, these paths are put on a stack. We store and update these partial paths for each component. When we process these components, we later try to enlarge these partial paths to simple cycles. The important point is, that even if we do not succeed in doing so, the partial paths are of even multiplicity and therefore valid for a decomposition. To avoid self-intersections, we have to redefine the meaning of an eligible edge. Now an edge is *eligible* if it carries a positive matching value $x_e > 0$ and if it does not create a self-intersection with respect to any partial path it belongs to. If no edge is eligible at some point, a path becomes a final path of even multiplicity in the current component.

Embedding phase. Having constructed the b-matching decomposition, we iterate once more over all components and embed the cycles and paths essentially in the same way as we do for the single component case. However, as in the decomposition phase, the embedding of some cycle with respect to one component B_1 may cause the embedding of a portion of a cycle with respect to some other component, say B_2. That is, when we process component B_2 the position of certain portions of cycles are already preassigned. To avoid potential conflicts from this preassignment, we now make sure that all

those cycles which never turn to the left or to the right are embedded to the "outmost tracks" of each macro element. By our assumption on the lower bound of the subdivision numbers, there are always two such cycles, one of them is embedded to the extreme left, and one other to the extreme right track. The remaining cycles can now use any position between the two special tracks as in Fig. 8.

3.4. The complete algorithm

So far we have only solved the case where the initial mesh is already conforming and has no self-intersecting dual cycles. For the general case, we propose the following algorithm which runs in four main steps:

(1) convert the coarse input mesh M_1 to a coarse all-quadrilateral mesh M_2 without self-intersecting dual cycles;
(2) solve a weighted perfect b-matching problem defined with respect to M_2 to meet the desired mesh density;
(3) decompose the matching into a collection of simple cycles and paths;
(4) translate these cycles and paths into a mesh refinement of M_2.

Steps (2)–(4) should be clear from the preceding discussion, but step (1) has to be explained. In step (1), one may take any suitable quadrilateral mesh generator with a very large desired edge length (for example, larger than the longest mesh edge) to produce a very coarse conforming mesh. Afterwards, we apply one of the methods described in Section 3.1 to remove all self-intersections which might be created. As the mesh is very coarse the absolute number of such self-intersections should be small.

In certain applications it might be demanded that all nodes and all edges (the latter possibly subdivided) of the input mesh must appear in the refined mesh. However, quite often, we have the freedom to change parts of the surface mesh (of course, without changing the geometry). Then sparsification techniques to assemble macro elements lying in the same plane may be used to coarsen the input mesh for the first phase.

4. Recursive decomposition of surface meshes

Motivation. As mentioned in Section 1, we may encounter a situation during the cycle elimination process where no cycle is attractive for an immediate elimination, although several cycles could be chosen for a feasible elimination. To remedy such a situation the idea is to insert an internal two-manifold which decomposes the component into two sub-components.

A simple example for such a situation is visualized in Fig. 11 showing a ball where any immediate cycle elimination would produce a layer of hexahedra with almost coplanar adjacent facets, i.e., with a very bad mesh quality. Another example (Fig. 12) shows a change of the local mesh density (typically occurring for a segment of a torus). Here, the optimal decomposition also requires a split into two components.

Another good reason for performing a split is that a balanced partition should speed-up the overall meshing time.

The next two subsections deal with the problem of how to find a suitable primal cycle for cutting the component and how to extend the surface mesh to the added two-manifold.

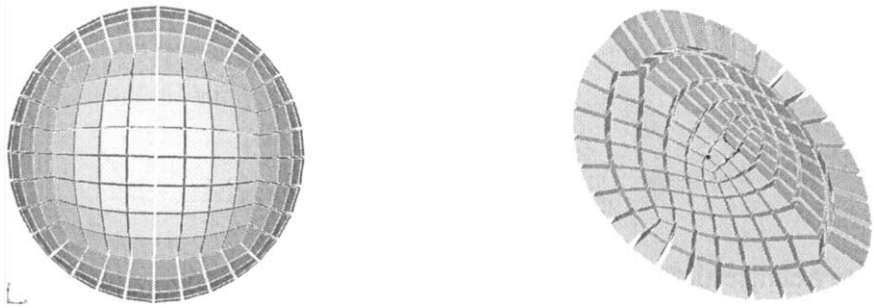

Fig. 11. The surface mesh of a complete ball is a simple example where no cycle is attractive for an immediate elimination. However, cutting the ball into halves or quarters leads to pleasant hexahedral meshes.

