

## Succinct representation of triangulations with a boundary<sup>\*</sup>

L. Castelli Aleardi<sup>1,2</sup>, Olivier Devillers<sup>2</sup>, and Gilles Schaeffer<sup>1</sup>

<sup>1</sup> Ecole Polytechnique, Palaiseau, France,  
amturing,schaeffe@lix.polytechnique.fr,

<sup>2</sup> Inria Sophia-Antipolis, Projet Geometrica, France  
olivier.devillers@sophia.inria.fr

**Abstract.** We consider the problem of designing succinct geometric data structures while maintaining efficient navigation operations. A data structure is said succinct if the asymptotic amount of space it uses matches the entropy of the class of structures represented.

For the case of planar triangulations with a boundary we propose a succinct representation of the combinatorial information that improves to 2.175 bits per triangle the asymptotic amount of space required and that supports the navigation between adjacent triangles in constant time (as well as other standard operations). For triangulations with  $m$  faces of a surface with genus  $g$ , our representation requires asymptotically an extra amount of  $36(g-1)\lg m$  bits (which is negligible as long as  $g \ll m/\lg m$ ).

### 1 Introduction

The problem of representing compactly the connectivity information of a two-dimensional triangulation has been largely addressed for compression purpose [2]. Indeed for a triangulation with  $m$  triangles and  $n$  vertices, the usual description of the incidence relations between faces, vertices and edges involves  $6m + n$  pointers (each triangle knows its neighbors and its incident vertices, and each vertex has a reference to an incident triangle). In practice, this connectivity information uses  $32 \times 7 = 224$  bits/triangle, or in theory  $7 \log m$  bits/triangle (as for a triangle mesh it holds  $n < m$ ), that is much more than the cost of point coordinates [4]. The enumeration of all different structures that the connectivity can assume shows that for the case of planar triangulations (with degree 3 faces) an encoding requires asymptotically 1.62 bits/triangle (or 3.24 bits/vertex, see

---

<sup>\*</sup> This work has been supported by the French “ACI Masses de données” program, via the Geocomp project, <http://www.lix.polytechnique.fr/~schaeffe/GeoComp/>

[12] for a recent optimal encoding). Similarly, 2.175 bits per triangle are needed to code triangulations when a larger boundary of arbitrary size is allowed (in [1] the entropy of this class of triangulations is computed). In this paper, our purpose is not to compress the data for storage or network transmission, but to design a compact representation that can be used in main memory and supports navigation queries. Since we care for coming down to the entropy bound, this work pertains to the algorithmics of *succinct* data structures, as discussed below.

**Contribution** At a conceptual level, our contribution is to show that two-dimensional geometric objects have nice local neighborhood relations that allow to apply a direct hierarchical approach to represent them succinctly, without even using separators or canonical orders. More precisely, given a triangulation of  $m$  triangles, we propose a structure that uses  $2.175m + O(m \frac{\lg \lg m}{\lg m})$  bits and supports access from a triangle to its neighbors in  $O(1)$  worst case time.

This storage is asymptotically optimal for the class of planar triangulations with a boundary. To our knowledge this is the first optimal representation supporting queries in constant time for geometric data structures.

Our approach extends directly to the more general case of triangulations with  $m$  triangles of a surface with genus  $g$ . In this case, the structure uses  $2.175m + 36(g-1) \lg m + O(m \frac{\lg \lg m}{\lg m} + g \lg \lg m)$  bits, which remains asymptotically optimal for  $g = o(m/\lg m)$ . For  $g = \Theta(m)$ , we still have an explicit dominant term, which is of the same order as the cost of a pointer-based representation. Finally, our approach allows to take advantage of low diversity in the local structure: for instance, when applied to the class of triangulations with vertex degree at most 10, our construction automatically adjusts to the corresponding entropy.

**Related work on compact representations of graphs** A first approach to design better geometric data structures is, as done in [9], to look from the programming point of view for practical solutions that improve by a constant factor on usual representations [4]. From a theoretical point of view however, standard representations are intrinsically non optimal since they use global pointers across the structure:  $\Theta(m \log m)$  bits are needed to index  $m$  triangles.

The seminal work of Jacobson [8] showed that it is possible to represent planar graphs with  $O(n)$  bits, allowing adjacency queries in  $O(\lg n)$  time. The result is based on a compact representation for balanced parenthesis systems, and on the four page decomposition of planar graphs. This two step approach was pushed further by improving on the representation of parenthesis systems or by using alternative graph encodings. Munro and Raman [10] achieve  $O(1)$  time for adjacency between vertices and degree queries: for planar triangulations with  $e$  edges and  $n$  vertices only  $2e + 8n$  bits are asymptotically required, that is, in terms of the number  $m$  of faces, between  $7m$  and  $12m$  bits depending on the boundary size. The best result for triangulations is due to Chuang *et al.* [6]  $2e + n$  bits (which is equivalent to  $3.5m$  for triangulations with a triangular boundary), with slightly different navigation primitives than ours. For the general case of planar graphs, Chiang *et al.* further extended and improved this result [5].

