Compact data structures for triangulations

Luca Castelli Aleardi, Olivier Devillers, Jarek Rossignac

Title:	Compact data structures for triangulations
Name:	Luca Castelli Aleardi[1], Olivier Devillers[2], Jarek Rossignac[3]
Affil./Addr. 1:	LIX, École Polytechnique, France
Affil./Addr. 2:	INRIA, France
Affil./Addr. 3:	Georgia Institute of Technology, USA
Keywords:	Triangulations; triangle meshes; compact data structures; succinct representations
SumOriWork:	2008; Castelli Aleardi, Devillers, Schaeffer 2009; Gurung, Rossignac 2012; Castelli Aleardi, Devillers, Rossignac

Compact data structures for triangulations

Luca Castelli Aleardi[1], Olivier Devillers[2], Jarek Rossignac[3]

[1] LIX, École Polytechnique, France
[2] INRIA, France
[3] Georgia Institute of Technology, USA

Years aud Authors of Summarized Original Work

2008; Castelli Aleardi, Devillers, Schaeffer
2009; Gurung, Rossignac
2012; Castelli Aleardi, Devillers, Rossignac

Keywords

Triangulations; triangle meshes; compact data structures; succinct representations

Problem Definition

The main problem consists in designing space-efficient data structures allowing to represent the connectivity of triangle meshes while supporting fast navigation and local updates.

Mesh structures: definition

Triangle meshes are among the most common representations of shapes. A *triangle mesh* is a collection of triangle faces that define a polyhedral approximation of a surface. A mesh is *manifold* if every edge is bounding either one or two triangles, and if the faces incident to a same vertex define a closed or open fan. Here we focus on manifold meshes. Assuming that the genus and the number of boundary edges is negligible when compared to the number n of vertices, the number m of faces is roughly equal to $2n$.

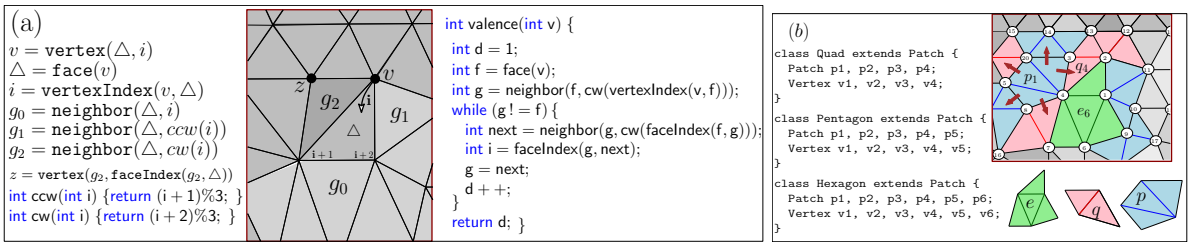


Fig. 1. (a) Triangle-based data structure: each triangle stores references to the 3 neighbors and to the 3 incident vertices yielding $13rpv$. (b) Catalog-based representation: using a catalog of size 3 one can guarantee that any quad is adjacent to at most two other quads, leading to a cost of $8.5\,rpv$.

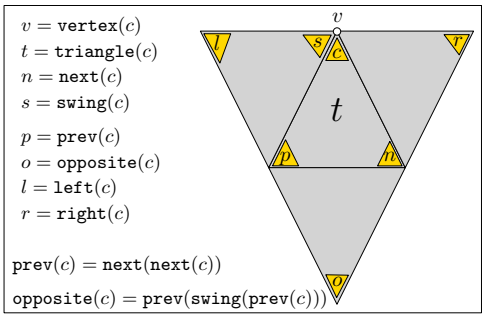
Data structures: classification

Mesh data structures can be compared with respect to several criteria. A basic requirement (the *traversability*) for mesh representations is to provide fast navigational operators allowing to perform a mesh traversal (such as walking around a vertex). Most representations are also *indexable*, allowing to access in constant time to the description of a given vertex or triangle, given its index. In order to support efficient processing of large meshes, one needs to reduce memory trashing during navigation. An effective way of doing so is to design *compact data structures* requiring small storage. Many applications ask for the *modifiability*: the manipulation of meshes requires to perform updates such as vertex insertions/deletions, edge collapses and edge flips. The choice of the data structure should also depend on the *simplicity* of its implementation and on its *practical efficiency* on common input data.

