Mesh data structures

Denis Zorin

Types of meshes

- Most general: polygon soup; polygons may have arbitrary number of vertices, vertices and edges can be shared by arbitrary polygons.
- Manifold polygonal meshes
  the union of all faces adjacent to a vertex can be continously deformed to a disk
- Orientable: orientation on faces can be chosen consistently
- Triangular meshes: all faces are triangles

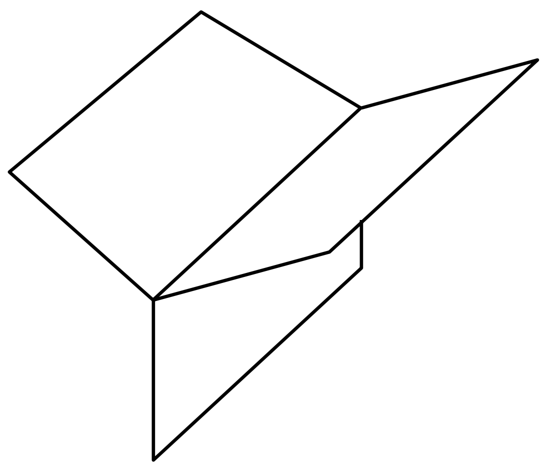
Types of meshes

Manifold

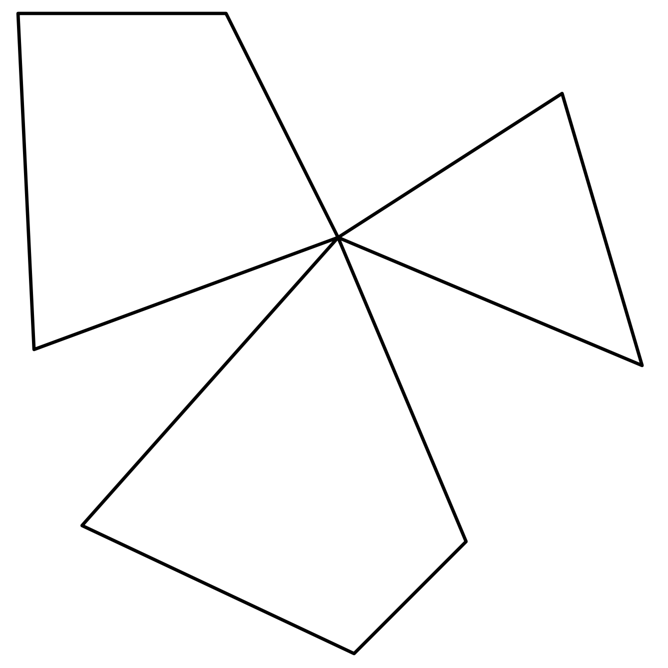
- Common assumption for many algorithms
- For a given vertex v, adjacent faces F_i can be ordered so that their vertices ≠ v form a simple chain, i.e. no 2 vertices coincide and two sequential are connected by an edge
- Each edge is shared by no more than 2 faces;

Types of meshes

Nonmanifold features



three faces sharing an edge



outer vertices do not form a simple chain

Types of meshes

Manifold property is commonly required, but available meshes often violate it;

- Solution: convert nonmanifold meshes to manifold by splitting along edges and vertices
- Increases mesh-parsing code complexity
- Arbitrary polygonal meshes often converted to triangular

Basic mesh queries

The choice of the data structure is determined by the elementary operations that have to be supported efficiently

- Simplest choice for a triangle mesh: list of vertex x,y,z coordinates, list of triples of vertex indices i1,i2,i3 for each triangle
- Finding two faces adjacent to an edge requires traversing the whole triangle list

Basic mesh queries

Adjacency:

- FV all vertices of a face
- EV both vertices of an edge
- VF all faces sharing a vertex
- EF all faces sharing an edge
- FE all edges of a face
- VE all edges sharing a vertex

