Istituto di Matematica Applicata e Tecnologie Informatiche

Consiglio Nazionale delle Ricerche

Genova - ITALY

JMeshLib

Version 1.0

A C++ API to manage manifold triangle meshes

MARCO ATTENE

QUICK INTRODUCTION

Index

1. Hello JMeshLib

This piece of code is an example of use of the API.

```
1 ) #include "jmesh.h"
2 )
3 ) int main(int argc, char *argv[])
4 ) {
5 )  JMesh::init();
6 )  Triangulation tin;
7 )
8 )  if (argc < 3)
9 )   JMesh::error("\nUsage: %s infile.wrl outfile.wrl\n",argv[0]);
10)
11)  if (tin.load(argv[1]) != 0) JMesh::error("Can't open file.\n");
12)  tin.saveVRML1(argv[2]);
13)
14)  return 0;
15) }
```

Code Walk-through

Line 1: Include all JmeshLib’s class definitions, static vars, constants, and so on. Always needed.

Line 5: Init JMeshLib's internal state machine. Always needed.

Line 6: Create an instance of a triangle mesh. Initially empty.

Line 9: Example of use of JMeshLib’s internal error reporting system.

Line 11: The method “load” of the class Triangulation is a wrapper that calls the proper loader depending on the format of the file. See section 4.

Line 12: The method “saveVRML1” writes the triangle mesh to a VRML 1.0 ascii file.

2. Manifold triangle meshes

In JMeshLib, particular care has been taken in maintaining a neat separation between connectivity and geometry. For this reason we make use of some notation adapted from [3], and denote a **triangle mesh** as a pair (*P*,*K*), where *P* is a set of *N* point positions $p_i = (x_i, y_i, z_i) \in R^3$ with $1 \le i \le N$, and *K* is an abstract simplicial complex which contains all the topological information. The complex *K* is a set of subsets of $\{1, ..., N\}$. These subsets are called simplices and come in 3 types: vertices $v = \{i\}$, edges $e = \{i,j\}$, and triangles $t = \{i,j,k\}$, so that any non-empty subset of a simplex of *K* is again a simplex of *K*, e.g., if a triangle is present so are its edges and vertices.

The abstract simplicial complex *K* describes a topology [4], or connectivity, on *P*. We refer to *P* as to the **geometry** of the triangle mesh *M*=(*P*, *K*), while we call **connectivity**, or topology, of *M* the connectivity defined on *P* through *K*. We say that *M* is *combinatorially manifold* iff *K* is a combinatorial manifold [5]. In its turn, *K* is a combinatorial manifold iff all its vertices are manifold, and a vertex of *K* is manifold if its neighborhood is homeomorphic to a disk in the topology of *K*.
A simplex σ of cardinality k+1 is also called a *k-simplex*. For each k-simplex σ we define a function φ:

$$\varphi : [0,k] \subset \mathrm{N} \to \mathrm{N} \text{ s.t. } \sigma = \bigcup_{i \in [0,k]} \varphi(i).$$

Now, for each k-simplex σ in *K*, let us consider the subset of R^3 formed by the points *x* that can be expressed as the convex combination of the vertex positions of σ:

$$x = \sum_{i=0}^{k} l_i p_{\varphi(i)}, \text{ with } \sum_{i=0}^{k} l_i = 1, l_i \ge 0$$

