Title:
A Compact Face-Based Topological Data Structure for Triangle Mesh Representation

Authors:
Yingzhong Zhang, zhangyz@dlut.edu.cn, Dalian University of Technology
Xiaofang Luo, lxf@dlut.edu.cn, Dalian University of Technology
Jia Jia, 49978527@qq.com, Dalian University of Technology

Keywords:
Triangle Mesh, Topological Data Structure, Mesh Representation, STL, Face-based Data Structure

DOI: 10.14733/cadconfP.2018.367-371

Introduction:

As triangle meshes have many unique characteristics, they are widely used to represent the surfaces of the geometric objects in computer graphics, CAD applications, 3D printing, and computational science [2]. In practical applications, a mesh model needs to carry out a lot of geometric information processing, such as Boolean operations, mesh segmentation, and feature reconstruction, which requires a lot of topological information among the mesh entities [2],[5]. A complete topological representation for triangle meshes is the foundation to meet above application requirements. On the other hand, as a surface is usually represented with an assembly of tiny triangle facets, in a complex mesh model, the number of triangular facets may reach millions. As a result, the ever-increasing size and complexity of meshes impose stress on both memory usage and processing time. The storage space, querying efficiency, and implementation complexity are important criteria for evaluating a mesh representation model.

The topological data structure has been researched intensively due to its importance. A wide variety of data structures for triangle meshes have been proposed. The half-edge data structure (HEDS) [7] is the most popular data structure and has been constantly improved [1-4],[8], which can succinctly describe the topological connection relations among geometric entities of triangle meshes and implement traversal search with high efficiency without any conditions. Typical implementations of HEDS include Open-Mesh, Surface_Mesh [2],[8], Directed Edge representation [3], Compact Array-Based data structures [1],[4], etc. Most HEDS employ arrays to store geometric and topological data in order to reduce memory footprints. However, using array structures many object references depend on the array index, which affects some computational performances. Hence, how to achieve a balance between memory requirements, performances and efficiency, and uniformly to represent and process manifold meshes and non-manifold meshes have been the challenges to be confronted [4].

This paper aims at above challenges, focuses on the STL mesh applications in reverse engineering, and presents a novel compact face-based geometric and topological representation for triangle meshes. The presented model makes full use of the implicit semantic relations among vertices, edges, and triangle faces, which results in memory consumption decrease and improves the flexibility, scalability, and usability of the mesh model.

Main Idea:

Vertex neighborhoods in triangle meshes

A vertex neighborhood of a vertex v is the set of vertices directly connected to v via an edge and v itself. In the mesh information processing, traversing the ring neighborhood of a vertex is an extremely important operation, which passes through the all incident faces attached to a given vertex and can obtain the set of the incident vertices connected to the vertex. The performing efficiency of this traversal operation is usually considered to be an important indicator to evaluate a topology data structure. Hence, at first we observe the topological dependence among the topological entities, i.e. vertices, edges and faces, by performing a 1-ring neighborhood traversal as shown in Fig. 1.

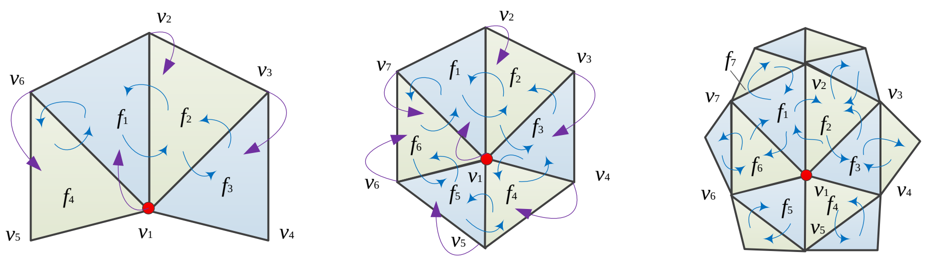

Fig. 1: Ring neighborhoods of a vertex: (a) An open 1-ring neighborhood, (b) A closed 1-ring neighborhood, and (c) More than a 1-ring neighborhood.