Standard mesh representations

Some common mesh representations are implemented in the explicit pointer-based form. References are used to describe incidence relations between mesh elements and navigation is performed throughout address indirection. For example, a face-based representation [2] provides operators $\texttt{vertex}(\triangle, i)$ (giving the i-th vertex of a triangle $\triangle$) and $\texttt{neighbor}(\triangle, i)$ (giving the i-neighbor of $\triangle$), as well as operator $\texttt{face}(v)$ (returning a triangle incident to vertex v). As illustrated in Fig. 1(a), the combination of these operators allows to implement operators $\texttt{faceIndex}(\triangle_1, \triangle_2)$ (giving the index of $\triangle_1$ among the neighbors of $\triangle_2$) and $\texttt{vertexIndex}(v, \triangle)$ (giving the index of a vertex in $\triangle$). An alternative solution is given by the *Corner Table* proposed by Rossignac and colleagues, which uses integer indices to integer tables and provides a triangulation interface involving the corner operators defined in Fig. 2.

The two abstract data types above fully support local navigation in the mesh: the face-based as well as corner operators support efficient mesh exploration (see Fig. 1 and 2). A simple implementation consists stores explicitly all incidence relations involving faces or corners, using 6 references per triangle plus one reference per vertex (describing the map from vertices to faces): according to Euler formula, this leads to a storage cost of 13 *references per vertex* (*rpv*). The results of $\texttt{triangle}(c)$ and $\texttt{next}(c)$ are not stored explicitly, but calculated assuming that the three corners of each triangle are assigned consecutive indices.




```
int c = seed;
visited[vertex(next(c))] = true;
visited[vertex(previous(c))] = true;
do {
   if(!visited[vertex(c)]){
     visited[vertex(c)] = true;
     explored[triangle(c)] = true;
   }
  else if(!explored[triangle(c)]) c = opposite(c);
  c = right(c);
}while(c! = opposite(s));
```

Fig. 2. The *Corner Table*: corners operators allow to implement local navigation, as illustrated by the code of the *Ring-Expander* procedure [10].

Key Results

A Theoretically Optimal Representation

From the information theory point of view, encoding a planar triangulation requires 3.24 *bits per vertex* (*bpv*), which is much less than the $13 \log n$ *bpv* used by standard representations. *Succinct representations* provide theoretically optimal encodings for triangulations, which match the optimal asymptotic bound of 3.24 *bpv* (or equivalently $1.62m$ bits), while efficiently supporting navigational operations [4; 5], as stated below.

Theorem 1. *Given a planar triangulation $\mathcal{T}$ of m triangles, there exists a succinct representation that uses $1.62m + O\left(\frac{m \log\log m}{\log m}\right)$ bits, supporting navigation in worst case $O(1)$ time.*

This result is achieved with a multi-level hierarchical structure. The initial triangulation of size m is decomposed into *small triangulations*, each having $\Theta(\log^2 m)$ triangles: such a decomposition leads to a map $\mathcal{F}$ describing adjacency relations between small triangulations. Small triangulations are then decomposed into *tiny triangulations* of size $\Theta(\log m)$, whose adjacency relations are described by a map $\mathcal{G}$. Map $\mathcal{F}$ has $O\left(\frac{m}{\log^2 m}\right)$ nodes and arcs and can be stored in sublinear space using $O\left(\frac{m}{\log^2 m}\right)$ references of size $O(\log m)$ (actually $O\left(\log \frac{m}{\log^2 m}\right) < O(\log m)$). Map $\mathcal{G}$ has $O\left(\frac{m}{\log m}\right)$ nodes and arcs: adjacencies between two tiny triangulation within the same small triangulation need references of size $O\left(\log \frac{\log^2 m}{\log m}\right) = O(\log\log m)$ while adjacencies crossing the small triangulation boundaries are accessed by referring to $\mathcal{F}$. In that way the storage of both $\mathcal{F}$ and $\mathcal{G}$ is sublinear. The structure of tiny triangulations is optimally encoded throughout look-up into a table storing all possible triangulations of size $O(\log m)$. Such a framework can be extended in order to support updates: vertex deletions and edge flips are performed in $O(\log^2 m)$ amortized time (vertex insertions require $O(1)$ amortized time). The optimality stated by Theorem 1 is obtained combining the two levels representation with a carefull decomposition of the mesh into tiny regions, involving a bijection between triangulations and a special class of vertex spanning trees [13].

A different approach, based on small separators, leads to compact representations [1] using $O(n)$ bits for more general classes of meshes (storage performances are difficult to evaluate precisely).

