ELSEVIER

Computational Geometry 9 (1998) 257–276

Computational Geometry
Theory and Applications

Converting triangulations to quadrangulations [☆]

Suneeta Ramaswami [a,*], Pedro Ramos [b], Godfried Toussaint [a]

[a] *School of Computer Science, 3480 University Street, McGill University, Montréal, Québec, Canada*

[b] *Departamento de Matematica Aplicada, EUIT Aeronautica, Universidad Politecnica de Madrid, Madrid, Spain*

Communicated by S. Ramaswami; submitted 14 November 1995; accepted 30 March 1997

Abstract

We study the problem of converting triangulated domains to quadrangulations, under a variety of constraints. We obtain a variety of characterizations for when a triangulation (of some structure such as a polygon, set of points, line segments or planar subdivision) admits a quadrangulation without the use of Steiner points, or with a bounded number of Steiner points. We also investigate the effect of demanding that the Steiner points be added in the interior or exterior of a triangulated simple polygon and propose efficient algorithms for accomplishing these tasks. For example, we give a linear-time method that quadrangulates a triangulated simple polygon with the minimum number of outer Steiner points required for that triangulation. We show that this minimum can be at most $\lfloor n/3 \rfloor$, and that there exist polygons that require this many such Steiner points. We also show that a triangulated simple n-gon may be quadrangulated with at most $\lfloor n/4 \rfloor$ Steiner points inside the polygon and at most one outside. This algorithm also allows us to obtain, in linear time, quadrangulations from general triangulated domains (such as triangulations of polygons with holes, a set of points or line segments) with a bounded number of Steiner points. © 1998 Elsevier Science B.V.

Keywords: Matchings; Triangulations; Quadrangulations; Simple polygons; Mesh-generation; Finite element methods; Scattered data interpolation

1. Introduction

A central problem in the manufacturing industry concerns the simulation of a wide variety of processes, such as fluid flow in injection molding, by solving complicated systems of partial differential equations [11]. To make this task easier, the method of *finite elements* is usually employed [20]. In this approach a solid model of the object under study (or its bounding surface) is divided up into small pieces determined by data points sampled on the object's surface.

[☆] The first and third authors were supported by NSERC Grant No. OGP0009293 and FCAR Grant No. 93-ER-0291. The second author's research was carried out during his visit to McGill University in 1995 and was self-supported.

[*] Corresponding author. E-mail: ramaswam@cs.mcgill.ca; rsuneeta@crab.rutgers.edu.

In scattered bivariate data interpolation one is required to construct a bivariate function (or surface) that fits data that has been collected at sampled points on the plane [34]. One application of such a problem in the area of computer cartography is the construction of approximate models of terrains from data consisting of the elevation at a given finite set of sampled points [15]. To facilitate this process the data points in the plane are used to divide it into small pieces. Each such piece then gives rise to a surface patch and these surface patches are finally "stitched" together to form the desired approximation to the surface.

One fundamental geometric problem in applications such as those mentioned above is the construction of a *mesh* from the given set of data points. For several decades the favored mesh used in such applications has been the *triangular mesh* or triangulation of the data points [15]. In a triangular mesh the finite elements are, as the name implies, triangles. As a result, triangulations of sets (such as sets of points, line segments, polygons, etc.) have been studied in depth and much is known about them [6]. However, in some situations for both the finite element and the scattered data interpolation problems, it is preferable that the finite elements be *quadrangles* (quadrilaterals) instead of triangles. For example, it has recently been shown that quadrangulations have several advantages over triangulations for the problem of scattered data interpolation [26] and that improvements in elasticity analysis can be obtained in finite element methods by using quadrangles rather than triangles [2]. Unfortunately, not much is known about quadrangulations of point sets and good quadrangular meshes are harder to generate than good triangular meshes [21]. In fact, if edges are allowed to be inserted only between the given data points (i.e., no extra points called Steiner points are permitted) then not all sets of points admit a quadrangulation. The characterization of quadrangulations of point sets and the design of algorithms for their efficient computation using the minimum number of Steiner points have only just begun [10]. In [10] it is shown that a set of points admits a quadrangulation without Steiner points if and only if the number of points on the convex hull is even.

In practical problems faced by engineers, the typical input consists of a set of points lying in the interior of a polygon with holes [19,22]. Since little is known about computing quadrangulations, whereas triangulations have been well studied for several decades [6], engineers have devoted some attention to the problem of *converting* triangulations to quadrangulations [19,22,35]. These methods however are heuristic, conceptually rather cumbersome and may require many Steiner points. For example, Johnston et al. [22] integrate several heuristics into a system that automatically converts a triangular mesh into a quadrangular mesh which runs in $O(n^2)$ time and may add more than n Steiner points in the process, where n is the size of the triangular mesh. No attempts appear to have been made to optimize either the number of Steiner points or the complexity of the corresponding algorithms.

We remark that quadrangulations of polygons (without given data points inside) have been investigated in the computational geometry literature for some time in several different contexts. First we note that, as with points, arbitrary simple polygons do not always admit a quadrangulation. In fact, it is not difficult to construct polygons that require $\Omega(n)$ Steiner points in order to complete a quadrangulation. On the other hand *orthogonal* polygons (also isothetic or rectilinear) always admit a quadrangulation *without* Steiner points. In fact, such polygons always admit quadrangulations in which every quadrangle is *convex*, a useful property not only in the context of polygonal region illumination or guarding but also in finite element methods. For this reason, non-convex quadrangulations of orthogonal polygons are not interesting and have not been studied. An existential proof that orthogonal polygons always admit convex quadrangulations was first given by Kahn et al. [23]. A constructive proof with an $O(n)$ time algorithm was first obtained by Sack and Toussaint [32] for star-shaped polygons and subsequently generalized to run in $O(n \log n)$ time for arbitrary simple orthogonal polygons by Sack [31]. Edelsbrunner et al. [16], Lubiw [27] and Sack and Toussaint [33], among others, later obtained additional constructive variants with similar time complexities. An orthogonal polygon with holes does not necessarily admit a convex quadrangulation and Lubiw [27] showed that to determine if this is possible is NP-complete. For references to additional special cases of quadrangulation problems, the reader is referred to [36].

In this paper we study the problem of *converting* general triangulated domains to quadrangulations, under a variety of constraints. We focus on a careful study of quadrangulating simple polygons and show later that these techniques extend to general triangulated domains such as polygons with holes and data points inside (the case of particular interest to engineers). We demand that the quadrangles obtained be *strict* quadrangles, i.e., that quadrangles not contain three collinear vertices, which would in effect make them triangles. For example, in some mesh generation methods [17] and in the recent efficient scattered data interpolation algorithms [26], the quadrangles must be strict. Although strict quadrangulations may be obtained by adding Steiner points on the boundary or diagonals, we obtain strict quadrangulations by considering only Steiner points added in the exterior or interior (and not on a diagonal) of the polygon. For the simple case when no Steiner points are allowed, i.e., when it is asked whether a quadrangulation can be obtained simply by removing a carefully selected subset of edges of the triangulation, we point out the connection between quadrangulations and *perfect matchings* of the dual graphs of the triangulations in question. We obtain a variety of characterizations for when a triangulation (of some structure such as a polygon, set of points, line segments or even a triangulated planar subdivision) admits a quadrangulation without using Steiner points (or with a bounded number of Steiner points). We also investigate the effect of demanding that the Steiner points be added in the interior or exterior of a triangulated simple polygon. Furthermore, we propose efficient algorithms for accomplishing these tasks.

In Section 2, we show that every n-gon may be quadrangulated in $O(n)$ time with at most $\lfloor n/3 \rfloor$ outer Steiner points, and that there exist polygons that require this many outer Steiner points (we define *outer Steiner points* to be Steiner points that are added outside the simple polygon). In the remainder of the paper, we describe algorithms for converting triangulations to quadrangulations; we call these *percolation algorithms*. In Section 3, we give a linear time algorithm for computing a maximum matching in a tree that also has the additional property that all unmatched nodes are leaves of the tree. This matching algorithm yields a method that quadrangulates a triangulated simple polygon with the *minimum* number of outer Steiner points required for that triangulation, this minimum being at most $\lfloor n/3 \rfloor$. In Section 4, we show that a triangulated simple polygon may be quadrangulated with at most $\lfloor n/4 \rfloor$ Steiner points *inside* the polygon and at most one outside. We should point out that it is not always possible to quadrangulate a simple polygon with Steiner points only on the inside; for example, a pentagon. We conclude Section 4 with a discussion of the applications of these percolation techniques to the problem of obtaining quadrangulations from general triangulated domains, such as triangulated sets of points, line segments or polygons with holes and data points inside. Finally, we conclude the paper by presenting some open problems in Section 5.