It can be seen from Fig. 1 that we can set an incidence relation between a vertex object and a face object. Specifically, a vertex object contains a reference (or pointer) to a face object where it locates. As a result, from a vertex, its incident face can be visited. At the same time, in a face object, except its three vertices, three incidence relations to its mate faces are also set. A triangle face has three adjacent faces at most. The adjacent faces of a face are defined as its mate faces each other. For example, in Fig. 1(c), the mate faces of the face f_1 are f_2, f_6, and f_7.

After setting the above incidence relations, we can easily implement a query of 1-ring neighborhoods for a given vertex. As shown in Fig. 1(b), from the given vertex v_1, f_1 can be obtained at first; and from f_1, v_2 and v_7 can be obtained. From the order of vertex v_7, a mate face of f_1, namely f_6, can be obtained. Performing the similar loop program, all vertices on the 1-ring neighborhood of vertex v_1 can be obtained. The performing operation is in constant time and independent on the size of meshes.

An object-oriented face-based topological representation model

Based on the above analysis and geometric characteristics of triangle meshes, we can abstract following topological object classes according to object-oriented methods.

- *CVertex* class

In triangle meshes, vertices are 0-cells of a mesh complex, which decide the position, shape, and size of a mesh. At the same time, a vertex is one of vertices of an edge, so it is incident to an edge. The vertex is a fundamental geomantic entity in a polygon mesh. Hence, we abstract a *CVertex* class to represent vertex objects.

Apart from the position coordinates of a vertex, we set a pointer to a triangle facet object where the vertex locates as an attribute of the *CVertex* object. Through the pointer, the incident relationship between the vertex and its incident faces can be set up.

- *CFace* class

A face is also a fundamental topological entity in the boundary representation of geometric models and OpenGL rendering. In triangle meshes, we employ the face entity to represent the 2-cells of a mesh complex. A face is a convex triangle surrounded by three edges which are sequentially connected by three vertices. As the characteristics of convex polygons, the boundary of a face can be defined implicitly by its vertices. The counterclockwise sequential connection of vertices of a face forms its triangle boundary and its normal vector. Hence, the edges of a face entity don't need to be explicitly defined, which can reduce the memory footprint.

- *CWireEdge class*

In general, the wire-edge is a degenerate triangle face as resulting from some cases and appears in non-manifold meshes. In this paper, we define the *CWireEdge* class as the subclass of the *CFace* class to represent wire-edges, which inherits all geometric attributes of the *CFace* class.

In fact, in a wire-edge object, only two vertices are used. When a wire-edge object needs to be represented, we can represent it as an instance of the *CWireEdge* class. As the *CWireEdge* class is the subclass of the *CFace* class, the object instance of the wire-edge can be stored in the face container and can be uniformly represented and processed.

- *CMesh* class

A mesh is a finite collection of 2-dimensional cells, or face entities. We employ the *CMesh* class to represent a whole mesh or a local sub-mesh of a three-dimensional geometric object. According to above analysis, only the face and vertex are needed to be explicitly defined in triangle meshes.

On the above analysis and class definition, a novel object-oriented face-based topological representation model for triangle meshes is presented as shown in Fig. 2.

