Explicit array-based compact data structures for triangulations

Luca Castelli Aleardi, Olivier Devillers

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Explicit array-based compact data structures for triangulations

Luca Castelli Aleardi — Olivier Devillers

N° ????

Septembre 2011

ISSN 0249-6399 ISRN INRIA/RR--????--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SOPHIA ANTIPOLIS - MÉDITERRANÉE

Explicit array-based compact data structures for triangulations

Luca Castelli Aleardi* , Olivier Devillers†

Thème : Algorithmique, calcul certifié et cryptographie
Équipe-Projet Geometrica

Rapport de recherche n° ???? — Septembre 2011 — 20 pages

Abstract: We consider the problem of designing space efficient solutions for representing triangle meshes. Our main result is a new explicit data structure for compactly representing planar triangulations: if one is allowed to permute input vertices, then a triangulation with n vertices requires at most $4n$ references ($5n$ references if vertex permutations are not allowed). Our solution combines existing techniques from mesh encoding with a novel use of minimal Schnyder woods. Our approach extends to higher genus triangulations and could be applied to other families of meshes (such as quadrangular or polygonal meshes). As far as we know, our solution provides the most parsimonious data structures for triangulations, allowing constant time navigation in the worst case. Our data structures require linear construction time, and all space bounds hold in the worst case. We have implemented and tested our results, and experiments confirm the practical interest of compact data structures.

Key-words: triangulations, compact representations, mesh data structures, codage de graphes, Schnyder woods

This work is partially supported by ERC under the agreement ERC StG 208471 - ExploreMap.

* LIX, École Polytechnique, France
† Projet Geometrica, INRIA Sophia-Antipolis, France

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Structures de données explicites compactes basées sur des tableaux pour les triangulations

Résumé : Nous nous intéressons là la conception de représentations efficaces pour les maillages triangulaires. Le résultat principal de ce travail est une nouvelle structure de données pour la représentation compacte des triangulations planaires: si on autorise la permutation des sommets du maillages, alors une triangulations à n sommets peut se représenter avec au plus $4n$ références ($5n$ références sont nécessaires, si on ne fait pas de permutation). Notre solution combine des techniques existantes de codage de graphes avec une nouvelle utilisation des Schnyder woods minimaux. Cette approche s'étend aussi au cas de triangulations de genre supérieur et pourrait s'appliquer à d'autres familles de maillages (tels que les maillages quadrangulaires ou polygonaux). A notre connaissance, ce résultat fournit la structure de données la plus compacte pour les triangulations, permettant la navigation dans le maillage en temps constant dans le pire des cas. Le temps de construction de nos représentations est linéaire, et toutes les bornes sont valables dans le pire des cas. Nous avons implémenté et testé nos résultats, et nos expériences confirment l'intérêt pratique des structures de données compactes.

Mots-clés : triangulations, représentations compactes, structures de données pour les maillages, Schnyder woods

1 Introduction

The large diffusion of geometric meshes (in application domains such as geometry modeling, computer graphics), and especially their increasing complexity has motivated a huge number of recent works in the domain of graph encoding and mesh compression. In particular, the *connectivity* information of a mesh (describing the incidence relations) represents the most expensive part (compared to the geometry information): for this reason most works try to reduce the first kind of information, involving the combinatorial structure of the underlying graph. Many works addressed the problem from the *compression* [23, 24] point of view: compression schemes aim to reduce the number of bits as much as possible, possibly close to theoretical minimum bound according to information theory. For applications requiring the manipulation of input data, a number of explicit (pointer-based) data structures [7, 3, 4, 17] have been developed for many classes of surface and volume meshes. Most geometric algorithms require data structures which are easy to implement, allowing fast navigation between mesh elements (edges, faces and vertices), as well as efficient update primitives. Not surprisingly common mesh representations are redundant and store a huge amount of information in order to achieve the prescribed requirements. In this work we address the problem above (reducing memory requirements) from the point of view of *compact data structures*: the goal is to reduce the redundancy of common explicit representations, while still supporting efficient navigation.

1.1 Existing mesh data structures

Classical data structures in most programming environments do admit *explicit pointer-based* implementations. Each pointer stores at most one reference: pointers allow to navigate in the data structure through address indirection, and storing/manipulating service bits within references is not allowed. Many popular geometric data structures (such as *Quad-edge*, *Winged-edge*, *Half-edge*) fit in this framework. In edge-based representations basic elements are edges (or half-edges): navigation is performed storing, for each edge, a number of references to incident mesh elements. For example, in the *Half-edge DS* each half-edge stores a reference to the next and opposite half-edge, together with a reference to an incident vertex (which gives 3 references for each of the $6n$ half-edges, for a triangulation having n vertices).

Compact practical solutions. Several works [9, 26, 21, 1, 10, 19, 18, 20] try to reduce the number of references stored by common mesh representations: this leads to more compact solutions, whose performances (in terms of running time) are still really of practical interest. In this case array-based implementations are sometimes preferred to pointer-based representations, depending on the flexibility of the programming environment. Many data structures (*triangle-based*, *array-based compact half-edge*, *SOT/SQUAD data structures*) make use of a slightly stronger assumption: when dealing with a mesh of size m, each memory word can store a $\lg m$ bits integer reference, and C bits are reserved as *service bits* (C is a small constant, commonly between 1 and 4) [1]. More-

[1] For a mesh with m elements (m can be the number of vertices, edges or faces), $\lg m := \lceil \log_2 m \rceil$ bits are required to distinguish all the elements. The length w of each memory word is assumed to be $\Omega(\lg m)$ so that a reference fit in a constant number of words.

over, basic arithmetic operations are allowed on references: such as addition, multiplication, floored division, and bit shifts/masks. An interesting general approach is based on the reordering of mesh elements: for example, storing consecutively the half-edges of a face allows to save 3 references per face (as in *Directed Edge* [9], which requires $13n$ references instead of the $19n$ stored by *Half-edge*). Or still, storing edges/faces according to the vertex ordering allows to implicitly represent the map from edges/faces to vertices. This is one of the ingredients used by the *SOT* data structure [19], which represents triangulations with $6n$ references. Adopting an interesting heuristic one may even obtain a more compact solution [18], requiring about $(4 + c)$ references per vertex: as shown by experiments c is a small value (between 0.09 and 0.3 for tested meshes), but there are no theoretical guarantees in the worst case.

Theoretically optimal solutions. For completeness, we mention that *succinct representations* [14, 6, 22, 12, 11, 27] are successful in representing meshes with the minimum amount of bits, while supporting local navigation in worst case $O(1)$ time. They run under the *word-Ram model*, where basic arithmetic and bitwise operations on words of size $O(\lg n)$ are performed in $O(1)$ time. One main idea (underlying almost all solutions) is to reduce the size, and not only the number, of references: one may use graph separators or hierarchical graph decomposition techniques in order to store in a memory word an arbitrary (small) number of tiny references. Typically, one may stores up to $O(\frac{\lg n}{\lg \lg n})$ sub-words of length $O(\lg \lg n)$ each. Unfortunately, the amount of auxiliary bits needed by the encoding becomes asymptotically negligible only for very huge graphs, which makes succinct representations of mainly theoretical interest.

