Title:

With or Without edges? Data Structure for Global Mesh Operations

Authors:

Gábor Fábián, insomnia@inf.elte.hu, ELTE Eötvös Loránd University, Budapest, Hungary

Keywords:

triangle mesh, mesh processing, topology, subdivision, half-edge

DOI: 10.14733/cadconfP.2024.174-178

Introduction:

In this paper, we design a data structure for representing polyhedra that allows the efficient execution of global operations. Our research is motivated by the following contradiction. In computer-aided design or modeling tasks, we generally represent surfaces using edge-based data structures as Winged edge [1], Half-edge [5], or Quad-edge [2]. 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 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, that is based on the concept of face-vertex meshes. We will discuss, what additional topological information need to be stored to obtain fast traversal algorithms.

In our experiments we used an efficient Half-edge implementation, and we implemented our proposed data structure. We have done several tests to measure time cost of some global operations. We compared the performance of the data structures for some complex manipulations as subdivision. Our results confirmed, that many 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.

Half-edge data structure

The Half-edge data structure is suitable for storing orientable 2-manifolds defined by polygonal faces [6]. As we will soon see, this representation effectively takes advantage of these properties. Let us assume that each edge is shared by exactly two faces (which means, the mesh is 2-manifold), and the orientation of the common edge is opposite on these two faces (which means our mesh is an orientable 2-manifold). See e.g. [3] for the detailed explanation of these topological properties. In this representation each edge is "split in two" obtaining the so-called half-edges, the central elements of the data structure. For each half-edge, we store the starting vertex (vertex), the associated face (face), its oppositely oriented pair (twin), and the next half-edge (next), as you can see on Fig. 1.

Faces and vertices store only a reference to a corresponding half-edge to which they are connected.

SolidMesh data structure

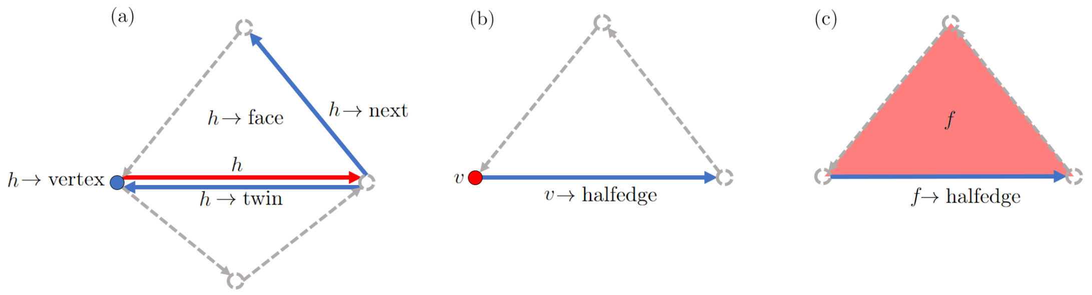

Fig. 1: The local neighborhood of (a) an h Half-edge; (b) a v vertex; (c) an f face in Half-edge data structure.

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

1. The representation should 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. Since every polygonal face can be decomposed into triangles, we choose triangular faces.

In the Half-edge data structure, 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. The following section describes how our data structure is created, which is suitable for storing and manipulating orientable compact 2-manifolds defined by triangular faces. It is important to note that the surface of a solid geometry is orientable compact 2-manifold, and conversely, each non-self-intersecting connected orientable compact 2-manifold defines a solid. Our data structure is designed specifically for storing and manipulating the surfaces of such solid geometries, where we extensively utilize these topological properties. Therefore, we will refer to this representation as SolidMesh in the following.

Maybe the simplest approach is to define the SolidMesh data structure using functions with finite domains. Let us suppose, that $I = \{0, \ldots, n-1\}$, $J = \{0, \ldots, m-1\}$ and $\tau = \{0, 1, 2\}$. Then a the common representation of a mesh in computer graphics is a (V, T) pair, where the V vertex-array and the T index-array can be defined by the following functions: $V : I \to \mathbb{R}^3$ and $T : J \times \tau \to I$. 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. τ is similar to the ring of integers modulo 3, we defined the addition for any $k \in \mathbb{Z}$ as

$$\forall j \in \tau \quad \forall k \in \mathbb{Z} \ : \ j \oplus k := (j + k) \mod 3.$$

We found, if we define our data structure with the following functions, we achieve a similarly strong topological descriptive capability as in the case of the Half-edge structure.

