SCARST: Schnyder Compact and Regularity Sensitive Triangulation Data Structure

Luca Castelli Aleardi, Olivier Devillers

SCARST: Schnyder Compact and Regularity Sensitive Triangulation Data Structure

Luca Castelli Aleardi
LIX, Ecole Polytechnique, Institut Polytechnique de Paris, Palaiseau, France

Olivier Devillers
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Abstract

We consider the design of fast and compact representations of the connectivity information of triangle meshes. Although traditional data structures (Half-Edge, Corner Table) are fast and user-friendly, they tend to be memory-expensive. On the other hand, compression schemes, while meeting information-theoretic lower bounds, do not support navigation within the mesh structure. Compact representations provide an advantageous balance for representing large meshes, enabling a judicious compromise between memory consumption and fast implementation of navigational operations. We propose new representations that are sensitive to the regularity of the graph while still having worst case guarantees. For all our data structures we have both an interesting storage cost, typically 2 or 3 r.p.v. (references per vertex) in the case of very regular triangulations, and provable upper bounds in the worst case scenario. One of our solutions has a worst case cost of 3.33 r.p.v., which is currently the best-known bound improving the previous 4 r.p.v. [Castelli et al. 2018]. Our representations have slightly slower running times (factors 1.5 to 4) than classical data structures. In our experiments we compare on various meshes runtime and memory performance of our representations with those of the most efficient existing solutions.

2012 ACM Subject Classification Mathematics of computing → Combinatoric problems; Theory of computation → Computational geometry

Keywords and phrases Meshes, compression, triangulations, compact representations

Digital Object Identifier 10.4230/LIPIcs.SoCG.2024.32

Related Version *Full Version*: `https://inria.hal.science/hal-04320292`

Supplementary Material *Software (Source Code and Data)*: `https://www.lix.polytechnique.fr/~amturing/software.html`

Funding *Luca Castelli Aleardi*: This work is supported by ANR grant "3DMaps" ANR-20-CE48-0018.

1 Introduction

Planar graphs and 3D surface meshes, which are among the most commonly used representations for discrete approximations of surfaces, have become ubiquitous in various fields such as geometric modeling, computer graphics, and computational geometry. Over the last two decades, the widespread adoption and growing complexity of such graphs have sparked numerous research efforts aimed at efficiently processing and representing these objects. The main focus is on reducing the memory cost for storing the *connectivity*[1] information of the mesh (requiring several hundreds of bits per vertex with conventional mesh representations), which is significantly more expensive to store when compared to geometric data (typically the point coordinates, requiring a few tens of bits per vertex).

Mesh representations. Indeed, various hierarchies of representations exist for storing and manipulating meshes, each offering a different balance between ease of use and memory consumption. The choice of which structure to use should rely on the specific needs of the

[1] The connectivity describes the incidence relations between the vertices, edges and faces of the mesh.

application, on the amount of computational and memory resources and on the type and combinatorial structure of the input graph. For instance, depending on the application one could seek to achieve very good compression rates on average in the case of regular real-world 3D meshes, or to provide worst-case upper bounds for dealing with irregular graphs. Simplicity and efficient navigation are essential features of mesh representations. A straightforward implementation allows for quicker processing of common navigational and access operators, improving overall performance. Mesh representations should ideally *preserve the original vertex ordering* of the input graph. This has a twofold advantage. Firstly, several algorithms rely on geometric and other additional data associated with vertices that cannot be arbitrarily rearranged (e.g. when utilizing geometric data structures for proximity queries). On the other hand, many real-world meshes exhibit excellent *vertex locality*, where neighboring vertices tend to have close indices on average. Preserving the vertex ordering in the case of good vertex locality significantly enhances the runtime efficiency by reducing cache misses. The storage cost of a mesh data structure can be measured in *bits per vertex* (b.p.v.) or in *references per vertex* (r.p.v.). A reference can be either a pointer or a vertex number of logarithmic size (in the number of vertices). In our results, for short we speak of x r.p.v., but for the detailed results we give precise bounds of the form $(x \log_2 n + y)$ b.p.v.

1.1 Related works: a hierarchy of structures

Explicit data structures. The simplest and most common way of representing the combinatorial structure of surface meshes is to store all incidence relations between vertices, edges and faces. For this reason conventional mesh representations [2, 3, 4, 14] tend to be highly redundant in order to allow fast local navigation and data access. They admit pointer-based and array-based implementations, which typically require between 13 and 19 r.p.v. to represent a triangulation.

Compression schemes. From the compression point of view there exist efficient schemes [10, 15, 23, 27] allowing us to encode triangle meshes with a few bits per vertex or matching asymptotically the information-theoretic lower bound of 3.245 b.p.v. [22]. Sometimes it is possible to exploit the *regularity* of the mesh and to get even better compression rates [26]. A commonly used measure of mesh regularity is the percentage of degree 6 vertices, denoted by d_6. However, it is worth noting that such encodings are primarily suitable for storage on disks or transmission over networks, as they do not efficiently support local navigation. To access the data, it is necessary to decompress the entire mesh first.

Succinct data structures. On the theoretical side, there exist data structures [8, 28] (called *succinct*) that attain the asymptotic optimal threshold of 3.245 b.p.v. and offer $O(1)$ time navigation within a mesh. While being extremely compact "for n large enough", they suffer from a significant limitation that renders them impractical for real-world use. Their storage bounds typically include a sublinear term $O(n\frac{\log\log n}{\log n})$, which becomes negligible only for very huge datasets.

Compact data structures. Compact data structures decrease the storage cost of explicit structures by reducing the number of stored incidence relations between mesh elements. This can be achieved by performing a convenient re-ordering and/or pairing of the faces and edges and exploiting some combinatorial decompositions. Their storage costs range between 2 and 7 r.p.v. (refer to Table 1): some representations [17] exhibit impressive compression rates for very regular 3D meshes (high values of d_6) but achieves bad results for

Table 1 Comparison between existing compact data structures for triangle meshes. Storage bounds are expressed in terms of *references per vertex* (r.p.v.) and hold for the case of genus 0 closed surfaces. The degree of the accessed vertex is denoted d°, while d_M denotes the maximum vertex degree of its neighbors. We report in columns 3-5 the average storage costs for both synthetic and real-world datasets obtained with our implementations of `SCARST` and `SQUAD` and LR representations. Column 3 reports the average costs obtained on the tested regular 3D meshes (for which $d_6 \geq 40\%$).

Compact data structure	storage cost (r.p.v.)					navigation time		preserves vertex ordering
	best case	average (tested meshes)			worst case	edge/face navigation	`target` operator	
		3D meshes	Delaunay	random				
2D catalogs [6]	-	-	-	-	7.67	$O(1)$	$O(1)$	yes
Star vertices [20]	7	7	7	7	7	$O(d^\circ)$	$O(1)$	yes
sorted `TRIPOD` [25]	6	6	6	6	6	$O(1)$	$O(d^\circ)$	yes
SOT [19]	6	6	6	6	6	$O(1)$	$O(d^\circ)$	yes
ESQ [7]	4	-	-	-	4.8	$O(1)$	$O(d^\circ)$	yes
SQUAD [16]	4	4.14	4.31	4.59	6	$O(1)$	$O(d^\circ)$	yes
LR [17]	2	2.27	3.04	6.15	> 12	$O(1)$	$O(1)$	no
Prop. 1=[5, Thm 2]	6	6	6	6	6	$O(1)$	$O(d^\circ)$	yes
[5, Thm 4]	5	5	5	5	5	$O(1)$	$O(d^\circ)$	yes
[5, Thm 5]	4	4	4	4	4	$O(1)$	$O(d^\circ)$	no
SCARST-OS, Thm 2	3	**3**	**3**	**3**	**3**	$O(d_M+d^\circ)$	$O(d^\circ)$	yes
SCARST-OT, Thm 3	3	**3.34**	**3.71**	**3.93**	**5**	$O(1)$	$O(d^\circ)$	yes
SCARST-RS, Cor 5	2	**2**	**2**	**2**	**2**	$O(d_M+d^\circ)$	$O(d^\circ)$	no
SCARST-RT, Thm 4	2	**2.26**	**2.54**	**2.65**	**3.67**	$O(1)$	$O(d^\circ)$	no
SCARST-WC, Thm 6	3	**3.03**	**3.08**	**3.04**	**3.33**	$O(1)$	$O(d^\circ)$	no

irregular graphs. Because of the simplicity of their implementation and the fact that, in most cases, they preserve the original vertex ordering, compact data structures provide very fast navigation. For instance the computation of the vertex degree requires typically some tens of nanoseconds per query, so they are only 1.5 to 4 times slower than explicit representations (these comparisons assume that the whole graph fits in memory).

Ultra-compact data structures. Combining vertex re-ordering approaches with efficient encodings of variable length references, it is possible to design *ultra compact data structures* whose compression rates are close to the information theory bounds. For instance, the `BELR` [17] (*Bit efficient LR)* and `Zipper` [18] data structures combine `LR` re-ordering approach with a better encoding of references, decreasing the storage bounds to some tens of bits per vertex. `PEMB` representation enriches the Turan encoding of planar graphs to support constant time navigation and gets even better compression rates: encoding a triangulation requires about 17 b.p.v.. Such storage [12, 13] performances are achieved at the cost of a much slower navigation. For instance, `BELR` is about five times slower than `LR`, and the `PEMB` [12] representation is between 15 and 200 times slower than a plain graph data structure depending on the type of query. This makes such structures advantageous only when the whole graph does not fit in main memory with alternative solutions. These structures do not preserve the input vertex ordering.

