


Topological Data Structure for Computer Graphics

G. Fábán¹ 

¹ ELTE Eötvös Loránd University, Budapest, Hungary

Abstract

This research is motivated by the following well-known contradiction. In computer-aided design or modeling tasks, we generally represent surfaces using edge-based data structures as winged edge [Bau75], half-edge [MP78] [CP98], or quad-edge [GS85]. In contrast, real-time computer graphics represents surfaces with face-vertex meshes, since for surface rendering, there is no need for the explicit representation of edges. In this research we introduce a novel data structure for representation of triangle meshes. Our representation is based on the concept of face-vertex meshes with adjacencies, but we use some extra information and new ideas that greatly simplify the implementation of algorithms.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—

1. Introduction

In this research, we design a data structure for representing polyhedra that allows the efficient execution of global operations. In most cases, when mainly local modifications are used (e.g. vertex split, edge flip, face removal), traditional winged edge and half-edge data structures perform well. However, for global operations (affecting large number of vertices, edges, faces), the advantages of edge-based data structures seem to diminish. In this research we will show a novel data structure for representation of triangle meshes. In our experiments we used the industry standard half-edge data structure, and we implemented our proposed data structure. We have done several tests to measure time cost of some local and global operations. We compared the performance of the data structures for some complex operations as subdivision. Our results confirmed, that many local and global operations can be easily implemented and efficiently performed without explicit representation of edges. Moreover, our surface representation stores less data than the half-edge data structure. We refer to our representation as SolidMesh, emphasizing that it is suitable only for storing triangulation of surfaces of solid geometries.

2. SolidMesh data structure

When designing our data structure, we formulated the following requirements.

1. The representation should be based on the vertex and index arrays used by the GPU.
2. Edges should not be explicitly represented.
3. A fixed amount of data should be stored for faces and vertices.
4. Global operations should be performed quickly.

Condition 1. and 2. imply, that the central elements of our data structure are necessarily faces. Condition 3. can not be fulfilled, unless each face has a same number of vertices, therefore we choose triangular faces. Efficient execution of local operations occurring during modeling tasks was not a crucial consideration. We would like to prepare the data structure for global operations where the entire vertex and index arrays need to be traversed (detach, cut, split, smooth, subdivide, etc.).

The half-edge data structure allows efficient execution of local modifications, geometric information of a neighborhood of an edge is encapsulated into half-edges. In the implementation of our data structure, we did not create a new face class, which would achieve similar encapsulation. Instead, we added some extra (one- and multi-dimensional) arrays containing all the necessary geometric information to the vertex and index arrays, similarly to the render dynamic meshes [TM06], see Figure 1 for an example.

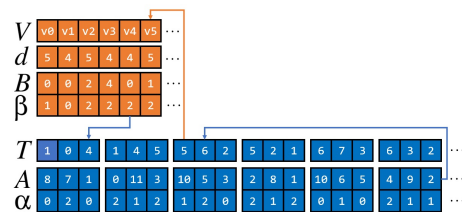


Figure 1: Example for SolidMesh data structure, a part of the representation of a cube model.

Maybe the simplest approach is to define the SolidMesh data structure using functions with finite domains. Consider a polyhedron with n vertices and m faces, and let us suppose, that

© 2024 The Authors.
 Proceedings published by Eurographics - The European Association for Computer Graphics.
 This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.



$I = \{0, \dots, n-1\}$, $J = \{0, \dots, m-1\}$ and $\tau = \{0, 1, 2\}$. Then a common representation of a mesh in CG is a (V, T) pair, where the V vertex-array and the T index-array can be defined by the following functions: $V : I \rightarrow \mathbb{R}^3$ and $T : J \times \tau \rightarrow I$. The I, J sets refer to the indices of the vertices and triangles, the τ set is responsible for storing the order of vertices within a triangle, i.e. $T(i, k) = j$ means: the k -th vertex of the i -th triangle is $V(j)$. In SolidMesh data structure a vertex stores its degree ($d : I \rightarrow \mathbb{N}$), and a reference for itself in an arbitrary triangle ($B : I \rightarrow J, \beta : I \rightarrow \tau$). Edges are only implicitly represented, any two circularly consecutive vertex of a triangle define an edge. Using this convention, a face stores references to its edge-adjacent triangles and endpoints of its own edges ($A : J \times \tau \rightarrow J, \alpha : J \times \tau \rightarrow \tau$). A brief overview of this concept can be seen in Figure 2.

