Triangle Meshes: Summary

1 Definitions

Disclaimer: Not all authors agree on the terminology and definitions provided below, but the concepts introduced here are commonly used and important. Hence, you must understand these definitions and learn them by heart.

A triangle mesh (**mesh**) is a collection of **cells**: **vertices** (0-cells), **edges** (1-cells), and **triangles** (2-cells).

Each edge is the line segment between 2 vertices and is **relatively open** (it does not contain its end-points).

Each triangle is the relative-interior of the convex hull of 3 vertices (does not contain its bounding edges, vertices).

A mesh is **consistent** if all cells are pair-wise disjoint.

A mesh is **clean** if each edge and vertex bounds a triangle of the mesh (**no hair**) and if the edges and vertices bounding each triangle are part of the mesh (**no cuts**).

We say that a triangle is **incident** upon its **bounding** edges and vertices.

A mesh is **edge-manifold** if each edge is bounding either one or two triangles.

Two triangles are **adjacent** if they share a bounding edge.

A mesh is **edge-connected** if for any two of its triangles T_1 and T_n there is an ordered list of triangles $\{T_1,T_2...,T_n\}$ such that any two consecutive triangles in the list are adjacent.

The **star** of a vertex is the set of all triangles and edges it bounds.

A **vertex** is **manifold** if its star is edge-manifold and edge-connected.

A **mesh** is **manifold** if all its vertices are manifold.

A mesh is **watertight** if each edge bounds an even number of triangles.

A **shell** is an edge-connected, watertight, manifold mesh.

The **genus** (number of handles) of a shell is $H=T/4–V/2+1$, where T is the number of triangles and V the number of vertices.

A mesh is **simple** (a topological sphere) if it is a shell with genus zero. Then, **$T=2V–4$**. It can be drawn as a consistent mesh on the plane (planar triangle graph).

Given a vertex v and an incident triangle t, v'=nvat(v,t) returns the **next vertex around triangle** t.

Given a triangle t and a bounding vertex v, t'= ntav(t,v) returns the **next triangle around vertex** v or null if t' does not exist.

A manifold mesh is **oriented** if for each pair of adjacent triangles t_1 and t_2, sharing vertices v_a and v_b, either we have v_b=nvat(v_a,t_1) and v_a=nvat(v_b,t_2), or we have v_b=nvat(v_a,t_2) and v_a=nvat(v_b,t_1). Note that a shell that is not consistent may not be **orientable** (Klein bottle).

2 Representation and queries

Popular representations of a mesh distinguish geometry from connectivity. *You must understand what these concepts mean and why they are important.*

Geometry: An **array G[] of points** representing the vertex locations. Typically the **order is arbitrary** but fixed. Hence, **each vertex is associated with an integer index**: V_0=G[0], V_1=G[1], …

Connectivity: **Additional** information providing **constant cost** support of **queries** such as:

- **Incidence**: access the three **vertices of a triangle t**, **in nvat order** (used for **rendering** the mesh)
- **Adjacency**: access the three **triangles adjacent to triangle t** (used for **traveling** on the mesh)
- **Star**: Access the **triangles incident upon a vertex v, in ntav order** (used for computing vertex **normals**)

A variety of representation schemes and low-level operators have been proposed for the connectivity.

For **edge-manifold meshes**, will use the **Corner Table**.

Each triangle has 3 **corners**, each one is **incident upon** (associated with) a different vertex. Hence, a corner is the association of a triangle with one of its bounding vertices. **A mesh has 3T corners**. **In a simple mesh, a vertex has, on average, about 6 incident corners.**

Our representation of the connectivity, the low-level queries, and most algorithms operate on <u>corners</u>.

Each **triangle** is associated with an **integer triangle index** in [0,T–1] and each vertex is associated with an **integer vertex index** in [0,V–1]. Each corner is associated with an **integer corner index** in [0,3T–1].

From now on, when we say "**corner**" we mean the "**integer corner index**" of that corner, we say "**triangle**" we mean the "**integer triangle index**" of that triangle, we say "**vertex**" we mean the "**integer vertex index**" of that vertex.

Given a corner c, we support the following **primary operators**:

t(c) is the **triangle** associated with c

v(c) is the **vertex** associated with c

n(c) is the **next** corner around t

o(c) is the **opposite** corner b, such that either v(n(c))=v(p(b)) and v(p(c))=v(n(b)), or if none can be found b=c

From these, we derive the following convenient **secondary operators**.