A More Practical Solution

Succinct representations run under the *word-RAM model* and are mainly of theoretical interest, since the amount of memory required in practice is quite important even for very large meshes. Some attempts to exploit the algorithmic framework of succinct representations in practice had lead to a space efficient dynamic data structure [3]. The main idea is to gather together neighboring faces into small groups of triangles (called *patches*). While references are still of size $\Theta(\log n)$, grouping triangles allows to save some references (corresponding to edges internal to a given patch). For example, using a catalog consisting only of triangles and quadrangles we encode a triangulation with at most $10.6\,rpv$ (a 19% improvement over simple representations mentioned earlier). More sophisticated choices of patches lead to dynamic structures with smaller storage (e.g. Fig 1-b), as stated below:

Theorem 2. *Given a triangulation (possibly having handles and boundaries), there exists a data structure using* $7.67\,rpv$*, which allows* $O(1)$ *time navigation and supports updates in* $O(1)$ *amortized time.*

Reducing redundancy throughout face reordering

The main idea used in the SOT data structure [9] is to implicitly represent the map from triangles to corners (`triangle` operator), and the map from corners to vertices (`vertex` operator), thourough face reordering. First match each vertex to an incident triangle (in such a way a triangle is matched with at most one vertex). Then permute triangles in such a way that the triangle associate with the i-th vertex v_i has number i (thus the first n triangles appearing in this ordering are the ones associated with a vertex). The corners of a triangle are listed consecutively, and the first one corresponds to the vertex matched for the triangle. The incidence relations are stored in an array O (of length $3m$) having 3 entries per triangle: $O[i]$ stores the index of the corner opposite to c_i (which is matched to vertex v_i, for $i \leq m$). Corners operators are supported in $O(1)$ time performing arithmetic operations (see Fig. 3(a)). Accessing a vertex v_i requires to walk around its incident faces until c_i is reached (v_i being matched to c_i).

Theorem 3. *[9] Given a triangulation (possibly having handles and boundaries), there exists a data structure using* $6\,rpv$ *which supports* $O(1)$ *time navigation (retrieving a vertex of degree* d *requires* $O(d)$ *time).*

More compact (static) representations Combining this reordering approach with a pairing of adjacent triangles into quads, the SQUAD data structure [8] reaches better storage requiring slightly more than $4\,rpv$ according to experimental results on common meshes (the worst case upper bound is still $6\,rpv$). If one is allowed to perform a reordering of the input vertices, it is possible to guarantee a storage of $4\,rpv$ in the worst case (with same time performances as before): the edge-based representation described in [6] matches this bound exploiting Schnyder woods decompositions [14]. Various heuristics allows to further reduce storage requirements in practice [10; 11; 12].

A Dynamic Representation

Combining the reordering approach described above with the decomposition into triangle patches, the ESQ data structure [7] exihibits the same navigation performaces as in SOT, while supporting local updates. As in [3] the mesh is decomposed into a collection of patches, each consisting of one or more triangles, and vertices are matched

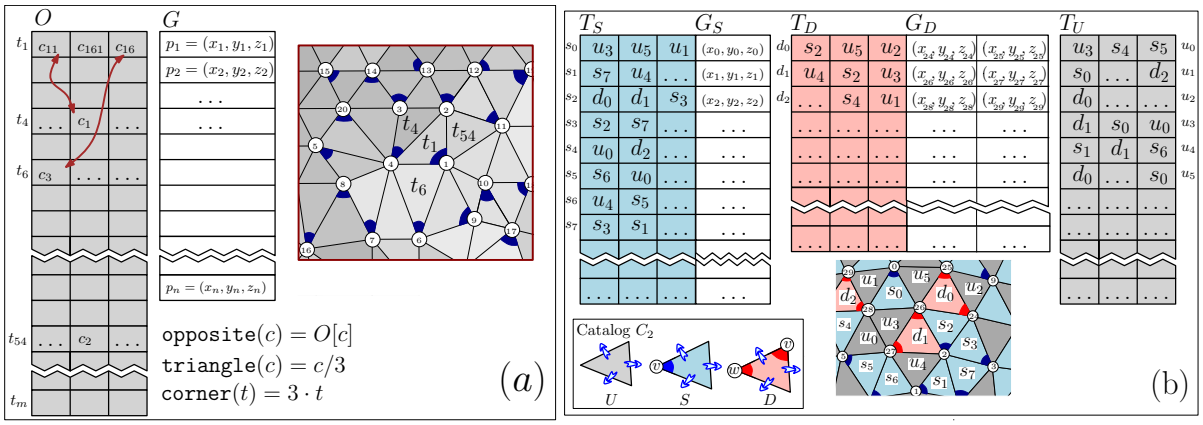


Fig. 3. Illustrations of the SOT (a) and ESQ (b) data structures.