Finally, we observe that the parsimonious use of references may affect the navigation time: for example, the access to some mesh elements requires more than $O(1)$ time in the worst case [19, 18, 21] (as reported in Table 1.1). Moreover, most compact mesh representations do not support efficient updates: local modifications (such as vertex insertion or edge flip) can be very expensive, implying the update of many references in the worst case.

1.2 Preliminaries

Combinatorial aspects of triangulations. In this work we exploit a deep and strong combinatorial characterization of planar triangulations. A planar triangulation is a simple planar map where every face (including the infinite face) has degree 3. Triangulations are *rooted* if there is one distinguished *root face*, denoted by (v_0, v_1, v_2) , with a distinguished incident *root edge* (v_0, v_1). *Inner edges* (and *inner vertices*) are those not belonging to the root face (v_0, v_1, v_2) [2]. As pointed out by Schnyder [25], the inner edges of a planar triangulation can be partitioned into three sets $\mathcal{T}_0$, $\mathcal{T}_1$, $\mathcal{T}_2$, which are plane trees spanning all inner nodes, and rooted at v_0, v_1 and v_2 respectively. This spanning condition can be derived from a local condition: the inner edges can be oriented in such a way that every inner node is incident to exactly 3 outgoing edges, and the orientation/coloration of edges must satisfy a special local rule (see Fig. 1 for an example).

[2] For the sake of simplicity, in our drawings the root coincides with the infinite face.

Data structure	size	navigation time	vertex access	dynamic updates
Edge-based DS [17, 3, 4]	$18n+n$	$O(1)$	$O(1)$	*yes*
triangle based [7]/Corner Table	$12n+n$	$O(1)$	$O(1)$	*yes*
Directed edge [9]	$12n+n$	$O(1)$	$O(1)$	*yes*
2D catalogs [10]	$7.67n$	$O(1)$	$O(1)$	*yes*
Star vertices [21]	$7n$	$O(d)$	$O(1)$	*no*
TRIPOD [26] + reordering	$6n$	$O(1)$	$O(d)$	*no*
SOT data structure [19]	$6n$	$O(1)$	$O(d)$	*no*
SQUAD data structure [18]	$(4+c)n$	$O(1)$	$O(d)$	*no*
(no vertex reordering) Thm 2	$6n$	$O(1)$	$O(d)$	*no*
(no vertex reordering) Thm 4	$5n$	$O(1)$	$O(d)$	*no*
(with vertex reordering) Thm 6	$4n$	$O(1)$	$O(d)$	*no*
(with vertex reordering) Cor 6	$6n$	$O(1)$	$O(1)$	*no*

Table 1: Comparison between existing data structures for triangle meshes. All bounds hold in the worst case, at the exception of SQUAD data structure, whose performances are interesting in practice for common meshes, but with no theoretical guarantees.

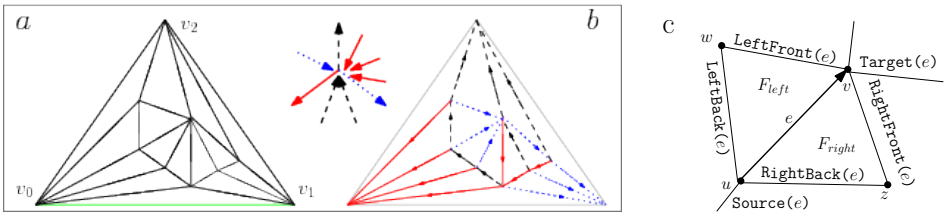

Figure 1: A planar triangulation with 9 vertices (a), endowed with a (minimal) Schnyder wood (b) (the local Schnyder condition around inner vertices is also shown). Picture (c) illustrates the navigational operations supported by our representations.

Definition 1 ([25]). *Let $\mathcal{G}$ be a planar triangulation with root face (v_0, v_1, v_2). A **Schnyder wood** of $\mathcal{G}$ is an orientation and labeling, with label in $\{0, 1, 2\}$ of the inner edges such that the edges incident to the vertices v_0, v_1, v_2 are all ingoing and are respectively of color 0, 1, and 2. Moreover, each inner vertex v has exactly three outgoing incident edges, one for each color, and the edges incident to v in* counter clockwise *(ccw) order are: one outgoing edge colored 0, zero or more incoming edges colored 2, one outgoing edge colored 1, zero or more incoming edges colored 0, one outgoing edge colored 2, and zero or more incoming edges colored 1 (this is referred to as **local Schnyder condition**).*

Navigational operators. Here are the operators supported by our representations. Let $e = (u, v)$ be an edge oriented toward v, which is incident to (u, v, w) (its left triangle) and to (u, v, z) (its right triangle), as depicted in Fig. 1(c).

- `LeftBack`(e), returns the edge (u, w).
- `LeftFront`(e), returns the edge (v, w).
- `RightBack`(e), returns the edge (u, z).
- `RightFront`(e), returns the edge (v, z).
- `Source`(e) (resp. `Target`(e)), returns the origin (resp. destination) of e;
- `Edge`(u), returns an edge incident to vertex u;
- `Point`(u), returns the geometric coordinates of vertex u.

The operators above are supported by most mesh representations [9, 3], and allow full navigation in the mesh as required in geometric processing algorithms: their combination allows to iterate on the edges incident to a given node, or to walk around the edges incident to a given face.

Overview of our solution In order to design new compact array-based data structures, we make use of many ingredients: some of them concerning the combinatorics of graphs, and some of them pertaining the design of compact (explicit) data structures. The main steps of our approach are the following:

- as done in [9, 19, 18], we perform a reordering of cells (edges), to implicitly represent the map from vertices to edges, and the map from edges to vertices;
- as done in [26], we exploit the existence of 3-orientations (edges orientations where every inner vertex has outgoing degree 3) for planar triangulations [25]. This allows to store only two references per edge.

Combining these two ideas one can easily obtain an array-based representation using $6n$ references, allowing $O(1)$ time navigation between edges and $O(d)$ time for the access to a vertex of degree d: for the sake of completeness, this simple solution will be detailed in Theorem 2. Our main contribution is to show how to get further improvements and generalizations, using the following ideas:

- we exploit the existence of *minimal* Schnyder woods, without cycles of directed edges oriented in ccw direction. And also the fact that, given the partition $(\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2)$, the two trees $\mathcal{T}_0$ and $\mathcal{T}_1$, are sufficient to retrieve the triangulation. With these ideas we store only $5n$ references (Theorem 4).
- we further push the limits of the previous reordering approach: by rearranging the input points according to a given permutation and using a special kind of order on plane trees (the so-called *DFUDS* order [5]), we are able to use only $4n$ references (Theorem 6);
- we also show how to reformulate a recent generalization of Schnyder woods [13], in order to deal with genus g triangulations: our representation requires at most $5(n + 4g)$ references (Theorem 8);

To our knowledge, these are the best (worst case and guaranteed) upper bounds obtained so far, which improve previous existing results.