Fig. 12. A simple example (left) where a separation is necessary for finding the natural decomposition into 8 hexahedra by successive cycle elimination. The primal graph is displayed in solid lines, the dual cycles are dashed. The separating primal cycle is indicated by a double line.

4.1. Cycle separators

Our goal to split the surface mesh (and its dual) into exactly two components can be achieved by simple primal cycles.

Fact 6. *If we delete the dual edges corresponding to a simple primal cycle of a quadrilateral surface mesh* (*i.e., a* 3*-connected planar graph*)*, then the remaining dual graph consists of exactly two connected components.*

We want to find a primal cycle C with the following properties:

(i) C is simple,
(ii) the edges of C lie almost in a plane parallel to the normals of the quadrilaterals incident to these edges,
(iii) the separated sets of quadrilaterals Q_1 and Q_2 each contain at least one dual cycle completely,
(iv) the cardinality of the sets Q_1 and Q_2 is balanced.

To this end, we specify edge costs in the dual graph. Dual edges get zero cost if the two adjacent quadrilaterals are “almost coplanar” or even concave with respect to the interior, and a larger cost otherwise. Then we look for a minimum cost balanced cut (Q_1, Q_2) in the dual graph. To make sure that the separated vertex sets both contain at least one dual cycle completely, we select two non-intersecting

dual cycles and contract each of them (including the enclosed vertices) to a single vertex. Note that the graph remains planar by these contractions.

For general graphs, finding a minimum cost balanced cut is NP-hard. However, efficient algorithms are known to find nearly optimal solutions for planar graphs [16,17] (which would be sufficient for our purposes). In our scenario, we propose an even simpler method: We first determine a minimum cut separating the two selected dual cycles (by solving a maximum flow problem on a planar graph), and then do local exchange steps to balance the cardinality of the separated sets.

4.2. Meshing an internal manifold

Suppose that C is a primal cycle which we have selected for a decomposition step. To perform the decomposition, we have to mesh the separating 2-manifold induced by C. This has to be done carefully

(i) to guarantee that the graphs of both components are planar, simple, and 3-connected,
(ii) to maintain the invariant that all dual cycles are simple with respect to the components, and
(iii) to make sure that all edges of C appear in both components.

As the quadrilateral surface mesh is bipartite, any primal cycle has even length. So assume that C has length $2k$ for some $k \geqslant 2$. By planarity and the fact that each dual cycle is simple, the cycle C crosses each dual cycle an even number of times. Hence, if we cut the dual edges corresponding to C, each thereby affected dual cycle is cut into an even number of pieces with one half of these pieces on each side.

For the construction of the quadrilateral mesh for the internal 2-manifold bounded by C we take the dual viewpoint of rebuilding a dual cycle arrangement from these pieces. Think of the ends of these pieces as being labeled by the name of the cycle they belong to. Then the meshing of C can be seen as connecting ends with the same label by a simple path inside C such that no two paths with the same label cross. Moreover, we want that at most two paths cross at the same point. Given these properties, the induced dual cycles for the component are well-defined and simple. The set of cycles belonging to a component remains essentially the same. Note that a cycle which crosses C more than twice may now be cut into several new cycles. Finally, we insert an additional dual cycle running parallel to C (called "outer cycle" in the algorithmic description), and, if necessary, a second dual cycle to remove parallel dual edges. These parallel edges may only occur if C has two neighbored edges with the same label.

Algorithm 1 gives a more formal description of a simple way to perform this meshing. The idea is to connect the ends of dual cycles with the same label step by step in a circular fashion. In each iteration, the algorithm inserts a new path such that it crosses only a minimal set of previously layed out paths. In this process, the predicate $head(v)$ denotes the tip of the current partial path starting at vertex v. The predicate $edge(w)$ returns the first edge starting at vertex w (if existing).

Fig. 13 shows two examples for the meshing of an internal two-manifold. Note that, in general, the resulting quadrilateral mesh is neither independent of the chosen starting point nor of the chosen direction of the traversal (clockwise or counterclockwise).

Before we prove the correctness of Algorithm 1 we state two simple facts for cycle arrangements.