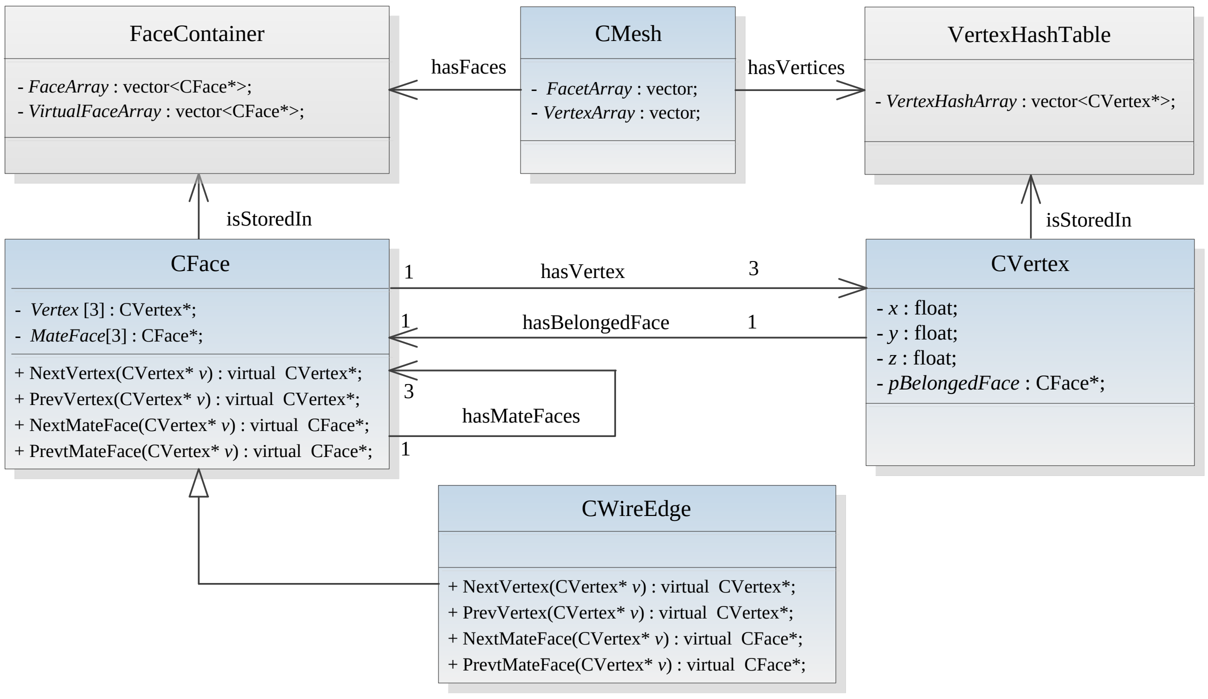
FaceContainer
- FaceArray : vector<CFace*>;
- VirtualFaceArray : vector<CFace*>;
CMesh
- FacetArray : vector;
- VertexArray : vector;
VertexHashTable
- VertexHashArray : vector<CVertex*>;
hasFaces
hasVertices
isStoredIn
isStoredIn
CFace
- Vertex [3] : CVertex*;
- MateFace[3] : CFace*;
+ NextVertex(CVertex* v) : virtual CVertex*;
+ PrevVertex(CVertex* v) : virtual CVertex*;
+ NextMateFace(CVertex* v) : virtual CFace*;
+ PrevtMateFace(CVertex* v) : virtual CFace*;
1
hasVertex
3
1
hasBelongedFace
1
3
hasMateFaces
1
CVertex
- x : float;
- y : float;
- z : float;
- pBelongedFace : CFace*;
CWireEdge
+ NextVertex(CVertex* v) : virtual CVertex*;
+ PrevVertex(CVertex* v) : virtual CVertex*;
+ NextMateFace(CVertex* v) : virtual CFace*;
+ PrevtMateFace(CVertex* v) : virtual CFace*;

Fig. 2: Class diagram of the presented model.

In the presented model, a mesh object sets up two containers to store vertex objects and face objects. Each face object contains three pointers to its three boundary vertex objects, and three pointers to its three mate face objects. Each vertex object contains a pointer to its incident face. As a result, to represent topological information, the data structure will occupy $4\times(3+3) = 24$ bytes for a face, and $4 \times 1 = 4$ bytes for a vertex.

In a manifold triangular mesh, according to Euler relation among vertices, edges and faces, the number of the triangular facet is approximately equal to two times the number of vertices, namely $F = 2V$; To represent topological relations of a mesh with n vertices, this data structure only requires $4n + 2\times 24n = 52n$ bytes memory spaces, which demonstrates that this data structure is compact. Detailed comparison with other data structures will be provided in the later section.