2. Triangulated polygons

In this section we restrict our attention to simple polygons. First we dispense with a remark concerning our non-standard term "quadrangle" for the ubiquitous "quadrilateral". All polygons except

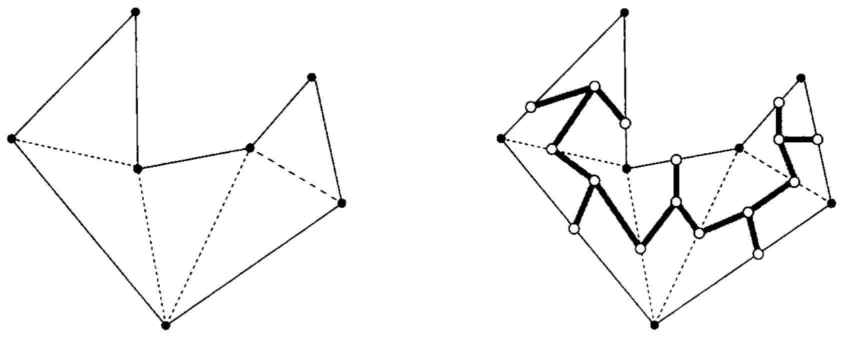
Fig. 1. Example of simplest construction of a quadrangulation from a triangulated polygon.

(for some unknown reason) the polygon of four vertices, are referred to by their number of vertices (angles) rather than their sides (latus). In the words of Coxeter [14], "*it is more usual to call this a quadrilateral, but to do so is unreasonable, as the word triangle refers to its vertices rather than its sides, and so too does the word pentagon*". We assume that a polygon has $n > 3$ vertices. As pointed out in the previous section, not all polygons admit a quadrangulation. In such cases, it is necessary to add "Steiner points" (i.e., points that are not vertices of the original polygon) in order to quadrangulate the polygon. In this and the following section, we address the question of obtaining a quadrangulation of a simple polygon after it has been triangulated. This implies we are allowed to delete existing diagonals, but no new diagonals between pairs of vertices are allowed to be inserted. Also, we do not allow deletion of vertices of the original polygon.

Probably the simplest method to obtain a quadrangulation of a triangulated polygon is to first insert a Steiner point in the interior of every edge and diagonal of the triangulated polygon (note that this violates our definition of "allowed" Steiner points, as described in the introduction). Then, for each triangle insert an extra Steiner point anywhere in the interior of the triangle (as long as it does not make three collinear Steiner points with any other pair of Steiner points in that triangle) and connect it to the three other Steiner points of that triangle. Such a quadrangulation is illustrated in Fig. 1. This method has several advantages. For one, by choosing the interior Steiner point carefully (i.e., in the interior of the triangle defined by the other three Steiner points) a convex quadrangulation can be obtained [17]. The algorithm is trivial to implement and it runs in linear time. Observe that this algorithm works for any triangulated domain. The problem with this approach is that although it leads to strict quadrangulations, it uses too many Steiner points when it is desirable to keep this number small. In fact, this approach will always use $3n - 5$ Steiner points in a triangulated simple n-gon.

Another approach that uses about one third as many Steiner points is via the Hamiltonian triangulation algorithm of Arkin et al. [3]. With a very different goal in mind, namely, fast rendering in computer graphics, Arkin et al. proposed an elegant method of obtaining what they call a Hamiltonian-cycle triangulation. Such a triangulation has the property that its dual graph admits a Hamiltonian cycle. Bose and Toussaint [10] recently proposed a method to obtain quadrangulations of point sets via what they called serpentine triangulations. A triangulation is serpentine if its dual graph admits a Hamiltonian path. By combining the ideas of [3,10] we can obtain an algorithm for quadrangulating a triangulated simple polygon as follows (refer to Fig. 2). First a Hamiltonian-cycle triangulation is obtained with the algorithm of Arkin et al. [3]. Consider a triangulated simple polygon as in Fig. 2(a). First, a planar dual

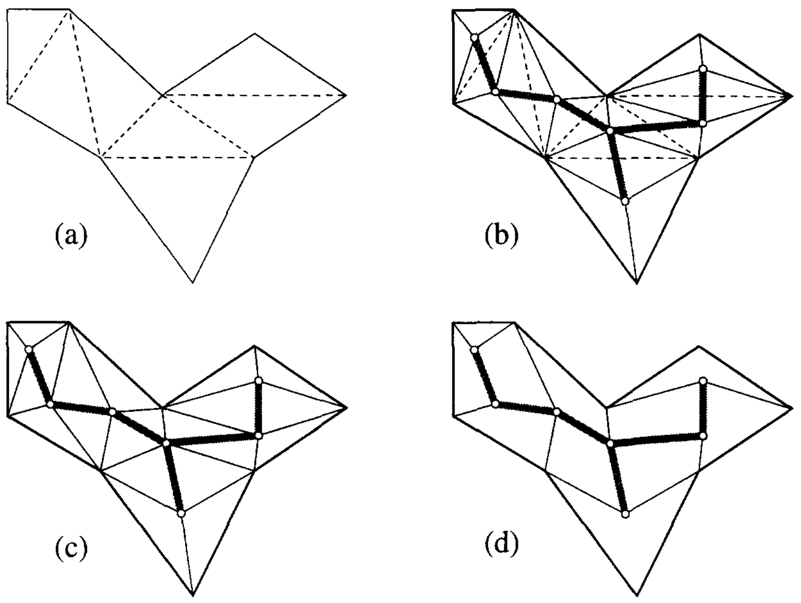

Fig. 2. Quadrangulation via Hamiltonian triangulation. (a) Original triangulated polygon. (b) Geometrical dual tree inserted with each node of the tree connected to the three vertices of its corresponding triangle. (c) Original diagonals removed. (d) A resulting quadrangulation with a single triangle remaining, where an outer Steiner point is inserted.

tree is inserted in the triangulated polygon. That this can always be done in a triangulation or convex quadrangulation was first proved by Bern and Gilbert [7]. Next, in each triangle the node in the dual tree corresponding to this triangle is connected with edges to the three vertices of the triangle. Finally, the original diagonals of the triangulated polygon are removed to yield the Hamiltonian triangulation shown in Fig. 2(c). The Hamiltonian cycle contained in the dual of the triangulation can be found by performing a tree traversal of the geometrical dual tree; this allows us to visit every triangle in the Hamiltonian order. To obtain a quadrangulation it suffices to follow the Hamiltonian order (starting at any triangle) and delete every other diagonal. A quadrangulation obtained in this way is illustrated in Fig. 2(d). Note that the last element may be a triangle in which case we may add one additional *outer* Steiner point to convert this triangle to a quadrangle. Although this algorithm is slightly more complicated than the previous one, it still runs in $O(n)$ time. Furthermore, at most one outer Steiner point is needed and the number of internal Steiner points is always $n-2$, i.e., at most $n-1$ Steiner points in all. Note that this method does not violate our conditions for converting the triangulation to a quadrangulation because even though it discards *all* diagonals, it does not insert new diagonals between pairs of vertices. Although the Hamiltonian approach gives a marked improvement in the number of Steiner points used, we show that by using coloring arguments for triangulated polygons [13,18], we can further reduce the number of Steiner points by a factor of three and this is optimal.

Before proceeding, we make our definition of Steiner points more precise. As pointed out in the introduction, no Steiner points may be placed on the boundary of the polygon or on diagonals. Therefore, we consider only two types of Steiner points: *inner* and *outer*. Inner Steiner points lie in the strict interior of the polygon (but not on a diagonal) and outer Steiner points in the strict exterior. Furthermore, for the case when only outer Steiner points are allowed, the boundary of the original

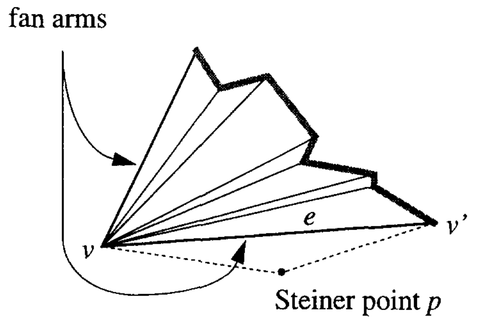

Fig. 3. A *fan* in the decomposition always begins and ends with a polygon edge.

polygon may be modified in the following way: each outer Steiner point p is affiliated with a single edge e of the original polygon, the edge e is deleted and two new edges are created by connecting p to the two end-points of e.