- $V : I \to \mathbb{R}^3$: the vertex coordinates. $V(i) \in \mathbb{R}^3$ defines the position of the i-th vertex.

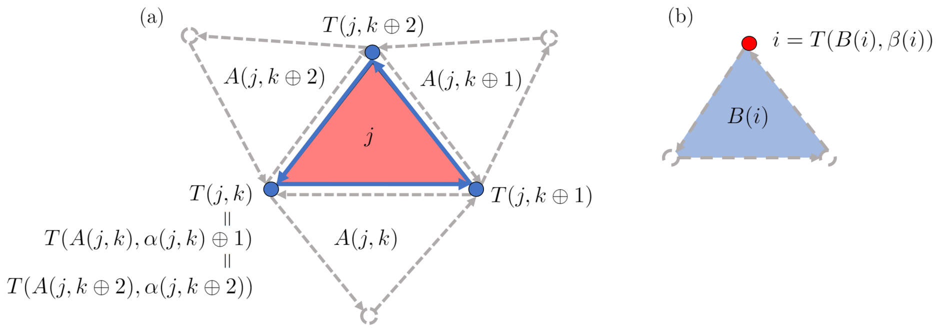

Fig. 2: The local neighborhood of (a) the j-th triangle; (b) the i-th vertex in SolidMesh data structure.

- $T : J \times \tau \to I^3$: the index triplets of the faces. Let us suppose that the j-th triangle of our surface is spanned by the $V(p), V(q), V(r)$ vertices $(p, q, r \in I)$. This fact can be formulated as follows:

$$T(j,0) = p \;\wedge\; T(j,1) = q \;\wedge\; T(j,2) = r.$$

  The edges determined implicitly by T. We will assume that the k-th edge of the j-th triangle is a directed edge from $V(T(j,k))$ to $V(T(j,k\oplus 1))$ for $k = 0,1,2$.

- $A : J \times \tau \to J$: the triangle adjacents of the triangles. $A(j,k) = p$ if and only if the j-th triangle is adjacent to the p-th triangle, and their shared edge is the k-th edge of the j-th triangle.

- $\alpha : J \times \tau \to \tau$: the edge index of the adjacent triangle. $\alpha(j,k) = q$ if and only if the shared edge between triangles j and $A(j,k)$ is the q-th edge of the adjacent ($A(j,k)$-th) triangle. By the definition of T, A, and α, we get

$$T(j,k) = T(A(j,k),\ \alpha(j,k)\oplus 1), \quad \text{and} \quad T(j,k\oplus 1) = T(A(j,k),\alpha(j,k)).$$

- $B : I \to J$: face index for a vertex. $B(i) = j$ means that the $V(i)$ vertex is a vertex of the j-th triangle.

- $\beta : I \to \tau$ vertex index for a vertex. $\beta(i) = q$ means that the $V(i)$ vertex is the q-th vertex of the $B(i)$-th triangle, i.e.

$$T(B(i),\beta(i)) = i.$$

- $d : I \to \mathbb{N}$: degree of vertices. The $V(i)$ vertex has exactly $d(i)$ vertex neighbours.

Formally, our mesh representation is a $(V, T, A, \alpha, B, \beta, d)$ tuple. Since (V, T) is the common representation of a surface in computer graphics, V and T can be considered as the vertex-array and the index-array, we pass to the GPU for rendering.

Loop subdivision

Due to content constraints, we cannot present the results of all our measurements, nor the details of individual algorithm implementations; these will be included in the full paper. Now we give a representative example, the application of the Loop subdivision scheme [4]. Loop subdivision is a global operation defined for a polyhedron defined by triangular faces. In each step of the subdivision algorithm, the following operations are performed.

