Explicit array-based compact data structures for triangulations

Luca Castelli Aleardi, Olivier Devillers

Explicit array-based compact data structures for triangulations *

Luca Castelli Aleardi[1] and Olivier Devillers[2]

[1] LIX (Ecole Polytechnique, France) amturing@lix.polytechnique.fr
[2] INRIA Sophia Antipolis – Méditerranée , France, olivier.devillers@inria.fr

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.
keywords: triangulations, compact representations, mesh data structures, graph encoding, 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* [24, 25] 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, 18] 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.

* This work is supported by ERC (agreement ERC StG 208471 - ExploreMap).

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, 27, 22, 1, 10, 20, 19, 21] 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: each memory word can store a $\lg n$ bits integer reference [3], and C bits are reserved as *service bits* (C is a small constant, commonly between 1 and 4). Moreover, 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 [20], which represents triangulations with $6n$ references. Adopting an interesting heuristic one may even obtain a more compact solution [19], 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* [15, 6, 23, 12, 11, 28] 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

[3] For a mesh with n elements, $\lg n := \lceil \log_2 n \rceil$ bits are required to distinguish all the elements. The length w of each memory word is assumed to be $\Omega(\lg n)$

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 [20, 19, 22] (as reported in Table 1.1).

Data structure	references	navigation	vertex access	dynamic
Edge-based data structures [18, 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 [22]	$7n$	$O(d)$	$O(1)$	*no*
TRIPOD [27] + reordering / Thm 1	$6n$	$O(1)$	$O(d)$	*no*
SOT data structure [20]	$6n$	$O(1)$	$O(d)$	*no*
SQUAD data structure [19]	$(4 + c)n$	$O(1)$	$O(d)$	*no*
(no vertex reordering) Thm 2	$5n$	$O(1)$	$O(d)$	*no*
(with vertex reordering) Thm 3	$4n$	$O(1)$	$O(d)$	*no*
(with vertex reordering) Cor 3	$6n$	$O(1)$	$O(1)$	*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.

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) [4]. As pointed out by Schnyder [26], 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).

[4] We will denote by the ordered pair (u, v) an edge oriented toward v, while $\{u, v\}$ will denote an edge regardless of its direction. In our drawings the root face coincide with the infinite exterior face.

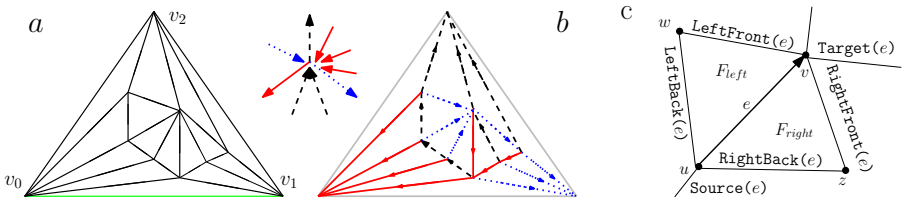

Fig. 1. A planar triangulation with 9 vertices (a), endowed with a (minimal) Schnyder wood (b) (the local Schnyder condition around inner vertices is also shown). Picture (c) illustrates the navigational operations supported by our representations.

Definition 1 ([26]). *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, 20, 19], 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 [27], we exploit the existence of 3-orientations (edges orientations where every inner vertex has outgoing degree 3) for planar triangulations [26]. 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 1. Our main contribution is to show how to get further improvements and generalizations [5], 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 to store only $5n$ references (Theorem 2).
- we further push the limits of the previous reordering approach: by arranging 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 3);
- in the full version of this paper, we also show how to reformulate a recent generalization of Schnyder woods [14], in order to deal with genus g triangulations: our representation requires at most $5(n + 4g)$ references;

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

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 [27]. 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. We firstly compute a Schnyder wood of the input triangulation $\mathcal{G}$ (in linear time, as shown in [26]). We define the tree $\overline{\mathcal{T}}_0$ by adding edges (v_1, v_0) and (v_2, v_0) to $\mathcal{T}_0$; we add the edge (v_2, v_1) to the tree $\mathcal{T}_1$, as depicted in Fig. 2. Each edge gets a color and an orientation: edges in $\overline{\mathcal{T}}_0$ are called red edges, those in $\mathcal{T}_1$ 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

[5] Detailed proofs of the results presented here can be found in [13].