1.2 Contributions

We propose SCARST (Schnyder Compact And Regularity Sensitive Triangulation) a new compact data structure (built on the starting representation described in [5, Thm 2]) for representing the connectivity of triangle meshes that improve the previous existing worst case bounds. SCARST has several variants, ensuring (or not) a constant navigation time in the worst case, preserving (or not) vertex ordering, being time-sensitive or storage-sensitive to the mesh regularity. All variants have a worst case storage guarantee. Table 1 provide a range for the memory storage and an experimental value, averaging on many practical meshes. More precisely, we design the representations below (they are all provided with upper bounds):

- SCARST-OS (Ordered preserved, constant Storage) and SCARST-OT (Ordered preserved, constant navigation Time) are two variants that preserve the original vertex ordering. SCARST-OS has a cost of 3 r.p.v. and its navigation time is regularity sensitive, while SCARST-OT get constant time at the price of some extra storage that is regularity sensitive. The storage of SCARST-OT is almost always below 4 r.p.v. for both real-world meshes and synthetic graphs, while having an upper bound of 5 r.p.v. in the worst case. On most datasets our data structure achieves better results when compared to `SQUAD` [16], the best known compact representation preserving vertex ordering, see Fig. 15.
- SCARST-RS (Reordered, constant Storage) and SCARST-RT (Reordered, constant navigation Time) are two variants that makes use of vertex reordering to further reduce the storage bounds: experimentally SCARST-RT storage is very close to 2 r.p.v. for regular meshes, while we can guarantee a worst-case upper bound of 3.67 r.p.v.. Its compression rates are comparable to the ones of `LR` [17], and in some cases even better.
- SCARST-WC (best Worst-Case) is a variant (using vertex reordering) providing constant time navigation and a 3.33 r.p.v. worst-case upper bound, while being close to 3 r.p.v. on regular meshes. As far as we know this is the best existing upper bound on the storage requirements for such compact data structures. Previous solutions either have larger provable upper bounds in the worst case [5, 6, 16, 19] or achieve poor compression rates for several classes of irregular graphs [17] (see Fig. 15).

We performed experimental evaluations on a broad spectrum of 2D and 3D triangle meshes, comparing our storage and runtime performances to previous solutions. Our results confirm the practical interest of our representations: while being much more compact than explicit representations, our data structures are still fast, being only between 2× and 4× slower in practice. We also establish new experimental evaluations on the storage bounds of existing compact representations [16, 17]: our results confirm that previous existing solutions [17] achieve very good compression rates for regular meshes (with many low degree vertices), but may require large storage in the case of irregular and random graphs.

2 Preliminaries

2.1 Navigation interface

We adopt the navigation interface of the *Winged-edge* data structure [2] that supports efficient local navigation in a surface mesh (see Fig. 1). Given an edge $\vec{e} = (v, w)$ (arbitrarily oriented) the incidences between edges and vertices can be performed through the six operators below: $\texttt{Source}(\vec{e})$ and $\texttt{Target}(\vec{e})$ are the two incident vertices, $\texttt{LFront}(\vec{e})$ and $\texttt{LBack}(\vec{e})$ are the two other edges of the triangle to the left of $\vec{e}$, and $\texttt{RFront}(\vec{e})$ and $\texttt{RBack}(\vec{e})$ are edges of triangle to its right (see Fig. 1). It is worth noting that edges are arbitrarily oriented and each edge is represented only once (with one of the two orientations), conversely to what occurs in the case of the Half-edge structure which stores pairs of half-edges (with opposite orientations).

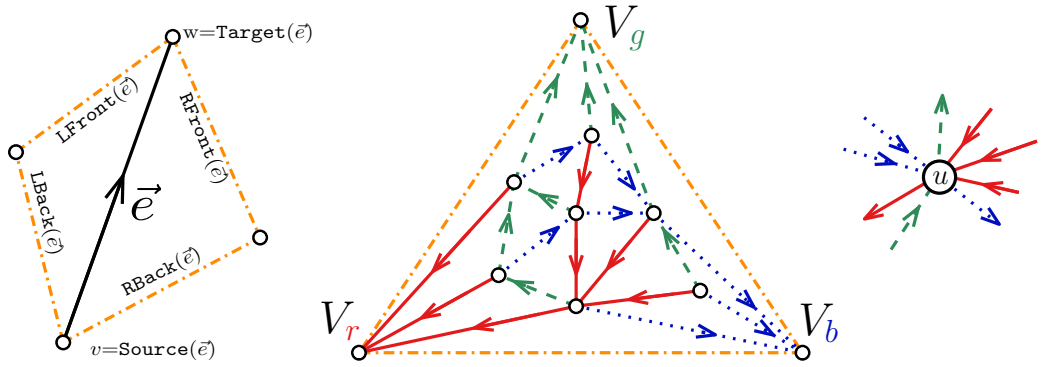


Figure 1 Left:Navigation interface of *Winged-edge*. Center: A planar triangulation endowed with a Schnyder wood. Right: the Schnyder rule for inner vertices. Plain lines correspond to red edges in T_r, while dotted blue edges are in T_b and green dashed edges are in T_g.

2.2 Schnyder woods for planar triangulations

Given a planar triangulation (or a genus 0 triangle mesh) with a distinguished *root* face (V_r, V_b, V_g), one can compute a *Schnyder wood* [24], which is a partition of the internal edges into three sets which are trees T_r, T_b and T_g spanning all internal vertices, rooted at the vertices V_r, V_b and V_g respectively (see Fig. 1-Center).

Internal edges are oriented and colored (with 3 colors) in such a way that each inner vertex has exactly three outgoing incident edges (one in each color). We will denote by T_c the tree consisting of edges having color c (where $c \in \{r, b, g\}$), with the convention: $r{+}1 = b$; $b{+}1 = g$; $g{+}1 = r$. The local Schnyder rules state that turning in counterclockwise (ccw) direction around an internal vertex u we must encounter: the outgoing red edge, a group of incoming green edges if any, the outgoing blue edge, a group of incoming red edges if any, the outgoing green edge, a group of incoming blue edges if any (see Fig. 1-Right).

A Schnyder wood can be computed in linear time by a simple shelling process [21]. A given triangulation may admit several distinct Schnyder woods, but there is a unique *minimal* Schnyder wood which is guaranteed to have no *ccw triangles* (triangles whose three edges are oriented in counter-counterclockwise direction).

2.3 Castelli&Devillers' starting data structure

For sake of completeness we review the approach adopted in our previous work [5, Thm. 2] and describe a simple and fast compact data structure that represents planar[2] triangulations with 6 r.p.v. Even if we start from the same point as [5], we proceed in a different manner to reduce redundancies allowing better performances and sensitivity to the regularity of the triangulation. The complexity proofs need new arguments.

Overview of Thm. 2 in [5]. We first compute an arbitrary Schnyder wood and complete its coloring. We assign colors and orientations to the edges on the root face: edges incident to V_r are red (oriented toward V_r) and edge (V_g, V_b) is blue and oriented toward V_b. Then each edge is uniquely identified by its origin and its color (see Fig. 2-Left).

[2] As done in [5, 12, 13], we deal in this work with planar graphs (genus 0 meshes): in Section 5 we briefly discuss how to address the case of higher genus graphs.

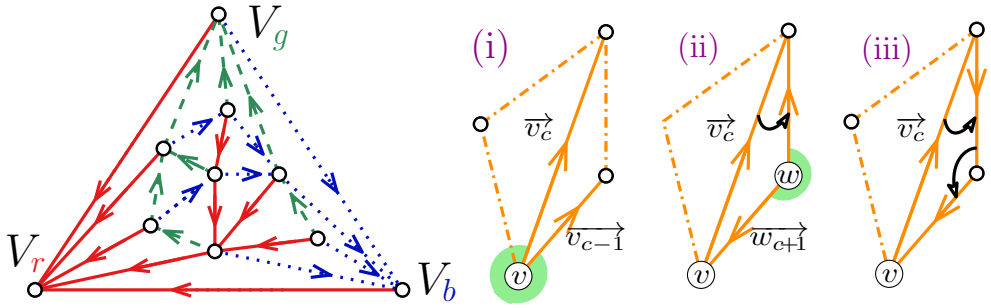


Figure 2 Left: Schnyder wood, coloring external face. Right: Retrieving right back edge.

In the sequel an edge will be denoted $\overrightarrow{v_c}$ for the edge of source v and color $c \in \{r, b, g\}$. Then the idea is for each edge $\overrightarrow{v_c}$ to store the orientation of the four neighboring edges and the source of the two front edges.