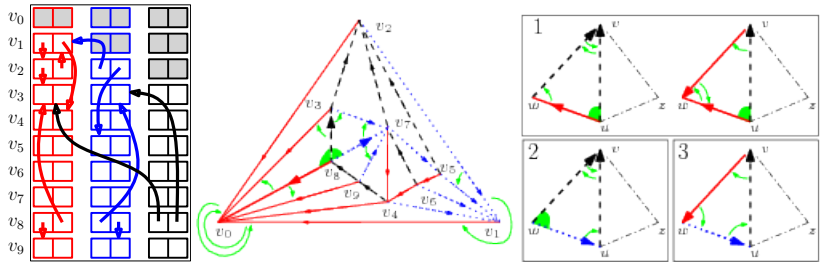

Figure 2: Our first solution. For each vertex v we store of 6 consecutive integer references, corresponding to the its 3 outgoing edges. To help intuition, table T is drawn as a bi-dimensional array of size $n \times 6$. The case analysis of Theorem 1 is illustrated on the right (where edge (u, v) has color 2).

2 Compactly representing triangulations

2.1 The first data structure: simple and still redundant

We first design a simple data structure requiring $6n$ references, which allows to perform all navigational operators in worst case $O(1)$ time, and `Target` operator in $O(d)$ time (when retrieving a degree d vertex). This is a preliminary step in describing a more compact solution. Observe that our first scheme achieves the same space bounds as the *Tripod data structure* by Snoeyink and Speckmann [26]. Although both solutions are based on the properties of Schnyder woods, the use of references (between edges) is different: this is one of the features which allow to make our scheme even more compact in the sequel.

Scheme description. In a first step we compute a Schnyder wood of the input triangulation $\mathcal{G}$ (this requires linear time, as shown in [25]). We then define the tree $\overline{\mathcal{T}}_0$ by adding edges (v_0, v_1) and (v_2, v_0) to $\mathcal{T}_0$ (oriented toward v_0); we also define a tree $\overline{\mathcal{T}}_1$ by adding the edge (v_2, v_1) to the tree $\mathcal{T}_1$ (oriented toward vertex v_1), as depicted in Fig. 2. Using this decomposition each edge gets a color and an orientation: edges in $\overline{\mathcal{T}}_0$ are called red edges, those in $\overline{\mathcal{T}}_1$ blue edges and those in $\mathcal{T}_2$ black edges. For each vertex, we store 6 integers representing the 3 incident outgoing edges.

Vertices will be identified by integers $0 \leq i < n$ and edges by integers $3 \leq j < 3n$ (some indices between 0 and 8 are omitted, as it will be shown in the sequel). Our data structure consists of an array T of size $6n$, two arrays of bits S_a, S_b of size $3n$, and an array P of size n storing the geometric coordinates of the points. The entries of T and P are sorted according to the order of input points, which facilitates the implementation of the `Point` operator. By convention, the three edges having vertex i as source are indexed $3i$, $3i+1$ and $3i+2$, where edge having index $3i+c$ has color c. For each oriented edge we store two references to 2 neighboring edges. References are arranged in table T, in such a way that for each inner node u of $\mathcal{G}$, the outgoing edges associated with u are stored consecutively in T (refer to Fig. 2). The adjacency relations of an inner edge j are stored in entries $2j$ and $2j+1$ of table T, as follows:

- $T[2j]$ contains the index of `LeftFront(`j`)`, and $T[2j+1] =$ `RightFront(`j`)`;

Arrays S_a and S_b have an entry for each edge and are defined as follows:

- $S_a[j] = 1$ if edge j and `LeftBack`(j) have the same source, 0 otherwise;
- $S_b[j] = 1$ if edge j and `RightBack`(j) have the same source, 0 otherwise.

Edges belonging to (v_0, v_1, v_2) are stored in a similar manner [3]: some edge indices are not used, since vertices on the outer face do not have outdegree 3 .

We can state the following:

Theorem 2. *Let $\mathcal{G}$ be a triangulation with n vertices. The representation described above requires $6n$ references, while allowing to support* `Target` *operator in $O(d)$ time (when dealing with degree d vertices) and all other operators in $O(1)$ worst case time.*

Proof. Let us consider operations involving an edge $e = (u, v)$ of color c, oriented from u toward v, whose incident left (resp. right) triangle is (u, v, w) (resp. (u, v, z)). With abuse of notation, we denote by e the index of this edge.
Operators `Edge`(u) *and* `Point`(u)*:* to get the index of an edge incident to a given vertex u we simply compute $3u$, which returns the edge of color 0 incident to u. The geometric coordinates of vertex u are naturally stored in $P[u]$.
Operator `Source`(e)*:* to get the vertex u which is origin of an edge e is also an easy procedure, and requires only to compute $e/3$ (the integer floored division).
Operator `LeftFront`(e)*:* by definition, we can get edge (v, w) just reading the reference stored in $T[2e]$.
Retrieving edge colors/orientations: Observe that, given edge e, its color is $e\%3$ (we use this notation for the remainder operator). Again, given $e' =$ `LeftFront`(e), we retrieve the orientation of e' from its color and the color of e (according to the Schnyder local rule). We get the orientation of edge (u, w) simply reading the service bit stored in $S_a[e]$.

Operator `LeftBack`(e)**:** to get edge (u, w) we have to distinguish three cases, depending on the orientation of edges (u, w) and (v, w) (as illustrated in Fig. 2):

Case 1: $e' = (u, w)$ is directed toward w, thus being of color $c' = (c+1)\%3$. This case is easy to handle, since (u, w) is an outgoing edge from u, then its index is $e' = e + 1$ if $c \neq 2$, and $e' = e - 2$ otherwise.

Case 2: $e' = (u, w)$ is directed toward u (of color $(c - 1)\%3$), and edge (v, w) is oriented toward v (being thus of color c). We first retrieve the index of $e'' = (v, w)$, by computing $e'' =$ `LeftFront`(e) as before. Then we have $e' = e'' - 1$ if $c > 0$, and $e' = e'' + 2$ otherwise (since e' and e'' have the same source, and thus are stored in the same block in T).

Case 3: $e' = (u, w)$ is directed toward u (of color $(c-1)\%3$), and edge (v, w) is oriented toward w (being thus of color $c' = (c+1)\%3$). We first retrieve edge $e'' = (v, w)$ as before by computing $e'' =$ `LeftFront`(e). By construction, we had stored in entry $T[2e'']$ a reference from edge (v, w) to edge (w, u), so we just compute $e' =$ `LeftFront`(e'').

[3] For the sake of simplicity, in our examples the first 3 vertices belong to the root face. But there is no such a restriction in our implementations.