The following theorem gives us tight bounds on the number of outer Steiner points that are required to quadrangulate a triangulated polygon under the above conditions.

Theorem 2.1. $\lfloor n/3 \rfloor$ *outer Steiner points are always sufficient, and sometimes necessary, to quadrangulate a triangulated simple polygon of* n *vertices. Furthermore, these Steiner points may be located in* $O(n)$ *time.*

Proof. Fisk [18] observed that since the vertices of a triangulated polygon can be three-colored, every triangulation of an n-gon P can be partitioned into $\leqslant \lfloor n/3 \rfloor$ *fans* by choosing the least-occurring color (a *fan* is a triangulation where one vertex, called the *fan center*, is shared by all the triangles). Observe that there is always a decomposition such that these fans start and end at edges of the polygon (this follows from the three-coloring argument used by Fisk to partition the triangulated polygon into fans). We refer to such edges of P as *fan-arms* (see Fig. 3). It follows that each fan-arm appears in only one fan.

Consider now a vertex v of P that is a fan center. Vertex v defines a sequence of triangles in the triangulation. These triangles can be paired up to form quadrangles. If the number of such triangles is odd, we will be left with one triangle, one of whose edges is a fan-arm e. One of the endpoints of e is v; let the other be v'. We can convert this to a quadrangle by adding a Steiner point p in a suitable location outside e, deleting the edge e and connecting p to the two vertices v and v'.

Thus we need to add at most one Steiner point per fan. Since P can be partitioned into $\leqslant \lfloor n/3 \rfloor$ fans, it follows that $\lfloor n/3 \rfloor$ outer Steiner points are always sufficient to quadrangulate a triangulated simple polygon.

In order to see that $\lfloor n/3 \rfloor$ outer Steiner points are sometimes necessary to quadrangulate a triangulated polygon, consider the triangulated polygon in Fig. 4 (this is similar to an example of a polygon that requires $\lfloor n/3 \rfloor$ guards). There are only three ways in which fans may be chosen:

- If v_1 is chosen as one of the fan centers, then the other fan centers must be the vertices $v_4, v_7, v_{10}, \ldots, v_{n-2}$. These fans consist of single triangles and hence they will each need one outer Steiner point for the quadrangulation.
- If $v_3, v_6, v_9, \ldots, v_{n-3}, v_n$ are chosen as the fan centers, each of the fans has an odd number of triangles, and hence each of them will need one outer Steiner point for the quadrangulation.

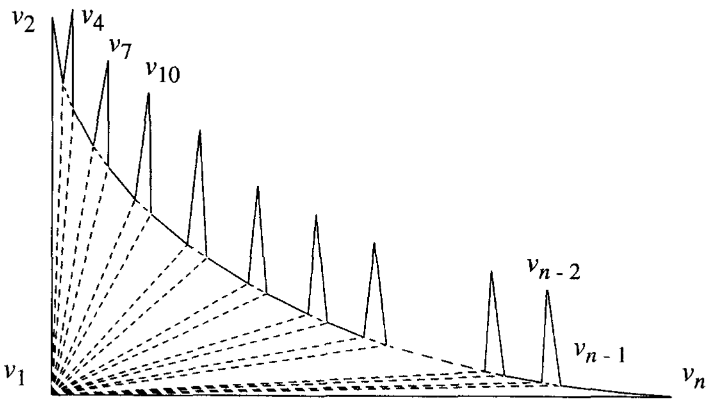

Fig. 4. Converting this triangulated polygon to a quadrangulation requires $\lfloor n/3 \rfloor$ outer Steiner points.

- If $v_2, v_5, v_8, \ldots, v_{n-1}$ are chosen as the fan centers, we have a case similar to the above.

We see that in each of the above cases, $\lfloor n/3 \rfloor$ outer Steiner points are necessary in order to obtain a quadrangulation from the triangulated polygon.

To see that these Steiner points can be located in $O(n)$ time, consider the following. The triangulated polygon can be three-colored in linear time (Kooshesh and Moret [25]). The edge on which a guard is placed gives us the fan-arm e outside which we place the Steiner point. To find an appropriate placement of the Steiner point, we may triangulate the simple polygon (or polygons) that lie outside P and within the convex hull of P, in $O(n)$ time using Chazelle's algorithm [12]. The Steiner point for e can be placed anywhere inside the triangle incident on e (and in the exterior of P). If e is an edge of the convex hull, then the Steiner point can be located in the interior of the region determined by the intersection of three half planes, one determined by the edge e in question and that does not contain P, and the other two determined by the edges of the convex hull adjacent to e and that contain P. It follows therefore that all Steiner points can be located in $O(n)$ time. □

Theorem 2.1 actually implies a more fundamental result concerning the quadrangulation of simple polygons in general, i.e., without reference to "converting a triangulated polygon". First, given a simple polygon, it can always be triangulated in $O(n)$ time [12] before applying the conversion algorithm. Second, the polygon in Fig. 4 admits only one possible triangulation (as shown) and, since no internal Steiner points are allowed, only these diagonals may be used in quadrangulating the polygon. We therefore have the following result.

Corollary 2.1. $\lfloor n/3 \rfloor$ *outer Steiner points are always sufficient, and sometimes necessary, to quadrangulate any simple polygon of* n *vertices. Furthermore, these Steiner points may be located in* $O(n)$ *time.*

3. Quadrangulations and matchings

Consider a planar subdivision which has the property that every face is classified in one of three ways: an *outer face*, an *object face* or *a hole*. The outer face is the only unbounded face. Bounded faces that do not belong to the object are called holes. By a *triangulation*, we mean a planar subdivision in which every object face is a triangle and every edge of the subdivision belongs to at least one object face. From now on, when we use the phrase "triangle of the triangulation", we refer exclusively to an object face of the triangulation. The *dual graph* of a triangulation is the graph in which there is a node for every triangle of the triangulation, and an edge between two nodes if the corresponding two triangles share a side.

Given a graph $G = (V, E)$ (possibly weighted) with V as the set of nodes and E as the set of edges, a *matching* M on G is a set of edges such that no two of them have a common node. The *maximum cardinality matching* problem is that of finding a matching of maximum size. Similarly, the *maximum weight matching* problem is that of finding a matching of maximum weight. A *perfect matching* is a matching such that every node in V belongs to an edge of the matching. Note that this is slightly different from some definitions of perfect matching found in the literature [1,30], where the definition allows one extra "free", i.e., unmatched node (when $|V|$ is odd, there will be at least one unmatched node in a maximum matching). Our definition is more appropriate in the context of quadrangulations.

When we obtain a quadrangulation from a triangulation, we would like to add as few Steiner points as possible with the constraint that diagonals between pairs of vertices can only be deleted and not inserted. Consequently, the idea of pairing up neighboring triangles in a triangulation to form quadrangles immediately implies that our goal is to find the maximum possible number of such pairings. This corresponds precisely to the maximum cardinality matching problem for the dual graph of the triangulation.

If a triangulation T can be quadrangulated without Steiner points, it means that we can eliminate some of the edges of the triangulation so that the resulting set of object faces are quadrangles. In other words, all the quadrangles are formed by pairs of triangles that share a side. In the dual graph, consider the set M of edges defined by these pairs of triangles. The matching M is perfect (since T can be quadrangulated). Conversely, let M be a perfect matching of the dual graph. Each edge in M gives us a quadrangle and, since M is perfect, there are no left-over triangles in the triangulation. It follows that we can obtain a quadrangulation of T without using Steiner points. Therefore, a triangulation admits a quadrangulation without Steiner points if and only if the dual graph of the triangulation admits a perfect matching.