Fact 7. *If a cycle arrangement is connected, it is already* 2*-connected.*

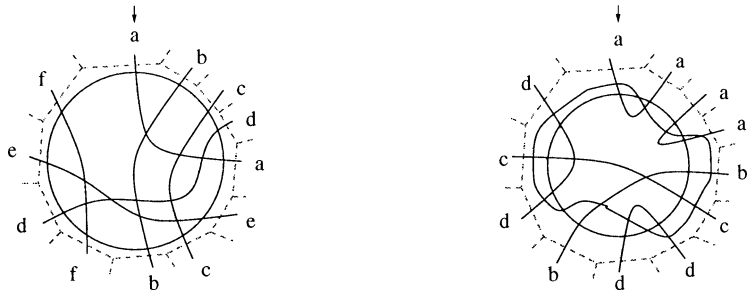

Fig. 13. Two examples for the meshing of an internal two-manifold; the resulting dual graphs are shown. The separating primal cycles are dashed; letters correspond to the labeling of the canonical cycles. The meshing algorithm proceeds in clockwise order, starting with the label a indicated by a small arrow. In the second example, an additional cycle is used to remove parallel edges.

Fact 8. *If the vertices u, v form a 2-separator of a 2-connected cycle arrangement* (*their removal disconnects the remaining graph*), *then there is a canonical cycle C in this arrangement containing both vertices u and v.*

Lemma 9. *Given a surface mesh component and a primal cycle, Algorithm 1 meshes an internal two-manifold bounded by C such that the induced sub-components are free of self-intersecting cycles if the original component satisfies this property.*

Proof. The insertion of the outer cycle guarantees that all primal edges of C will reappear in both components. Parallel edges which might occur because cycles with the same label are neighbored in C can always be avoided, in the worst case with the help of an additional outer cycle as in Fig. 13. It remains to show the 3-connectedness for both resulting components. So let us assume that the dual graph G^d of one component is not 3-connected when the algorithm terminates. As G^d is clearly connected, it is even 2-connected by Fact 7. We now assume that there is a 2-separator u, v with u and v belonging to some canonical cycle C (by Fact 8). This means that there are two nodes, s and t which are disconnected after the removal of u and v.

Case 1: $C = C^{out}$ *is the new outer cycle.*

In the following discussion we assume an embedding where all new vertices and edges created in the remeshing lie "inside" C^{out}. It is not possible that s and t both lie outside C as the subgraph outside C is connected by Fact 6. Thus, assume that s lies inside C. As each vertex belongs to two canonical cycles, there are four node disjoint paths from s to vertices of C. At most two of these paths can end in u or v. Hence, there is a path from s to some node $s' \in C$ with $s' \neq u, v$. Likewise, there is a path from t to some $t' \in C$ with $t' \neq u, v$. Using the fact that s' and t' are connected by a path outside C, we can construct a path from s to t avoiding the separator u, v. Finally, for any pair of vertices on C, there are three disjoint paths (two on the cycle, and one outside). This contradicts our assumption that u, v is a separator.

Case 2: *The two separator does not lie on* C^{out}, *i.e.,* $C \neq C^{out}$.

By construction, every vertex belonging to the interior of C^{out} lies on two cycles, but any two cycles cross at most once inside C^{out}. Hence, for each interior vertex there are four disjoint paths to the outer cycle. Hence, it is not possible that s and t are separated and lie both inside C^{out}. So we assume next that

Algorithm 1: Meshing an internal two-manifold.

input : a separating primal cycle C of a quadrilateral surface mesh without self-intersecting dual cycles; the edges of C are labeled with the dual cycle passing through it

output : a quadrilateral mesh for the internal two-manifold such that both induced sub-components of the original mesh are free of self-intersecting dual cycles

```
begin
    comment: build the dual of the quadrilateral mesh first
    create an artificial vertex v for each edge in C and label it with the name of the
    corresponding edge label;
    put these artificial vertices in order of appearance in C into list L and mark
    them as unprocessed;
    initialize head(v) := v for all vertices;
    initialize edge(v) := nil for all vertices;
    take the first unprocessed vertex v := L.first();
    repeat
        mark v as processed;
        w := L.next(v);
        break := false;
        while (w ≠ nil) and (break == false) do
            if w is processed then
                create new node x;
                split (u, w) := edge(w) into edges (u, x), (x, w);
                insert new edge e := (head(v), x);
                set head(v) := x;
                set edge(w) := (x, w);
            if (w is unprocessed) and (label(w) == label(v)) then
                mark w as processed;
                insert new edge e := (head(v), w);
                set edge(w) := e;
                break := true;
            set w := L.next(w);
        v := L.next(v);
    until all vertices of L are processed;
    insert outer cycle;
    if parallel edges exist then
        insert extra cycle to remove them;
    identify all artificial vertices contained in L to a single vertex;
    dualize the obtained planar graph and output it;
end
```

both s and t lie outside C^{out}. By the 3-connectivity of the original graph, there have been three disjoint paths between s and t before the remeshing of the interior of C^{out}. A path can only be destroyed by the remeshing if it crosses C^{out}. With the help of C^{out} we can rebuild these paths, although these paths are not necessarily disjoint on C^{out}. But by our assumption that the 2-separator does not lie on C^{out} it is not possible to disconnect all three of them. It remains the final case that s lies inside C^{out} and t outside.