Operator `Target`(e)**:** unfortunately we have not stored enough information to return v in $O(1)$ time: the idea is to iteratively turn around vertex v, starting from edge (u, v) in cw direction (or ccw direction) until we get an edge $e' = (v, x)$ having v as source. Then compute `Source`(e') in $O(1)$ time as above, which results in the target of (u, v). This procedures ends after at most $d - 3$ steps (for a vertex v of degree d), since each vertex has 3 outgoing edges.
Operators `RightBack`(e) *and* `RightFront`(e)*:* observe that the traversal of the right face (u, v, z) incident to (u, v) can be handled in a similar manner as above, because of the symmetry of Schnyder woods and of our use of references. □

References encoding. As T is an array of integers $< 3n$ and S_a, S_b are arrays of bits, from a practical point of view, arrays T and S_a, S_b can be stored in a single array. Just encode the service bits within the references stored in T, where first k bits of an integer represent the index of an edge. Since we have at most $3n$ edges, we can set $k = \lceil \log 3n \rceil$. In this section we use only 2 service bits per edge: having 2 references per edge, we just store 1 service bit in each reference. Assuming 32 bits integers, we can encode triangulations having up to 2^{31} edges.

2.2 More compact solutions, via minimal Schnyder woods

In order to reduce the space requirements, we exploit the existence of a special kind of Schnyder wood, called *minimal*, not containing ccw oriented triangles:

Lemma 3 ([8])**.** *Let $\mathcal{G}$ be a planar triangulation. Then it is possible to compute in linear time a Schnyder wood without ccw oriented cycles of directed edges.*

New scheme. We modify the representation described in previous section: the first step is to endow $\mathcal{G}$ with its minimal Schnyder wood (no ccw oriented triangles). Outgoing edges of color 0 and 1 will be still represented with two references each, while we will store only one reference for each outgoing edge of color 2 (different cases are illustrated by Fig. 3, top pictures). More precisely, let $e = (u, v)$ be an edge having face (u, v, w) at its left and face (u, v, z) at its right, and let q be the vertex defining the triangle (v, z, q). For a vertex u we store 5 entries $T[5u] \dots T[5u + 4]$ as follows (edge $e = (u, v)$ being outgoing of color c)
• for $c = 1$ (as in Theorem 2), we store in $T[5u+2]$ and $T[5u+3]$ two references, respectively to (v, w) and (v, z); (edges cw and ccw around v)
• for $c = 2$, we store in $T[5u + 4]$ a reference to:
  – edge (v, z), if (z, u) is directed toward u, (edge ccw around v)
  – edge (v, w) otherwise; (edge cw around v)
• for $c = 0$, we store one reference in $T[5u]$ to (v, w) (edge cw around v) and one reference in $T[5u + 1]$ to:
  – edge (v, q) if (v, q) is of color 1 oriented toward v (and thus (v, z) must be of color 2), (second edge ccw around v)
  – edge (v, z) otherwise (edge ccw around v), as in Theorem 2.

As before, the values of $S_a[e]$ and $S_b[e]$ describe the orientations of edges `LeftBack`(e) and `RightBack`(e): service bits and modulo 3 computations suffice to retrieve the orientation of edges and to distinguish all cases (since we need 2 per edge, and we have 5 references, we have to use 2 service bits per reference).

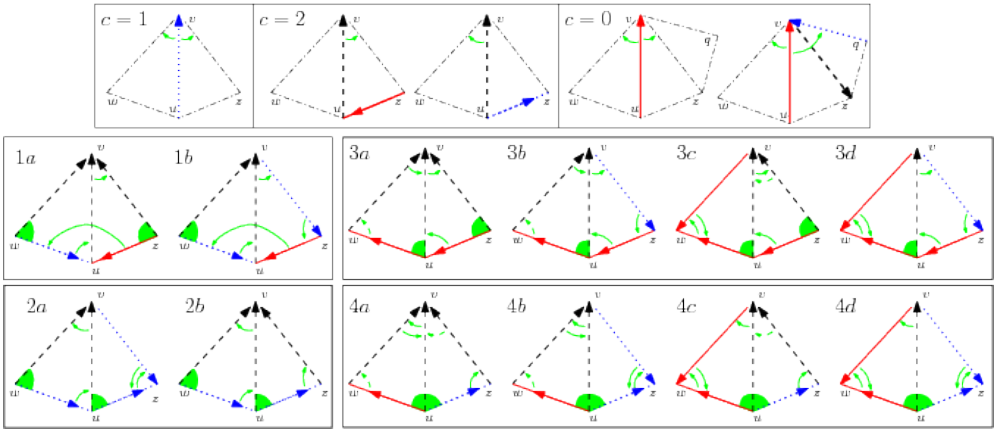

Figure 3: More compact scheme. Neighboring relations between edges are represented by tiny oriented (green) arcs corresponding to stored references, and by filled (green) corners which implicitly describe adjacency relations between outgoing edges incident to a same vertex: because of local Schnyder rule, we do not need to store references between neighboring outgoing edges.

Theorem 4. *Let $\mathcal{G}$ be a triangulation with n vertices. There exists a representation requiring $5n$ references, allowing efficient navigation, as in Theorem 2.*

Proof. The first missing information compared to the simple scheme (with $6n$ references) is $\texttt{RightFront}(e)$ when e is of color 0, edge (v, z) is oriented toward z and edge (z, u) is of color 1 oriented toward u (Fig. 3top-right): but in this case the information is easily retrieved as $\texttt{RightFront}(\texttt{e}) = \texttt{LeftFront}(e')$, where e' is stored in $T[5u+1]$ for $u = \texttt{Source}(e)$. The other missing information is either $\texttt{RightFront}(e)$ or $\texttt{LeftFront}(e)$ when e is of color 2.

It is easy to see that if the traversal does not involve edges of color 2, then the same argument as in Theorem 2 apply. Also, operators `Source`, `Target` and `Edge` are implemented similarly as in Theorem 2 (just recall that table T has size $5n$). To prove the correctness of our claim when color 2 edges are involved, our argument uses an exhaustive, and somehow tedious, case analysis (please refer to pictures in Fig. 3). Let us consider an edge $e = (u, v)$ of color 2 oriented toward v. Here is a detailed case analysis.

Case 1 The most involved case (Pictures 1a,1b) occurs when edges (w, u) and (z, u) are oriented toward u, of colors 1 and 0 respectively.

Operator `RightFront`: to get edge (z, v) we simply look at entry $T[5u + 4]$, which by definition contains the corresponding reference.

Operator `RightBack`: if edge (z, v) is oriented toward v being thus of color 2 (Picture 1a), then the index of (z, u) is $e'' = e' - 2$, where $e' = \texttt{RightFront}(e)$. Otherwise, edge (v, z) is of color 1 and oriented toward z (Picture 1b): thus edge (u, z) is given by $\texttt{RightFront}(e')$ (which is well defined since (v, z) is of color 1), where $e' = \texttt{RightFront}(e)$.

Operator `LeftBack`: first compute $e'' = \texttt{RightBack}(u, v)$, the index of edge (z, u) as above. Then the reference to (w, u) is just stored by definition in $T[5z + 1]$, where $z = \texttt{Source}(e'')$.