As we will see in the remainder of this paper, this relation between quadrangulations and matchings gives us a powerful unified approach to handle the problem of obtaining quadrangulations from triangulations while possibly adding Steiner points. For example, let us assume we are given a triangulated domain such as a triangulated polygon that contains holes. Applying any of the classical maximum matching algorithms to the dual graph of the triangulated polygon maximizes the number of quadrangles obtained while minimizing the number of left over unpaired triangles. Thus computing a maximum matching answers the question of whether the triangulation admits a quadrangulation without the use of Steiner points. Using the fastest matching algorithm available due to Micali and Vazirani [28] this can be accomplished in $O(n^{1.5})$ time. On the other hand, if the domain is a triangulated simple polygon then the dual graph is a tree and maximum matchings can be computed faster by exploiting this added structure. Recently a general theory has been developed for solving a variety of optimization problems on a class of graphs (called tree-decomposable) in linear time using dynamic programming [4,8]. This class of problems includes maximum matchings and the class of graphs includes trees. On the other hand a simpler and more straight-forward linear-time algorithm for computing maximum matchings of acyclic graphs was discovered by Klee and Van den Driessche [24] twenty years ago, although this work seems to be unknown in graph theory circles. We can use the latter algorithm to determine if the triangulated polygon admits a quadrangulation without Steiner points. However, if such is not the case, we are also still interested in obtaining a quadrangulation with the minimum number of outer Steiner points. Using the above algorithms, if we are lucky, the unpaired triangles will each have an edge on the boundary of the polygon so that they can be converted to quadrangles using outer Steiner points. We would then obtain a quadrangulation with the minimum number of *outer* Steiner points possible for the given triangulation. We would like to point out that it is not true in general that the number of unmatched nodes in a maximum matching of the dual graph is equal to the number of Steiner points required to quadrangulate the given triangulation. For example, by adding just one inner Steiner point, we can obtain a quadrangulation of the triangulation with dual $K_{1,3}$ (whose maximum matching has two unmatched nodes).

In the rest of this section, we show that a maximum matching of the dual tree with all unmatched nodes at the leaves can be computed in linear time. This immediately yields algorithms to obtain a quadrangulation from any given triangulated polygon in linear time with the minimum number of outer Steiner points (for that triangulation). We will also give bounds on the number of Steiner points that may be necessary for the quadrangulation. In addition, the methods used in these algorithms also give rise to efficient algorithms for obtaining quadrangulations by adding a bounded number of Steiner points only *inside* the polygon. Before we proceed, we give some basic definitions and properties of matchings that are relevant for our purpose (see, for example, [1,30]).

The edges in a matching M of an undirected graph $G = (V, E)$ are called *matching edges* and the edges not in M are called *free*. A node is *matched* if it is one of the nodes of a matching edge and *free* (or *unmatched*) otherwise. If (u, v) is an edge in M, then the node u is called the mate of v in the matching M. An alternating path is a simple path in G whose edges are alternately matching and free. If both the end nodes of an alternating path are free, then the path is called an augmenting path. If M has an augmenting path, then M cannot be a maximum matching: this is because we can obtain a matching of size $|M| + 1$ by interchanging the matching and free edges along the path. Less obviously, the converse is also true and we have the following lemma.

Lemma 3.1 [5,29]. *M is a maximum matching of a graph $G = (V, E)$ if and only if G has no augmenting paths with respect to M.*

We now give the description of simple linear-time algorithms, which we call the *percolation algorithms*, that give us maximum matchings for binary trees. Note that we focus on binary trees because the graphs that are of interest to us are either duals of triangulations of polygons or the spanning trees of dual graphs. The nodes in these dual graphs have degree at most 3. (Some of these techniques will generalize to general trees, but we will not go into that here.) Interestingly, the methods described here provide us with an alternate proof of Theorem 2.1. More importantly, the technique behind the percolation algorithms can be used to obtain quadrangulations from *general* triangulated domains by adding Steiner points. For example, we will be able to give upper bounds on the number of Steiner points for obtaining quadrangulations from triangulated polygons with holes and triangulated sets of line segments. It is useful to observe that the number of outer Steiner points given by the percolation algorithms for triangulated simple polygons will, in general, be less than the number of Steiner points given by the fan-decomposition approach described in Theorem 2.1; this follows from the fact that the matching implied by that approach is not, in general, a maximum matching.

Let $T = (V, E)$ be a binary tree, and without loss of generality, assume that T is rooted at a node of degree one (this makes no difference to our algorithm, but makes the discussion simpler). Let h be the number of levels in T (with the root being at level 1). For any node u in T, we use level(u) to denote the level at which u occurs. Consider now the following matching algorithm, which we call the *percolate-up algorithm*. Let V_h be the set of nodes at level h of T. Although all these nodes are leaves, clearly not all leaves of T are in V_h. Let $v \in V_h$ and let par(v) represent v's parent. We have the following cases:

- *Case 0*. If par(v) is a node of degree one, then T consists of two nodes joined by an edge. In this case, match v and par(v). Note that if par(v) is NIL (i.e., v does not have a parent) then T consists of just a single node and we leave it unmatched.
- *Case 1*. par(v) is a node of degree 2. In this case, match v and par(v).
- *Case 2*. par(v) is a node of degree 3 and v is the left child of par(v). In this case, match v and par(v).
- *Case 3*. par(v) is a node of degree 3 and v is the right child of par(v). In this case, leave v unmatched.

For each $v \in V_h$, perform the above matching step and then prune T in the following way. If Case 0 applies, delete v and par(v) from T (if par(v) is NIL, then just v is deleted). If Case 1 applies, delete v and par(v) from T. Note that for every Case 2, there must be a Case 3. Hence if Case 2 applies, we delete v, par(v) and v's sibling (v and par(v) are matched, and v's sibling remains unmatched). After the matching and pruning steps have been carried out for all $v \in V_h$, we have a new tree in which the number of levels is either $h - 1$ or $h - 2$. Let $T^{(1)}$ denote this pruned version of T. Repeat the above matching and pruning step on all nodes at the lower-most level of $T^{(1)}$, and obtain a new pruned tree denoted by $T^{(2)}$. Continue this step with successively pruned trees $T^{(3)}$, $T^{(4)}$ and so on until we obtain $T^{(k)}$, where $T^{(k)}$ is the empty tree. Note that $k \leqslant h$.

The matching M found by the percolate-up algorithm for the tree T cannot have any augmenting paths with respect to M and hence it follows from Lemma 3.1 that M is a maximum matching. To see this, consider any two unmatched nodes u and v in T and refer to Fig. 5. Without loss of generality, let level$(u) \geqslant$ level(v). Note that since level$(u) \geqslant$ level(v), u cannot be the root. By our algorithm, u can be an unmatched node only if it is the right child of a node of degree 3. In this case, par(u) is matched to its left child (u's sibling). Since level$(u) \geqslant$ level(v) the path from u to v must go through par(u) and par(par(u)). It follows therefore that there cannot be a path of alternating free and matching edges from u to v. In other words, T cannot have any augmenting paths.

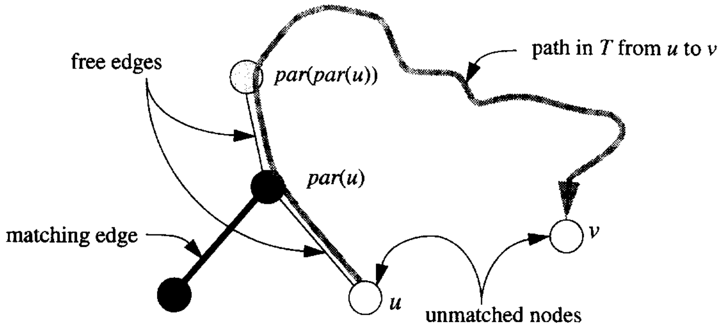

Fig. 5. There cannot be an augmenting path in T from u to v.

Consider now the time complexity of the algorithm. Let V_i denote the set of vertices of T at level i, where $1 \leqslant i \leqslant h$. Each V_i can be found by using well-known strategies (such as depth-first or breadth-first search) to traverse through the tree. We assume that each set is maintained as a linked list and that each node in T maintains a pointer to its location in one of the V_i. These steps can be done in $O(n)$ time. Every time the percolate-up algorithm deletes a node from T, that node is also deleted from the V_i to which it belongs. In addition, the degree of the parent (if undeleted) of that node is also updated. Thus there is a constant amount of work done per node during the matching and pruning steps of the algorithm. It follows that the total run-time of percolate-up is $O(n)$.

The percolate-up algorithm gives a maximum matching in which some of the unmatched nodes are internal. However, we are interested in a maximum matching in which the unmatched nodes are at the leaves. This is because, for simple polygons, the quadrangulation can then be obtained immediately by adding a Steiner point in constant time for each unmatched node (which corresponds to a boundary triangle in the triangulation). We now show that the maximum matching obtained by the percolate-up algorithm can be modified appropriately, while maintaining linear run-time, to yield a maximum matching with all its unmatched nodes occurring at the leaves.