Although this literature is mainly focused on the asymptotic behaviors, it has been demonstrated by Blandford *et al.* [3] that  $O(n)$  data structures can be competitive in practice: using graph separators and local labellings, they propose a compact representation for separable graphs that supports adjacency and degree queries in constant time and saves space already for middle size graphs. The design of compact data structures ultimately relies on partitioning into small regions, inside which local labels can be used, and on describing efficiently inter-region relations. A compact data structure is called *succinct* when its space requirement matches asymptotically the entropy bound at first order. For the previous approach to yield a succinct data structure, the partitioning must be done without increase of entropy, and the inter-region relations must be described within sub-linear space. To our knowledge, this was done successfully only for simpler structures like bit vectors, dictionaries or trees [13, 10, 11].

**Overview of our structure** As opposed to the previous approaches for triangulations, we apply the partitioning process directly to the triangulation, following a three level scheme similar to what was done for trees in [11].

The initial triangulation of  $m$  triangles is divided in pieces (*small triangulations*) having  $\Theta(\lg^2 m)$  triangles, and each small triangulation is then divided into *planar* sub-triangulations (*tiny triangulations*) of size  $\Theta(\lg m)$ . Any such subdivision is acceptable for our approach. We produce one in linear time using a tree partitioning algorithm on a spanning tree of the dual graph.

Then we construct a three level structure. The first level is a graph linking the  $\Theta(\frac{m}{\lg^2 m})$  small triangulations. This graph is classically represented with pointers of size  $O(\lg m)$ , and in view of its number of nodes and edges, its storage requires  $o(m)$  bits. The second level consists in a graph linking the tiny triangulations, or more precisely a map, since the relative order of neighbors around a tiny triangulation matters here. The nodes of this map are grouped according to the small triangulation they belong to. This allows to use local pointers of size  $O(\lg \lg m)$  to store adjacencies between tiny triangulations. The combinatorial information of a tiny triangulation is not explicitly stored at this second level, we just store a pointer to the third level: the catalog of all possible tiny triangulations. The whole size of all these pointers to the third level can be proved to be 2.175 bits per triangle and all other informations are sub-linear.

As such, the structure would not describe completely the initial triangulation: the combinatorics of combining tiny triangulations into a big triangulation is more involved than, *e.g.*, the combinatorics of combining subtrees into a big tree as in [11]. The second level must therefore be enriched with a coloring of the vertices on the boundary of tiny triangulations according to the number of tiny triangulations they belongs to. This coloring describes how the boundary edges of a tiny triangulation are distributed between its neighbors (which are given by the second level map). Like its combinatorial description, the coloring of a tiny triangulation is encoded through a pointer to a catalog of all possible border colorings. The subtle point is that the total size of these pointers is sub-linear, even though the total length of the borders themselves can be linear (recall no

assumption is made on the quality of the decomposition in tiny triangulations). The space requirement is dictated by the cost of the pointers to tiny pieces: since these are planar triangulations with boundary the representation is succinct for this class. On the other hand, restraining the catalog to a subclass (like triangulations with degree at most 10) immediately cuts the pointer sizes and reduces the cost to the associated entropy.

The construction of our representation can be performed in linear time and a complete analysis is provided in [1].

## 2 Preliminaries

**Model of computation** As in previous works about succinct representations of binary trees, our model of computation is a RAM machine with  $O(1)$  time access and arithmetic operation on words of size  $\log_2 m$ . Any element in a memory word can be accessed in constant time, once we are given a pointer to a word and an integer index inside. The machine word size matches the problem size, in the sense that a word is large enough to store the input problem size. We use  $\lg m$  to denote  $\lceil \log_2(1 + m) \rceil$ . From now on, when we speak about the time complexity of an algorithm we refer to the number of elementary operations on words of size  $\lg m$ , and about storage we refer to the size of an object in term of bits.

**Notations and vocabulary** The initial triangulation is denoted  $\mathcal{T}$  and its size  $m$  (from now on the size of any triangulation is its number of triangles). When the triangulation  $\mathcal{T}$  is not planar, we denote by  $g$  its genus. The *small* triangulations, of size between  $\frac{1}{3} \lg^2 m$  and  $\lg^2 m$ , are denoted  $\mathcal{ST}_i$ . Finally the *tiny* triangulations, of size between  $\frac{1}{12} \lg m$  and  $\frac{1}{4} \lg m$ , are denoted  $\mathcal{TT}_j$ .