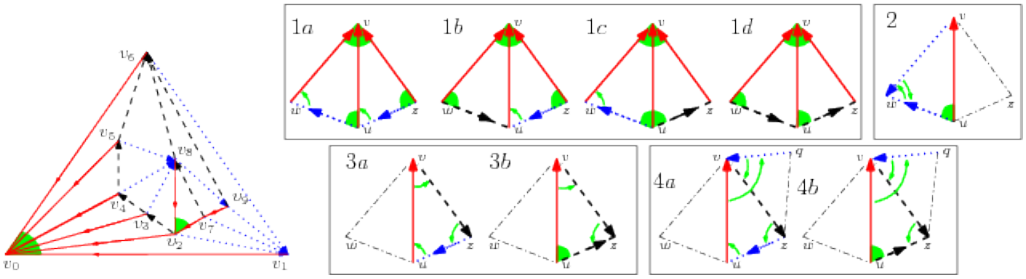

Figure 4: A planar triangulation whose vertices are labeled according to a DFUDS traversal of tree $\overline{\mathcal{T}}_0$ (left). On the right are shown all cases involved in the proof of Theorem 6: we now store only one reference for edges of color 0 (red edges), since most adjacency relations are implicitly described by the DFUDS labels.

Operator `LeftFront`: finally, given $e''' = \texttt{LeftBack}(u, v)$, we know that the index of (w, v) is $e''' + 1$, since e''' precedes (w, v) in ccw direction around w.

Case 2 The case where edges (w, u) and (u, z) are of color 1 is easier to handle. To perform `LeftFront` we retrieve the reference e' to edge (w, v) which is just stored in $T[5u + 4]$ by definition. For `LeftBack`, we obtain edge (w, u) observing that is the outgoing edge incident to w of color 1: so its index is $e' - 1$ (see Pic. 2a, 2b).

For the right traversal we proceed similarly: given the index of e, the index of (u, z) is just $e' = e - 1$ (being the outgoing edge of color 1 incident to u). Then we retrieve edge (v, z) by computing $\texttt{LeftFront}(e')$ (which is always easy to compute, for edges of color 1).

Remaining cases depicted in Fig. 3 can be handled in a similar manner.

Before concluding, observe that it is possible to distinguish between cases in $O(1)$ time, just using service bits stored in S_a, S_b, and computing colors (with the % operator). □

2.3 Further reducing the space requirement

Allowing to exploit a permutation of the input vertices (reordering the vertices according to a given permutation), we are able to save one more reference per vertex. Let us first recall a result concerning the traversal of plane trees, which has been already applied to the encoding of trees [5] and labeled graphs [2]:

Lemma 5 ([5])**.** *Let $\mathcal{T}$ be a plane tree whose nodes are labeled according to the DFUDS (Depth First Unary Degree Sequence) traversal of $\mathcal{T}$. Then the children of a given node $v \in \mathcal{T}$ have all consecutive labels.*

Scheme description We first compute a minimal Schnyder wood of $\mathcal{G}$, and perform a DFUDS traversal of $\overline{\mathcal{T}}_0$ starting from its root v_0: as $\overline{\mathcal{T}}_0$ is a spanning tree of all vertices of $\mathcal{G}$, we obtain a vertex labeling such that, for every vertex $v \in \mathcal{G}$, the children of v in $\overline{\mathcal{T}}_0$ have consecutive labels (as illustrated in Fig. 4). We then reorder all vertices (their associated data) according to their DFUDS label, and we store entries in table T accordingly. This allows us to save one

reference per vertex: roughly speaking, we do not store a reference to `LeftFront` for edges in $\overline{\mathcal{T}}_0$, which leads to store for each vertex 4 references in table T. Let $e = (u, v)$ be of color c (incident to faces (u, v, w) and (u, v, z)), and let (q, v, z) the triangle sharing edge (v, z) (as illustrated in Fig. 4). For edges of color $c = 1$, we store in $T[4u + 1]$ and $T[4u + 2]$ two references, to edges (v, w) and (v, z) respectively. For edges of color $c = 0$, we store in $T[4u]$ a reference to edge (q, v), if (v, z) is oriented toward z of color 2 and (q, v) is oriented to v of color 1. We store a reference to edge (v, z) otherwise. For edges of color $c = 2$, we store in $T[4u + 3]$ a reference to edge (v, z), if (z, u) is oriented toward u; and a reference to edge (v, w) otherwise. Service bits are stored in arrays S_a, S_b as in Theorem 4. We can state the following

Theorem 6. *Let $\mathcal{G}$ be a triangulation with n vertices. If one is allowed to permute the input vertices (their associated geometric data) then $\mathcal{G}$ can be represented using $4n$ references, supporting navigation as in previous representations.*

Proof. Compared to the representation of (Theorem 4) we loose the information concerning `LeftFront` for edges of color 0 (which was previously explicitly stored in T). An important remark is that, given an edge $e = (u, v)$ of color 0, we retrieve its left siblings in $\overline{\mathcal{T}}_0$ just using its DFUDS label.

Here is a detailed case analysis concerning the correctness of Theorem 6. We first distinguish several situations for the left traversal (cases are numbered according to Fig. 4).

Case 1: retrieving edges (v, w) and (w, u) is a rather easy task when edge (v, w) is oriented toward v of color 0, as (u, v). Instead of using references as in Theorems 2 and 4, we can navigate just exploiting the DFUDS labels of vertices u and w. Observe that vertex u immediately follows vertex w in DFUDS order, since u is the right sibling of w in the tree $\overline{\mathcal{T}}_0$: so we have just $w = u - 1$: the index of (w, v) is just $3w$. To get edge (u, w) we proceed as follows. If (u, w) is oriented toward w then just compute $e + 1$ (see Pictures 1a, 1c). Otherwise, compute first $w = \texttt{Source}(v, w)$ as above, and then return $3w + 2$, since (w, u) is the outgoing edge of color 2 incident to w (see Pictures 1b, 1d).

Case 2: if (v, w) is oriented toward w (which is of color 1), then there is only one case to consider: (u, w) must be oriented toward w of color 1, since a minimal Schnyder wood does not contain ccw oriented triangles. We proceed as follows. We get first the index of (w, u) by computing $e' = e + 1$ (since it corresponds to the outgoing edge of color 1 incident to the same vertex u). Then we have just to return $\texttt{RightFront}(e')$ (easy to compute, since e' has color 1).

Performing right traversals. We have again different cases to consider.

Case 1: if (v, z) is of color 0 oriented toward v, then we proceed similarly as showed above (case 1 for the left traversal).

Case 3: (v, z) is of color 2 oriented toward z and (q, v) is not oriented toward v. Then, by definition, $\texttt{RightFront}(e)$ is stored in $T[4u]$, where $u = \texttt{Source}(e)$. To get (u, z) just compute $\texttt{RightFront}(v, z)$.