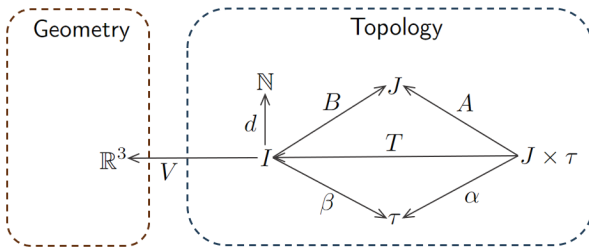


Figure 2: Visualization of the of $V, T, A, \alpha, B, \beta, d$ functions with their domain and range sets.

3. Results

In our experiments we used an efficient half-edge data structure [MP78], and we implemented our proposed data structure in C# language using Unity game engine. We have done several tests to measure time cost of some local and global operations. We compared the performance of the data structures for some complex operations as subdivision. We give a representative example, the application of the Loop subdivision scheme [Loo87]. Loop subdivision is a global operation working on polyhedrons defined by triangular faces. The subdivision operation is not trivial; we need to break down each face of the polyhedron and compute the coordinates of each vertex (new and old). By calculation of vertex positions the edge-adjacent triangle pairs play an important role, therefore the half-edge data structure is often chosen for implementing this subdivision scheme.

Model name	Number of vertices	Subdivision time [ms]	
		Half-edge	SolidMesh
Cube	8	0.10	0.02
Sphere	482	5.63	0.79
Torusknot	880	11.36	1.46
2-tori	1156	14.37	2.05
Bunny	2503	34.78	4.81
Ducky	5084	60.87	8.83
Mug	6390	71.99	11.12
Armadillo	15002	370.69	31.74

Table 1: Execution times for Loop subdivision.

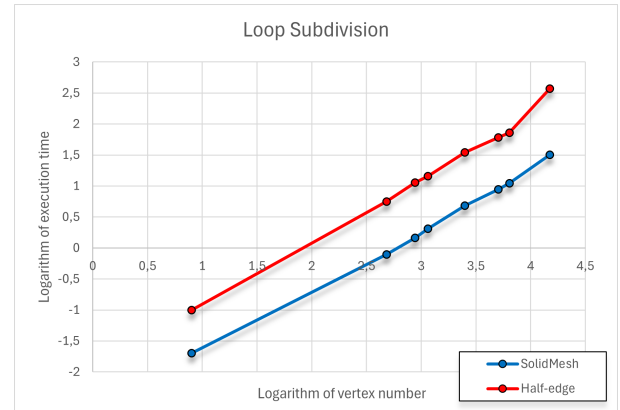


Figure 3: Execution times for Loop subdivision (log-log plot).

Our results seem to support that, despite the lack of explicit edge representation in our data structure, complex operations can be executed much faster with it. According to our measurements, Loop subdivision implemented in SolidMesh data structure ran approximately 10 times faster than the half-edge implementation, see Table 1 and Figure 3. Similar promising results have been obtained for algorithms such as mesh smoothing and mesh slicing as well. Although the computational complexity of these algorithms is linear in the number of faces, it can be proven in many cases that the SolidMesh implementation utilizes significantly fewer operations compared to other topological data structures.

Finally, let us discuss briefly storage requirements. The half-edge data structure is often criticized for storing topological information with high overhead. We examined the storage requirements of data structures for low-genus polyhedra with n vertices. We found that while the half-edge structure requires $30n$ floating point or integer numbers to store the complete topological and geometric information, SolidMesh requires only $24n$.

References

- [Bau75] BAUMGART B. G.: A polyhedron representation for computer vision. In *National Computer Conference and Exposition (AFIPS '75)* (1975), pp. 589–596. doi:10.1145/1499949.1500071. 1
- [CP98] CAMPAGNA S., PREPARATA F. P.: Directed edges – a scalable representation for triangle meshes. *Journal of Graphics, GPU & Game Tools* 3 (1998), 1–11. doi:https://doi.org/10.1080/10867651.1998.10487494. 1
- [GS85] GUIBAS L., STOLFI J.: Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics* 4, 2 (1985), 74–123. doi:10.1145/282918.282923. 1
- [Loo87] LOOP C. T.: *Smooth Subdivision Surfaces based on Triangles*. PhD thesis, University of Utah, 1987. 2
- [MP78] MÜLLER D. E., PREPARATA F. P.: Finding the intersection of two convex polyhedra. *Theoretical Computer Science* 7, 2 (1978), 217–236. doi:10.1016/0304-3975(78)90051-8. 1, 2
- [TM06] TOBLER R. F., MAIERHOFER S.: A mesh data structure for rendering and subdivision. In *Proceedings of the January 30 - February 3, 2006, Winter School of Computer Graphics* (Plzen, Czech Republic, 2006), WSCG '2006. 1