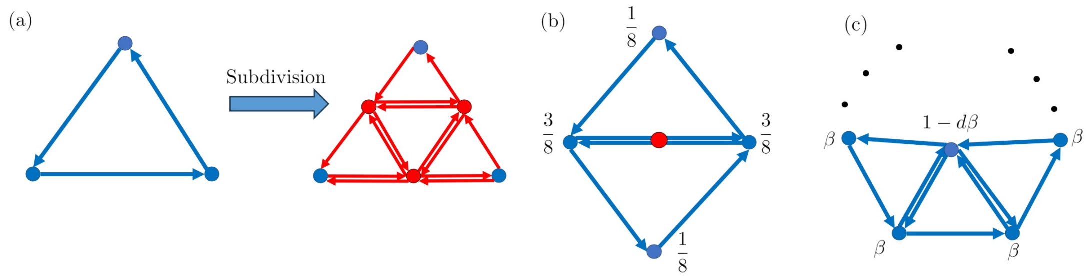

Fig. 3: Loop subdivision

1. Create a new vertex on each edge.
2. Connect the new vertices to split each triangle into 4 smaller triangles (see Fig. 3 (a)).
3. Calculate the positions of the new vertices as the barycentric combination of the two vertices spanning the edge and the third vertices of the triangles that are sharing the edge (see Fig. 3 (b)).
4. Recalculate the positions of the old vertices as the barycentric combination of the adjacent old vertices (see Fig. 3 (c)).

As we can see, the subdivision operation is not trivial; we need to break down every face of the polyhedron and compute the coordinates of every 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. The operation of the algorithm illustrated in Fig. 3.

Results

Our results seem to support that, despite the lack of explicit edge representation in our data structure, subdivision can be executed much faster with it. According to our measurements, the Loop subdivision implemented in SolidMesh data structure ran approximately 5-10 times faster than the Half-edge implementation, as you can see in Fig. 4 and in Table 1.

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: Time costs of Loop subdivision

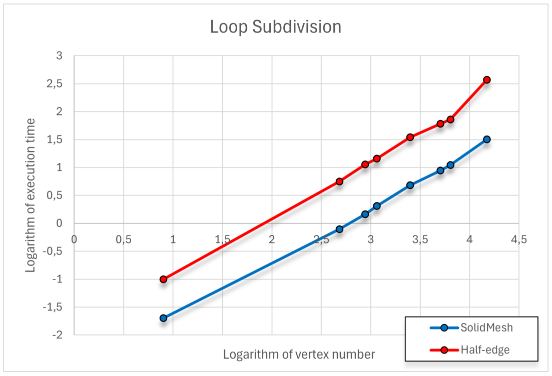

Fig. 4: Results visualized on a log-log graph

This extended abstract contains only one interesting result; the implementation details and further measurement results will be available in the full paper. Similar results were obtained for other global operations: many important operations can be performed significantly faster in SolidMesh data structure, and efficient implementation of local operations is also possible. Moreover, this data structure has a smaller storage requirement than the Half-edge.

Acknowledgement:

Supported by the ÚNKP-23-4 New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund.

Új Nemzeti Kiválóság Program MINISTRY OF CULTURE AND INNOVATION

Gábor Fábián, https://orcid.org/0000-0003-0255-5379

References:

[1] Baumgart, B. G.: A polyhedron representation for computer vision. In Proceedings of the May 19-22, 1975, National Computer Conference and Exposition (AFIPS '75). Association for Computing Machinery, New York, USA, 1975, 589–596. https://doi.org/10.1145/1499949.1500071

[2] 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. https://doi.org/10.1145/282918.282923

[3] Lee, J. M.: Introduction to Topological Manifolds, Graduate Texts in Mathematics, Springer, 2000.

[4] Loop, C. T.: Smooth Subdivision Surfaces based on Triangles, M.S. Mathematics thesis, University of Utah, USA, 1987. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/thesis-10.pdf

[5] Müller, D. E.; Preparata, F. P.: Finding the intersection of two convex polyhedra, Theoretical Computer Science, 7(2), 1978, 217–236. https://doi.org/10.1016/0304-3975(78)90051-8

[6] Mäntylä, M.: An Introduction to Solid Modeling, Principles of computer science series, Springer, Computer Science Press, 1988.