We refer to the union of all such subsets as to the *geometric realization* $S \subset R^3$ of the triangle mesh *M=(P,K)*. Thus, the geometric realization is a set of points of R^3 for which an Euclidean topology exists, and we say that *S* is manifold iff the neighborhood of each point in *S* is homeomorphic to a disk. Throughout the reminder of this paper we say that *M* is *geometrically* manifold, or manifold in the Euclidean sense, if *S* is manifold with respect to the Euclidean topology.
Note that a triangle mesh may be manifold in the combinatorial sense and not in the Euclidean one, for example when the mesh self-intersects. Also, a geometrically manifold mesh may be not combinatorially manifold. To obtain such a model, for example, start from a triangle mesh which is both combinatorially and geometrically manifold, pick an edge $e = \{i,j\}$, add a new triangle $t = \{i,j,k\}$ and set $p_k = p_j$.
If we relax the requirement of homeomorphism with a disk to the weaker condition of homeomorphism with a disk **or** with a half-disk, we say that *M* is manifold *with boundary*, which holds both in the Euclidean and in the combinatorial sense.
We define an *orientation* of an edge as an ordering of its two vertices. Furthermore, we call an *orientation* of a triangle an equivalence class of ordering of its vertices where $(v_1, v_2, v_3) \sim (v_{\psi(1)}, v_{\psi(2)}, v_{\psi(3)})$ are equivalent orderings if the parity of the permutation ψ is even. Two triangles sharing an edge *e* are *consistently oriented* if they induce different orientations on *e*. A triangle mesh is *orientable* iff all its triangles can be oriented consistently.

3. A data structure for manifold triangle meshes

The definition of a data structure for boundary representations, such as triangle meshes, requires the coding of topological entities (with the associated geometric information) and of a suitable subset of the topological relationships between such entities. In particular, it is desirable that all of the following requirements are satisfied:

- The structure must be **complete**, that is, it must be possible to extract *all* of the entities and relationships which are not explicitly stored, without ambiguity.

- The structure should be **non-redundant**, that is, if an entity (or a relationship) can be computed in *optimal* time, then it should not be explicitly coded in the data structure.
- Each relationship which is not explicitly stored must be **computable in optimal time**, that is, the number of operations required must be linear in the number of elements of the relationship.

3.1 Scheme of the relationships

A topological relation is essentially a function which associates to each element σ of a given type (a vertex, an edge or a triangle) the set of all the elements of another given type having a topological connection with σ. For example, if V is the set of vertices of a triangle mesh M = (V,E,T), then the set of pairs VE = $\{(v, Y) \mid v \in V, Y \subseteq E \text{ and } \forall \varphi \in Y, v \subset \varphi\}$ is the topological relationship which relates each vertex with the set of all the edges incident at it. Clearly, the set of pairs VE may be viewed as a function VE: $V \rightarrow \wp(E)$ which maps each element of V into a subset of E.

Let M = (V,E,T) be a manifold triangle mesh. The following Figure 1 depicts all of the possible relationships between elements of M. Notice that a good data structure should not store them all in order to avoid redundancy.

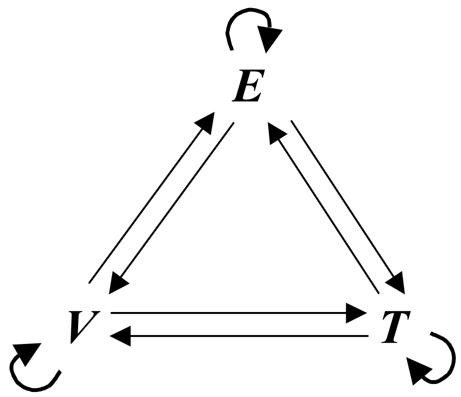

Figure 1: A scheme of all of the possible relationships between topological entities in a triangle mesh.

Let *v* be a vertex. VV(*v*) is the set of all the vertices which are connected to *v* through an edge[1]. VE(*v*) is the set of all the edges which are incident at *v*. VT(*v*) is the set of all the triangles of which *v* is a vertex.

Let *e* be an edge. EV(*e*) is the set containing the two ending vertices of *e*. ET(*e*) is the set of the triangles of which *e* is an edge. Notice that, since we consider only manifold triangle meshes (possibly with boundary), each set ET(*e*) contains either two elements (when *e* is an internal edge) or one element (when *e* is a border edge). The set EE(*e*) is the set of the edges bounding the triangles of which *e* is an edge, excluding *e* itself. Thus, if *e* is an internal edge, the set EE(*e*) contains four edges, while if *e* is on the boundary, |EE(*e*)| = 2.

Let *t* be a triangle. The set TV(*t*) is the set of the three vertices of *t*. TE(*t*) is the set of the three edges bounding *t* and, finally, TT(*t*) is the set of the triangles sharing an edge with *t*.