For a list-of-triangles representation, only FV, EV, FE are efficient

Mesh modification

- Add/remove face, edge or vertex
- Split face or edge

To keep the mesh manifold (or triangular), operations need to be performed in a consistent way;

e.g. an edge split requires one or two face splits in a triangular mesh

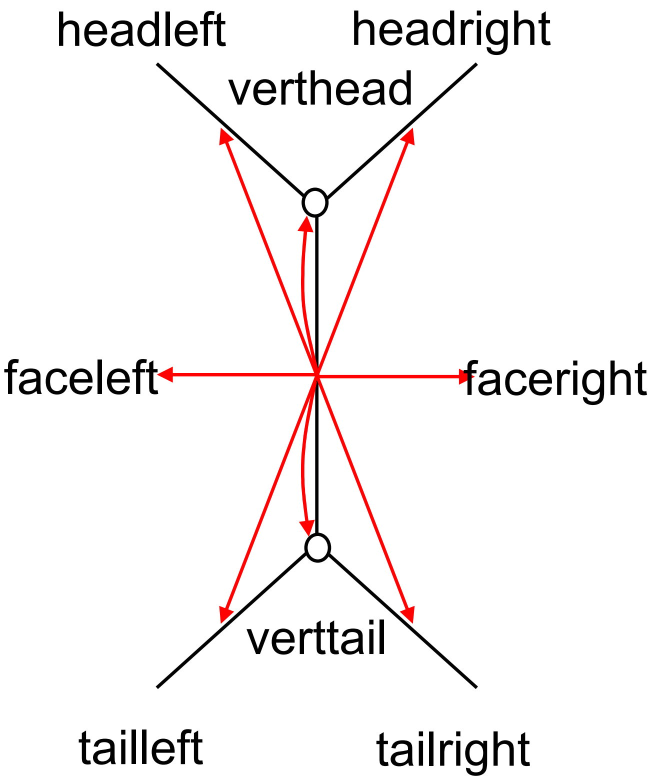
Mesh data structures

Typical requirement: constant storage per data structure

- For manifold meshes, only the number of faces per edge and vertices per edge requires constant storage;
- Most popular data structures for storing adjacency information are edge-based
- Face-based are also used for triangular meshes (vertices and edges per face are constant)

Winged-edge

```
struct Edge {
   Edge *headleft,*headright,
*tailleft,*tailright;
   Face *faceleft,*faceright;
   Vertex *verthead, *verttail;
   // edge data
} ;
struct Face {
  Edge* edge;
  // face data
};
struct Vertex {
  Edge* edge;
 // vertex data
};
```



red arrows indicate pointers

Winged-edge

6-8 pointers per edge E

- 4 to next/previous edges in two faces sharing E
- 2 to faces (if information is stored in faces)
- 2 to vertices (if information is stored in vertices)

Vertices and faces store

a single pointer to an edge

Pointer storage for a regular triangular grid:

faces ~ 2 #vertices

edges ~ 3#vertices

If #vertices = N

~27*N pointers need to be stored

Winged-edge

Trivial: EF, EV

FE (all edges of a face)

```
e0 = f->edge; e = e0;
do {
  if(e->faceleft== v)
    e = edge->headleft;
  else
    e = edge->tailright;
 } while (e != e0);
```

