TetGen
Develop Document

Hang Si

August 17, 2008

Abstract

This document is used for developing `TetGen`.

Contents

Chapter 1

Mesh Data Structure

Let Ω be a three-dimensional domain with boundary $\partial\Omega$. A tetrahedral mesh of Ω is a three-dimensional simplicial complex $\mathcal{T}$. It contains a subcomplex $\mathcal{F}$ which is the surface mesh of $\partial\Omega$. $\mathcal{F}$ a two-dimensional simplicial complex.

For storing and maintaining a tetrahedral mesh $\mathcal{T}$ as well as its subcomplex $\mathcal{F}$, we use the following lists of elements:

- a list of vertices of $\mathcal{T}$;
- a list of tetrahedra of $\mathcal{T}$;
- a list of subfaces of $\mathcal{F}$; and
- a list of subsegment of $\mathcal{F}$.

Where subfaces and subsegments are those faces and edges belong to both $\mathcal{T}$ and F, respectively. Since we store all tetrahedra, there is no need to store faces and edges of $\mathcal{T}$. While subfaces and subsegments are stored explicitly.

There are four types of mesh elements: vertices, tetrahedra, subfaces, and subsegments. The elements of all the four types consist of a tetrahedral mesh of a 3D domain. Three data structures are declared: *tetrahedron*, *shellface*, and *point*. A tetrahedron is a tetrahedron; a shellface can be either a subface or a subsegment; and a point is a point. These three data types, linked by pointers comprise a mesh.

1.1 Point

The *point* data structure is a list of floating-point numbers. This list contains the following data of the point:

- 3 coordinates;
- n user-defined attributes (optional);

- 1 user-defined metric tensor (optional);
- 1 pointer to a simplex (tet, tri, edge);
- 1 pointer to a parent (or duplicate) point;
- 1 pointer to a tet in background mesh (optional);
- 1 integer index (also for boundary marker);
- 1 integer for the vertex type.

1.2 Shellface

The *shellface* data structure represents either a subface or a subsegment. It contains the following data (see Fig. **??** left):

- 3 pointers to adjacent subfaces;
- 3 pointers to its points;
- 2 pointers to adjacent tetrahedra;
- 3 pointers to adjacent subsegments;
- 1 pointer to a badface (optional);
- 1 double of maximum area (optional);
- 1 integer of boundary marker;
- 1 integer of type;

Fig. 1.1 shows a subface with its adjacent subfaces, subsegments, and tetrahedra. Note that the three adjacent subsegments is not always exist, if so, they are set to `dummysh`).

Note that a shellface is also used to store a subsegment. In such case, the third point of it is set to `null`. And it has only one pointer to an adjacent subface. Since there may be arbitrary number of subfaces containing a subsegment, chose one arbitrarily.

1.3 Tetrahedron

The *tetrahedron* data structure represent a tetrahedron. It contains the following data (see Fig. 1.2 right):

- 4 pointers to adjacent tetrahedron;
- 4 pointers to its points;

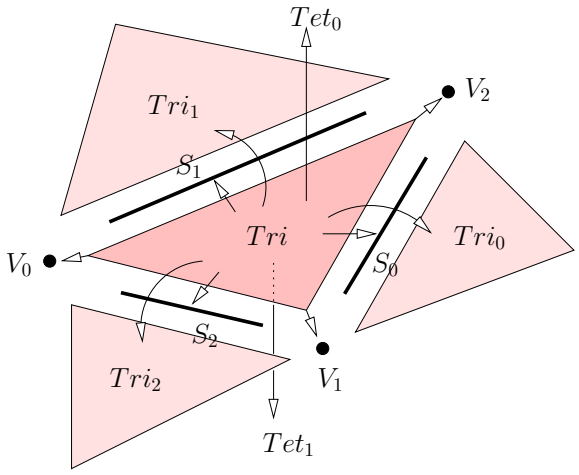


Figure 1.1: The shellface data structures.