Moreover, a topological relation may map each element of its domain into a set of constant or variable cardinality. Hence, when dealing with closed triangle meshes, one can classify the relationships in:

A. **Constant relations.** These relations map edges and triangles into sets of neighboring elements with constant cardinality. Specifically, |EV(e)| = 2, |EE(e)| = 4, |ET(e)| = 2, |TV(t)| = 3, |TE(t)| = 3 and |TT(t)| = 3.

B. **Variable relations.** These relations are vertex-based and the cardinality of the set may vary depending on which vertex is being mapped (VV, VE and VT).

[1] The set VV(*v*) is also known as the *link* of *v*.

In the design of a data-structure, however, it is useful to extend the concept of constant relation to the case of manifold triangle meshes with boundary. In fact, although the cardinality of some image-sets is no longer *constant*, it can assume a finite number of values. Specifically the cardinality of the EE may be 2 or 4, the one of the ET may be 1 or 2, and the one of the TT may be 0, 1, 2 or 3. All the others are still *properly* constant.

3.2 A non-redundant data structure

Clearly, a data-structure coding all the relations depicted in Figure 1 is redundant. Conversely, when dealing with manifold triangle meshes with boundary, the scheme depicted in Figure 2 meets all of the requirements discussed above [1].

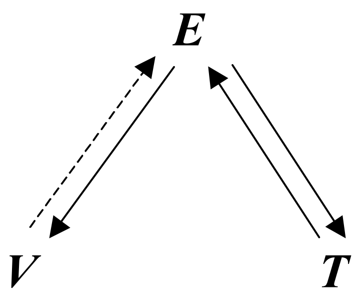
$$\begin{tikzcd}
 & E \arrow[dl] \arrow[dr, shift left] & \\
V \arrow[ur, dashed] & & T \arrow[ul, shift left]
\end{tikzcd}$$

Figure 2: Scheme of relationsused in JMeshLib from which it is possible to derive all of the other (non-stored) relations in optimal time. The dotted line representing the VE indicates that such a relation is only partially stored.

In Figure 2 the VE is indicated with a dotted line, meaning that such a relation is only partially stored. From now on, we denote with VE* such a restricted VE. VE*(*v*) maps *v* into **one of its incident edges**. The complete VE(*v*) can be computed starting from the VE*(*v*) by "turning around" *v* through successive applications of the coded relations, keeping track of the already traversed triangles. In particular, the initial VE is initialized as the VE*, then choose one triangle of the ET(VT*(v)), let it be *t*, and choose the edge *e* of TE(*t*) such that v ∈ EV(*e*) and e ≠ VE*. Now add *e* to the set VE and repeat the same operations by considering *e* as the new VE*. If *v* is not on the boundary, the process terminates when *e* becomes equal to the original VE*(*v*). If *v* is on the boundary, *e* may become a boundary edge; In this case, the process continues by considering the unconsidered triangle of ET(VE*(*v*)) and turns in the opposite sense. An example of this process is depicted in the following Figure 3.

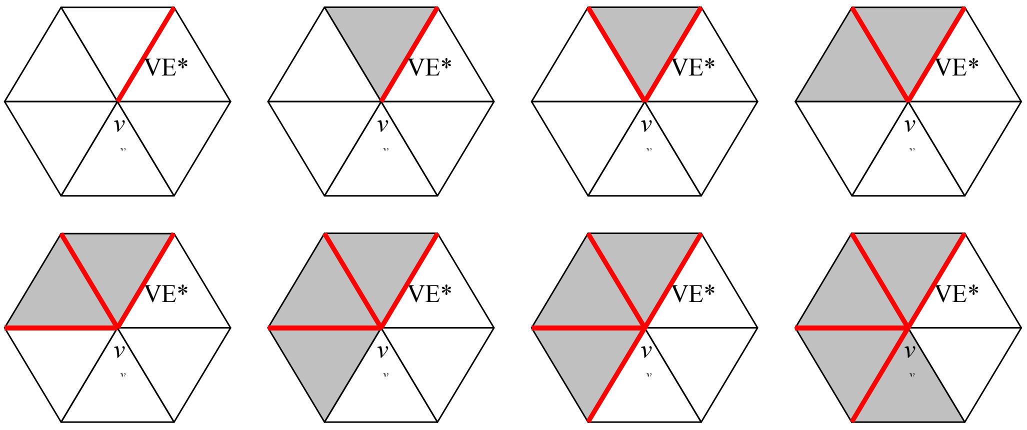

Figure 3: Reconstruction of the VE relation starting from the VE.*

Note that this process requires a number of operations that is linearly proportional to the number of elements of the final VE, therefore it is optimal.