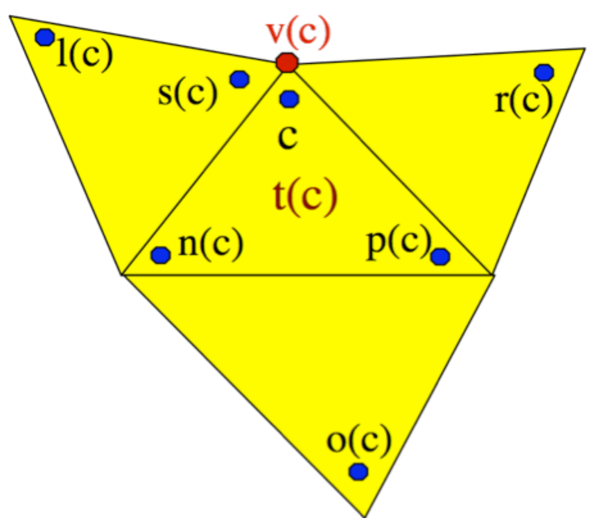
g(c) is the point where vertex v(c) is located: g(c)=G[v(c)]

p(c) is the **previous** corner around t: p(c)=n(n(c))

l(c) is the **left neighbor** of c: l(c)=o(p(c))

r(c) is the **right neighbor** of c: r(c)=o(n(c))

s(c) is the **swing** of c (next corner around v(c)): s(c)=p(l(c)). Useful to walk around hole border loops.



3 Implementation

The mesh is stored as 3 arrays:

pt G[V]: an array of **points**, one per vertex

int V[3T]: an array of **integer vertex indices**, 3 per triangle

int O[3T]: an array of **integer corner indices**, 3 per triangle

We cache v(c) in V[c] and o(c) in O[c]. The three corners {a,b,c} of each triangle are consecutive in these tables and are stored in an order such that b=n(a) and c=n(b).

O[c]=c when c has **no opposite**. *(This is new! In the papers and notes, I used to set O[c]=– 1 when c has no opposite.)*

```
class Mesh {
  int nv;                  // number of vertices
  pt G[nv];               // geometry (vertices)
  int nt;                  // number of triangles
  int nc;                  // number of corners (3 per triangle)
  int V[nc];              //  corner/vertex incidence
  int O[nc];              // opposite corners
int t (int c) {return int(c/3);};        // triangle of corner
int n (int c) {return 3*t(c)+(c+1)%3;};   // next corner in the same t(c)
int p (int c) {return n(n(c));};  // previous corner in the same t(c)
int v (int c) {return V[c] ;};   // id of the vertex of c
pt g (int c) {return G[v(c)];};  // point of the vertex v(c) of corner c
boolean b (int c) {return O[c]==c;};      // if faces a border (has no opposite)
int o (int c) {return O[c];}; // opposite (or self if border)
int l (int c) {return o(n(c));}; // left neighbor or next if b(p(c))
int r (int c) {return o(p(c));}; // right neighbor or next if b(r(c))
int s (int c) {return p(l(c));}; // swings around v(c) or around a border loop

// Additional book-keeping masks and attributes
int [] color = new int[nt];   // color of triangles (0 means invisible)
boolean[] Mt = new boolean[nt];   // mask indicating that triangle was visited
boolean[] Mv = new boolean[nt];   // mask indicating that vertex was visited
vec[] N = new vec[nv];   // vertex normal
...}
```

Algorithms that students should know how to re-invent:

Compute the O table from the V table of a manifold mesh
Identify the non-manifold edges and vertices of a mesh
Compute the estimate of the normal to a vertex
Compute the number of edge-connected components in a manifold mesh
Identify the edge-connected components of a manifold mesh and tag the triangles with the component number
Compute the number of holes in an edge-connected manifold mesh
Trace the border of a hole and make a triangle fan to fill it
Flip an edge identified by one of its opposite corners
Collapse an edge identified by one of its opposite corners
Compute the concentric rings around a seed triangle T and mark their triangles with a graph distance from T
Identify the concave edges of a manifold mesh
Given a watertight manifold mesh representing the boundary of a solid, orient its shells
Compute the genus of a shell
Smooth a shell