All tiny triangulations shall be *planar* triangulations with *one* boundary cycle. As subtriangulations of  $\mathcal{T}$ , these tiny triangulations will share their boundary edges. More precisely a boundary edge can be shared by two different tiny triangulations or can also appear twice on the boundary of one tiny triangulation. We call *multiple vertices* those vertices that are incident to at least 3 boundary edges (generically they are shared by more than two tiny triangulations, but self-intersections of boundaries also create multiple vertices). A *side* of a tiny triangulation  $\mathcal{TT}_j$  is a sequence of consecutive boundary edges between two multiple vertices: edges of a same side are shared by  $\mathcal{TT}_j$  with a same tiny triangulation  $\mathcal{TT}_{j'}$  (possibly with  $j' = j$ ). The boundary of a tiny triangulation is divided in this way in a cyclic sequence of sides, called the *coloring* of the boundary. As just seen, this coloring is induced by the distinction multiple/normal vertices.

The exhaustive set of all possible tiny triangulations with at most  $\frac{1}{4} \lg m$  triangles is stored in a structure denoted  $A$  while the set of all colorings of a boundary with less than  $\frac{1}{4} \lg m$  vertices is stored in a structure called  $B$ . The adjacencies between the small triangulations  $\mathcal{ST}_i$  are stored in a graph denoted  $F$ , those between tiny triangulations  $\mathcal{TT}_j$  in a graph  $G$ . The part of  $G$  corresponding to pieces of  $\mathcal{ST}_i$  is denoted  $G_i$ . To be more precise  $G$  must be a *map*: at each node the set of incident arcs (one for each side) is stored in an

array, sorted to reflect the circular arrangement of the sides of the triangulation. For  $F$  we could content with a graph structure, but it is convenient, as discussed in Appendix A, to construct both  $F$  and  $G$  by the same simplification process: in particular, although  $F$  and  $G$  can have loops (corresponding to boundary self intersections) and multiple arcs (corresponding to two subtriangulations sharing different sides), their number of edges is linearly bounded in the genus and number of vertices because they are constructed with all faces of degree at least 3.

For the sake of clarity, from now on we will use the word *arcs* and *nodes* to refer to edges and vertices of the maps  $F$ ,  $G$  and  $G_i$ , and keep the word edges and vertices only for the edges and vertices of  $\mathcal{T}$  and of the subtriangulations.

**Operations on the triangulation** The following primitive operations are supported by our representation in  $O(1)$  time.

- $Triangle(v)$ : returns a triangle incident to vertex  $v$ ;
- $Index(\Delta, v)$ : returns the index of vertex  $v$  in triangle  $\Delta$ ;
- $Neighbor(\Delta, v)$ : returns the triangle adjacent to  $\Delta$  opposite to vertex  $v$  of  $\Delta$ ;
- $Vertex(\Delta, i)$ : returns the vertex of  $\Delta$  of index  $i$ .

With marginal modifications, the structure could also allow for other local operations, like degree queries or adjacency between vertices in constant time.

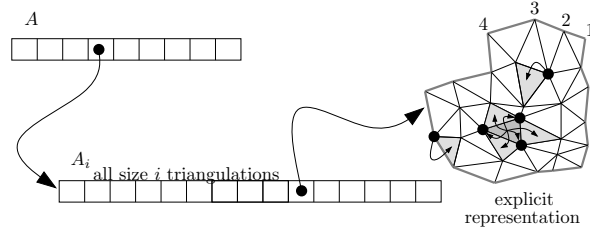
### 3 Exhaustive list of all tiny triangulations

All possible triangulations having  $i$  triangles ( $i \leq \frac{1}{4} \lg m$ ) are generated and their explicit representations are stored in a collection  $A$  of tables  $A_i$ . A reference to a tiny triangulation in  $A_i$  costs asymptotically  $2.175i$  bits because there are at most  $2^{2.175i}$  triangulations with  $i$  triangles (for more details refer to [1]).

In the rest of this section we describe the organization of the structure (see also Figure 1) and we analyze the storage. The construction of the structure can be done in sub-linear time (see [1]).

#### Description of the representation

- $A$  is a table of size  $\frac{1}{4} \lg m$ , in which the  $i$ th element is a pointer to Table  $A_i$ .
- $A_i$  is a table containing all possible triangulations having exactly  $i$  triangles. The  $j$ th element is a pointer to an explicit representation  $A_{i,j}^{explicit}$  of the triangulation  $A_{i,j}$ .
- $A_{i,j}^{explicit}$  contains at least two fields:
  - $A_{i,j}^{explicit}.vertices$  is the table of the vertices of  $A_{i,j}$ . Each vertex just contains the index of an incident triangle in Table  $A_{i,j}^{explicit}.triangles$ . By convention, the boundary vertices appear first in that table, and are stored in the counter-clockwise order of the boundary of  $A_{i,j}$ . For boundary vertices, the incident triangle stored is required to be the one incident to next edge on the boundary.
  - $A_{i,j}^{explicit}.triangles$  is the table of the triangles of  $A_{i,j}$ . Each triangle contains the indices of its vertices in  $A_{i,j}^{explicit}.vertices$  and of its neighbors in  $A_{i,j}^{explicit}.triangles$ . Triangles on the boundary have *null* neighbors.