- 4 pointers to adjacent subfaces (optional, -p, -r options);
- 2 pointers to adjacent subsegments (optional, -p, -r options);
- n user-defined attributes (optional);
- 1 double of maximum volume (optional);
- 1 pointer to a list of higher order nodes (-o2 option);

1.4 Mesh Handles

Two special data types, 'triface' and 'face' are defined for maintaining and updating meshes. They are like pointers (or handles), which allow you to hold one particular part of the mesh, i.e., a tetrahedron, a triangle, an edge and a vertex. However, these data types do not themselves store any part of the mesh. The mesh is made of the data types defined above.

1.4.1 Face

Let the three vertices of a triangle be ordered and numbered as v_0, v_1, and v_2. The six versions of the directed edges (i, j) of this triangle (see Fig. 1.3) are defined as follows:

0	(v_0, v_1)
1	(v_1, v_0)
2	(v_1, v_2)
3	(v_2, v_1)
4	(v_2, v_0)
5	(v_0, v_2)

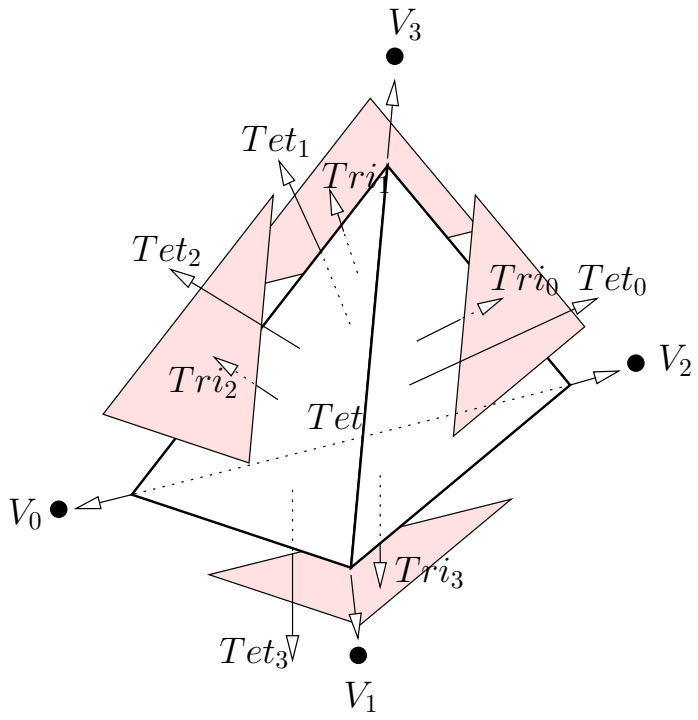


Figure 1.2: The tetrahedron data structures.

The *face* handle holds a directed edge of a shellface. A set of mesh manipulation primitives are defined for various operation on faces.

1.4.2 Triface

Let the four nodes of a tetrahedron be ordered and numbered from $v_0, ..., v_3$. The order of the nodes is chosen in such a way that the signed volume of the tetrahedron evaluated by $v_0, ..., v_3$ is negative. Intuitively, the last node (v_3) "sees" the first three nodes (v_0, v_1, v_2) in counterclockwise order.

Let the four faces of a tetrahedron be numbered from $f_0, ..., f_3$. Each face has 6 directed edges (edge versions are the same as in Fig. 1.3). We stipulate the four faces and the directed edges in each face as in Fig. 1.4.

The *triface* handle holds a directed edge of a terahedron. A set of mesh manipulation primitives are defined for various operation on trifaces.

The mesh data structures additionally store geometric informations which help for fast queries.