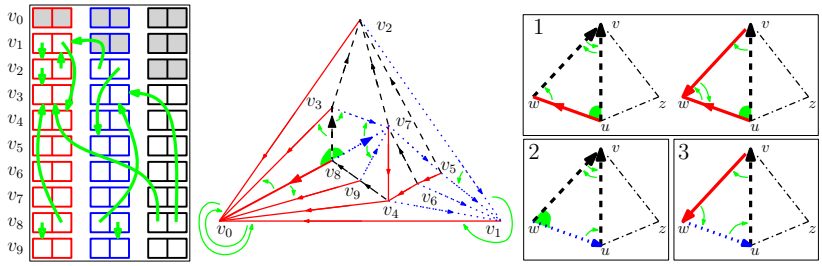

Fig. 2. Our first solution. For each vertex we store 6 references, corresponding to the its 3 outgoing edges. Table T is drawn as a bi-dimensional array of size $n \times 6$. The case analysis of Theorem 1 is illustrated on the right (where edge (u, v) has color 2).

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:

- $T[2j] =$ `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
- $S_b[j] = 1$ if edge j and `RightBack(`j`)` have the same source, 0.

Edges belonging to (v_0, v_1, v_2) are stored in a similar manner [6]: some edge indices are not used, since vertices on the outer face do not have outdegree 3 .

References encoding. As T is an array of integers $< 3n$ and S_a, S_b are arrays of bits, in practice these three arrays 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.

Theorem 1. *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 (a detailed proof can be found in [13]).*

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 1 ([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.*

[6] 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.

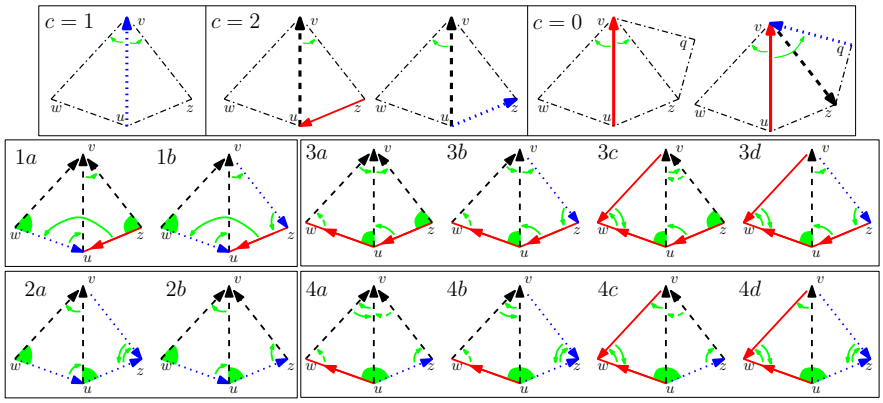

Fig. 3. More compact scheme. Neighboring relations between edges are represented by tiny oriented (green) arcs corresponding to stored references, and by filled (green) corners which implicitly describe adjacency relations between outgoing edges incident to a same vertex: because of local Schnyder rule,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

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 (let $e = (u, v)$ be of color c)

- for $c = 1$ (as in Theorem 1), 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 $\{q, v\}$ if $\{q, v\}$ 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 1.

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).

Theorem 2. *Let $\mathcal{G}$ be a triangulation with n vertices. There exists a representation requiring $5n$ references, allowing efficient navigation, as in Theorem 1.*

A detailed proof is in [13], the case analysis is illustrated by pictures in Fig. 3.