**Fig. 1.** Storage of all tiny triangulations

**Storage analysis** *The storage of Table A, and of all the information associated with Tables  $A_i$  requires asymptotically  $O(m^{0.55})$  bits.*

- $A$  is a table of size  $\frac{1}{4} \lg m$  of pointers of size  $\lg m$  and thus costs  $O(\lg^2 m)$ .
  - $A_i$  is a table of at most  $2^{2 \cdot 175i}$  pointers on  $\lg m$  bits, thus the storage of  $A_i$  requires less than  $O(2^{2 \cdot 175i} \lg m)$  bits.
  - The explicit representation  $A_{i,j}^{explicit}$ :
    - $A_{i,j}^{explicit}.triangles$  (resp.  $A_{i,j}^{explicit}.vertices$ ) is a table of size less than  $i \leq \lg m$  (resp. less than  $i + 2 \leq \lg m + 2$ ). Each element consists in several indices of value less than  $i$ , thus representable on  $\lg \lg m$  bits.
- Thus the size of one  $A_{i,j}^{explicit}$  is  $O(\lg m \lg \lg m)$  bits and the total size of the  $A_{i,j}^{explicit}$  indexed in Table  $A_i$  is less than  $O(2^{2 \cdot 175i} \lg m \lg \lg m)$  bits.

Finally the storage requirement for the whole structure  $A$  is obtained by summing over  $i$ , which yields  $O(2^{2 \cdot 175 \frac{1}{4} \lg m} \lg m \lg \lg m) = O(m^{0.55})$ .

## 4 Boundary descriptions

As already explained, we need to distinguish some vertices on the boundary of each tiny triangulation. This will be done with the help of a structure essentially equivalent to a bit vector supporting rank and select operations in constant time. This problem was addressed very much in detail in the literature and compact solutions have been proposed (see [14], [13], [7] and ref. therein). Since the bit vectors we use have size at most  $\frac{1}{4} \lg m$ , we can content with a simple explicit encoding of all bit-vectors of size  $p$  and weight  $q$  in a collection  $B$  of tables  $B_{pq}$ . Then  $B_{pq}$  contains  $\binom{p}{q}$  elements and a reference to one entry of  $B_{pq}$  has size  $\lg \binom{p}{q} \leq \min(q \lg p, p)$  bits (observe that the size of a reference is at most  $\frac{1}{4} \lg m$ , which allows to index in tables  $B_{pq}$  in  $O(1)$  time). In the rest of this section we provide the description and analysis of the structure.

### Description of the representation

- $B$  is a bi-dimensional array of size  $\frac{1}{4} \lg m \times \frac{1}{4} \lg m$ : each entry  $B(p, q)$  is a pointer to Table  $B_{pq}$ .

- $B_{pq}$  is a table containing for the  $k$ th bit-vector of size  $p$  and weight  $q$  a pointer to a structure  $B_{pqk}^{RS}$  allowing Rank/Select in constant time.
- $B_{pqk}^{RS}$  is a table of length  $p$  with two fields storing the precomputed result for  $Rank_1$  and  $Select_1$ :
  - $B_{pqk}^{RS}(i).rank$  is the number of '1's that precede the  $i$ -th bit.
  - $B_{pqk}^{RS}(i).select$  is the position of the  $i$ -th '1' in the vector.

**Storage analysis** *The storage of Table  $B$ , and of all the information associated with Tables  $B_{pqk}$  requires asymptotically  $O(m^{\frac{1}{4}} \lg m \lg \lg m)$  bits.*

- $B$  is a table of  $(\frac{1}{4} \lg m)^2$  pointers of size  $\lg m$ , its size is  $O(\lg^3 m)$  bits.
- $B_{pq}$  is a table containing  $\binom{p}{q}$  pointers of size  $O(\lg m)$ .
- $B_{pqk}^{RS}(i).rank$ ,  $B_{pqk}^{RS}(i).select$  are all integers less than  $\frac{1}{4} \lg m$  and then representable on  $\lg \lg m$  bits. The size of  $B_{pqk}^{RS}$  is  $O(\lg m \lg \lg m)$ .

The total amount of space required for storing all the bit-vectors of size (and weight) less than  $\frac{1}{4} \lg m$  is then  $\sum_{p,q} \binom{p}{q} O(\lg m \lg \lg m) = (\sum_p 2^p) O(\lg m \lg \lg m)$ , which is bounded by  $2^{\frac{1}{4} \lg m + 1} O(\lg m \lg \lg m) = O(m^{\frac{1}{4}} \lg m \lg \lg m)$ .