For each vertex v and each color c we store three boolean informations: $\texttt{Leaf}[v, c]$ which is true if v is a leaf in T_c, $\texttt{LOrient}[v, c]$ which is true if $\texttt{LFront}(\overrightarrow{v_c})$ is oriented towards $\texttt{Target}(\overrightarrow{v_c})$, and $\texttt{ROrient}[v, c]$ similar on the right. We also store two vertex numbers $\texttt{Source}(\texttt{LFront}(\overrightarrow{v_c}))$ and $\texttt{Source}(\texttt{RFront}(\overrightarrow{v_c}))$. When the orientations are known, colors can be determined using the Schnyder rule and back neighbors can be deduced (see Fig. 2-Right). The right back neighbor $\texttt{RBack}(\overrightarrow{v_c})$ of $\overrightarrow{v_c}$ can be retrieved as: (i) $\overrightarrow{v_{c-1}}$, if $\texttt{Leaf}[v, c+1]$ is true; (ii) $\overrightarrow{w_{c+1}}$ with $w = \texttt{Source}(\texttt{RFront}(\overrightarrow{v_c}))$, if $\texttt{Leaf}[v, c+1]$ is false and $\texttt{ROrient}[v, c]$ is true; (iii) $\texttt{RFront}(\texttt{RFront}(\overrightarrow{v_c}))$, otherwise. In a similar way one can retrieve $\texttt{LBack}(\overrightarrow{v_c})$.

Storage and runtime complexity. This structure can be implemented using 6 tables of size n containing vertex numbers: $\mathbf{R}_\mathrm{L}$ (with $\mathbf{R}_\mathrm{L}[v] = \texttt{Source}(\texttt{LFront}(\overrightarrow{v_r}))$) , $\mathbf{R}_\mathrm{R}$, $\mathbf{B}_\mathrm{L}$, $\mathbf{B}_\mathrm{R}$, $\mathbf{G}_\mathrm{L}$, and $\mathbf{G}_\mathrm{R}$, and 9 tables of size n containing booleans $\texttt{Leaf}[., r]$, $\texttt{LOrient}[., r]$, $\texttt{ROrient}[., r]$, $\texttt{Leaf}[., b]$, $\texttt{LOrient}[., b]$, $\texttt{ROrient}[., b]$, $\texttt{Leaf}[., g]$, $\texttt{LOrient}[., g]$, and $\texttt{ROrient}[., g]$. The implementation of the operators $\texttt{Source}(\overrightarrow{v_c})$, $\texttt{LFront}(\overrightarrow{v_c})$, $\texttt{RFront}(\overrightarrow{v_c})$, $\texttt{LBack}(\overrightarrow{v_c})$, and $\texttt{RBack}(\overrightarrow{v_c})$ requires a constant number of accesses to these different tables, while answering $\texttt{Target}(\overrightarrow{v_c})$ needs a number of access bounded by its degree. We get:

▶ **Proposition 1** ([5]-Thm 2). *The above structure represents a triangulation of n vertices with $O(1)$ time access to neighboring edges using $9n + 6n \log_2 n$ bits.*

3 SCARST data structures

We name our structure SCARST for *Schnyder Compact And Regularity Sensitive Triangulation*. In all cases we use minimal Schnyder woods (with no ccw triangles). We propose several trade-offs between runtime and storage cost. The runtime and/or the storage cost are sensitive to the degree of vertices, i.e. the mesh regularity.

Overview of our approach. We exploit the fact that in a planar (or bounded genus) graph the number of high degree vertices is small: so we distinguish between low and high degree vertices. Consider a vertex w having low indegree in color c (at most three in our implementation): then for each edge $\overrightarrow{v_c}$ such that $w = \texttt{Target}(\overrightarrow{v_c})$ we store only one reference to one of its left (or right) neighboring edges at w. Retrieving the remaining neighbors (whose reference is not stored) can be performed by turning around w in cw (or ccw) direction, which takes a constant number of steps. For a vertex w having high indegree in color c we

store a few *extra* skipping references (depicted as black arrows in our pictures), which allow us to jump and to efficiently traverse the siblings of an edge of color c incoming at w. In our implementation we store the fourth edge turning around w in cw (or ccw) direction. In practice the number of such extra references is small and from a novel counting argument involving Schnyder woods we derive worst case upper bounds.

Extra references and address indirection. We distinguish between an *essential* reference that exist for all vertices and an *extra* reference that exist only for some (high degree) vertices. Retrieving extra references can be done through address indirection. We use the memory word allocated for an essential reference as a pointer to an auxiliary table: in this table we have two corresponding entries, one for the extra reference and the second one for the lost essential reference. So, for storing each extra reference we consume two integer numbers.

3.1 SCARST-OS, a regularity sensitive navigation time data structure

Data Structure. This first structure is pretty similar to the one of Section 2.3: it uses 3 tables of size n containing vertex numbers: $\mathbf{R}_{\mathrm{L}}$ $\mathbf{B}_{\mathrm{R}}$, and $\mathbf{G}_{\mathrm{R}}$, and the 9 tables of n booleans $\texttt{Leaf}[.,c]$, $\texttt{LOrient}[.,c]$, and $\texttt{ROrient}[.,c]$ for $c \in \{r, b, g\}$. storing the same information as in Section 2.3. It remains to explain how to retrieve the missing information.

Retrieving $\texttt{LFront}(\overrightarrow{v_g})$. (see Fig. 3). The right front neighbor of $\overrightarrow{v_g}$ is either blue or green, and is explicitly stored. Thus we can iterate the right front operator until we find a blue edge $\overrightarrow{w_b}$. Notice that w is the target of $\overrightarrow{v_g}$. Then we can access to $\texttt{LFront}(\overrightarrow{w_r})$ which must be red, since there is no ccw triangle. Let $\overrightarrow{u_r} = \texttt{LFront}(\overrightarrow{w_r})$. Either $u = v$ and the searched edge $\texttt{LFront}(\overrightarrow{v_g})$ is $\overrightarrow{w_r}$. Or $u \neq v$, we can iterate the right front operator starting from $\overrightarrow{u_g}$ until we find $\overrightarrow{v_g}$ and the searched edge $\texttt{LFront}(\overrightarrow{v_g})$ is the previous edge in the iteration.

Retrieving $\texttt{LFront}(\overrightarrow{v_b})$. (see Fig. 3). The right front operator for blue edges is quite similar to the one for green edges. In the case of green edges, we needed to access the left front of red edges which is explicitly stored in Table $\mathbf{R}_{\mathrm{L}}$. In the case of blue edges, we need the left front of green edges which is not explicitly stored but retrieved at the previous paragraph.

Retrieving $\texttt{RFront}(\overrightarrow{v_r})$. (see Fig. 4-Left). For red edges, the situation is symmetrical. The left front neighbor is accessible and the right front one must be retrieved. We access it by turning clockwise around $w = \texttt{Target}(\overrightarrow{v_r})$ until we reach $\overrightarrow{u_r}$ whose right front neighbor is a blue edge $\overrightarrow{w_r}$. Now w is reach and we can continue to turn clockwise around w from $\overrightarrow{w_g}$ until we find again $\overrightarrow{v_r}$, the last seen edge is $\texttt{RFront}(\overrightarrow{v_r})$.

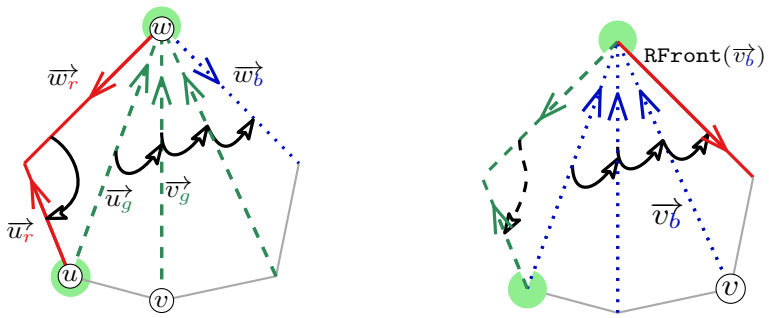


Figure 3 SCARST-OS. Left: green navigation. Right: blue navigation.

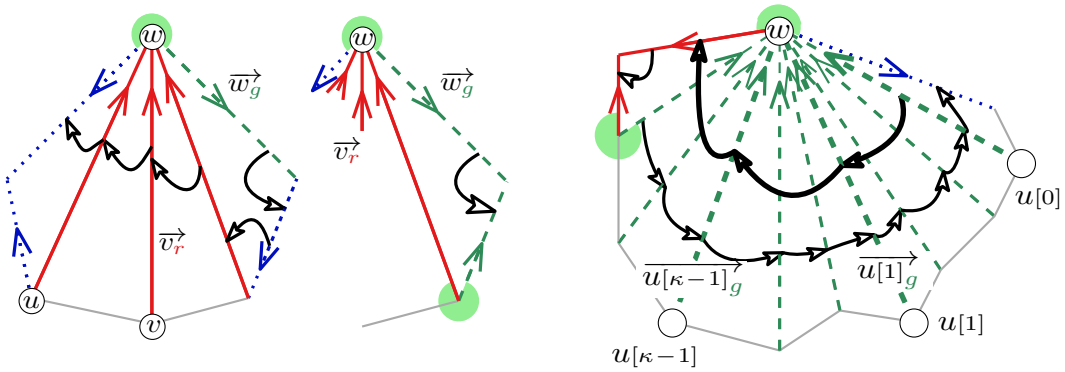


Figure 4 Left: SCARST-OS Red navigation to the right. Right: SCARST-OT Green navigation (high degree).