Topological relation representation

The topological relations between geometric entities can mainly be categorized into incidence relations and adjacency relations. The following will analyze the capability of this data structure to represent the two relations:

- Incidence relation representation

According to the incidence relation definition, two simplices are incident to each other if one of them is a part of the other. Typical incidence relations include the vertex-edge (or edge-vertex) relation, the vertex-face (or face-vertex) relation, and the edge-face (or face-edge) relation.

In this representation, edges are implicitly represented, namely, an edge is represented by two sequential vertices on the same face. Hence, given a vertex, we can get edges that are incident the vertex. A face contains three vertices, but in this model, a vertex is only to set to an incident face pointed by *pBlongedFace* value.

- Adjacency relations representation

In the same face, if a vertex is given, to get its adjacent vertices it is required to find the array index of the given vertex. Suppose the array index of a vertex as *i*, the array index of the next adjacent vertex is (*i*+1)%3, and the index of the previous adjacent vertex is (*i*+2)%3. The symbol "%" is the division remainder.

In this representation, the adjacent relation between two triangles can be obtained by their mate face pointers which are set in each face object. A face shares an edge with one of its mate faces. The storing order of mate faces is same as the order of vertices. As a result, from a vertex in a face, the corresponding mate faces of the face can be easily obtained.

Construction of mesh models from STL files

It is also very important for a topological representation model to be constructed effectively and efficiently. The following will simply describe how to construct a mesh model with the data structure as mentioned above from STL files.

In reverse engineering, STL is an exchange file format for triangular mesh data [9]. Currently, STL has become a standard for data input of all types of rapid prototyping systems and has been widely used in many fields. However, A STL file just lists the geometry information of each single triangle, resulting in a large amount of redundant data, and lacks topological information between the geometric entities. Hence, we construct the topological representation of a STL mesh according to the following procedures.

- Construction of vertex objects

The construction of vertex objects needs to finish two tasks: creating a geometric vertex and setting an incidence relation with it's an incident face object. The setting operation is usually performed after completing the incident face object.

A geometric vertex can be easily created by its geometric coordinates provided in STL files. The key issue is to remove the vertex repetition. The efficiency of searching vertices is very important for a complex mesh. Therefore, many methods to improve the efficiency of searching vertices were proposed in the past, such as the balanced binary tree, octree, and hash table methods [6]. In this paper, we employ a hash table method to search a vertex. In addition, we also make full use of the adjacency relations between vertices to speed up the search, such as querying 1-ring neighbor vertices.

- Construction of face objects

From the representation model presented above, it is evident that identifying the adjacent mate face is crucial for constructing a face object. We take the following steps: (1) In the process of constructing face objects, at first, the mate face pointers in each newly created face object are set as NULL. (2) When a new face object is constructed each time, check its face adjacent relations along its vertex order by counterclockwise. If there is an adjacent relation between two faces, their corresponding mate face pointers are set to point each other. (3) Check borders of non-manifold meshes. If there are NULL pointers in a face object, the corresponding edges of the face object are mesh borders. It is necessary to set its mate faces properly with some mapping rules.

Comparison and discussion:

Compared with other data structures, the presented topological data structure has the following characteristics:

- Much more flexibilities and extensibilities

Object-oriented representation can provide much more flexibilities, extensibilities, and usability for various mesh applications than the array-based or corner-based data structure. In the presented model, each mesh face or vertex is described as an instance of the respective geometric entity class. According to application requirements, users can simply derive their subclasses to create the user-defined mesh object, which makes it easy to implement hybrid meshes consisting of different types of mesh faces. As a result, various attributes, such as texture or color attributes, can be easily appended to the face object. In addition, the mesh instance can be flexibly accessed.

- Much more compact and succinct

The famous half-edge data structure needs more memory footprints. In commonly used half-edge data structure, such as the Surface_mesh data structure, a half-edge needs 20 bytes; a vertex needs a half-edge; an edge needs two half-edge; and a face needs a half-edge [2],[8]. As a result, for a triangle mesh with n vertices, the total memory consumption is: $20n + 20\times2\times3n + 20\times2n = 180n$ bytes. The array-based compact data structure [1],[4] can reduce the memory footprint of a half-edge to 8 bytes; but it also consumes memory at least 80 bytes/vertex. If a half-edge is encoded to an unsigned long integer, the memory footprint will be reduced to 64 bytes/vertex.