## 5 Map of tiny triangulations

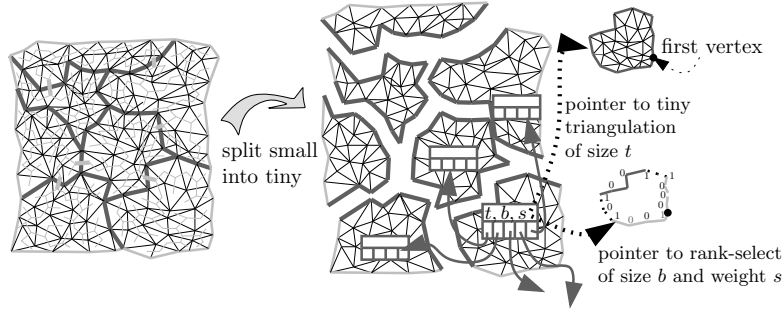
The main triangulation is split into small triangulations which are themselves split into tiny triangulations. In this section we describe the map  $G$  that stores the incidences between tiny triangulations. The memory for this map is organized by gathering nodes of  $G$  that correspond to tiny triangulations that are part of the same small triangulation  $\mathcal{ST}_i$  in a sub-map  $G_i$ . The purpose of this partition is to allow for the use of local pointers of small size for references inside a given sub-map  $G_i$ .

The map  $G$  may have multiple arcs or loops but all its faces have degree  $\geq 3$ . Each arc of  $G$  between  $\mathcal{TT}_j$  and  $\mathcal{TT}_{j'}$  corresponds to a side shared by  $\mathcal{TT}_j$  and  $\mathcal{TT}_{j'}$ .

**Description of the representation** The memory dedicated to  $G$  is organized in a sequence of variable size zones, each dedicated to a  $G_i$ . The memory requirements are analyzed afterward.

In the zone for  $G_i$ , for each node  $G_{i,j}$  corresponding to a tiny triangulation  $\mathcal{TT}_{i,j}$ , we have the following informations:

- $G_{i,j}^t$  is the number of triangles in  $\mathcal{TT}_{i,j}$ .
- $G_{i,j}^b$  is the size of the boundary of  $\mathcal{TT}_{i,j}$ .
- $G_{i,j}^A$  is the index of the explicit representation of  $\mathcal{TT}_{i,j}$  in Table  $A_{G_{i,j}^t}$ .
- $G_{i,j}^s$  is the degree of the node  $G_{i,j}$  (it is also the number of sides of  $\mathcal{TT}_{i,j}$ )
- $G_{i,j}^B$  is the index in Table  $B_{G_{i,j}^b, G_{i,j}^s}$  of a bit vector of size  $G_{i,j}^b$  and weight  $G_{i,j}^s$ . (This bit vector encodes the way the boundary of  $\mathcal{TT}_{i,j}$  splits into sides: the  $i$ th bit is 0 if the  $i$ th vertex on the boundary of  $\mathcal{TT}_{i,j}$  is inside a side, or 1 if this is a multiple vertex that separates two sides)



**Fig. 2.** This Figure shows the decomposition of a small triangulation into tiny triangulations and the map  $G_i$  that describes their adjacency relations

— Each of the  $G_{i,j}^s$  arcs of  $G_i$  that are incident to  $G_{i,j}$  is described by some additional information (beware that loops appear twice). Assume that the  $k$ th such arc connects  $G_{i,j}$  to a neighbor  $G_{i',j'}$  in  $G$ , then we store:

- $G_{i,j,k}^{address}$  the relative address of the first bit concerning the node of the neighbor in the memory zone associated to its small triangulation  $G_{i'}$ .
- $G_{i,j,k}^{back}$  the index  $k'$  of the side corresponding to the current arc in the numbering of sides at the opposite node  $G_{i',j'}$ .
- $G_{i,j,k}^{small}$  the index of the small triangulation  $G_{i'}$  in the table of the neighbors of  $G_i$  in the main map  $F$  (if  $i' = i$  then this index is set to 0).

**Storage analysis** *The storage of map  $G$  requires asymptotically  $2.175m + O(g \lg \lg m) + O\left(m \frac{\lg \lg m}{\lg m}\right)$  bits.*

For each node:

- $G_{i,j}^t$ ,  $G_{i,j}^b$  and  $G_{i,j}^s$  are less than  $\frac{1}{4} \lg m$ : each is stored in  $\lg \lg m$  bits.
- $G_{i,j}^A$  is an index in Table  $A_{G_{i,j}^t}$  stored in  $2.175G_{i,j}^t$  bits (see Section 3)
- $G_{i,j}^B$  is an index in  $B_{G_{i,j}^b, G_{i,j}^s}$  stored in  $G_{i,j}^s \lg G_{i,j}^b$  bits (see Section 4)
- The number of tiny triangulations neighboring  $G_{i,j}$  is  $G_{i,j}^s < \frac{1}{4} \lg m$ . We have for each:

- the pointers  $G_{i,j,k}^{address}$  are stored in  $K \lg \lg m$  bits ( $K$  chosen below).
- $G_{i,j,k}^{back}$  is less than  $\frac{1}{4} \lg m$  and thus can be stored in  $\lg \lg m$  bits.
- $G_{i,j,k}^{small}$  requires  $2 \lg \lg m$  bits of storage: indeed a small triangulation has at most  $\lg^2 m$  triangles, hence at most  $\lg^2 m$  edges on its boundary, thus the table of the neighbors of  $G_i$  in  $F$  has less than  $\lg^2 m$  entries.