Runtime complexity. This structure uses 3 tables of size n containing vertex numbers and 9 tables of size n containing booleans. All operators on $\overrightarrow{v_c}$ are implemented using a number of access to these tables bounded by $\mathrm{d}^\circ_c(w)$ (and by $\mathrm{d}^\circ_g(\texttt{Target}(\overrightarrow{w_g}))$ when $c = b$).

▶ **Theorem 2.** SCARST-OS *represents a triangulation of n vertices with access time to neighboring edges linear in the degree with storage cost $9n + 3n\log_2 n$ bits.*

3.2 SCARST-OT, a regularity sensitive storage cost data structure

Computing $\texttt{LFront}(\overrightarrow{v_g})$ whenever $w = \texttt{Target}(\overrightarrow{v_g})$ has a high indegree $\mathrm{d}^\circ_g(w)$ in the green tree takes more than $O(1)$ time. To reach constant navigation time we store an extra reference to a sibling in clockwise direction for some edges. More precisely, we store an extra reference only when $\mathrm{d}^\circ_g(w) = \delta \geq 4$ and we choose $\kappa = \lfloor \frac{\delta}{3} \rfloor$ children of w in the green tree, $u_{[0]}, u_{[1]}, \ldots, u_{[\kappa-1]}$, so that three conditions are verified: (i) $\overrightarrow{u_{[0]}}_g$ is the right most child of w, (ii) there are no more than three green edges between $\overrightarrow{u_{[i]}}_g$ and $\overrightarrow{u_{[i+1]}}_g$, and (iii) there are no more than four green edges between $\overrightarrow{u_{[\kappa-1]}}_g$ and $\overrightarrow{w_r}$ (see Fig. 4-Right).

Data Structure. As before, we have 3 tables of size n containing numbers (vertex numbers or indirection indices): $\mathbf{R}_\mathrm{L}$ $\mathbf{B}_\mathrm{R}$, and $\mathbf{G}_\mathrm{R}$; and 9 tables of size n containing booleans $\texttt{Leaf}[., c]$, $\texttt{LOrient}[., c]$, and $\texttt{ROrient}[., c]$ for $c \in \{r, b, g\}$. We add one table of size n of boolean $\texttt{Extra}[., c]$, one table of size η of booleans $\texttt{Extra0}(.)$, and two tables of size η of vertex numbers $\mathbf{F}[.]$ and $\mathbf{E}[.]$, where η is the number of extra references needed. The fact that an edge, e.g. $\overrightarrow{u_{[i]}}_g$, needs an extra reference, is stored in $\texttt{Extra}[u_{[i]}, g]$. In this case $\mathbf{G}_\mathrm{R}[u_{[i]}]$, instead of storing $\texttt{Source}(\texttt{RFront}(\overrightarrow{u_{[i]}}_g))$, contains a number α to represent an indirection. $\mathbf{F}[\alpha]$ contains $\texttt{Source}(\texttt{RFront}(\overrightarrow{u_{[i]}}_g))$. $\mathbf{E}[\alpha]$ contains the source of the extra neighbor ($\overrightarrow{u_{[i+1]}}_g$ or $\overrightarrow{w_r}$) and $\texttt{Extra0}(\alpha)$ contains a boolean to determine its color.

Retrieving missing references. To compute $\texttt{LFront}(\overrightarrow{v_g})$ we turn ccw around w, following the stored essential reference: at each step we check whether the visited edge has an extra reference going cw, and if so we use it to jump to the left and then continue ccw up to $\overrightarrow{v_g}$. The process is exactly the same for blue edges and symmetrical for red edges.

Runtime complexity. The storage uses 3 tables of size n and 2 of size η containing numbers and 12 tables of size n and 1 of size η containing booleans. Navigation operators are implemented with a constant number of access to these different tables, except $\texttt{Target}(\overrightarrow{v_c})$ which still has a cost bounded by its degree.

▶ **Theorem 3.** SCARST-OT *represents a triangulation of n vertices with access to neighboring edges in $O(1)$ time and storage cost $12n + \eta + 3n\log_2 n + 2\eta\log_2 n$ bits where η is sensitive to the number of high degree vertices in the triangulation. In the worst case storage remains below $13n + 5n\log_2 n$ bits.*

Proof. It remains to bound η. Observe that $\forall c, \sum_{v\in T_c} \left\lfloor \frac{\mathrm{d}^\circ_T(v)_c}{3} \right\rfloor < \frac{n}{3}$ (see full version), so that the number of extra references per color is less than $\frac{n}{3}$ and thus $\eta < 3\frac{n}{3} = n$. This ensures that an indirection to $\mathbf{E}[.]$ fits in $\log_2 n$ bits and thus in Tables $\mathbf{R}_\mathrm{L}$, $\mathbf{B}_\mathrm{R}$, and $\mathbf{G}_\mathrm{R}$. ◀

3.3 SCARST-RS, SCARST-RT, reordering the vertices

Making use of vertex re-ordering one can save at least one reference per vertex, which allows us to get below the barrier of 3 references per vertex for many classes of triangle meshes. As in Castelli et al. [5], we re-order the vertices in red BFS order so that neighboring red edges can be obtained with basic arithmetic operations. More precisely, we use the minimal Schnyder wood without ccw triangles and we re-order the vertices such that turning ccw around a vertex enumerates its children in the red tree with consecutive indices (see Fig. 5).

Data Structure. With respect to SCARST-OT, we drop Table $\mathbf{R}_\mathrm{L}$. We also modify the meaning of Table $\mathbf{B}_\mathrm{R}$ in some cases: for an edge $\overrightarrow{v_b}$ whose right back and front edges are red, instead of storing $\texttt{Source}(\texttt{RFront}(\overrightarrow{v_b}))$ we store $\texttt{Target}(\overrightarrow{v_r})$. We create a new operator $\texttt{SideTarget}(\overrightarrow{v_b})$ (see Fig. 5).

Retrieving $\texttt{RFront}(\overrightarrow{v_b})$. When right back and front edges are red, $\texttt{RFront}(\overrightarrow{v_b})$ can be obtained as $\texttt{LFront}(\overrightarrow{v_r})$, otherwise $\texttt{Source}(\texttt{RFront}(\overrightarrow{v_b}))$ is stored in Table $\mathbf{B}_\mathrm{R}$.

Retrieving $\texttt{LFront}(\overrightarrow{v_r})$ and $\texttt{RFront}(\overrightarrow{v_r})$. If $\texttt{LOrient}[v, c]$ is true then the searched edge, denoted $\overrightarrow{u_r}$, is red and we just compute $u = v - 1$. Otherwise the searched edge is a blue edge $\overrightarrow{w_b}$: since there is no ccw triangle $\texttt{LBack}(\overrightarrow{v_r})$ is the blue edge $\overrightarrow{v_b}$ and $\texttt{LFront}(\overrightarrow{v_r})$ is obtained as $\texttt{RFront}(\overrightarrow{v_b})$ (see Fig. 6). If $\texttt{RFront}(\overrightarrow{v_r})$ is a red edge $\overrightarrow{u_r}$ then u is obtained easily as $u = v + 1$. Otherwise $\texttt{RFront}(\overrightarrow{v_r})$ is $\overrightarrow{w_g}$ and in order to retrieve it we have two cases: (i) if $v - i$ for $i \in [0, k)$ is a green leaf then w is obtained as $\texttt{SideTarget}(\overrightarrow{(v-i)_b})$. This is true $\forall k < \mathrm{d}^\circ_r(w)$, but to have a constant access time we use $k = 3$. (ii) Otherwise, we still have to use an extra reference to store w.

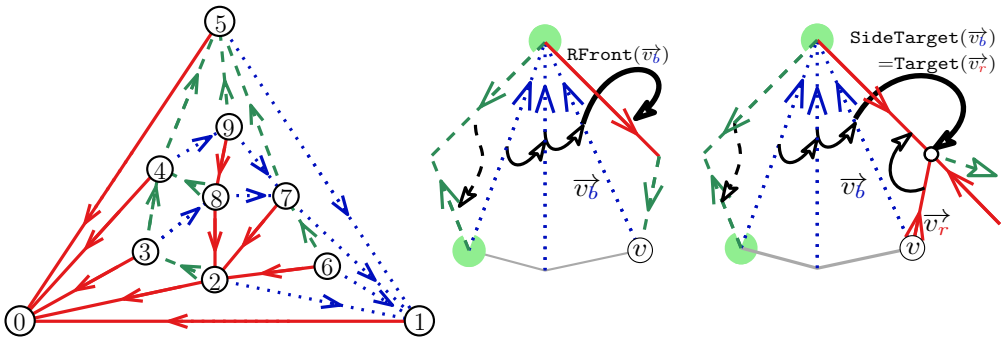


Figure 5 Left: Minimal Schnyder wood with red BFS order. Right: SCARST-RT: blue navigation.