Lemma 3.2. *There exists a maximum matching for a tree T such that all its unmatched nodes are leaves of T.*

Proof. Let M be the maximum matching found by the percolate-up algorithm. Let u be an unmatched interior node in T and refer to Fig. 6. First, we show that there is an alternating path from u to a leaf.

Note that u's children must be matched nodes because otherwise we could match u and an unmatched child to increase the size of M by one, which contradicts the fact that M is a maximum matching. Let u_1 be the child of u; if u has two children, then let u_1 be the left child of u. Let u_2 be the node matched with u_1. The children of u_2 must be matched nodes because otherwise there exists an augmenting path from u to an unmatched child, which contradicts the fact that M is a maximum matching. Let u_3 be the child of u_2; u_3 is the left child if u_2 has two children. Let u_4 be the node matched with u_3. In this manner, continue to find the remaining nodes u_5, u_6, etc. until we reach a leaf node. Call this leaf node l_u (we use this notation because, as we shall see shortly, each such leaf can be affiliated with only one unmatched node). This leaf node must be a matched node, since otherwise we have found an augmenting path. We use P_u to denote this path from u to l_u. In other words, P_u is the following path in T: $u \to u_1 \to u_2 \to u_3 \to u_4 \to \cdots \to u_{m-1} \to u_m = l_u$. Observe that such a path is uniquely defined for each internal unmatched node u. Furthermore, P_u is an alternating path, where (u_1, u_2), (u_3, u_4), (u_5, u_6), $\ldots$, (u_{m-1}, l_u) are matching edges.

We now show that for every two unmatched internal nodes u and v, the paths P_u and P_v are disjoint. If u and v are at the same level then P_u and P_v are disjoint because they lie in the subtrees rooted at u and v, respectively, and these sub-trees are disjoint. Therefore assume without loss of generality that level$(u) \leqslant$ level(v). If v does not lie in the sub-tree rooted at u, then P_u and P_v are disjoint for the same reason, mentioned above. If v lies in the right sub-tree (if it exists) of u, then P_v must lie entirely in the right sub-tree of u and hence P_u and P_v cannot overlap. If v lies in the left sub-tree of u, then v cannot lie on the path P_u since all nodes along this path are matched nodes. This means that v must lie in a sub-tree coming from one of the nodes u_i along the path P_u, such that the sub-tree is completely disjoint from P_u. In other words, v must lie in one of the sub-trees marked T_{u_1}, T_{u_2}, $\ldots$, $T_{u_{m-1}}$ in Fig. 6. It follows therefore that P_u and P_v cannot overlap. Thus each matched leaf can be

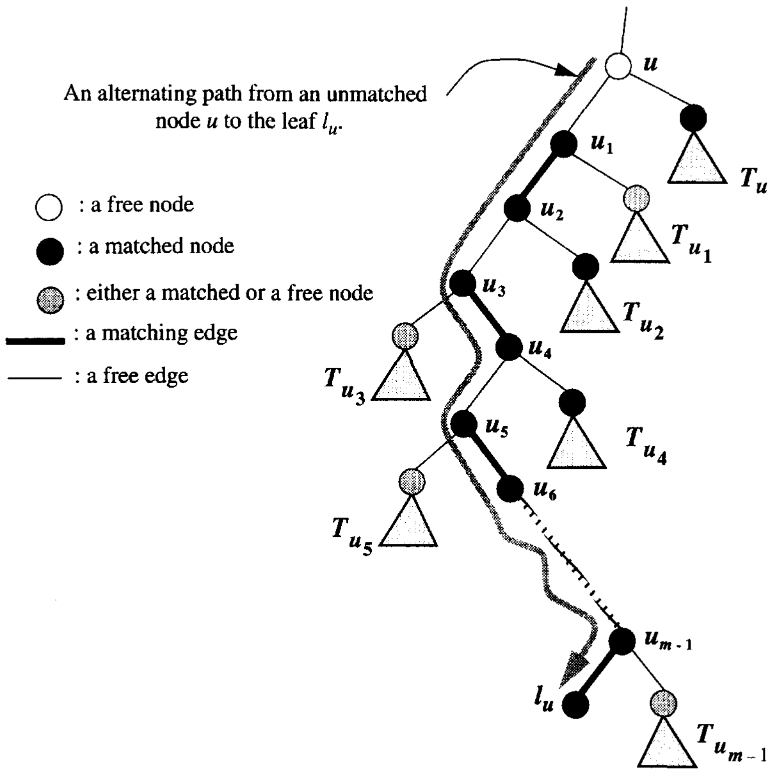

Fig. 6. Every internal unmatched node given by the percolate-up algorithm corresponds to a matched leaf.

affiliated (in the manner described above) with at most one unmatched internal node through a unique path.

The matching M can now be modified in the following manner so that we obtain another maximum matching with all unmatched nodes at the leaves. As above, let u be an internal node that is unmatched with respect to M and let P_u be the path from u to l_u. We can exchange the matching and free edges along P_u so that (u, u_1), (u_2, u_3), (u_4, u_5), ..., (u_{m-2}, u_{m-1}) are now matching edges and l_u is an unmatched node. Observe that the new matching M' has the same size as M and hence is also maximum. We can do this for every unmatched node u given by the matching M. Since the paths P_u are disjoint, the exchange of matching and free edges in one path will not interfere with the exchange on any other path. It follows therefore that M' is a maximum matching for T, with the property that all unmatched nodes are leaves of the tree. □

The above proof suggests a modification of the percolate-up algorithm to give a linear-time algorithm for finding a maximum matching with all its unmatched nodes occurring at the leaves. We do this as follows. First find a maximum matching M for the rooted tree T by using the percolate-up algorithm. Then perform a tree-traversal on T by using a pre-order tree-walk, where the root is examined and then recursively its left sub-tree followed by the right sub-tree (if it exists). The idea is that if an unmatched internal node u is encountered while walking through T, it is percolated *down* along the path P_u by swapping matching and free edges one edge at a time. At the end of the tree-walk, each such u will be matched and the leaf l_u will be unmatched.

Let M be the matching found by the percolate-up algorithm. If (u, v) is a matching edge in M, then we say that mate$(u) = v$ and mate$(v) = u$. If a node u is unmatched, then mate$(u) = \phi$. The tree traversal is adapted for our purposes, as described below in the procedure *MatchTreeWalk*. MatchTreeWalk will modify the matching M to obtain the new matching M' with all unmatched nodes at the leaves. We call this new algorithm the *percolate-up-and-down algorithm*:

- Run the *percolate-up* algorithm on the tree T to find the matching M.
- Perform a tree-walk on T by calling MatchTreeWalk(T, root):

```
MatchTreeWalk($T$, $u$);

If ($u \neq$ NIL) then
   If (mate($u$) = $\phi$ and $u$ is an internal node) then
      {mate($u$) $\leftarrow v$, where $v$ is $u$'s child (left child, if $u$ has two children);
       $v' \leftarrow$ mate($v$);
       mate($v$) $\leftarrow u$;
       mate($v'$) $\leftarrow \phi$; note that the unmatched node has been percolated down
                           by an edge along the path $P_u$
      }
   MatchTreeWalk($T$, left child of $u$);
   MatchTreeWalk($T$, right child of $u$).
```

At the end of MatchTreeWalk, we have the required matching M' as given by the function mate. We thus have the following result.

Theorem 3.1. *The percolate-up-and-down algorithm gives a quadrangulation of a triangulated simple n-gon by using the minimum number of outer Steiner points required to quadrangulate the given triangulation. In the worst case, at most $\lfloor n/3 \rfloor$ outer Steiner points are used. This algorithm runs in $O(n)$ time.*

Proof. Observe that this method gives the minimum number of outer Steiner points that are required to quadrangulate the given triangulation, since percolate-up-and-down finds a maximum matching for the dual tree.

To see that this algorithm uses at most $\lfloor n/3 \rfloor$ outer Steiner points, it is enough to show that the number of unmatched nodes in the dual tree T (as given by the percolate-up algorithm) is at most $\lfloor n/3 \rfloor$. This is because one Steiner point is added for each unmatched node. Observe that the percolate-up algorithm gives an unmatched node only when the rooted tree T, or one of the pruned versions $T^{(1)}, T^{(2)}, T^{(3)}, \ldots, T^{(k)}$ has a node of degree three such that both its children are leaves. In this case, the node and its left child are matched and the right child is left unmatched, after which all three nodes are deleted. Thus, every time the percolate-up algorithm gives an unmatched node, three nodes are pruned from the tree. If the tree consists of a single node, then that node remains unmatched. Thus the number of unmatched nodes in a tree T with t nodes is at most $\lceil t/3 \rceil$. Since $t = n - 2$ for the dual tree T, it follows that the number of unmatched nodes in the dual tree of a triangulated simple n-gon is at most $\lfloor n/3 \rfloor$.