Since each arc appears on at most two nodes, the cost per arc can be evaluated independently as  $2(K + 3) \lg \lg m$  bits per arc. It then remains for node  $G_{i,j}$  of  $G_i$  a cost of  $3 \lg \lg m + 2.175G_{i,j}^t + G_{i,j}^s \lg G_{i,j}^b$ .

The number of nodes is at most  $12 \lg m$  and the number of arcs (including arcs directed to other  $G_{i'}$ ) is bounded by the number of edges of  $\mathcal{T}$  incident to triangles of  $\mathcal{ST}_i$ , that is by  $\lg^2 m$ .

The cost for  $G_i$  is thus  $C_i \leq 2(K+3) \lg^2 m \lg \lg m + 12 \lg m (3 \lg \lg m + 2.175 \frac{1}{4} \lg m + \frac{1}{4} \lg m \lg \lg m)$ . Taking  $K = 5$ , we have  $\lg C_i < K \lg \lg m$  for all  $m \geq 2$ , which validates our hypothesis for the storage of  $G_{i,j,k}^{address}$ .

The overall cost for the complete map  $G$  is obtained by summing over  $i, j$ :

$$\begin{aligned} & \sum_i \sum_j (3 \lg \lg m + 2.175 G_{i,j}^t + G_{i,j}^s (\lg G_{i,j}^b) + G_{i,j}^s \cdot 8 \lg \lg m) \\ & \leq 2.175 \sum_{i,j} G_{i,j}^t + 9 \lg \lg m \sum_{i,j} G_{i,j}^s + 3 \lg \lg m \cdot 12 \frac{m}{\lg m}. \end{aligned}$$

The sum over  $G_{i,j}^t$  is the total number of triangles, *i.e.*  $m$ . The sum over  $G_{i,j}^s$  is the sum of the degrees of the nodes of the map  $G$ , or equivalently, twice its number of arcs.

Since  $G$  has only faces of degree  $\geq 3$ , its number  $a$  of arcs linearly bounds its number  $f$  of faces:  $2a = \sum_f d(f) \geq 3f$ . Euler's formula can then be written (with  $n$  for the number of nodes and  $g$  for the genus of  $G$  which is also the genus of  $\mathcal{T}$ ):

$$3(a+2) = 3n + 3f + 6g \quad \Leftrightarrow \quad a \leq 3n + 6(g-1).$$

Finally the number  $n$  of nodes of  $G$  is bounded by  $12m/\lg m$ , so that the cost of representing  $G$  is

$$C = 2.175m + 9 \lg \lg m \cdot 2 \left( 3 \cdot 12 \frac{m}{\lg m} + 6(g-1) \right) + 3 \lg \lg m \cdot 12 \frac{m}{\lg m},$$

and the lemma follows. Observe also that the bound  $g \leq \frac{1}{2}m + 1$  yields  $\lg C \leq 2 \lg m + 8$  for all  $m$  which will be used in the next section.

## 6 Graph of small triangulations

The last data structure needed is a graph  $F$  that describes the adjacency relations between small triangulations. The circular arrangement of neighbors is not used here so do not need a map structure as for  $G$ . However,  $F$  is obtained by construction as a map and it is convenient for the storage analysis to observe that, as a map,  $F$  has a genus smaller or equal to the genus of  $G$  and contains no faces of degree less than 3. We adopt here an explicit pointer based representation.

**Description of the representation** We store for each node of  $F$  its degree, a link to the corresponding part of  $G$  and the list of its neighbors. More precisely, for a node  $F_i$  corresponding to a small triangulation  $\mathcal{ST}_i$ :

- $F_i^s$  is the degree of node  $F_i$  in the map  $F$  (it corresponds to the number of small triangulations adjacent to  $\mathcal{ST}_i$ );
- $F_i^G$  is a pointer to the sub-map  $G_i$  of  $G$  that is associated to the small triangulation  $\mathcal{ST}_i$ .
- A table of pointers to neighbors:  $F_{i,k}^{address}$  is the address of the  $k$ th neighbor of  $F_i$  in  $F$ .

**Storage analysis** *The graph  $F$  uses  $36(g-1)\lg m + O\left(\frac{m}{\lg m}\right)$  bits.*

Recall that a small triangulation contains between  $\frac{1}{3}\lg^2 m$  and  $\lg^2 m$  triangles, thus map  $F$  has at most  $3m/\lg^2 m$  nodes.

- $F_i^s$  is less than  $\lg^2 m$ , and thus representable on  $2\lg \lg m$  bits.
- the address of  $G_i$  is a pointer of size bounded by  $2\lg m + 8$ .
- the pointers  $F_{i,k}^{address}$  are stored on  $K'\lg m$  bits each ( $K'$  chosen below).