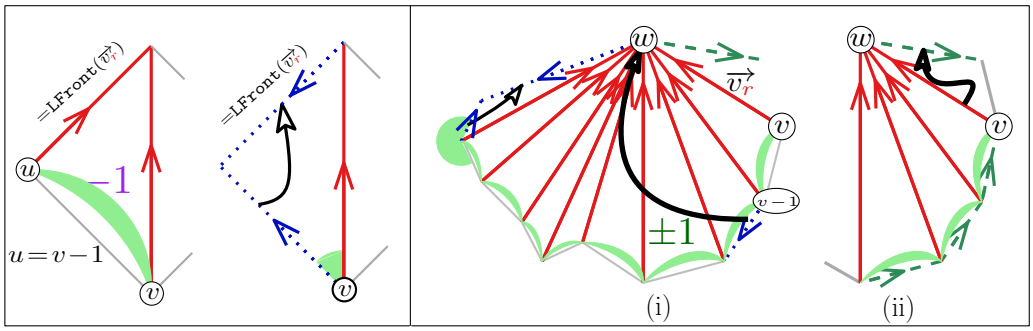


Figure 6 Red navigation for SCARST-RT. Left: retrieving left front. Right: retrieving right front.

Complexity. With respect to the previous solution, we use exactly the same tables except $\mathbf{R}_{\mathrm{L}}$ saving one reference per vertex. Previously we need some extra references for each vertex with high ingoing red degree, while now we only need one in some cases reducing the total number of extra references.

▶ **Theorem 4.** SCARST-RT *represents a triangulation T of n vertices with constant access time to neighboring edges with storage cost $12n+\eta+2n\log_2 n+2\eta\log_2 n$ bits where $\eta\in[0,n)$ is sensitive to the number of high degree vertices in the triangulation. In the worst case storage remains below $12.83n+3.67n\log_2 n$ bits.*

Proof. We have to bound η. We need an extra red reference when a vertex has at least 4 children in the red tree and three of them are internal nodes in the green tree. Such a situation should not arise too often: to get a conservative bound, we can permute the color so that the green tree is the one with the largest number of leaves. In such a case the number of extra red references cannot be larger than $\frac{n}{6}$ (see full version). The number of extra references for the blue and green tree is $\frac{n}{3}$ each, as in the previous section and $\eta\leq\frac{5}{6}n$. ◀

SCARST-OS can be viewed as a simplified version of SCARST-OT. In the same way, we can simplify SCARST-RT, forgetting the extra references at the price of an access time to some neighboring edges of $\overrightarrow{v_c}$ proportional to the degree of relevant vertices around $\overrightarrow{v_c}$.

▶ **Corollary 5.** SCARST-RS *represents a triangulation T of n vertices with access time to neighboring edges proportional to the degree and with storage cost $9n+2n\log_2 n$ bits.*

3.4 SCARST-WC, improving worst case storage cost

Our last data structure achieves the best worst case bounds known so far. As in the previous section, we use a minimal Schnyder wood numbered in redBFS order.

Data structure. (see Fig. 7). As in previous section, an extra reference to $\texttt{RFront}(\overrightarrow{v_r})$ is stored only if v is the rightmost child of some vertex w with $\mathrm{d}_r^\circ(w)\geq 4$ and v, $v-1$, and $v-2$ are internal nodes of T_g. For each blue edge we store exactly two references: one at right to $\texttt{Target}(\texttt{RFront}(\overrightarrow{v_b}))$ if both $\texttt{RFront}(\overrightarrow{v_b})$ and $\texttt{RBack}(\overrightarrow{v_b})$ are red or to $\texttt{RFront}(\overrightarrow{v_b})$ otherwise. And we store one reference at left: to $\texttt{LFront}(\overrightarrow{v_b})$ if it is blue (normal case) or to $\texttt{RFront}(\texttt{LFront}(\overrightarrow{v_b}))=\overrightarrow{w_\beta}$ if it is green (special case), where the color β of $\overrightarrow{w_\beta}$ can be either green or blue (we add a new table of booleans $\texttt{LROrient}(.)$ to store β). For each green edge we store a single reference, to $\texttt{RFront}(\overrightarrow{v_g})$ if $\texttt{LBack}(\overrightarrow{v_g})$ is red or to $\texttt{LFront}(\overrightarrow{v_g})$ otherwise.

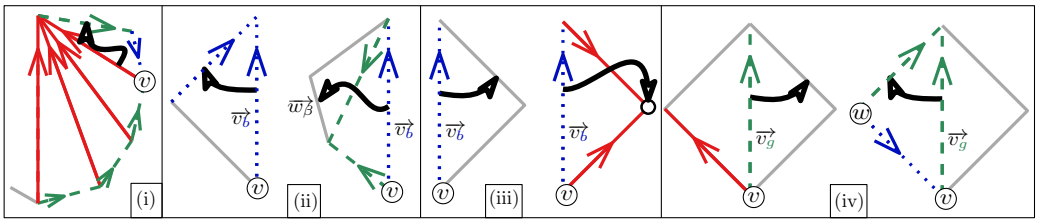


Figure 7 SCARST-WC. Definition of references: (i) red extra (ii) left blue (iii) right blue (iv) green.

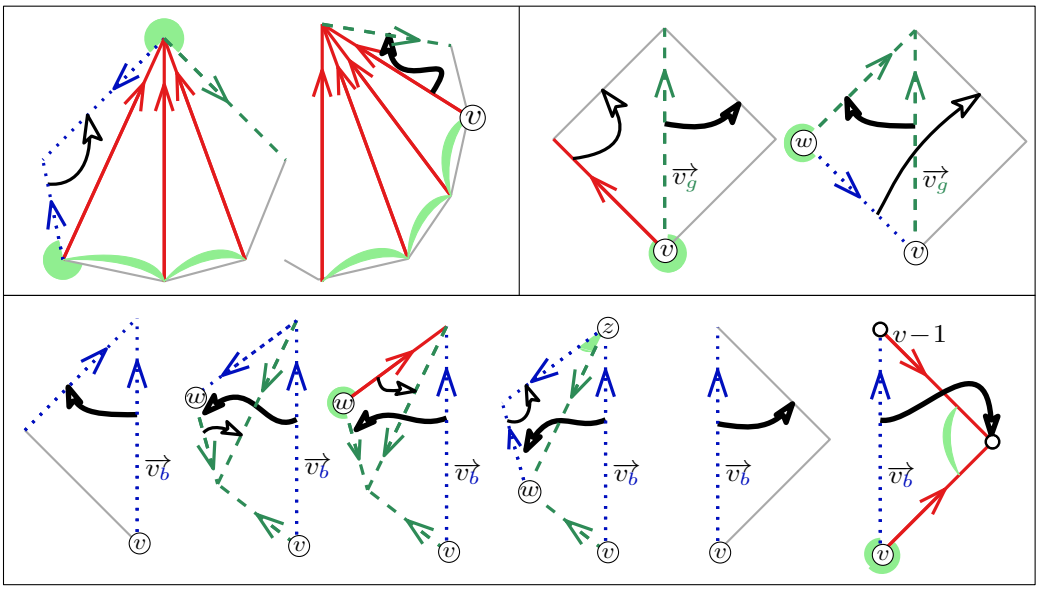


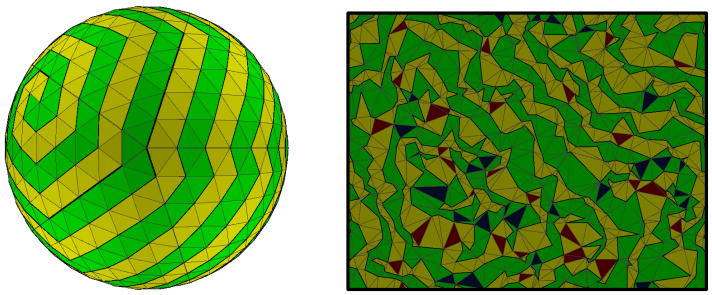
Figure 8 SCARST-WC. Top: red and green navigation. Bottom: blue navigation.

Retrieving neighbors. (see Fig. 8). Red navigation is identical to SCARST-RT. For green navigation, we retrieve $\texttt{LFront}(\overrightarrow{v_g})$ as $\texttt{RFront}(\overrightarrow{v_r})$ when $\texttt{RFront}(\overrightarrow{v_g})$ is stored. Otherwise $\texttt{LFront}(\overrightarrow{v_g})$ is stored and $\texttt{RFront}(\overrightarrow{v_g})$ is the special case of left neighbor of $\overrightarrow{w_b}$ where $w = \texttt{Source}(\texttt{RFront}(\overrightarrow{v_g}))$.

When $\texttt{LFront}(\overrightarrow{v_b})$ is not stored in Table $\mathbf{B}_\mathrm{R}$, then $\overrightarrow{w_\beta} = \texttt{RFront}(\texttt{LFront}(\overrightarrow{v_b}))$ is stored and $\texttt{LFront}(\overrightarrow{v_b})$ can be retrieved, either as $\texttt{LFront}(\overrightarrow{w_g})$ if $\beta = g$ and $\texttt{LBack}(\overrightarrow{w_g})$ is blue, $\texttt{RFront}(\overrightarrow{w_r})$ if $\beta = g$ and $\texttt{LBack}(\overrightarrow{w_g})$ is red, or as $\overrightarrow{z_g}$, where $z = \texttt{Source}(\texttt{RFront}(\overrightarrow{w_b}))$ otherwise. When $\texttt{RFront}(\overrightarrow{v_b})$ is not stored in Table $\mathbf{B}_\mathrm{L}$ $\texttt{RFront}(\overrightarrow{v_b})$ is $\overrightarrow{(v-1)_r}$.