There are four disjoint paths from s to the outer cycle, and at least three paths from t to C^{out}. Hence, again it is not possible to disconnect s and t by the removal of two vertices. This finishes the proof that the graph is 3-connected. □

Lemma 10. *Algorithm 1 can be implemented to run in linear time in the output, i.e., linear in the number of generated quadrilaterals.*

Proof. Every statement of the algorithm can clearly be executed in linear time except for the inner while-loop. The two if-statements within the while loop both require constant time for a single execution. As both create a new edge or node, they contribute to the output and so are linear in total. However, our implementation has to avoid spending too much time in finding the next vertex where one of the two if-statements evaluates to true. More precisely, we want an implementation which finds the next relevant case in constant time. To this end, we do some additional bookkeeping. Each vertex keeps a pointer to the next vertex in the cyclic order with the same label (these pointers can be determined in a linear time preprocessing). Moreover, we maintain an additional array of pointers. For each processed vertex this pointer gives access to the next processed vertex in the cyclic order, and returns nil otherwise. Initially, all these pointers are set to nil, but when a further node becomes marked as processed, we can update the relevant pointers in constant time. In summary, we can unroll the while-loop and always identify the next event for which we have to create a new edge in constant time. □

5. Conclusions and experimental results

In this paper, we have presented the first method which avoids the generation of self-intersecting dual cycles in the quadrilateral surface mesh generation process.

Other methods which do not integrate this objective into the surface mesh generation and use local mesh transformations only afterwards typically suffer from a degraded mesh quality. Our approach reuses a slight variation of a b-matching model for conformal mesh refinement. The key difference to previous methods lies in a careful decomposition and combinatorial layout of the b-matching solution. We also proposed a method to split components into subcomponents in order to find better candidates for a cycle elimination and showed how this can be done without creating new self-intersections.

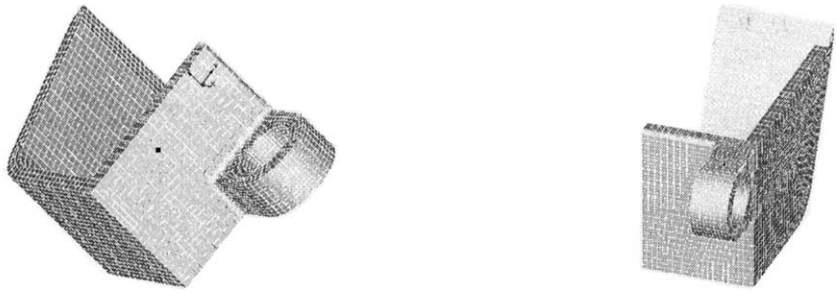

Fig. 14. Two different views on our surface mesh for a CAD model of a shovel. The surface mesh is free of self-intersecting cycles.

Fig. 15. Two different views on our hexahedral mesh for the most difficult part of the shovel. The right figure is a zoom into a typical detail.

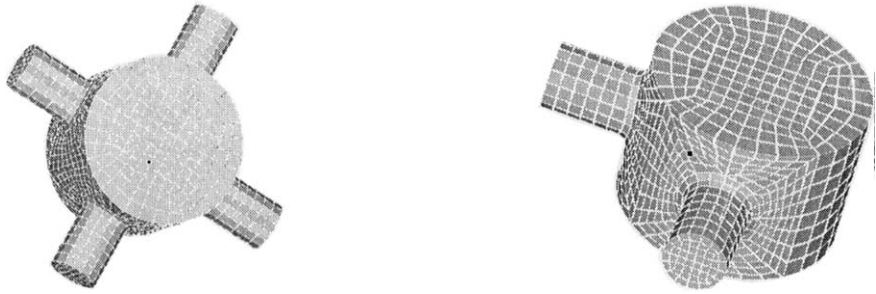

Fig. 16. Two further test instances where we obtained surface meshes without self-intersections which have perfect elimination schemes.