Summing on all the small triangulations we obtain that the bit size of  $F$  is  $(2\lg m + 8) \cdot 3m/\lg^2 m + K'\lg m \sum_i F_i^s$ . The sum of the  $F_i^s$  is the sum of the degrees of nodes of  $F$ , which is also twice its number of arcs.

In analogy with what was done for the map  $G$ , the number of arcs of  $F$  can be bounded more precisely by three times its number of nodes, which is less than  $3m/\lg^2 m$ , plus six times the genus minus one of  $F$ , which is bounded by the genus of  $\mathcal{T}$ . Using the bound  $g < \frac{1}{2}m + 1$  on the genus, the value  $K' = 3$  is seen to satisfy the constraints for  $m \geq 5$ . Finally the total bit cost for  $F$  is thus:  $36(g-1)\lg m + O(m/\lg m)$ .

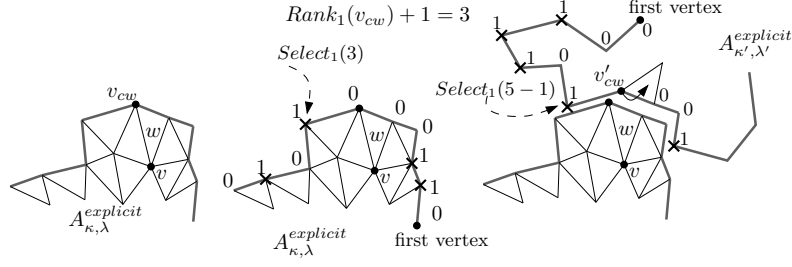
## 7 Navigation

**Triangle and vertex representations** In our structure, a triangle  $t$  is represented by a triple  $(F_i, a, w)$  where  $F_i$  is a node of  $F$  such that  $t \in \mathcal{ST}_i$ ,  $a$  is the address of  $G_{ij}$  in the memory zone of  $G_i$  such that  $t \in \mathcal{TT}_{ij}$  and  $w$  is the index of the triangle corresponding to  $t$  in  $A_{\kappa,\lambda}^{explicit}$  where  $A_{\kappa,\lambda}$  is the triangulation to which  $G_{ij}$  points.

Similarly a vertex is represented by a triple  $(F_i, a, v)$ . Observe that, as such, the representation of a vertex is not unique since, as opposed to triangles, vertices may be shared by several tiny triangulations. As sketched in Section 8, upon adding a negligible amount of information in the map  $G$ , a unique representation could be defined if needed (*e.g.* to test adjacency of vertices in constant time, or to attach data to vertices). However this is not needed for the four operations we have chosen to describe.

**Operations on the triangulation** Given a triangle  $(F_i, a, w)$  or a vertex  $(F_i, a, v)$  the operations *Triangle*, *Index* and *Vertex* are implemented directly by performing the operation in the explicit representation  $A_{\kappa,\lambda}^{explicit}$ .

The difficulty is with *Neighbor* $((F_i, a, w), (F_i, a, v))$ .



**Fig. 3.** Going to the neighbor

- Check if the corresponding neighbor  $w'$  of  $w$  exists in the explicit representation  $A_{\kappa, \lambda}$ : if it does return  $(F_i, a, w')$ .

Otherwise, the neighbor must be found across the boundary of the current tiny triangulation:

- Find in  $A_{\kappa, \lambda}^{explicit}$  the vertex  $v_{cw}$  following  $v$  in clockwise order around  $w$ .
- Compute  $l = Rank_1(v_{cw}) + 1$  in the bit vector associated to  $G_{ij}$ : it says that we are on the  $l$ th side of  $TT_{ij}$  ( $l = 3$  in Figure 3);
- Compute  $l' = Select_1(l) - v_{cw}$ : it says that  $v_{cw}$  is  $l'$ th vertex before the end of the side; ( $l' = 1$  in Figure 3); recall that  $Select_1(l)$  gives the position of the last vertex of the current  $l$ th side.

— Let  $x = G_{i,j,l}^{address}$ ,  $y = G_{i,j,l}^{back}$  and  $z = G_{i,j,l}^{small}$ .

— If  $z > 0$  then we must change also small triangulation: let  $G_{i'}$  be the sub-map of  $G$  pointed at by the  $z$ th neighbor  $F_{i'}$  of  $F_i$  in  $F$ .

Otherwise (that is,  $z = 0$ ) let  $G_{i'}$  be equal to  $G_i$ .

— Let  $G_{i',j'}$  be the node of  $G$  at address  $x$  in the memory zone of  $G_{i'}$  and  $A_{\kappa', \lambda'}^{explicit}$  the tiny triangulation it points at ( $y = 5$  in Figure 3, the  $y$ th side of  $G_{i',j'}$  matches the  $l$ th side of  $G_{i,j}$ ).

— Let  $v'_{cw} = Select_1(y - 1) + l'$  in the bit vector associated to  $G_{i',j'}$ : then  $v'_{cw}$  in  $A_{\kappa', \lambda'}^{explicit}$  matches  $v_{cw}$  in  $A_{\kappa, \lambda}^{explicit}$ .