FV and VF are similar

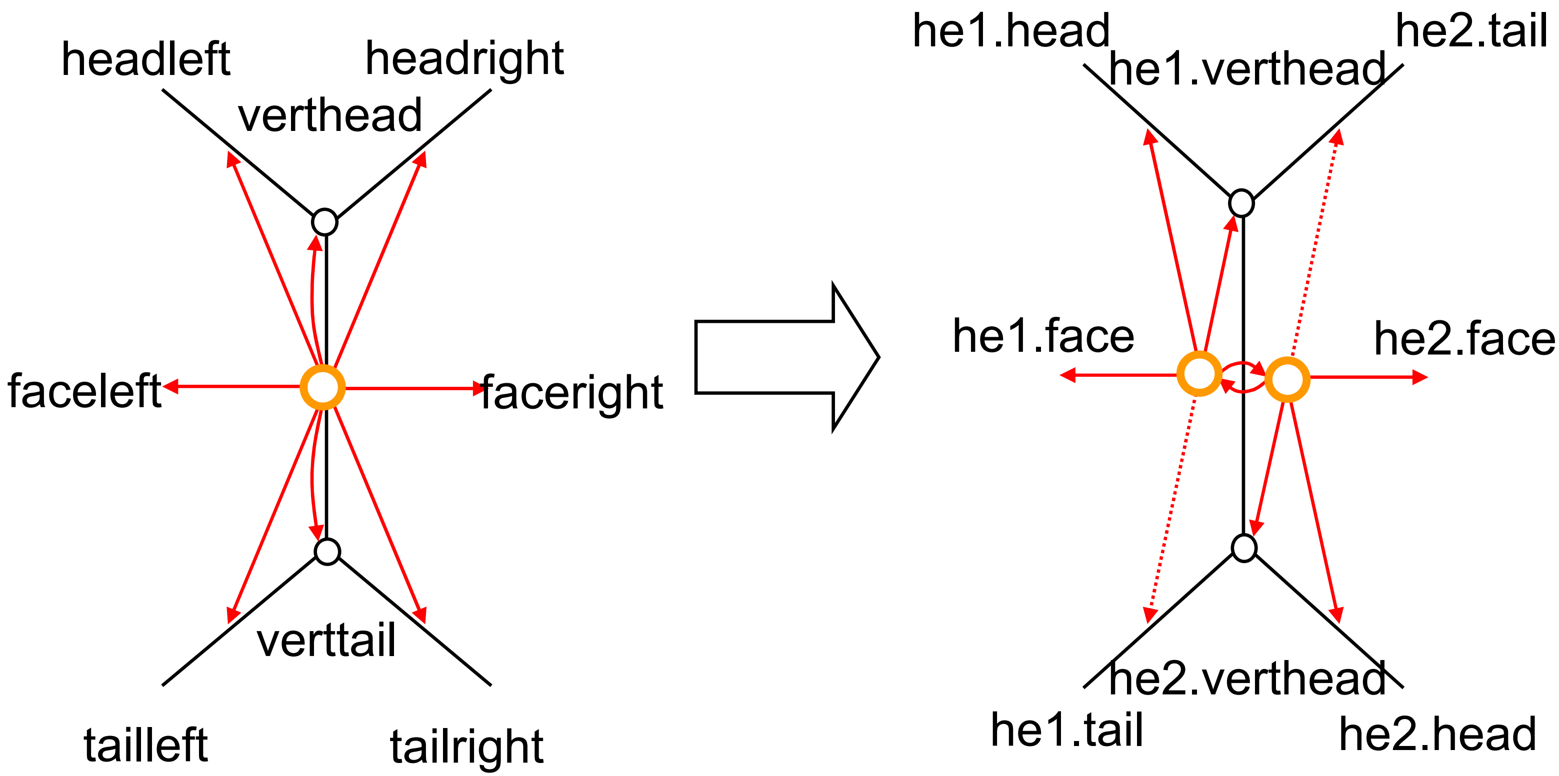
VE (all edges sharing a vertex, interior vertices)

```
e0 = v-> edge; e = e0;
do {
     if(e->verthead == v)
       e = edge->headright;
     else
       e = edge->tailleft;
   } while (e != e0);
```

Half-edge

Split each winged-edge data structure into 2;

- advantage: FE, VE traversals do not require "ifs" in code, consistent orientation



Half-edge

```
HalfEdge {
   HalfEdge* head, *tail;
   // tail pointer is optional
   HalfEdge* opposite;
   Face* face;
   Vertex* verthead;
};
```

FE (all edges of a face)

```
e0 = f->halfedge; e0 = e;
do { e = e->head; }
while (e != e0);
```

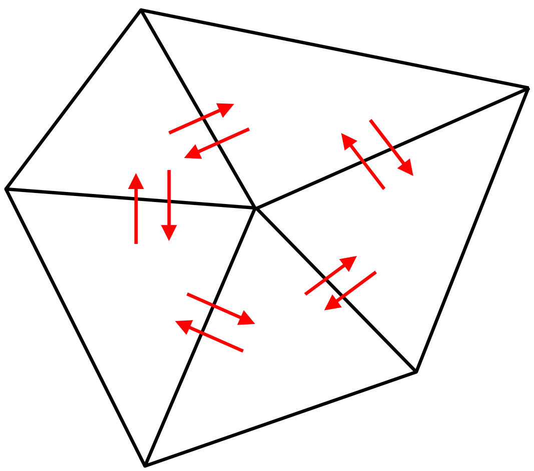
VE (all edges sharing a vertex, interior vertices)

```
e0 = v->halfedge;  e0 = e;
do {
  e = e->opposite->head;
} while ( e!= e0);
```

Both traversals do not require if’s

Face-based data structure

Primarily for triangle meshes



```
Face {
  Face*nbr[3];
  Vertex* vert[3];
}
```

6 pointers per triangle

1 per vertex, no edge records

(3/2*6+1)*N = 10*N vs. 27N for winged-edge