Case 4: the most involved case occurs when (v, z) is oriented toward z and (q, v) is oriented toward v, being of color 1. This situation can be handled as in Theorem 4: just observed that the references involved are stored in the same way as in case 1a and 1b of Theorem 4. □

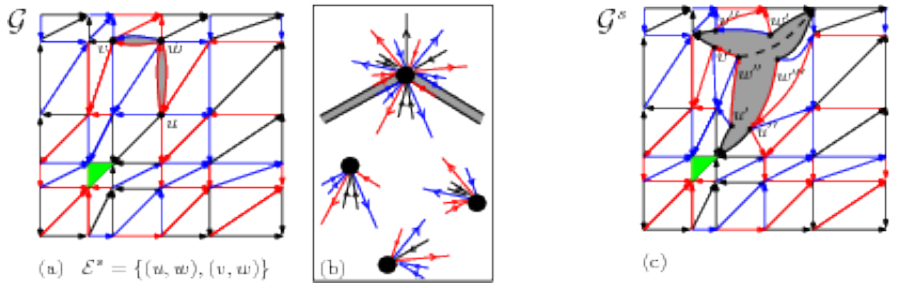

Figure 5: A genus 1 rooted triangulation endowed with a g-Schnyder wood (a): the local rule around multiple vertices is also shown (b). Every multiple vertex is incident exactly to an outgoing edge of color 2. Map $\mathcal{G}^s$ is obtained from $\mathcal{G}$ by splitting multiple vertices (c).

We can further exploit the redundancy in our representation in order to improve the computational cost of `Target` operator.

Corollary 7. *If one is allowed to permute input points, then there exists a compact representation requiring* $6n$ *references which supports all navigation operators (including* `Target`*) in worst case* $O(1)$ *time.*

Proof. The solution relies on a very simple idea: for edges of color 1 and 2 we add two entries in table T storing explicitly a reference to the target vertex v, while we do not need to store more references when dealing with edges of color 0. Observe that edges of color 0 require to store a reference for supporting the operator `RightFront` efficiently, only for the case where (v, z) is outgoing from v. So, when (v, z) is of color 0 and oriented toward v we can simply store in table T a reference to vertex v, which allows to perform `Target` in $O(1)$ time. When (v, z) is of color 2 oriented toward z, it suffices to perform `Source(RightFront`(e)`)`. □

2.4 Higher genus triangulations

We can get a compact representation requiring about 5 references per vertex, applying to a g-Schnyder wood the approach relying on DFUDS order:

Theorem 8. *Let* $\mathcal{G}$ *be a triangulation of genus* g *with* n *vertices. If one is allowed to permute input points, then there exists a representation requiring at most* $5(n + 4g)$ *references, supporting efficient navigation as in Theorem 2.*

The key ingredient to deal with higher genus surfaces is the recent generalization of Schnyder woods proposed in [13]. For a genus g (rooted) triangulation it is possible to define a coloration/orientation of inner edges such that: all the inner edges (not lying on the root face) can be oriented in one direction (having one color $i \in \{0, 1, 2\}$), at the exception of a small set $\mathcal{E}^s$ of *special* edges, which have possibly two orientations and two colors. Notice that there are at most $2g$ special edges. Moreover, all inner vertices (not lying on the root face) have outgoing degree 3, at the exception of at most $4g$ *multiple vertices*, which may have outgoing degree at most $O(g)$ (multiple vertices are the extremities of special edges).

An important fact is that a spanning property also holds in the higher genus case. Given a triangulation $\mathcal{G}$ with root face (v_0, v_1, v_2), the oriented graph $\mathcal{T}_2$ consisting of edges of color 2 is a spanning tree of all vertices in $\mathcal{G} \setminus \{v_0, v_1\}$, and $\mathcal{T}_2 \cup \mathcal{E}^s$ plus edges (v_2, v_0) and (v_2, v_1) is a cut graph of $\mathcal{G}$ (or, in other terms, a genus g map with exactly one face). A slightly different property holds for graphs $\mathcal{T}_0$ and $\mathcal{T}_1$, which are also maps of genus g spanning all inner vertices.

This generalization has been successfully applied to obtain a linear time compression scheme requiring at most $4n + O(g \log n)$ bits to encode a genus g triangulation of size n [13]. In this section we combine g-Schnyder woods with the techniques developed in Theorems 2 and 4 in order to design a space-efficient explicit data structure for dealing with genus g meshes.

For our purposes, we prefer to provide a reformulation of the structure defined in [13]: from a triangulation endowed with a g-Schnyder wood, we get a new map where all vertices have outgoing degree at most 3.

We obtain $\mathcal{G}^s$ by splitting $O(g)$ edges and vertices of $\mathcal{G}$, as follows. We split every special edge $e \in \mathcal{E}^s$ into a pair of edges e' and e'', each oriented in exactly one direction and having one color, according to the original double orientation of e. We split every multiple vertex $u \in \mathcal{G}$: if u has multiplicity m (u is incident to m special edges), then $\mathcal{G}^s$ will have $m + 1$ copies $u_1, \ldots, u_m$ of u. We also split, for each multiple vertex $u \in \mathcal{G}$, its outgoing edge of color 2: observe that only one sector of edges around u does contain an outgoing edge of color 2, as depicted in Fig. 5(b) (please refer to [13] for more details). We added in this way some new faces (at most $O(g)$), each corresponding to a sub-set of special edges which are pairwise incident (they share multiple vertices as extremities), as illustrated in Fig. 5(c).

Lemma 9. *Let $\mathcal{G}^s$ be a map obtained from a genus g rooted triangulation $\mathcal{G}$ with n vertices and f triangles, as explained above. Then $\mathcal{G}^s$ has the following properties:*

- *for every non multiple vertex $v \in \mathcal{G}$, the orientation/coloration of edges in $\mathcal{G}^s$ still satisfies the local Schnyder rule, according to Definition 1;*
- *every vertex v in $\mathcal{G}$ has outgoing degree at most 3;*
- *$\mathcal{G}^s$ is a map of genus g having $n + 4g$ vertices and at most $f + 2g$ faces, where all faces are triangles, except at most $2g$ faces corresponding to the special edges of $\mathcal{G}$.*
- *graph $\mathcal{T}_2^s$, obtained by adding to $\mathcal{T}_2$ the copies of multiple vertices, is a spanning tree of all inner vertices.*

Combining the lemma above with Theorem 2 it is straightforward to obtain a representation using about 6 references per vertex. Unfortunately, we cannot directly combine Lemma 9 and Theorem 4, since in the higher genus case the notion of minimal Schnyder woods is not still well defined: in particular, we cannot avoid ccw oriented triangles (which was a crucial point in the proof of Theorem 4).

Scheme Description First compute map $\mathcal{G}^s$ from $\mathcal{G}$ (endowed with a g-Schnyder wood) as explained above, and let denote by F the set of non triangular

faces (obtained after the splitting), and by E the set of its boundary edges. Thus E does contain both copies of a special edge of $\mathcal{G}$, as well as n_m pairs of edges of color 2, which are obtained by splitting outgoing edges of color 2 incident to multiple vertices of $\mathcal{G}$ (where n_m is the number of multiple vertices of $\mathcal{G}$). According to the example in Fig. 5(c), F has only one face, and E contains 10 edges, including the 6 edges of color 2 corresponding to vertices u', u'' and v', v'', as well as w', w'''. The first step consists in performing a re-numbering of the vertices, according to a DFUDS traversal of $\mathcal{T}_2^s$: this is well defined, since $\mathcal{T}_2^s$ is a tree, spanning all inner nodes.