— Let  $w'$  be the triangle pointed at by  $v'_{cw}$  in  $A_{\kappa', \lambda'}^{explicit}$ .

— Return triangle  $(F_{i'}, x, w')$ .

## 8 Concluding remarks

*Unique representation for vertices* A vertex on boundary of a tiny triangulation has several representations  $(F_i, a, v)$ . To show how to test that two such representations correspond to the same vertex of  $\mathcal{T}$  in constant time, let us distinguish three types of ambiguous vertices: vertices incident to only two boundary edges, multiple vertices incident to at most two small triangulations, and multiple vertices incident to at least three small triangulations. Identity can already be tested for the first type. For the  $O(n/\lg n)$  vertices of the second type, a  $\lg \lg n$  labelling (local to each  $\mathcal{T}_i$ ) can be used to describe the multiple vertices on the boundary

of  $\mathcal{T}\mathcal{T}_{ij}$  in an ordered array at each  $G_{ij}$ , and with the boundary description this allows to test identity. Finally upon listing in a table  $F_i^{vertex}$  the vertices of the third type appearing in each  $\mathcal{ST}_i$ ,  $O(\lg \lg n)$  indices to this table can be added in  $G_{ij}$  to allow for the test. The extra storage is negligible at first order.

*Attaching information* The proposed structure represents only the connectivity of the triangulation. One may want to attach information to vertices or triangles, such as vertices coordinates (or colors...). This should be done by adding the information to nodes of  $G$ . For instance one can add to  $G_{i,j}$  a table  $G_{i,j}^{coordinate}$  describing the coordinates of the vertices of  $\mathcal{T}\mathcal{T}_{i,j}$ . This Table contains the coordinates of all the internal vertices of  $\mathcal{T}\mathcal{T}_{i,j}$  and a selection of its boundary vertices, so that vertices lying on the side between two tiny triangulations are stored only once. To retrieve vertices shared by several tiny triangulations, one uses the above unique representation. Basic compression on these coordinates can be obtained by giving them in a frame local to  $G_{i,j}$ .

*Practical implementation* The result here is mainly theoretical: if  $m$  is one billion,  $\frac{1}{4} \lg m$  is only 7. However the value  $\frac{1}{4}$  is chosen to ensure that the table  $A$  can be constructed in sub-linear time: looking at the actual number of triangulations with  $p$  faces for small  $p$ , one can check that constructing the table of all tiny triangulations up to size 13 is actually be feasible. In particular Table  $A$  can be computed once and for all and stored. We intend to implement and test a simplified version of this work, by gathering triangles in small groups of 3 to 5 triangles and making a map of these groups.

## References

1. L. Castelli Aleardi, O. Devillers, and G. Schaeffer. Compact representation of triangulations. Technical report, RR-5433 INRIA, 2004. available at <http://www.inria.fr/rrrt/rr-5433.html>.
2. P. Alliez and C. Gotsman. Recent advances in compression of 3d meshes. In N.A. Dodgson, M.S. Floater, and M.A. Sabin, editors, *Advances in Multiresolution for Geometric Modelling*, pages 3–26. Springer-Verlag, 2005.
3. D. Blanford, G. Blelloch, and I. Kash. Compact representations of separable graphs. In *Proc. of the Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 342–351, 2003.
4. J.-D. Boissonnat, O. Devillers, S. Pion, M. Teillaud, and M. Yvinec. Triangulations in CGAL. *Comput. Geom. Theory Appl.*, 22:5–19, 2002.
5. Y.-T. Chiang, C.-C. Lin, and H.-I. Lu. Orderly spanning trees with applications to graph encoding and graph drawing. *SODA*, pages 506–515, 2001.
6. 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. *Automata, Languages and Programming*, pages 118–129, 1998.
7. D. R. Clark and J. I. Munro. Efficient suffix trees on secondary storage. In *SODA*, pages 383–391, 1996.
8. G. Jacobson. Space efficient static trees and graphs. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 549–554, 1989.
9. M. Kallmann and D. Thalmann. Star-vertices: a compact representation for planar meshes with adjacency information. *Journal of Graphics Tools*, 6:7–18, 2002.

10. J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM J. on Computing*, 31:762–776, 2001.
11. J. I. Munro, V. Raman, and A. J. Storm. Representing dynamic binary trees succinctly. In *SODA*, pages 529–536, 2001.
12. D. Poulalhon and G. Schaeffer. Optimal coding and sampling of triangulations. In *Proc. Intern. Colloquium ICALP'03*, pages 1080–1094, 2003.
13. R. Raman, V. Raman, and S.S. Rao. Succinct indexable dictionaries with application to encoding k-ary trees and multisets. In *SODA*, pages 233–242, 2002.
14. V. Raman and S.S. Rao. Static dictionaries supporting rank. In *ISAAC*, pages 18–26, 1999.