Exemplary results are shown in Fig. 10 for a pipe model with a junction (with 12092 hexahedra in our decomposition) and Figs. 14 and 15 for the CAD model of a shovel. The complete decomposition of the shovel consists of 24240 hexahedra (for both halves). Fig. 16 displays variations of the pipe model which possess non-trivial perfect elimination schemes after the surface meshing without self-intersections, and Fig. 17 shows a cut through one of the corresponding hexahedral meshes.

Mesh quality. To evaluate the mesh quality, we here report on two important criteria, namely, scaled Jacobians and the condition number of the Jacobian matrices [7,8]. For a vertex of a hexahedron the Jacobian matrix is formed as follows. Denote by $x \in \mathbb{R}^3$ the position of this vertex and let $x_i \in \mathbb{R}^3$ for $i = 1, 2, 3$ be the position of its three neighbors in some fixed order. Using edge vectors $e_i = x_i - x$ with $i = 1, 2, 3$ the Jacobian matrix is then $A = [e_1, e_2, e_3]$. The determinant of the Jacobian matrix is usually called *Jacobian.* If the edge vectors are scaled to unit length, we get the *scaled Jacobian* with values in the range -1.0 to 1.0. An element is said to be *inverted* if one of its Jacobians is less or equal to zero. The minimal requirement for a hexahedral mesh to be useful for the finite element method is that all hexahedra are non-inverted, and we would like to have all scaled Jacobians as close as possible to 1.0. The *condition number* $\kappa(A)$ of a Jacobian matrix A is the quantity $\kappa(A) = |A|\,|A^{-1}|$ for which we use the *Frobenius norm,* defined as $|A| = (\mathrm{tr}(A^{\mathrm{T}}A))^{1/2}$. Condition number should be as small as possible, the minimum is achieved with 3.0.

Table 1 reports the mesh quality with respect to scaled Jacobians and condition number for the instances shown in this paper. All hexahedral meshes are valid (i.e., contain only non-inverted elements),

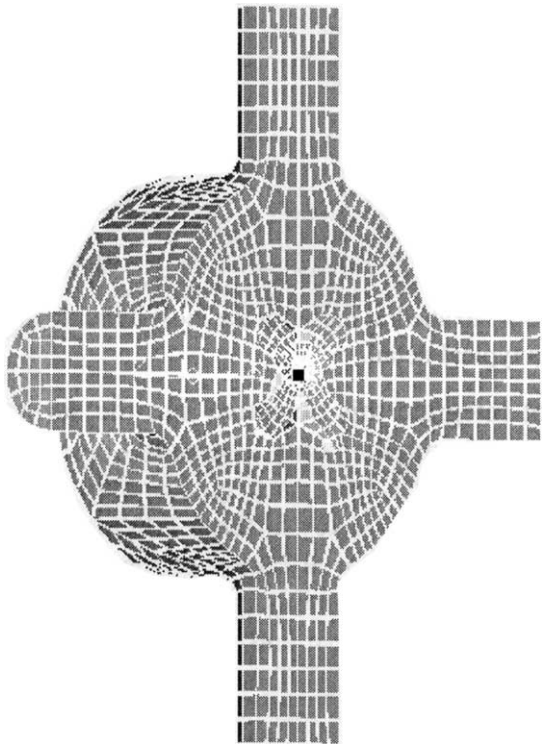

Fig. 17. Cut through the hexahedral mesh for the left instance of Fig. 16.

Table 1
Quality statistics for several hexahedral meshes

Instance	Number of	Scaled Jacobian		Condition number	
	elements	min	aver.	aver.	max
Ball (Fig. 11)	1088	0.14	0.92	3.57	18.02
Pipe 1 (Fig. 10)	12092	0.25	0.95	3.70	11.22
Pipe 2 (Fig. 16)	7452	0.04	0.85	3.79	58.08
Pipe 3 (Fig. 16)	22780	0.06	0.89	3.74	40.54
Shovel (Fig. 14)	24240	0.05	0.89	3.65	43.40

and the average mesh quality is typically high. Hence, we conclude that the methods proposed in this paper worked well for our test instances. Future work has to evaluate the benefits of these methods on more complicated test instances.