Complexity. In this structure we use 3 tables of size n containing vertex numbers: $\mathbf{B}_\mathrm{L}$, $\mathbf{B}_\mathrm{R}$, and $\mathbf{G}$; the usual 9 tables of size n containing booleans $\texttt{Leaf}[.,c]$, $\texttt{LOrient}[.,c]$, $\texttt{ROrient}[.,c]$ $c \in \{r, b, g\}$; 2 other tables of size n containing booleans $\texttt{Extra}[v, r]$ storing the existence of an extra reference for $\overrightarrow{v_r}$ and $\texttt{LROrient}(v)$ storing the orientation of $\texttt{RFront}(\texttt{LFront}(\overrightarrow{v_b}))$ when needed; 2 tables of size η of vertex numbers $\mathbf{F}[.]$ and $\mathbf{E}[.]$ for extra red references; and finally one table of size η of booleans $\texttt{Extra0}(.)$. Navigation operators require a constant number of accesses to these different tables, except $\texttt{Target}(\overrightarrow{v_c})$ which remains degree dependant.

▶ **Theorem 6.** *The above structure represents a triangulation T of n vertices with constant access time to neighboring edges with storage cost $11n + \eta + 3n\log_2 n + 2\eta\log_2 n$ bits where $\eta \in [0, n)$ is sensitive to the number of high degree vertices in the triangulation. In the worst case storage remains below $11.17n + 3.33n\log_2 n$ bits.*



sphere Delaunay

Figure 9 Laced-ring computed by our implementation of the *Ring-Expander* procedure of LR.

Proof. As in proof of Theorem 4, $\eta \leq \frac{n}{6}$ using a suitable choice of colors. ◀

4 Experimental results

We have written Java implementations of our data structures[3] and compared their performances to previous works on a wide collection of datasets. In order to obtain fair runtime comparisons, we have written array-based implementations of several mesh representations: *Half-edge* (HE, 19 r.p.v.), the *Compact Half-edge* [1] (13 r.p.v.) and *Winged-edge* (19 r.p.v.) which are edge-based data structures, as well as *Corner Table* (CT, 13 r.p.v.), SOT (6 r.p.v.) and SQUAD which are all face-based. For SQUAD we follow the well detailed description in [16]: we use the *Matching&Pairing* procedure to compute a matching between vertices and faces and a pairing of triangles into quadrangles. For the evaluation of LR we have implemented its construction phase:[4] its *Ring-Expander* procedure (whose pseudo-code is provided in [17]), computes in linear time a quasi-hamiltonian circuit of the graph (see Fig. 9). We also provide a comparison with our previous structure [5, Thm. 4] (whose code is publicly available).

4.1 Experimental setting and datasets

All our tests are performed on a laptop Latitude 7410 equipped with an Intel Core i7-10610U CPU (1.80 GHz, with 8MB of L3 cache), running under Ubuntu 22.04 and with Java 18 64-bit. We run our experiments on a single core allocating 4GB of RAM (we use the option `-server -Xmx4G` for the JVM).

Our tests involve real-world meshes from the aim@shape and the Thingi10K repositories as well as synthetic datasets of various size and type including grid-like meshes, Delaunay triangulations (of random points in a disk), stacked and random planar triangulations (see Figure 10 and full version for more details on datasets and implementations).

[3] Datasets, runnable programs and source code (pure Java) are made available at: www.lix.polytechnique.fr/~amturing/software.html.

[4] This allows a direct comparison of storage performances but not on runtimes. An indirect comparison of the runtimes of SCARST and LR is also possible, since the runtimes of LR have been evaluated and compared to CT in [17]: in the case of regular 3D meshes CT and LR achieve very close running times. To our knowledge the source code of LR is not publicly available.

4.2 Runtime performances

As in previous works, we evaluate the timings by comparing the runtime performances for the procedures `degree`, `adjacent` and `normal` that perform a traversal of the edges incident to a given vertex (`adjacent` and `normal` also involves the computation of the `Target`(v) operator; `normal` requires a few numerical calculations) and `bfs` (performing a BFS graph traversal from a random seed). Fig. 11, 12 and 13 report a comparison of the average timings per query (the average is computed over 200 runs of our navigational operators). For the `degree` and `normal` operators vertices are accessed sequentially according to the original vertex ordering of the input mesh: the vertex orderings coincide for all data structures, except the one described by our Cor. 5. In the case of the `adjacent` operator we select randomly $10K$ pairs of adjacent vertices (*real edges*) and $10K$ pairs of non adjacent vertices.

Discussion. As one would expect, our data structures are in most cases slower than explicit data structures (or some previous compact representations) being much more compact. According to our results we lose a factor between 1.2 and 3.8 with respect to uncompressed storage, depending on the type of query and the class of tested graphs. It is worth noting that SCARST-OS achieves most of the time better runtimes than SCARST-OT despite the fact that the worst case $O(1)$ time is not guaranteed, in practice the computation of vertex degrees has a global linear complexity when iterating over all vertices. SCARST-OT becomes more advantageous only when the number of high degree vertices is not negligible, which occurs for irregular graphs (e.g. `random` triangulations, see Fig. 12). Observe that SCARST-RS is almost always slower than the others, which is due to two facts: because of its compactness the retrieval of service bits is more involved and, more important, the original vertex ordering is lost, which decreases the runtime performances for graphs having good vertex locality.

Observe that SCARST data structures are slower than `SOT` and `SQUAD` (between 1.2 and 2.1 times) when processing vertices in a sequential manner (as for `degree` operator). SCARST data structures become much more competitive, from both the storage and runtimes point of view, when accessing the data in a random manner (`adjacent` operator) or performing non local graph traversals (`bfs` operator), as illustrated in Fig. 13. In some cases compact data structures may be even faster than conventional representations because of their smaller memory footprint which leads to reduce cache misses (see left and middle plots in Fig. 13).

Construction phase

We plot in Fig. 14 the running time and memory usage of the construction phase of the SCARST-OT data structure for the `Delaunay` datasets (SCARST-OS and SCARST-RS achieve similar performances and are omitted). The timing cost consists of two parts: the computation of the Schnyder wood and the computation of references stored by SCARST-OT (we use the

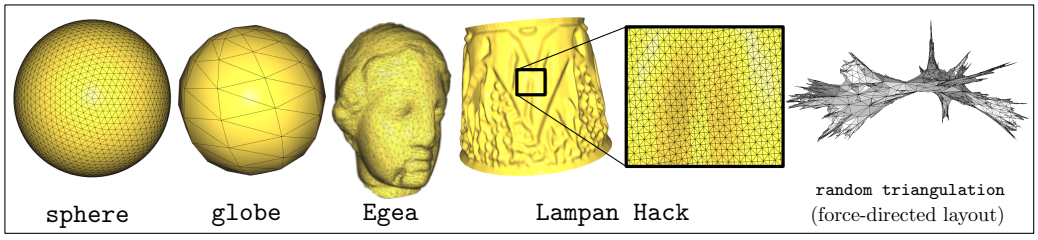


Figure 10 Some examples of 3D meshes and graphs tested in our experiments.

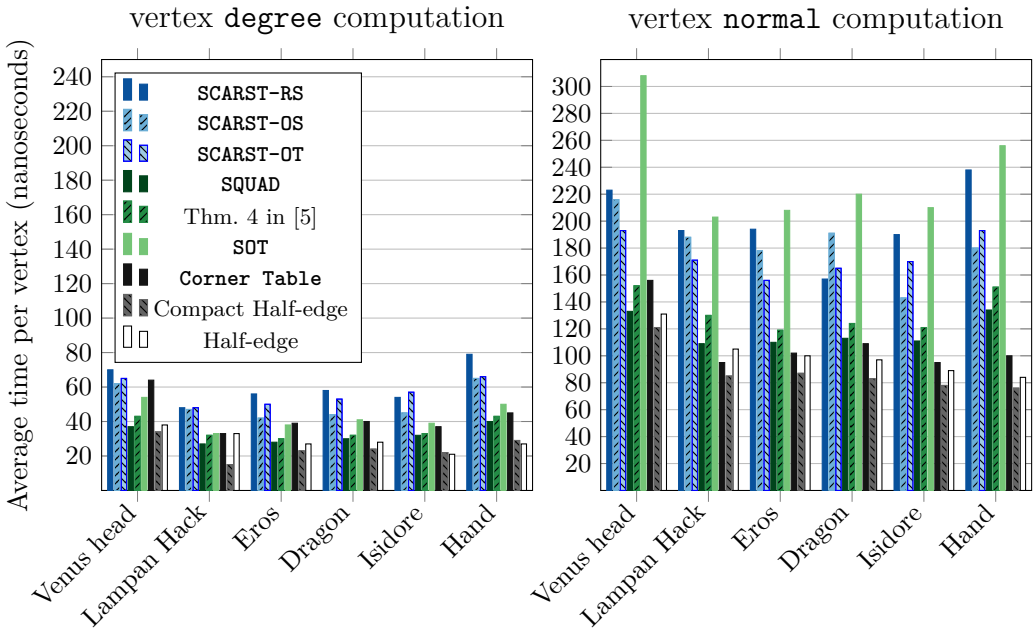


Figure 11 Comparison of runtime performances for the `degree` and `normal` operators on real-world 3D meshes. We report the average timings (over 200 runs) expressed in nanoseconds per vertex: vertices are accessed sequentially. Meshes are listed from left to right by increasing size.