- Good time performance

In the presented model, all topological queries are independent of the mesh size. The operation of querying 1-ring neighborhoods mentioned above can demonstrate the efficiency of accessing mesh entities. In addition, theoretically, the number of variables in the model can determine the reconstructing efficiency largely. In the presented model, there are only four variables for topological relations. Compared with the compact array-based data structure, the presented model doesn't need to perform operations of encoding and decoding half-edges. Encoding and decoding operations will take a few moments.

- Uniformly representing and processing manifold meshes and non-manifold meshes.

Currently, most of existing mesh data structures are incapable of representing and processing non-manifold meshes well. By defining the C*WireEdge* class and a few mapping rules for mate faces, the presented model can uniformly represent and process mesh boundary and non-manifold meshes.

Conclusions:

A large number of triangle mesh applications need a compact topological representation that can uniformly represent manifold meshes and non-manifold meshes with less memory consumption and more flexibilities and extensibilities. In this paper, a novel compact face-based topological representation for triangle meshes is presented. The presented model makes full use of the semantic relations and implicit information among faces, edges, and vertices. Compared with the general edge-based data structure, the memory footprint for topology information is reduced. Compared with the array-based compact data structure, the present model has much more flexibility and extensibility. In addition, the presented model can uniformly represent and process hybrid meshes, manifold meshes, mesh boundaries, and non-manifold meshes. Furthermore, a reconstruction experiment for STL format data has been carried out, which verifies that the presented model is feasible and effective.

Acknowledgements:

This work is supported by the National Science Foundation of China (Grant No. 51775081 and 51375069). The authors thank the anonymous reviewers for their helpful suggestions on this study.

References:

[1] Alumbaugh, T. J.; Jiao, X.: Compact array-based mesh data structures, Proceedings of the 14th International Meshing Roundtable, Berlin Heidelberg: Springer, 2005.

[2] Botsch, M.; Kobbelt, L.; Pauly, M.; Alliez, P.; Lévy, B.: Polygon mesh processing, A K Peters, Natick, Massachusetts, 2010.

[3] Campagna, S.; Kobbelt, L.; Seidel, H.-P.: Directed edges—A scalable representation for triangle meshes, Journal of Graphics Tools, 3(4), 1998, 1-12. https://doi.org/10.1080/10867651.1998.10487494

[4] Dyedov, V.; Ray. N.; Einstein, D.; Jiao, X.; Tautges, T. J.: AHF: Array-based half-facet data structure for mixed-dimensional and non-manifold, Engineering with Computers, 31(3), 2015, 389-404. https://doi.org/ 10.1007/s00366-014-0378-6

[5] Feito, F. R.; Ogayar, C. J.; Segura, R. J.; Rivero, M. L.: Fast and accurate evaluation of regularized Boolean operations on triangulated solids, Computer-Aided Design, 45(3), 2013, 705-716. https://doi.org/10.1016/j.cad.2012.11.004

[6] Hrádek, J.; Kuchar, M.; Skala, V.: Hash functions and triangular mesh reconstruction, Computers & Geosciences, 29(6), 2003, 741-751. https://doi.org/10.1016/S0098-3004(03)00037-2

[7] Mäntylä, M.: An introduction to solid modeling, Computer Science Press, 1988.

[8] Sieger, D.; Botsch, M.: Design, implementation, and evaluation of the surface mesh data structure, Proceedings of the 20th International Meshing Roundtable, Berlin Heidelberg: Springer, 2011.

[9] Várady, T.; Martin, R. R; Cox, J.: Reverse engineering of geometric models-an introduction, Computer-Aided Design, 29(4), 1997, 255-68. https://doi.org/10.1016/S0010-4485(96)00054-1