to patches. The assumption that each vertex is matched to a different triangle is relaxed. The catalog thus consists of a collection of k patch types (having possible one or more marked corners, describing how vertices are matched). Adjacency relations between faces are stored in k tables $T_1, \ldots, T_k$, one for each patch type (see Fig. 3b). Extending the approach introduced in SOT, a reordering of the input vertices allows to represent the maps from vertices to triangles and from triangles to vertices. A table T_S of type $S = (c, b)$ (with b boundary edges, and c matched vertices) contains b references for each entry; the entries in the associated table G_S (containing geometric coordinates) are ordered accordingly. The decomposition into patches is maintained under local modifications with a constant number of memory updates in tables T_i.

Theorem 4 ([7]). *Given a triangulation (possibly having handles and boundaries), there exists a dynamic data structure using* 4.8 *rpv, which allows* $O(1)$ *time navigation and* $O(d)$ *time access to a vertex of degree* d*. Updates (vertex insertions/deletions and edge flips) are supported in* $O(1)$ *amortized time.*

Experimental Results

In [8] are reported timing comparisons of operators for SOT, SQUAD and Corner Table data structures: experimental evaluations concern adjacency and navigational operations. On the tested mesh (the 55 million triangles David) SQUAD requires 20 seconds and uses 2.2GB of RAM for the construction (on a MacbookPro, equipped with 2.66 GHz Intel Core i7, 8GB). When the whole mesh fits in main memory, compact data structures (SQUAD and SOT) performs slower than Corner Table. When the allowed memory is reduced, SQUAD performances are comparable and sometimes even better than Corner Table performances for high-level tasks (e.g. valence computations).

Cross-References

- *Compressed representations of graphs* (I. Munro, area editor: G. Navarro)
- *Delaunay triangulation and randomized constructions* (O. Devillers, area editor: Haverkort)

Recommended Reading

1. D. Blanford, G. Blelloch, and I. Kash. Compact representations of separable graphs. In *SODA*, 342–351, 2003. url

2. J.-D. Boissonnat, O. Devillers, S. Pion, M. Teillaud, and M. Yvinec. Triangulations in CGAL. *Comp. Geometry*, 22:5–19, 2002. doi
3. L. Castelli Aleardi, O. Devillers, and A. Mebarki. Catalog Based Representation of 2D triangulations. In *Internat. J. Comput. Geom. Appl.*, 21(4): 393-402, 2011. doi
4. L. Castelli Aleardi, O. Devillers, and G. Schaeffer. Succinct representation of triangulations with a boundary. In *WADS*, 134–145, 2005. doi
5. L. Castelli Aleardi, O. Devillers, and G. Schaeffer. Succinct representations of planar maps. *Theor. Comput. Sci.*, 408(2-3):174–187, 2008. doi
6. L. Castelli Aleardi and O. Devillers. Explicit array-based compact data structures for triangulations. *ISAAC*, 312–322, 2011. doi
7. L. Castelli Aleardi, O. Devillers and J. Rossignac. ESQ: Editable SQuad Representation for Triangle Meshes. *SIBGRAPI*, 110–117, 2012. doi
8. T. Gurung, D. Laney, P. Lindstrom, and J. Rossignac. *SQUAD*: Compact representation for triangle meshes. In *Comput. Graph. Forum*, 30(2):355-364, 2011. doi
9. T. Gurung and J. Rossignac. *SOT*: compact representation for tetrahedral meshes. In *Proc. of the ACM Symp. on Solid and Physical Modeling*, 79–88, 2009. doi
10. T. Gurung, M. Luffel, P. Lindstrom, and J. Rossignac. *LR*: compact connectivity representation for triangle meshes. In *ACM Trans. Graph.*, 30(4):67, 2011. doi
11. T. Gurung, M. Luffel, P. Lindstrom, and J. Rossignac. *Zipper*: A compact connectivity data structure for triangle meshes. In *Computer-Aided Design*, 45(2):262-269, 2013. doi
12. M. Luffel, T. Gurung, P. Lindstrom, and J. Rossignac. *Grouper*: A Compact, Streamable Triangle Mesh Data Structure. In *IEEE Trans. Vis. Comput. Graph.*, 20(1):84-98, 2014. doi
13. D. Poulalhon and G. Schaeffer. Optimal coding and sampling of triangulations. *Algorithmica*, 46:505–527, 2006. doi
14. W. Schnyder. Embedding planar graphs on the grid. In *SODA*, 138–148, 1990. url