Compact Half-edge structure for storing the input graph). Our results confirm the linear behaviour of the construction: once the mesh is loaded in main memory, our construction algorithm is extremely fast, being able to process about $1.56M$ vertices per second. The right plots show the peaks of the memory consumption during the construction phase (we distinguish two phases: the computation of the Schnyder wood and the data structure initialization). The memory requirements for processing a Delaunay triangulation with $20M$ vertices never exceed $2.5GB$, including the cost of geometric coordinates (12 bytes/vertex). According to our experiments these performances are similar to the ones of `SQUAD`.

4.3 Storage comparison

The charts in Fig. 15 show the storage performances of our data structures compared to the ones of `LR` [17] and `SQUAD` [16] on both synthetic and real-world datasets. The results in Fig. 15 clearly confirm that our data structures are sensitive to the vertex degree distribution: the storage costs decrease to the best case bounds reported in Table 1 as d_6 tends to 100%. It is worth noting that our SCARST representations achieve compression rates which are always well below the worst case bounds established in Section 3.

Structures preserving vertex ordering. SCARST-OT preserves vertex ordering and allows $O(1)$ time edge/face navigation as `SQUAD` does. The storage of SCARST-OT is almost always below 4 r.p.v. which makes SCARST-OT between 10% and 25% more compact than `SQUAD` in practice. The only exception are grid-like meshes such as `Lampan Hack`, having roughly half of the vertices of high degree. SCARST-OS only requires 3 r.p.v., which make it at least 25% more compact than `SQUAD` (and for some datasets even more).

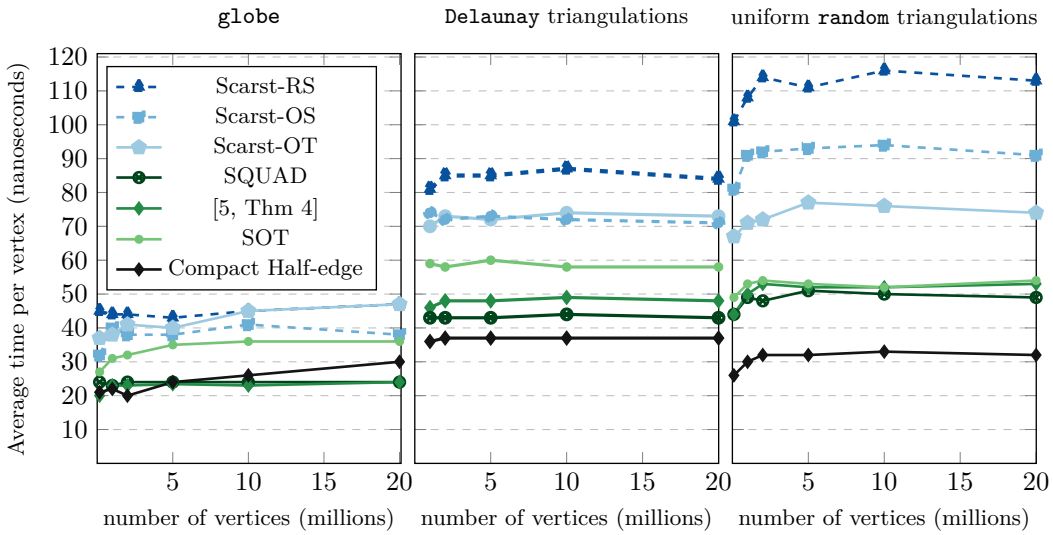


Figure 12 Runtime performances of the `degree` operator on synthetic datasets. We plot the average timings over 200 runs expressed in nanoseconds per vertex as a function of the mesh size.

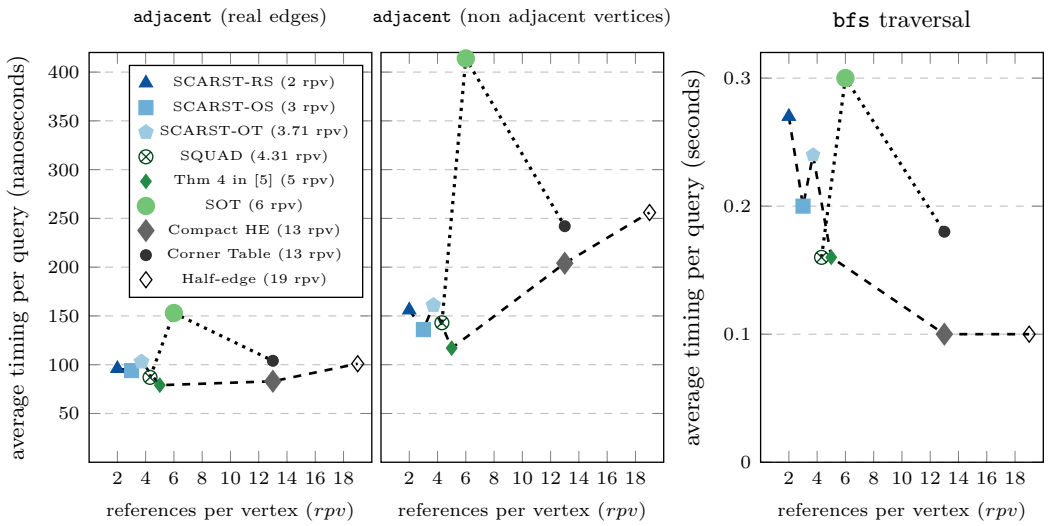


Figure 13 Trade-off between storage and runtime for a `delaunay1M` graph (vertices are accessed at random). We plot time versus representation size for adjacency testing. Left: with positive answer. Middle: negative answer. Right: rutime for a BFS traversal (with a random seed). Results for triangle-based representations are joined by a dotted line and edge-based by a dashed line.

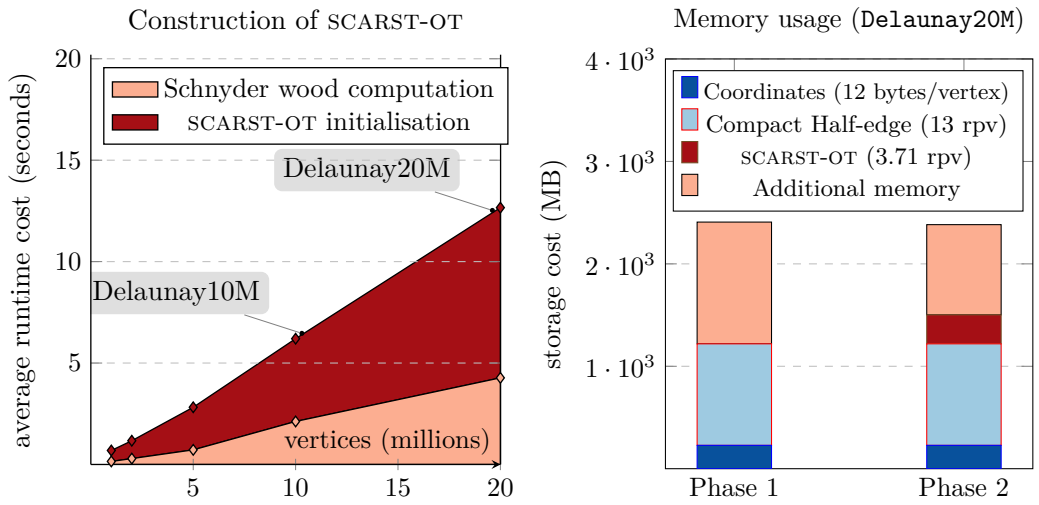


Figure 14 Construction of `SCARST-OT`. Left: Runtime cost. Right: peak of memory consumption during the Schnyder wood computation (phase 1) and the `SCARST-OT` initialization (phase 2).

Structures not preserving vertex ordering. Concerning structures reordering the vertices, SCARST-RT and SCARST-WC have similar compression rates compared to `LR` while providing worst case provable guarantees, and performing much better in some cases. More precisely, our experiments confirm the intuition that for regular 3D meshes (with $d_6 \geq 50\%$), `LR` achieves in practice very good compression rates ranging between 2 and 2.5 r.p.v..

But we have observed that in the case of irregular meshes (few degree 6 vertices and many separating triangles) the storage cost of `LR` increases significantly. According to our experiments it is more than 6 r.p.v. for random triangulations and even worse for some graphs (e.g. for stack triangulation we got more than 12 r.p.v.), which makes `LR` far less compact than SCARST (a few more details on the storage of `LR` can be found in full version).

5 Concluding remarks

Topology extensions. The results stated here and in [25, 5, 12, 13] hold for planar triangulations.[5] We mention that it could be possible to adapt our data structures to deal with genus 1 triangulations making use of toroidal Schnyder woods [11]. In the higher genus case one can planarize the surface, cutting the graph along non contractible cycles of length $O(\sqrt{n})$: as mentioned in [9], this approach is suitable for dealing with bounded genus, since the number of duplicated vertices remains asymptotically negligible.

Sensitivity to regularity. Observe that the number of additional skipping references (e.g. ccw pointers in Fig. 4) can be modified in order to obtain different trade-offs between running time and storage cost (the bounds depending on η in our theorems will change accordingly). Our algorithms are sensitive to the mesh regularity. This sensitivity appears in the complexities through the parameter η for which we give worst case bounds. These bounds can be improved if we have some knowledge of the input graph: this occurs for instance for some classes of triangulations (such as Delaunay triangulations of random points or uniform planar triangulations) for which the vertex distribution can be estimated asymptotically.