Acknowledgements

The author was partially supported by grants Mo 446/2-3 and Mo 446/2-4 of the Deutsche Forschungsgemeinschaft (DFG). The author wishes to thank G. Krause for providing us with the finite element preprocessor ISAGEN (which we used for our illustrations), and Dirk Feuchter for the CAD model of the shovel. Special thanks goes to my students Benjamin Feldhahn and Christian Trinks who helped in the implementation of our algorithms.

References

[1] T.D. Blacker, R.J. Meyers, Seams and wedges in plastering: A 3D hexahedral mesh generation algorithm, Engineering with Computers 9 (1993) 83–93.
[2] N.A. Calvo, S.R. Idelsohn, All-hexahedral element meshing by generating the dual mesh, in: S. Idelsohn, E. Oñate, E. Dvorkin (Eds.), Computational Mechanics: New Trends and Applications, CIMNE, Barcelona, Spain, 1998.
[3] S.A. Canann, Plastering: A new approach to automated, 3D hexahedral mesh generation, Amer. Inst. Aeronautics and Astronautics, Reston, VA, 1992.
[4] D. Eppstein, Linear complexity hexahedral mesh generation, Computational Geometry—Theory and Applications 12 (1999) 3–16.
[5] N.T. Folwell, S.A. Mitchell, Reliable whisker weaving via curve contraction, Engineering with Computers 15 (1999) 292–302.
[6] J.E. Hopcroft, R.E. Tarjan, Dividing a graph into triconnected components, SIAM J. Comput. 2 (1973) 135–158.
[7] P.M. Knupp, Matrix norms and the condition number: A general framework to improve mesh quality via node-movement, in: Proceedings of the 8th International Meshing Roundtable, South Lake Tahoe, CA, 1999, pp. 13–22.
[8] P.M. Knupp, Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities, Part II—A framework for volume mesh optimization, Internat. J. Numer. Methods Engrg. 48 (2000) 1165–1185.
[9] S.A. Mitchell, A characterization of the quadrilateral meshes of a surface which admit a compatible hexahedral mesh of the enclosed volume, in: Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS'96), Lecture Notes in Computer Science, Vol. 1046, Springer, Berlin, 1996, pp. 465–476.
[10] R.H. Möhring, M. Müller-Hannemann, K. Weihe, Mesh refinement via bidirected flows: Modeling, complexity, and computational results, J. ACM 44 (1997) 395–426.
[11] R.H. Möhring, M. Müller-Hannemann, Complexity and modeling aspects of mesh refinement into quadrilaterals, Algorithmica 26 (2000) 148–171.
[12] M. Müller-Hannemann, Hexahedral mesh generation by successive dual cycle elimination, Engineering with Computers 15 (1999) 269–279.
[13] M. Müller-Hannemann, Shelling hexahedral complexes for mesh generation, Technical Report 632/1999, Fachbereich Mathematik, Technische Universität Berlin, 1999.
[14] M. Müller-Hannemann, A. Schwartz, Implementing weighted b-matching algorithms: Insights from a computational study, ACM Journal of Experimental Algorithmics 5 (2000) Article 8.
[15] S. Owen, Meshing research corner, http://www.andrew.cmu.edu/user/sowen/mesh.html.
[16] J.K. Park, C.A. Phillips, Finding minimum-quotient cuts in planar graphs, in: Proceedings of the 25th Annual ACM Symposium on the Theory of Computing, 1993, pp. 766–775.

[17] S. Rao, Faster algorithms for finding small edge cuts in planar graphs, in: Proceedings of the 24th Annual ACM Symposium on the Theory of Computing, 1992, pp. 229–240.
[18] R. Schneiders, Open problem, available online at http://www-users.informatik.rwth-aachen.de/~roberts/open.html, 1995.
[19] R. Schneiders, Information on finite element mesh generation, http://www-users.informatik.rwth-aachen.de/~roberts/meshgeneration.html.
[20] T.J. Tautges, S.A. Mitchell, Whisker weaving: Invalid connectivity resolution and primal construction algorithm, in: Proceedings of the 4th International Meshing Roundtable, SAND95-2130, Sandia National Laboratories, Albuquerque, NM, 1995, pp. 115–127.
[21] T.J. Tautges, T. Blacker, S.A. Mitchell, The whisker weaving algorithm: A connectivity-based method for constructing all-hexahedral finite element meshes, Internat. J. Numer. Methods Engrg. 39 (1996) 3327–3349.
[22] W. Thurston, Hexahedral decomposition of polyhedra, Posting to Sci.Math., 25 October 1993, http://www.ics.uci.edu/~eppstein/gina/Thurston-hexahedra.html.