Finally, note that both the percolate-up algorithm as well as MatchTreeWalk take $O(n)$ time each. Therefore the entire algorithm takes $O(n)$ time. □

To conclude, we remind the reader that from Corollary 2.1, it follows that $\lfloor n/3 \rfloor$ outer Steiner points is the best possible.

4. Inner Steiner points and quadrangulating general triangulated domains

We now introduce a percolation algorithm that we call the *Q-percolation* algorithm, which converts a triangulated polygon to a quadrangulation while adding Steiner points *inside* the polygon (we call these *inner Steiner points*), with at most one outer Steiner point. Notice that we cannot always avoid adding one Steiner point outside, i.e., there are polygons that cannot be quadrangulated with *only* inner Steiner points. Since an n-gon has exactly $n + 2s - 2$ triangles in any triangulation with s inner Steiner points, it follows immediately that inner Steiner points alone will not suffice when n is odd (this fact is also used in [10]). Inner Steiner points are an important consideration when the goal is to quadrangulate a simple polygon without modifying the boundary of the polygon. Before we proceed, we define inner Steiner points more precisely. As with outer Steiner points, we allow the deletion of diagonals from the original triangulation and we do not allow any new diagonals to be added between vertices of the input polygon. We only allow the addition of diagonals between an inner Steiner point and vertices of the polygon.

The Q-percolation algorithm for quadrangulating a triangulated simple polygon uses ideas similar to those in the percolate-up algorithm. First consider the following simpler version of the algorithm, which gives us an upper bound of $\lfloor n/2 \rfloor$ *inner* Steiner points (and at most one outer Steiner point) for quadrangulating a triangulated simple polygon. We will then refine this argument to tighten the bound. As before, let T be the dual tree of the triangulated simple polygon which we assume to be rooted at a node of degree one and let h be the number of levels in T (with the root being at level 1). As in the percolate-up algorithm, the Q-percolation algorithm starts at the lower-most level of T and prunes the tree as it proceeds up the tree. Let V_h be the set of nodes at level h of T. Let $v \in V_h$ and let $\mathrm{par}(v)$ represent v's parent. We have the following cases, analogous to the cases in the percolate-up algorithm:

- *Case 0.* If $\mathrm{par}(v)$ is a node of degree one, then v and $\mathrm{par}(v)$ (i.e., the triangles corresponding to these nodes) form a quadrangle. Remove these two nodes from T. If $\mathrm{par}(v)$ is NIL, then we have simply a triangle which can be quadrangulated with one *outer* Steiner point, which is possible because this is a boundary triangle. Note that this is the only outer Steiner point added in this method. Remove v from T.
- *Case 1.* If $\mathrm{par}(v)$ is a node of degree two, then v and $\mathrm{par}(v)$ form a quadrangle. Remove these two nodes from T.
- *Case 2.* If $\mathrm{par}(v)$ (call this u) is a node of degree three, then let w be v's sibling. Then, as illustrated in Fig. 7, we can add a Steiner point p in the triangle Δ_u corresponding to node u. Connect p to the three vertices of Δ_u, thus dividing it into three smaller triangles Δ_{u1}, Δ_{u2} and Δ_{u3} such that Δ_{u2} is adjacent to the triangle Δ_v and Δ_{u3} is adjacent to the triangle Δ_w. Thus the triangles Δ_v and Δ_{u2} can be paired up to form one quadrangle, as can the triangles Δ_w and Δ_{u3}. Now in the tree T, delete nodes v and w. The node u now corresponds to the triangle Δ_{u1}.

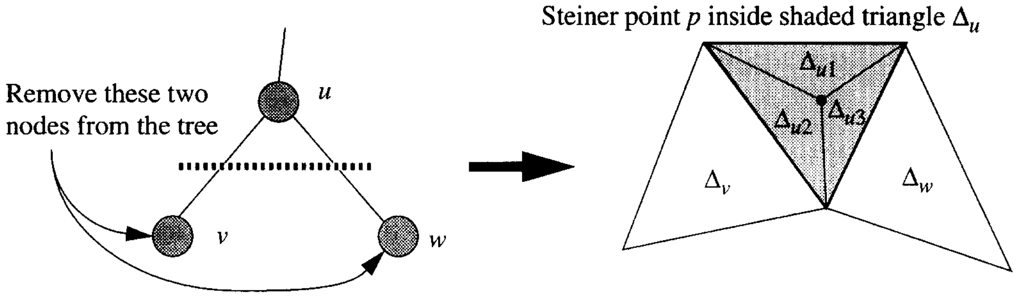
Remove these two nodes from the tree
u
v
w
Steiner point p inside shaded triangle Δ_u
Δ_{u1}
Δ_{u2}
Δ_{u3}
Δ_v
Δ_w

Fig. 7. A Steiner point p may be added in the triangle Δ_u corresponding to a node of degree three in the dual tree, as shown on the right.

After the above step is carried out for all nodes in V_h, we continue with the set of nodes in the lower-most level of the pruned version of T. The step is repeated on successively pruned trees until we are left with the empty tree. As in the percolate-up algorithm, the set of nodes at every level of T can be maintained as linked lists. Observe that all Steiner points (except possibly one) are added in the interior of the polygon. Furthermore, the number of Steiner points added is equal to the number of triangles in the triangulation that correspond to nodes of degree three in the dual tree T. Since two nodes are deleted every time a Steiner point is added, it follows that in the worst case this algorithm adds at most $\lfloor n/2 \rfloor$ inner Steiner points and at most one outer Steiner point.

This method adds Steiner points conservatively. In other words, we can tighten the upper bound by exploiting the structure of the tree T. We will now show that it is possible to delete at least four nodes of T every time an inner Steiner point is added. In order to prove the tighter bound, we use the property that pentagons are star-shaped from some point in its interior. Recall that a polygon is star-shaped if it contains a point x such that for all y in the polygon, the closed line segment xy lies in the polygon.

Theorem 4.1. *The following Q-percolation algorithm computes a quadrangulation of a triangulated simple n-gon with at most $\lfloor n/4 \rfloor$ inner Steiner points and at most one outer Steiner point in $O(n)$ time.*

Proof. We enumerate the following case analysis where V_h is as before.

Step 1. Do the following for each node $v \in V_h$. If v is such that par(v) is NIL then we have a single triangle that can be quadrangulated with one *outer* Steiner point, which is possible because this is a boundary triangle. We delete v from T. If par(v) is a node of degree 1, then these two nodes correspond to a quadrangle and we delete v and par(v) from T. For all remaining nodes $v \in V_h$ such that par(v) is a node of degree 2, delete v and par(v) from T (v and par(v) will form a quadrangle) and update the degree of the parent of par(v).

Step 2. If V_h is not empty, do the following for each $v \in V_h$. Note that all remaining v in V_h will be such that par(v) is a degree 3 node. Let w be the sibling of v. Refer to Fig. 8: the thick dotted line indicates the part of T that is deleted in this step and the shaded triangle refers to the region where the polygon possibly continues. We assume that whenever nodes are deleted from T, the degree of an affected node is updated appropriately. One of the following cases applies:

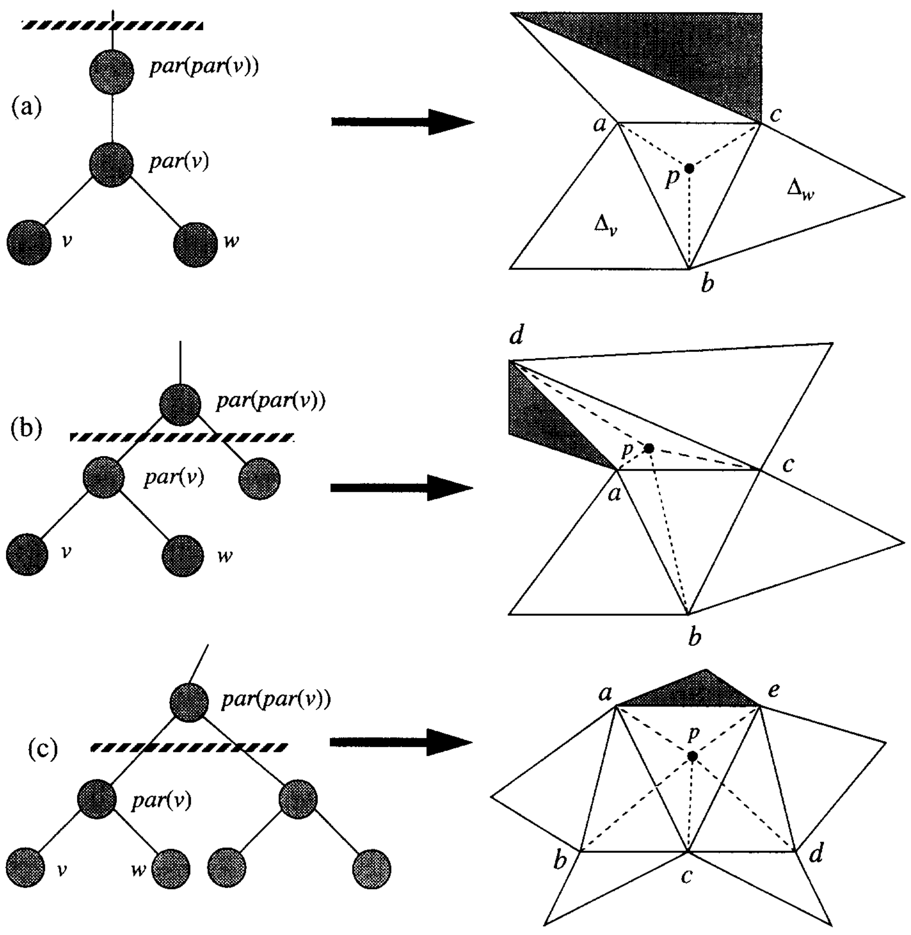

Fig. 8. The three cases that arise in the Q-percolation algorithm.

- *Case 1.* par(par(v)) *is a node of degree 1 or 2* (see Fig. 8(a)). Let triangle abc correspond to par(v). In this case we add a Steiner point p in the interior of triangle abc such that it does not create three collinear points with any vertices of the four triangles in question. Insert diagonals pa, pb and pc forming three quadrangles: the union of triangles pab and Δ_v, the union of triangles pbc and Δ_w, and the union of triangles pac and the triangle corresponding to par(par(v)). Delete v, w, par(v) and par(par(v)) from T.
- *Case 2.* par(par(v)) *is a node of degree 3.* Observe that because of Step 1 above, the sibling of par(v) must be a node of degree 1 or of degree 3. Hence we have the following two sub-cases.
  - *Case 2.1. The sibling of* par(v) *is a node of degree 1* (see Fig. 8(b)). The five triangles corresponding to the five nodes in question are converted to three quadrangles and one triangle as follows. Let $abcd$ denote the quadrangle formed by the union of the two triangles abc and acd corresponding, respectively, to par(v) and par(par(v)). Delete diagonal ac. Quadrangle $abcd$ must be star-shaped (at least from any point in the interior of segment ac). Pick a Steiner point p in the interior of the kernel of $abcd$ such that it does not create three collinear points with the vertices of the triangles in question including the parent of par(par(v)). Insert diagonals

from p to a, b, c and d creating four new triangles with p as apex and the sides of $abcd$ as bases. Now delete diagonals ab, bc and cd to form the three new quadrangles. Triangle pad is now the new triangle corresponding to par(par(v)). Delete v, w, par(v) and the sibling of par(v) from T. The node par(par(v)) now represents the smaller triangle obtained by adding the four diagonals.

- *Case 2.2. The sibling of* par(v) *is a node of degree 3* (see Fig. 8(c)). The seven triangles corresponding to the seven nodes in question are converted to four quadrangles and one triangle as follows. Let $abcde$ denote the pentagon formed by the union of the three triangles abc, cde and ace corresponding, respectively, to par(v), the sibling of par(v), and par(par(v)). Delete the diagonals ac and ce. The pentagon $abcde$ must be star-shaped from a non-zero-measure region in its interior. Pick a Steiner point p in the interior of the kernel of pentagon $abcde$ such that it does not form three collinear points with any vertices of the triangles in question including the parent of par(par(v)). Insert diagonals from p to a, b, c, d and e creating five new triangles with p as apex and the sides of pentagon $abcde$ as bases. Now delete diagonals ab, bc, cd and de to form the four new quadrangles. Triangle pae is now the new triangle corresponding to par(par(v)) in T. Now delete the following nodes from T: v, w, par(v), the sibling of par(v) and the two (leaf) children of this node.

Repeat Steps 1 and 2 on the pruned version of T, and continue doing so until the remaining tree is empty. Observe that every time the Q-percolation algorithm adds an inner Steiner point, at least four nodes are removed from T. At the very last step before the tree becomes empty, one outer Steiner point may be added. □

We can also solve several optimization versions of the quadrangulation problem with maximum *weighted* matching algorithms. For example, if we assign a weight of magnitude *one* (say) to all edges in the dual graph of the triangulation that correspond to non-convex quadrangles, and we assign an appropriate higher weight to the edges corresponding to convex quadrangles, then a maximum weighted matching algorithm will give us the quadrangulation that maximizes the number of convex quadrangles, a property that sometimes is desirable in practice [21]. Similarly, we can assign weights that measure other properties of the quadrangles besides convexity, such as *fatness*, and obtain corresponding optimal quadrangulations.

We close this section with a discussion of an important feature of the Q-percolation algorithm, which is that it can be used to obtain quadrangulations from any triangulated domain (that is, not necessarily triangulations of simple polygons). Let Γ be any triangulation, as in the definition given at the beginning of Section 3. We can quadrangulate Γ by constructing a spanning tree of the dual graph of Γ, and then applying the Q-percolation algorithm to the resulting tree (to each tree in a forest of spanning trees, if the dual of Γ is not a connected graph). Observe that the method used in the percolate-up-and-down algorithm is not particularly useful for the spanning tree of the dual graph of Γ. This is because the leaves of the spanning tree do not necessarily correspond to boundary triangles and hence unmatched leaves cannot be dealt with in the straightforward manner of the percolate-up-and-down algorithm.

The Q-percolation algorithm adds at most one Steiner point outside a triangle of the triangulation. This triangle corresponds to the root node of the dual tree. Thus in order to use the Q-percolation algorithm on Γ, we just have to ensure that the root node of the spanning tree of the dual graph of Γ corresponds to a *border triangle*. By a border triangle, we mean a triangle of the triangulation that has at least one edge that belongs either to the outer face or to a hole. The number of Steiner points required to quadrangulate these triangulations is at most $\lfloor t/4 \rfloor$, where t is the number of triangles in the triangulation Γ.

It follows therefore that we can quadrangulate triangulated polygons with holes as well as triangulated line segments. In particular, we have the following bounds for a triangulated n-gon with h holes (since it can always be decomposed into exactly $n + 2h - 2$ triangles).

Theorem 4.2. *A triangulated polygon P with n vertices and h holes can be quadrangulated in linear time with at most $\lfloor (n + 2h - 2)/4 \rfloor$ Steiner points inside the polygon and at most one outside.*

Observe also that *any* triangulation of a set of n points can be converted into a quadrangulation with at most $\lfloor (2n - 2 - h)/4 \rfloor$ Steiner points, where h is now the number of points on the convex hull of the input set of points (the number of triangles in the triangulation is exactly $2n - 2 - h$). All Steiner points will lie within the convex hull, except possibly one that lies outside.

Similarly, any triangulation of a set of n line segments can be converted into a quadrangulation with at most $\lfloor (4n-2-h)/4 \rfloor$ Steiner points, since a triangulation of n line segments is a triangulation of the $2n$ points that are vertices of the line segments. Note that for line segments, the dual graph is defined in the usual way except for the following: when a common boundary between two triangles is a line segment from the input set, the dual will not contain an edge between the two nodes corresponding to these two triangles. In practical problems of interest to engineers, the triangulated domain is derived from a polygon with holes and data points in the interior of the polygon. Our algorithms can also be used to efficiently convert these triangulations into quadrangulations.

We would like to point out that it is actually possible to show that for any triangulated domain, $\lfloor h/3 \rfloor$ outer Steiner points are always sufficient to quadrangulate the triangulation, where h is the number of edges of the triangulation that are adjacent to the outer face or a hole. This result can be derived from a basic theorem in graph theory (Petersen's theorem) that says that every 3-regular graph without cut edges has a perfect matching. We will not go into the specifics here, but refer the interested reader to [9] for details, where we present experimental results on computing quadrangulations of random sets of points by utilizing some of the ideas presented in this paper.