We make use of an array-based representation of $\mathcal{G}^s$, using a table T having 5 entries for every vertex: according to the lemma above, this requires to store $5(n+4g)$ integer references. We store two references for each outgoing edge of color 0 and 1 (as in the first simple solution of Theorem 2). The last integer concerns the outgoing edge of color 2 incident to a node u: this is a reference to the edge (v,w) or to the edge (v,z) depending on the orientation of edges (u,w) and (u,z). More precisely, we store

– a reference to edge (v,z) if (v,z) is of color 1 oriented toward z, and (v,w) is of color 0 oriented toward w;

– a reference to edge (v,w) otherwise.

In order to distinguish edges in E from normal edges, we use a further service bit: this can be also used to distinguish between copies of multiple vertices and normal vertices.

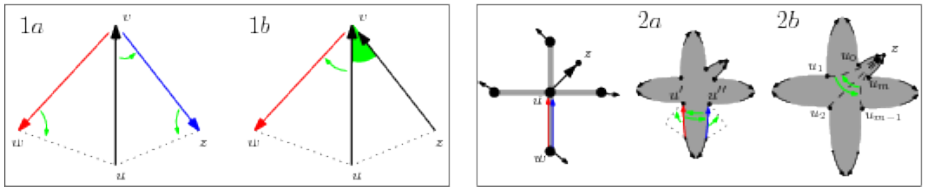

Figure 6: Case analysis involved in the proof of Theorem 8.

Proof of correctness We first provide a navigation strategy for edges which are not in E, we then explain how to handle the few edges arising from the splitting phase (there are at most $O(g)$ such edges).

Case 1: performing navigation for edges not in E. Our construction of map $\mathcal{G}^s$ ensures that the local Schnyder rule for edge colorations/orientations is consistent in every corner of a triangle face, so many arguments from previous section still apply. Again, we have to distinguish among different cases, depending on the coloration/orientation of edges. It is easy to check that many cases, when the edges involved are not of color 2, can be handled with the approach described in Theorem 2.

Let us consider an edge $e=(u,v)$ of color 2, oriented toward v (refer to Fig. 5). The most involved case occurs when edges (w,u) and (z,u) are oriented toward u: we have to consider several sub-cases depending on the orientation/coloration of edges (v,w) and (v,z).

Case 1a: the most difficult case is when edge (v, w) is of color 0 oriented toward w and edge (v, z) is of color 1 oriented toward z: the traversal is performed with the following steps.

Operator `RightFront`: by construction, we get the index of edge (v, z) simply computing $T[5u + 4]$.

Operator `RightBack`: it can be handled similarly, as we get (z, u) by simply computing `RightFront`(v, z).

Operator `LeftFront`: the trick is here is to firstly compute the index $e' =$ `RightFront`(v, z) (as before), then observing that edge (v, w) is an outgoing edge of color 0 incident to v: so the answer is just $e'' = e' - 1$ (since (v, z) and (v, w) have the same origin).

Operator `LeftBack`: it is now straightforward to obtain edge (w, u) from edge (v, w), which results in computing `LeftBack`(e'') (e'' is retrieved as in Theorem 2, as is of color 0).

Case 1b: we now consider the case where (v, w) is oriented toward w (of color 0), and (z, v) is oriented toward v (of color 2). We proceed in a similar manner as before. The left traversal is easy to perform since we have stored all needed references. For the right traversal, we exploit the DFUDS order on edges of color 2. More precisely given the index e of (u, v) we know that the index e' of (z, v) is simply given by $e' = 3(u + 1) + 2$, since (z, v) is the right sibling of (u, v) in the tree $\mathcal{T}_2^s$. We finally get (z, u) from (z, w), by computing $e'' = e - 1$ or $e'' = e' - 2$, depending on the orientation of (u, z) (observe that the orientation of (u, z) is described by service bits).

It is easy to check that remaining cases can be handled similarly, using the DFUDS order on vertices, when references between adjacency edges are not explicitly stored.

Case 2: navigation for edges in E. We have to take care of edges on the boundary of a split face $f_s \in F$, especially if we want to "jump" in $O(1)$ worst case time, from one side of f_s to another side: jumping from one side to another in $\mathcal{G}^s$ corresponds to perform a traversal of a special edge in $\mathcal{G}$.

Case 2a: let us consider an edge $e' = (u', w')$ which is paired to $e'' = (u'', w'')$. Since e' and e'' come from multiple edges, they cannot be of color 2. Observe that for e' there is one reference which still available: only one reference has been used for navigating in the triangle face incident to e' (a similar property holds for e'' as well). So, we can just use these 2 references to describe the fact that e' and e'' do coincide in $\mathcal{G}$.

Case 2b: let $e' = (u_0, z)$ and $e'' = (u_m, z)$ two edges of color 2 arising from the splitting of an outgoing edge incident to a multiple vertex u in $\mathcal{G}$, with multiplicity m. If $m > 2$, then there are at least two vertices u_1 and u_{m-1} which are not incident to an outgoing edge of color 2: observe that for each multiple vertex $u \in \mathcal{G}$ only one sector does contain an outgoing edge of color 2 (as shown in [13]). Then, we can use the entry $T[u_1 + 4]$ and $T[u_{m-1} + 4]$ (unused entries corresponding to vertex u_1 and u_{m-1}) in order to describe the neighboring relation between e' and e'': observe that edge (u_1, z) is retrievable from (u_0, z) in $O(1)$ because of the DFUDS labels of vertices in $\mathcal{T}_2^s$ (analogously, (u_{m-1}, z) and (u_m, z) are retrievable one from each other).

If $m < 2$, then u_0 and u_m can be obtained in $O(1)$ one from the other just using their DFUDS labels. Recall that, by construction, they are stored consecutively since they have a common parent vertex in $\mathcal{T}_2^s$.

3 Experimental results

Experiments setting. We have written Java array-based implementations of mesh data structures and performed tests [4] on various kinds of data (3D models and random triangulations generated with an uniform random sampler [23]). As in previous works [9, 19] we consider two geometric processing procedures: computing *vertex degrees* (involving edge navigation) and *vertex normals* (involving vertex access operators and geometric calculations). Vertex ordering of input points is the same for all tested data structures (vertices are accessed sequentially according ot their original order). To obtain a fair comparison, we do not use the power of the object-oriented java programming (no use of interfaces or Java *Generics* in our tests, in order to avoid small but not negligible overhead costs).

Implemented data structures Table 2 reports comparisons with existing data structures: *Half-edge* and *Winged-edge*. We have a compact representation using $6n$ references (*Compact* $6n$ *Basic*), encoded following the scheme described in Theorem 2: the orientation is not necessarily minimal, we use only 1 bit per reference, and colors/orientations of edges are retrieved with by modulo computations.

We have also a faster version (referred to as *Compact* $6n$ *Fast*), where division/modulo computations are replaced by bit shifts/masks (using 2 service bits per reference). Moreover, the use of minimal Schnyder woods further speeds up the data structure (reducing the number of cases to consider): as shown in Table 2(A)-(B) the obtained speed up is not negligible.