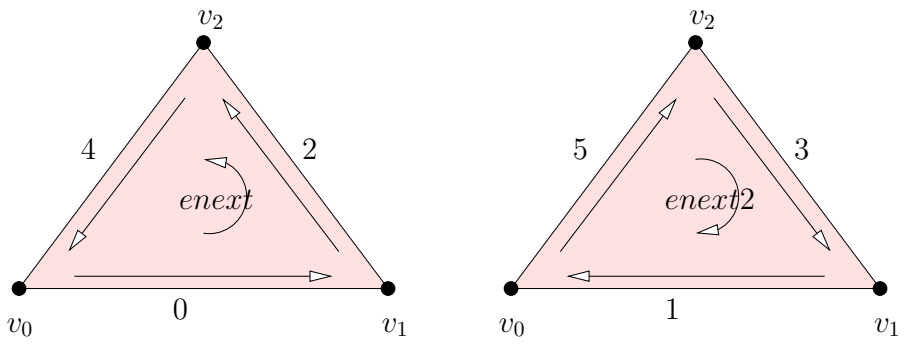
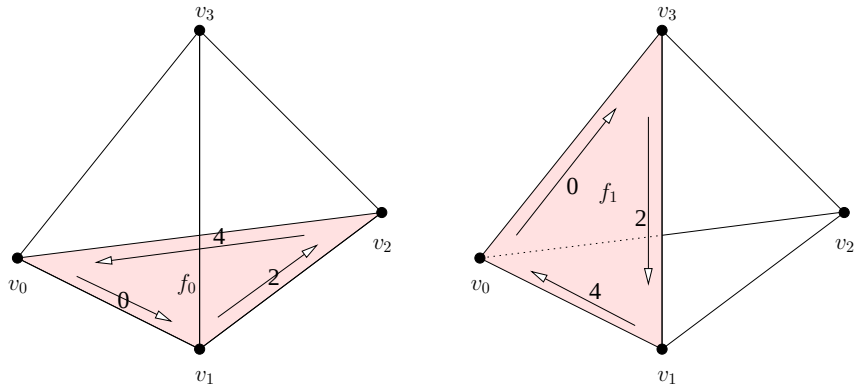
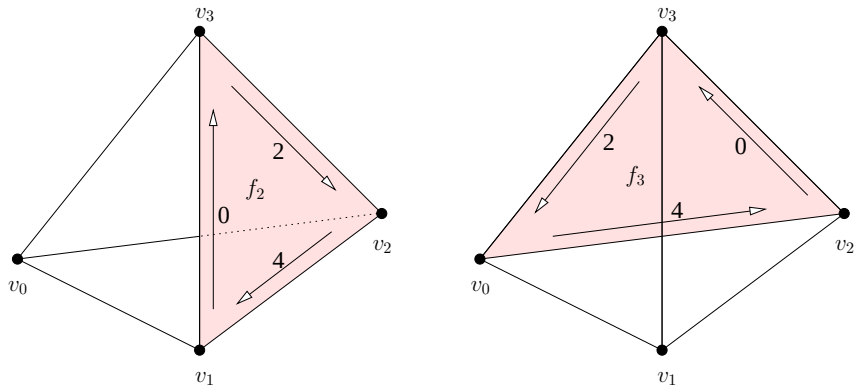


Figure 1.3: The six versions of the directed edges of a triangle.



$f_0 : (v_0, v_1, v_2)$ $f_1 : (v_0, v_3, v_1)$



$f_2 : (v_1, v_3, v_2)$ $f_3 : (v_2, v_3, v_0)$

Figure 1.4: The faces and directed edges of a tetrahedron.

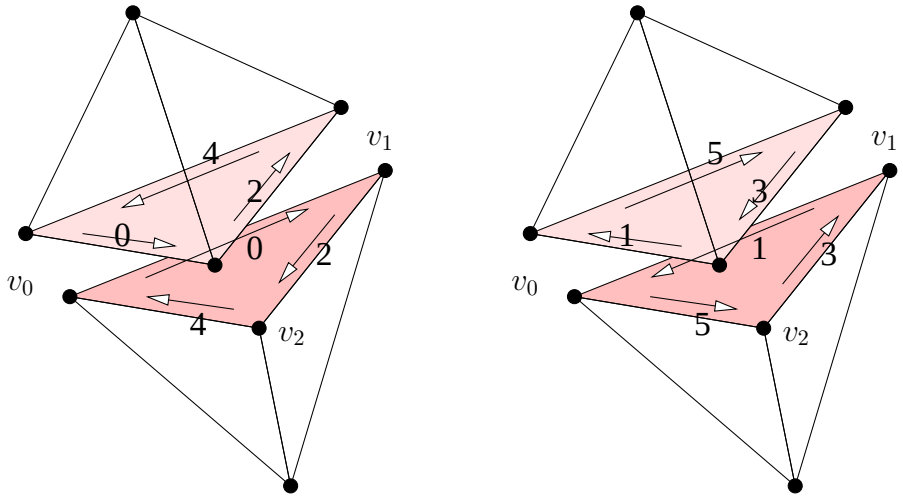


Figure 1.5: The `fnext()` primitive.