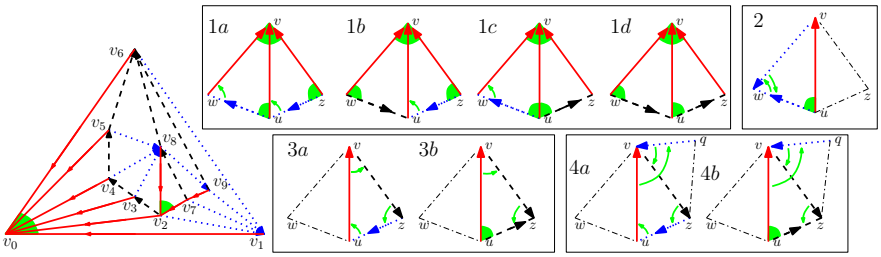

Fig. 4. A planar triangulation whose vertices are labeled according to a DFUDS traversal of tree $\overline{\mathcal{T}}_0$ (left). On the right are shown all cases involved in the proof of Theorem 3: we now store only one reference for edges of color 0 (red edges), since most adjacency relations are implicitly described by the DFUDS labels.

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 2 ([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 2. We can state the following result (the case analysis is partially illustrated by pictures in Fig. 4, see [13] for more details):

Theorem 3. *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.*

Corollary 1. *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.*

For dealing with the *higher genus case*, the key ingredient are g-Schnyder woods, a generalization of Schnyder woods for genus g triangulated surfaces [14]. We get a compact representation requiring about 5 references per vertex, applying to a g-Schnyder woods the approach relying on DFUDS order:

Theorem 4. *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 1.*

Our approach is quite general and could be 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 [17, 16].

3 Experimental results

We have written Java array-based implementations of mesh data structures and performed tests [7] on various kinds of data (3D models and random triangulations generated with an uniform random sampler [24]). As in previous works [9, 20] we consider two geometric processing procedures: computing *vertex degrees* (involving edge navigation) and *vertex normals* (involving vertex access operators and geometric calculations). 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 1. 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). 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. 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). 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).

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] 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)	Compact 6n Thm 1 (Basic)	Compact 6n Thm 1 (Fast)	Compact 5n Thm 2
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).

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 and O. Devillers. Explicit array-based compact data structures for triangulations. *INRIA research report 7736*, 2011. (`http://hal.archives-ouvertes.fr/inria-00623762/`)
14. 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.
15. 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.
16. H. de Fraysseix and P. Ossona de Mendez. On topological aspects of orientations. *Disc. Math.*, 229:57–72, 2001.
17. S. Felsner. Convex drawings of planar graphs and the order dimension of 3-polytopes. *Order*, 18:19–37, 2001.
18. 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.
19. T. Gurung, D. Laney, P. Lindstrom, and J. Rossignac. *SQUAD*: Compact representation for triangle meshes. In *Comput. Graph. Forum*, 30(2):355-364, 2011.
20. 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.
21. T. Gurung, M. Luffel, P. Lindstrom, and J. Rossignac. *LR*: compact connectivity representation for triangle meshes. In *ACM Trans. Graph.*, 30(4):67, 2011.
22. M. Kallmann and D. Thalmann. Star-vertices: a compact representation for planar meshes with adjacency information. *Journal of Graphics Tools*, 6:7–18, 2002.
23. J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM J. on Computing*, 31(3):762–776, 2001.
24. D. Poulalhon and G. Schaeffer. Optimal coding and sampling of triangulations. *Algorithmica*, 46:505–527, 2006.
25. J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *Transactions on Visualization and Computer Graphics*, 5:47–61, 1999.
26. W. Schnyder. Embedding planar graphs on the grid. In *SODA*, 138–148, 1990.
27. J. Snoeyink and B. Speckmann. Tripod: a minimalist data structure for embedded triangulations. In *Workshop on Comput. Graph Theory and Combinatorics*, 1999.
28. K. Yamanaka and S. Nakano. A compact encoding of plane triangulations with efficient query supports. *Inf. Process. Lett.*, 110:803–809, 2010.