Comparison. As one could expect, non-compact mesh representations are faster (*Half-edge* being slightly faster than *Winged-edge*). Our data structures achieves good trade-offs between space usage and runtime performances. While being 3 or 4 times more compact for connectivity, our structures are slightly slower, loosing a factor between 1.16 and 1.90 (comparing *Compact* $6n$ *Fast* to *Winged-edge*) on tested data for topological navigation (see Table 2).

Concerning the implementation of vertex access operator, our tests show how in practice `Target` can be performed efficiently (`Target` requires $O(d)$ time in the worst case). The main reason is that very often `Target` requires only two bit inspections and a memory access to be performed: as observed in our tests, for most of edges at least one of the two edges (v, z) or (v, w) is outgoing from v (this occurs for more than 67% of edges, on all tested meshes). This also explains why our representations are even more competitive when considering geometric calculations: as shown in Table 2, our *Compact* $6n$ *Fast* is just slightly slower than *Winged-edge* (between 1.19 and 1.52 times slower).

[4] We tested on a Dell XT2, with Core 2 Duo 1.6GHz, 32bit Windows 7, Java 1.6

(A) Computing vertex degree

Mesh type	Halfedge (19n)	Winged edge (19n)	Thm 1 (Basic) 6n	Thm 1 (Fast) 6n	Thm 2 5n
Bunny	77	90	138	104	244
Iphigenia	73	91	145	108	255
Eros	58	66	133	98	233
Pierre's hand	40	48	119	91	223
Random (500K vert.)	68	84	163	126	278
Random (1M vert.)	67	81	157	120	277

(B) Computing vertex normals (with simple floating precision)

Mesh type	vertices	faces	Winged edge	Thm1 (Basic)	Thm 1 (Fast)	Thm 2
Bunny	26002	52K	607	797	724	1117
Iphigenia	49922	99K	549	820	726	1165
Eros	476596	953K	548	756	684	1061
Pierre's hand	773465	1.54M	477	724	646	980

Table 2: Comparison of mesh data structures. Runtime performances are expressed in nanoseconds per vertex. Vertex ordering of input points is the same for all tested data structures (vertices are accessed sequentially according to their original order).

4 Conclusion

Our approach is quite general and could be easily applied to other important classes of meshes, such as polygonal or quadrangular meshes homeomorphic to the sphere: just observe that nice (minimal) orientations (with bounded outgoing degree) also exist for planar quadrangulations and 3-connected graphs [16, 15].

References

[1] T. J. Alumbaugh and X. Jiao. Compact array-based mesh data structures. In *Proc. of the 14th Intern. Meshing Roundtable (IMR)*, 485–503, 2005.

[2] J. Barbay, L. Castelli-Aleardi, M. He, and J. I. Munro. Succinct representation of labeled graphs. to appear in *Algorithmica*, 2011. (preliminary version in *ISAAC 2007*)

[3] B. G. Baumgart. Winged-edge polyhedron representation. Technical report, Stanford, 1972.

[4] B. G. Baumgart. A polyhedron representation for computer vision. In *AFIPS National Computer Conference*, 589–596, 1975.

[5] D. Benoit, E. D. Demaine, J. I. Munro, R. Raman, V. Raman, and S. S. Rao. Representing trees of higher degree. *Algorithmica*, 43(4):275–292, 2005.

[6] D. Blanford, G. Blelloch, and I. Kash. Compact representations of separable graphs. In *SoDA*, 342–351, 2003.

[7] J.-D. Boissonnat, O. Devillers, S. Pion, M. Teillaud, and M. Yvinec. Triangulations in CGAL. *Comp. Geometry*, 22:5–19, 2002.

[8] E. Brehm. 3-orientations and Schnyder-three tree decompositions. Master's thesis, Freie Universitaet Berlin, 2000.

[9] S. Campagna, L. Kobbelt, and H. P. Seidel. Direct edges - a scalable representation for triangle meshes. *Journal of Graphics tools*, 3(4):1–12, 1999.

[10] L. Castelli-Aleardi, O. Devillers, and A. Mebarki. Catalog Based Representation of 2D triangulations. In *Internat. J. Comput. Geom. Appl.*, 21(4): 393-402, 2011.

[11] L. Castelli-Aleardi, O. Devillers, and G. Schaeffer. Succinct representation of triangulations with a boundary. In *WADS*, 134–145. Springer, 2005.

[12] L. Castelli-Aleardi, O. Devillers, and G. Schaeffer. Succinct representations of planar maps. *Theor. Comput. Sci.*, 408(2-3):174–187, 2008.

[13] L. Castelli-Aleardi, E. Fusy, and T. Lewiner. Schnyder woods for higher genus triangulated surfaces, with applications to encoding. *Discr. & Comp. Geom.*, 42(3):489–516, 2009.

[14] R. C.-N. Chuang, A. Garg, X. He, M.-Y. Kao, and H.-I. Lu. Compact encodings of planar graphs via canonical orderings and multiple parentheses. *ICALP*, 118–129, 1998.

[15] H. de Fraysseix and P. Ossona de Mendez. On topological aspects of orientations. *Disc. Math.*, 229:57–72, 2001.

[16] S. Felsner. Convex drawings of planar graphs and the order dimension of 3-polytopes. *Order*, 18:19–37, 2001.

[17] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and computation of Voronoi diagrams. *ACM Trans. Graph.*, 4(2):74–123, 1985.

[18] T. Gurung, D. Laney, P. Lindstrom, and J. Rossignac. *SQUAD*: Compact representation for triangle meshes. In *Comput. Graph. Forum*, 30(2):355-364, 2011.

[19] T. Gurung and J. Rossignac. *SOT*: compact representation for tetrahedral meshes. In *Proc. of the ACM Symp. on Solid and Physical Modeling*, 79–88, 2009.

[20] T. Gurung, M. Luffel, P. Lindstrom, and J. Rossignac. *LR*: compact connectivity representation for triangle meshes. In *ACM Trans. Graph.*, 30(4):67, 2011.

[21] M. Kallmann and D. Thalmann. Star-vertices: a compact representation for planar meshes with adjacency information. *Journal of Graphics Tools*, 6:7–18, 2002.

[22] J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM J. on Computing*, 31(3):762–776, 2001.

[23] D. Poulalhon and G. Schaeffer. Optimal coding and sampling of triangulations. *Algorithmica*, 46:505–527, 2006.

[24] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *Transactions on Visualization and Computer Graphics*, 5:47–61, 1999.

[25] W. Schnyder. Embedding planar graphs on the grid. In *SODA*, 138–148, 1990.

[26] J. Snoeyink and B. Speckmann. Tripod: a minimalist data structure for embedded triangulations. In *Workshop on Comput. Graph Theory and Combinatorics*, 1999.

[27] K. Yamanaka and S. Nakano. A compact encoding of plane triangulations with efficient query supports. *Inf. Process. Lett.*, 110:803–809, 2010.

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399