All the other relations which are not explicitly stored in the data structure may be derived in optimal time as follows:

C. VE(v) = construction described above;

D. VV(v) = $\{w \in EV(e) \mid e \in VE(v)$ **and** $w \neq v\}$

E. VT(v) = $\{t \in ET(e) \mid e \in VE(v)\}$

F. EE(e) = $\{f \in TE(t) \mid t \in ET(e)$ **and** $f \neq e\}$

G. TV(t) = $\{v \in EV(e) \mid e \in TE(t)\}$

H. TT(t) = $\{y \in ET(e) \mid e \in TE(t)$ **and** $y \neq t\}$

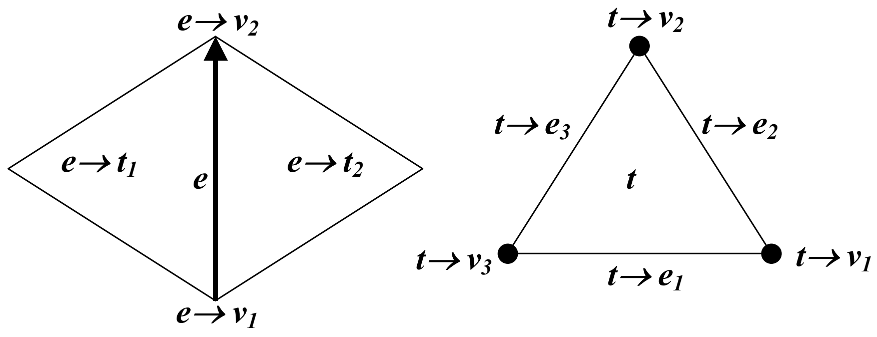

*Figure 4: Orientation of the neighbors of an edge **e** and of a triangle **t** as stored within JmeshLib's data structure*

4. Loading polygon meshes

While loading, the data structure is initialized. JMeshLib data structure has been optimized to efficiently manage manifold and oriented meshes, possibly with boundary. Most graphic formats supported by the loader (VRML, OFF, IV, …), however, may represent non-manifold and/or non-orientable sets of polygons. In this case the loader runs the algorithm described in [2]. If the resulting manifold surface is not oriented, however, JMeshLib assigns an orientation to one triangle for each connected component, and propagates the orientation to neighboring triangles; once all of the triangles have been visited, the mesh is cut along edges having non-consistently oriented incident triangles. Further operations include the triangulation of non-triangular faces, the removal of isolated vertices, and the duplication of non-manifold vertices.
All the operations described are performed automatically by the loader, so that the user always works on manifold and oriented meshes.

Bibliography

[1] Bruzzone, E. and De Floriani, L. 1990. *Two Data Structures for Constructing Tetrahedralizations*. The Visual Computer, 6, 5, 266-283.

[2] Guéziec, A., Taubin, G., Lazarus, F. and Horn, B. 2001. *Cutting and stitching: Converting sets of polygons to manifold surfaces*. IEEE Transactions on Visualization and Computer Graphics, 7, 2, 136–151.

[3] Lee, A. W. F., Sweldens, W., Schröder, P., Cowsar, L. and Dobkin, D. 1998. *MAPS: Multiresolution adaptive parameterization of surfaces*. In Proceedings of ACM SIGGRAPH '98, 95-104.

[4] Munkres, J. R. 2000. *Topology.* Prentice Hall, New Jersey, USA.

[5] De Floriani, L., Morando, F. and Puppo, E. 2003. *Representation of Non-manifold Objects through Decomposition Into Nearly manifold parts.* In Proceedings of ACM Solid Modeling '03, 304-309.