[5] It is worth noting that a large proportion of real-world 3D meshes have planar topology: among the 5806 manifold 3D meshes of the `Thingi10K` repository, 2474 do have genus 0.

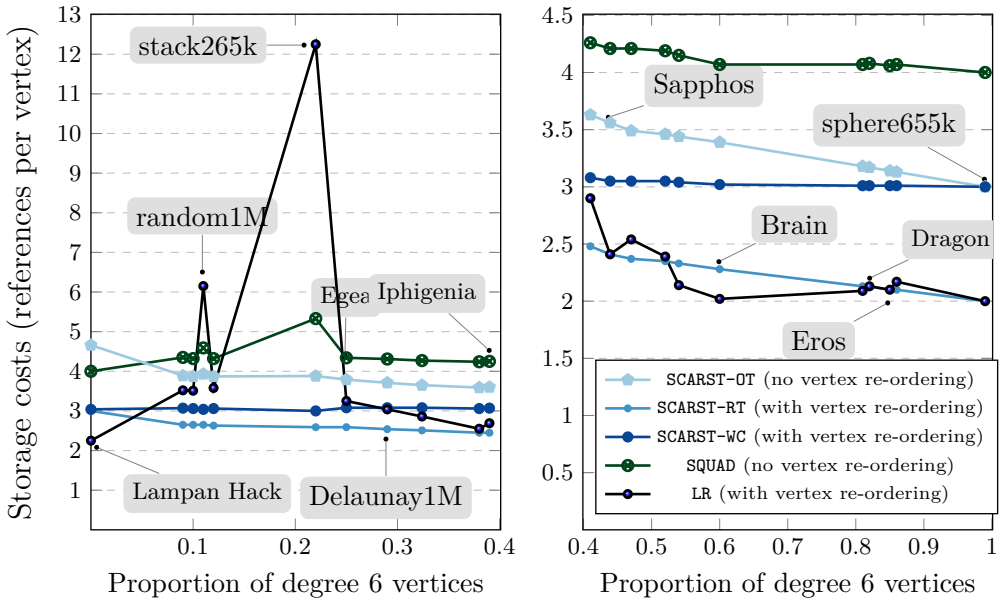


Figure 15 Comparison of storage results (expressed in terms of r.p.v.). Left: irregular and synthetic graphs ($d_6 < 40\%$). Right: 3D real-world regular meshes ($d_6 \geq 40\%$).

Streamable decompression from compressed format. In our previous paper [5, Section 3], we describe a procedure to construct the compact representation of Proposition 1 directly from a compressed format of size $4n$ bits without the usage of extra storage. This can be adapted to our SCARST-OS and SCARST-OT data structures.

References

1 Tyler J Alumbaugh and Xiangmin Jiao. Compact array-based mesh data structures. In *Proceedings of the 14th International Meshing Roundtable*, pages 485–503. Springer, 2005. doi:10.1007/3-540-29090-7_29.

2 Bruce G Baumgart. Winged edge polyhedron representation. Technical report, DTIC Document, 1972. URL: http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=AD0755141.

3 Bruce G Baumgart. A polyhedron representation for computer vision. In *Proceedings National Computer Conference and Exposition*, pages 589–596. ACM, 1975. doi:10.1145/1499949.1500071.

4 Jean-Daniel Boissonnat, Olivier Devillers, Sylvain Pion, Monique Teillaud, and Mariette Yvinec. Triangulations in CGAL. *Computational Geometry: Theory & Applications*, 22:5–19, 2002. doi:10.1016/S0925-7721(01)00054-2.

5 Luca Castelli Aleardi and Olivier Devillers. Array-based compact data structures for triangulations: Practical solutions with theoretical guarantees. *J. Comput. Geom.*, 9(1):247–289, 2018. doi:10.20382/jocg.v9i1a8.

6 Luca Castelli Aleardi, Olivier Devillers, and Abdelkrim Mebarki. Catalog based representation of 2d triangulations. *International Journal of Computational Geometry & Applications*, 21:393–402, 2011. doi:10.1142/S021819591100372X.

7 Luca Castelli Aleardi, Olivier Devillers, and Jarek Rossignac. ESQ: editable squad representation for triangle meshes. In *25th Conference on Graphics, Patterns and Images, SIBGRAPI 2012*, pages 110–117, 2012. doi:10.1109/SIBGRAPI.2012.24.

8 Luca Castelli Aleardi, Olivier Devillers, and Gilles Schaeffer. Succinct representations of planar maps. *Theor. Comput. Sci.*, 408(2-3):174–187, 2008. doi:10.1016/j.tcs.2008.08.016.

9 Luca Castelli Aleardi, Éric Fusy, and Thomas Lewiner. Optimal encoding of triangular and quadrangular meshes with fixed topology. In *Proceedings of the 22nd Annual Canadian Conference on Computational Geometry*, pages 95–98, 2010. URL: http://cccg.ca/proceedings/2010/paper27.pdf.

10 Markus Denny and Christian Sohler. Encoding a triangulation as a permutation of its point set. In *Canadian Conference on Computational Geometry*, 1997. URL: https://api.semanticscholar.org/CorpusID:26767430.

11 Vincent Despré, Daniel Gonçalves, and Benjamin Lévêque. Encoding toroidal triangulations. *Discrete & Computational Geometry*, 57(3):507–544, 2017. doi:10.1007/s00454-016-9832-0.

12 Leo Ferres, José Fuentes-Sepúlveda, Travis Gagie, Meng He, and Gonzalo Navarro. Fast and compact planar embeddings. *Comput. Geom.*, 89:101630, 2020. doi:10.1016/j.comgeo.2020.101630.

13 José Fuentes-Sepúlveda, Diego Seco, and Raquel Viaña. Succinct encoding of binary strings representing triangulations. *Algorithmica*, 83(11):3432–3468, 2021. doi:10.1007/s00453-021-00861-4.

14 Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM transactions on graphics (TOG)*, 4(2):74–123, 1985. doi:10.1145/282918.282923.

15 Stefan Gumhold and Wolfgang Straßer. Real time compression of triangle mesh connectivity. In *Proc. of SIGGRAPH 1998*, pages 133–140, 1998. doi:10.1145/280814.280836.

16 Topraj Gurung, Daniel Laney, Peter Lindstrom, and Jarek Rossignac. SQuad: Compact representation for triangle meshes. *Computer Graphics Forum*, 30(2):355–364, 2011. doi:10.1111/j.1467-8659.2011.01866.x.

17 Topraj Gurung, Mark Luffel, Peter Lindstrom, and Jarek Rossignac. LR: compact connectivity representation for triangle meshes. *ACM transactions on graphics (TOG)*, 30(4), 2011. doi:10.1145/2010324.1964962.

18 Topraj Gurung, Mark Luffel, Peter Lindstrom, and Jarek Rossignac. Zipper: A compact connectivity data structure for triangle meshes. *Computer-Aided Design*, 45(2):262–269, 2013. doi:10.1016/j.cad.2012.10.009.

19 Topraj Gurung and Jarek Rossignac. SOT: compact representation for tetrahedral meshes. In *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 79–88. ACM, 2009. doi:10.1145/1629255.1629266.

20 Marcelo Kallmann and Daniel Thalmann. Star-vertices: a compact representation for planar meshes with adjacency information. *Journal of Graphics Tools*, 6(1):7–18, 2001. doi:10.1080/10867651.2001.10487533.

21 Stephen G. Kobourov. Canonical orders and Schnyder realizers. In *Encyclopedia of Algorithms*, pages 277–283. Springer, 2016. doi:10.1007/978-1-4939-2864-4_650.

22 Dominique Poulalhon and Gilles Schaeffer. Optimal coding and sampling of triangulations. *Algorithmica*, 46(3-4):505–527, 2006. doi:10.1007/s00453-006-0114-8.

23 Jarek Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *Visualization and Computer Graphics, IEEE Transactions on*, 5(1):47–61, 1999. doi:10.1109/2945.764870.

24 Walter Schnyder. Embedding planar graphs on the grid. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, volume 90, pages 138–148, 1990. URL: http://departamento.us.es/dma1euita/PAIX/Referencias/schnyder.pdf.

25 Jack Snoeyink and Bettina Speckmann. Tripod: a minimalist data structure for embedded triangulations. In *Workshop on Comput. Graph Theory and Combinatorics*, 1999. URL: https://www.win.tue.nl/~speckman/papers/Tripod.pdf.

26 Andrzej Szymczak, Davis King, and Jarek Rossignac. An edgebreaker-based efficient compression scheme for regular meshes. *Comput. Geom.*, 20(1-2):53–68, 2001. doi:10.1016/S0925-7721(01)00035-9.

27 Costa Touma and Craig Gotsman. Triangle mesh compression. In *Proc. of the Graphics Interface 1998 Conference*, pages 26–34, 1998. doi:10.20380/GI1998.04.

28 Katsuhisa Yamanaka and Shin-ichi Nakano. A compact encoding of plane triangulations with efficient query supports. *Information Processing Letters*, 110(18):803–809, 2010. doi:10.1016/j.ipl.2010.06.014.