5. Conclusions

We presented efficient algorithms for converting triangulated domains to quadrangulations, while giving bounds on the number of Steiner points that might be required to obtain the quadrangulations. We showed that, in linear time, a triangulated simple n-gon can be quadrangulated with the least number of outer Steiner points required for that triangulation. We showed that $\lfloor n/3 \rfloor$ outer Steiner points are sufficient, and sometimes necessary, to quadrangulate a triangulated simple n-gon. We also showed that $\lfloor n/4 \rfloor$ inner Steiner points (and at most one outer Steiner point) are sufficient to quadrangulate a triangulated simple n-gon, and this can be done in linear time. Moreover, this method can also be used to quadrangulate arbitrary triangulated domains.

Some open questions arise from these results. For instance, are $\lfloor n/4 \rfloor$ inner Steiner points sometimes necessary to quadrangulate a simple n-gon? In other words, are there simple n-gons that would necessarily require $\lfloor n/4 \rfloor$ Steiner points, where we allow the Steiner points to be added only inside the polygon (with possibly one Steiner point outside)? We do not know of any non-trivial lower bounds for this problem. The least number of Steiner points required to quadrangulate a simple polygon, over all triangulations, is also an open problem. In addition, it would be interesting to look into the question of obtaining better bounds on the number of Steiner points required to quadrangulate more general triangulated domains, such as triangulated polygons with holes or triangulated sets of line segments.

Acknowledgements

The authors thank Prosenjit Bose, Bill Cook, Francisco Gómez, Peter Gritzman, Arvind Gupta, Fady Habra, Victor Klee, Anna Lubiw, Sunil Shende, Thomas Shermer and Xun Xue for interesting and useful discussions. We also thank Joe O'Rourke and anonymous referees for helpful comments and suggestions.

References

[1] R. Ahuja, T. Magnanti, J. Orlin, Network Flows: Theory, Algorithms and Applications, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] D.J. Allman, A quadrilateral finite element including vertex rotations for plane elasticity analysis, Internat. J. Numer. Methods Engrg. 26 (1988) 717–730.

[3] E. Arkin, M. Held, J. Mitchell, S. Skiena, Hamiltonian triangulations for fast rendering, in: J. van Leeuwen, ed., Algorithms – ESA '94, Lecture Notes in Computer Science 855, Utrecht, The Netherlands, September 1994, pp. 36–47.

[4] S. Arnborg, J. Lagergren, D. Seese, Easy problems for tree-decomposable graphs, J. Algorithms 12 (1991) 308–340.

[5] C. Berge, Two theorems in graph theory, in: Proc. Nat. Acad. Sci. 43 (1957) 842–844.

[6] M. Bern, D. Eppstein, Mesh generation and optimal triangulation, in: F.K. Hwang, D.-Z. Du (eds.), Computing in Euclidean Geometry, World Scientific, 1992.

[7] M. Bern, J.R. Gilbert, Drawing the planar dual, Inform. Process. Lett. 43 (1992) 7–13.

[8] M.W. Bern, E.L. Lawler, A.L. Wong, Linear-time computation of optimal subgraphs of decomposable graphs, J. Algorithms 8 (1987) 216–235.

[9] P. Bose, S. Ramaswami, G. Toussaint, A. Turki, Experimental comparison of quadrangulation algorithms for sets of points, in: Proc. of the 12th European Workshop on Computational Geometry, Münster, Germany, March 1996.

[10] P. Bose, G.T. Toussaint, No quadrangulation is extremely odd, in: Proc. of the International Symposium on Algorithms and Computation, Cairns, Australia, 4–6 December 1995.

[11] J. Bown, Injection Moulding of Plastic Components, McGraw-Hill, New York, 1979.

[12] B. Chazelle, Triangulating a simple polygon in linear time, Discrete Comput. Geom. 6 (1991) 485–524.

[13] V. Chvatal, A combinatorial theorem in plane geometry, J. Combin. Theory Ser. B 18 (1975) 39–41.

[14] H.S.M. Coxeter, Projective Geometry, Springer, New York, 1987.

[15] L. De Floriani, B. Falcidieno, C. Pienovi, Delaunay-based representation of surfaces defined over arbitrarily shaped domains, Computer Vision, Graphics and Image Processing 32 (1985) 127–140.

[16] H. Edelsbrunner, J. O'Rourke, E. Welzl, Stationing guards in rectilinear art galleries, Computer Vision, Graphics and Image Processing 27 (1984) 167–176.

[17] H. Everett, W. Lenhart, M. Overmars, T. Shermer, J. Urrutia, Strictly convex quadrilateralizations of polygons, in: Proc. of the 4th Canadian Conference on Computational Geometry, St. Johns, Newfoundland, 1992, pp. 77–82.

[18] S. Fisk, A short proof of Chvatal's watchman theorem, J. Combin. Theory Ser. B 24 (1978) 374.

[19] E. Heighway, A mesh generator for automatically subdividing irregular polygons into quadrilaterals, IEEE Trans. Magnetics 19 (6) (1983) 2535–2538.

[20] K. Ho-Le, Finite element mesh generation methods: A review and classification, Computer Aided Design 20 (1988) 27–38.

[21] B. Joe, Quadrilateral mesh generation in polygonal regions, Computer Aided Design 27 (3) (1995) 209–222.

[22] B.P. Johnston, J.M. Sullivan, A. Kwasnik, Automatic conversion of triangular finite meshes to quadrilateral elements, Internat. J. Numer. Methods Engrg. 31 (1) (1991) 67–84.

[23] J. Kahn, M. Klawe, D. Kleitman, Traditional galleries require fewer watchmen, SIAM J. Algorithms Discrete Methods 4 (2) (1983) 194–206.

[24] V. Klee, P. van den Driessche, Linear algorithms for testing the sign stability of a matrix and for finding z-maximum matchings in acyclic graphs, Numer. Math. 28 (1977) 273–285.

[25] A.A. Kooshesh, B.M.E. Moret, Three-coloring the vertices of a triangulated simple polygon, Pattern Recognition 25 (1992).

[26] M.J. Lai, L.L. Schumaker, Scattered data interpolation using C^2 piecewise polynomials of degree six, in: Third Workshop on Proximity Graphs, Mississippi State University, Starkville, MS, 1–3 December 1994.

[27] A. Lubiw, Decomposing polygonal regions into convex quadrilaterals, in: Proc. of the 1st ACM Symposium on Computational Geometry, 1985, pp. 97–106.

[28] S. Micali, V.V. Vazirani, An $O(|V|^{1/2}|E|)$ algorithm for finding maximum matchings in general graphs, in: Proc. 21st Annual IEEE Symposium on the Foundations of Computer Science, 1980, pp. 17–27.

[29] R.Z. Norman, M.O. Rabin, An algorithm for a minimum cover of a graph, in: Proc. Amer. Math. Soc. 10 (1959) 315–319.

[30] C.H. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[31] J.-R. Sack, An $O(n \log n)$ algorithm for decomposing simple rectilinear polygons into convex quadrilaterals, in: Proc. 20th Annual Allerton Conference on Communication, Control and Computing, Urbana, IL, October 1982, pp. 64–75.

[32] J.-R. Sack, G.T. Toussaint, A linear-time algorithm for decomposing rectilinear star-shaped polygons into convex quadrilaterals, in: Proc. 19th Annual Allerton Conf. on Communications, Control and Computing, Urbana, IL, 1981, pp. 21–30.

[33] J.-R. Sack, G.T. Toussaint, Guard placement in rectilinear polygons, in: G.T. Toussaint (ed.), Computational Morphology, North-Holland, Amsterdam, 1988, pp. 153–175.

[34] L.L. Schumaker, On the dimension of spaces of piecewise polynomials in two variables, in: W. Schempp, K. Zeller (eds.), Multivariate Approximation Theory, Birkhäuser, Basel, 1979, pp. 396–411.

[35] V. Srinivasan, L.R. Nackman, J.-M. Tang, S.N. Meshkat, Automatic mesh generation using the symmetric axis transformation of polygonal domains, Proc. IEEE (Special Issue on Computational Geometry) 80 (9) (1992) 1485–1501.

[36] G.T. Toussaint, Quadrangulations of planar sets, in: Proc. 4th International Workshop on Algorithms and Data Structures (WADS), Lecture Notes in Computer Science 955, Springer, New York, 1995, pp. 218–227.